

Robot Builder's Bonanza

Over 100 projects!



Fourth Edition

- 10 all-new robot construction plans
- Updated with the latest technologies and techniques
- Now with bonus online projects and tutorials



Gordon McComb

ROBOT BUILDER'S BONANZA

ABOUT THE AUTHOR

Gordon McComb has written 65 books and thousands of magazine articles—over a million copies of his books are in print, in more than a dozen languages. For 13 years, Gordon wrote a weekly syndicated newspaper column on computers and high technology, which reached several million readers worldwide. He's a regular contributor to *SERVO Magazine* and other publications, and maintains an active Web site dedicated to teaching the art and science of robot building.

ROBOT BUILDER'S BONANZA

GORDON McCOMB

FOURTH EDITION

McGraw-Hill
New York Chicago San Francisco Lisbon London Madrid
Mexico City Milan New Delhi San Juan Seoul
Singapore Sydney Toronto

Copyright © 2011, 2001, 1987 by Gordon McComb. All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

ISBN: 978-0-07-175035-6

MHID: 0-07-175035-5

The material in this eBook also appears in the print version of this title: ISBN: 978-0-07-175036-3,

MHID: 0-07-175036-3.

All trademarks are trademarks of their respective owners. Rather than put a trademark symbol after every occurrence of a trademarked name, we use names in an editorial fashion only, and to the benefit of the trademark owner, with no intention of infringement of the trademark. Where such designations appear in this book, they have been printed with initial caps.

McGraw-Hill eBooks are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. To contact a representative please e-mail us at bulksales@mcgraw-hill.com.

Information contained in this work has been obtained by The McGraw-Hill Companies, Inc. ("McGraw-Hill") from sources believed to be reliable. However, neither McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein, and neither McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that McGraw-Hill and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

TERMS OF USE

This is a copyrighted work and The McGraw-Hill Companies, Inc. ("McGrawHill") and its licensors reserve all rights in and to the work. Use of this work is subject to these terms. Except as permitted under the Copyright Act of 1976 and the right to store and retrieve one copy of the work, you may not decompile, disassemble, reverse engineer, reproduce, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish or sublicense the work or any part of it without McGraw-Hill's prior consent. You may use the work for your own noncommercial and personal use; any other use of the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

THE WORK IS PROVIDED "AS IS." MCGRAW-HILL AND ITS LICENSORS MAKE NO GUARANTEES OR WARRANTIES AS TO THE ACCURACY, ADEQUACY OR COMPLETENESS OF OR RESULTS TO BE OBTAINED FROM USING THE WORK, INCLUDING ANY INFORMATION THAT CAN BE ACCESSED THROUGH THE WORK VIA HYPERLINK OR OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. McGraw-Hill and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither McGraw-Hill nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. McGraw-Hill has no responsibility for the content of any information accessed through the work. Under no circumstances shall McGraw-Hill and/or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

*For Lane and Firen,
keeping the legacy alive*

This page intentionally left blank

CONTENTS

Acknowledgments	xxi
Photo and Illustration Credits	xxiii
Introduction	xxv

Part 1—The Art and Science of Robot Building

Chapter 1—Welcome to the Wonderful World of Robotics!	3
What the Adventure Holds	3
Why Build Robots?	4
The Building-Block Approach	6
Lower Costs, Better Bots	6
Skills You Need	7
Do It Yourself, Kits, or Ready-Made?	9
Thinking Like a Robot Builder	12
Chapter 2—Anatomy of a Robot	13
Stationary versus Mobile Robots	13
Autonomous versus Teleoperated Robots	14
Tethered versus Self-Contained Robots	15
So, What’s a Robot, Anyway?	16
The Body of the Robot	17
Locomotion Systems	22
Power Systems	24
Sensing Devices	25
Output Devices	27
Where the Word “Robot” Comes From	27
Chapter 3—Getting Parts	29
Local Electronics Stores	30
Online Electronics Outlets	30
Using FindChips.com to Locate Parts	30

Specialty Online Robotics Retailers	31
Hobby and Model Stores	31
Craft Stores	32
Hardware and Home Improvement Stores	33
Samples from Electronics Manufacturers	33
Finding What You Need on the Internet	33
Shop Once, Shop Smart	34
Haunting the Surplus Store	34
Getting Parts from Specialty Stores	35
Scavenging: Making Do with What You Already Have	36
Getting Organized	37

Part 2—Robot Construction

Chapter 4—Safety First (and Always) 43

Project Safety	43
Battery Safety	44
Soldering Safety	44
Fire Safety	44
Avoiding Damage by Static Discharge	45
Working with House Current	47
First Aid	47
Use Common Sense—and Enjoy Your Robot Hobby	48

Chapter 5—Building Robot Bodies—the Basics 49

Picking the Right Construction Material	49
In Review: Selecting the Right Material	52
Robots from “Found” Parts	53
Basic Tools for Constructing Robots	54
Optional Tools	57
Hardware Supplies	57
Setting Up Shop	58

Chapter 6—Mechanical Construction Techniques 59

First Things First: Eye and Ear Protection	59
Plan, Sketch, Measure, Mark	60
Drilling Holes in Things	60
Cutting Things to Size	64
Using Portable Power Tools	67
Getting Work Done Fast with Air Tools	68

Chapter 7—Working with Wood	69
Hardwood versus Softwood	69
Planks or Ply	69
The Woodcutter’s Art	72
Chapter 8—Build a Motorized Wooden Platform	80
Making the Base	80
Building and Attaching the Motors	83
Building and Mounting the Wheels	84
Attaching the Ball Caster	84
Using the PlyBot	85
Variations on a Theme	85
Chapter 9—Working with Plastic	87
Main Kinds of Plastics for Bots	87
Best Plastics for Robotics	89
Where to Buy Plastic	89
The Ins and Outs of Rigid Expanded PVC	90
How to Cut Plastic	91
How to Drill Plastic	93
Making Plastic Bases	93
Making Plastic Frames	95
How to Bend and Form Plastic	96
How to Smooth the Edges of Plastic	96
How to Glue Plastic	96
Using Hot Glue with Plastics	98
How to Paint Plastics	98
Household Plastics for Bot Constructions	98
Chapter 10—Build a Motorized Plastic Platform	100
Making the Base	100
Attaching the Motors	102
Fitting the Wheels	104
Attaching the Ball Caster	104
Using the PlastoBot	104
Altering the PlastoBot Design	105
Chapter 11—Working with Metal	107
All About Metal for Robots	107
Measuring the Thickness of Metal	109

What's This about Heat Treatments?	110
Where to Get Metal for Robots	110
Recap of Metals for Robotics	110
Metal from Your Home Improvement Store	111
Metal from Craft and Hobby Stores	113
The Metalsmith's Art	114
Chapter 12—Build a Motorized Metal Platform	122
Making the Base	122
Using the TinBot	127
Chapter 13—Assembly Techniques	129
Screws, Nuts, and Other Fasteners	129
Brackets	135
Selecting and Using Adhesives	136
Chapter 14—Rapid Prototyping Methods	144
Selecting Lightweight Robot Materials	144
Cutting and Drilling Substrate Sheets	146
Rapid Construction with Semipermanent Fasteners	147
Chapter 15—Drafting Bots with Computer-Aided Design	152
Making Drilling and Cutting Layouts	152
File Formats for Vector Graphics	158
Using Laser-Cutting Services	158
Producing “Quick-Turn” Metal and Plastic Prototypes	159
Chapter 16—Constructing High-Tech Robots from Toys	160
Erector Sets	160
Fischertechnik	161
K'NEX	162
Other Construction Sets to Try	162
Construction with Snap-Together Components	163
Specialty Toys for Robot Hacking	165
Making Robots from Converted Toy Vehicles	166
Chapter 17—Building Bots from Found Parts	170
A Dozen Ideas to Get You Started	170
Experimenting with “No-Cut” Metal Platform Designs	171
Using Wood and Plastic Samples	175
Keep Your Eyes Peeled and Your Tape Measure Out	176

Part 3—Power, Motors, and Locomotion

Chapter 18—All about Batteries	179
An Overview of Power Sources	179
Batteries for Your Robots	180
Understanding Battery Ratings	183
Recharging Batteries	187
Robot Batteries at a Glance	187
Common Battery Sizes	187
Increasing Battery Ratings	189
Chapter 19—Robot Power Systems	190
Power and Battery Circuit Symbols	190
Using a Premade Battery Pack	191
Making Your Own Rechargeable Battery Pack	192
Using Battery Cells in a Battery Holder	193
Best Battery Placement Practices	195
Wiring Batteries to Your Robot	196
Preventing Reverse Battery Polarity	197
On the Web: How to Solder a Barrel Plug onto a Battery Holder or DC Wall Transformer	198
Adding Fuse Protection	198
Providing Multiple Voltages	199
Regulating Voltage	201
Dealing with Power Brownouts	207
Battery Voltage Monitors	208
Chapter 20—Moving Your Robot	209
Choosing a Locomotion System	209
Locomotion Using Wheels	211
Locomotion Using Tracks	215
Locomotion Using Legs	216
Locomotion Using Other Methods	217
On the Web: Managing the Weight of Your Robot	218
Chapter 21—Choosing the Right Motor	219
AC or DC Motor?	219
Continuous or Stepping Motor?	220
Servo Motors	220
Motor Specs	221

Testing Current Draw of a Motor	224
Dealing with Voltage Drops	228
Avoiding Electrical Noise	229
Chapter 22—Using DC Motors	230
The Fundamentals of DC Motors	230
Reviewing DC Motor Ratings	232
Controlling a DC Motor	232
Motor Control by Switch	232
Motor Control by Relay	234
Motor Control by Bipolar Transistor	239
Motor Control by Power MOSFET Transistor	241
Motor Control by Bridge Module	244
Controlling the Speed of a DC Motor	247
Bonus Projects: Interfacing to Motor Bridge Modules	248
Chapter 23—Using Servo Motors	249
How R/C Servos Work	249
Control Signals for R/C Servos	251
The Role of the Potentiometer	253
Special-Purpose Servo Types and Sizes	253
Gear Trains and Power Drives	254
Output Shaft Bushings and Bearings	254
Typical Servo Specs	255
Connector Styles and Wiring	256
Analog Versus Digital Servos	257
Electronics for Controlling a Servo	258
Using Continuously Rotating Servos	260
Modifying a Standard Servo for Continuous Rotation	261
Using Servo Motors for Sensor Turrets	265
Chapter 24—Mounting Motors and Wheels	266
Mounting DC Motors	266
Mounting and Aligning Motors with Aluminum Channel	269
Mounting R/C Servos	270
Mounting Drivetrain Components to Shafts	272
Mounting Wheels to DC Gear Motors	273
Mounting Wheels to R/C Servos	274

Attaching Mechanical Linkages to Servos	276
Drivetrain Components for Robotics	277
Using Rigid Flexible Couplers	278
Working with Different Shaft Types	282
Everything You Always Wanted to Know about Gears	283
Chapter 25—Robot Movement with Shape Memory Alloy	287
Shape Memory Alloy Comes to Robotics	287
Basics of Shape Memory Alloy	287
Using Shape Memory Alloy	288
Operating SMA Using a Microcontroller	290
Experimenting with SMA Mechanisms	291
Using Ready-Made SMA Mechanisms	292
Part 4—Hands-on Robotic Projects	
Chapter 26—Build Robots with Wheels and Tracks	297
Basic Design Principles of Rolling Robots	297
Two-Motor BasicBot	304
Bonus Project: Double-Decker RoverBot	306
Building 4WD Robots	306
Building Tank-Style Robots	309
Chapter 27—Build Robots with Legs	318
An Overview of Leggy Robots	318
Selecting the Best Construction Material	321
Scratch Build or Parts Kits	322
Leg Power	324
Walking Gaits for Legged Robots	327
Build a 3-Servo Hexapod	328
Creating X-Y Servo Joints	335
Bonus Project: Build a 12-Servo Hexapod	338
Chapter 28—Experimenting with Robotic Arms	339
The Human Arm	339
Degrees of Freedom in a Typical Robotic Arm	340
Arm Types	340
Actuation Techniques	343
Build a Robotic Wrist	344

Build a Functional Revolute Coordinate Arm	345
Build a Robotic Arm from a Kit	350
Chapter 29—Experimenting with Robotic Grippers	352
Concept of the Basic Gripper	352
Two-Pincher Gripper	353
Tool Clamp Gripper	355
On the Web: More Gripper Plans	359
Part 5—Robot Electronics	
Chapter 30—Building Robot Electronics—the Basics	363
Tools for Electronics You Should Have	363
Making Electronic Circuits—the Basics	370
Understanding Wires and Wiring	370
How to Solder	371
Using Headers and Connectors	375
Using Clip-on Jumpers	377
Good Design Principles	377
RoHS Demystified	379
Chapter 31—Common Electronic Components for Robotics	381
But First, a Word about Electronics Symbols	381
Fixed Resistors	382
Potentiometers	388
Capacitors	390
Diodes	394
Light-Emitting Diodes (LEDs)	396
Transistors	399
Integrated Circuits	400
Switches	402
Relays	404
. . . And the Rest	405
On the Web: Stocking Up on Parts	406
Chapter 32—Using Solderless Breadboards	407
Anatomy of a Solderless Breadboard	407
Steps in Constructing a Solderless Breadboard Circuit	411

Making Long-Lasting Solderless Circuits	411
Mounting the Breadboard to Your Robot	412
Tips for Using a Solderless Breadboard	413
Chapter 33—Making Circuit Boards	414
Overview of Your Primary Circuit Board Options	414
Clean It First!	415
Making Permanent Circuits on Solder Breadboards	415
Using Point-to-Point Perforated Board Construction	416
Using Pre drilled Stripboards	417
Creating Electronic Circuit Boards with PCB CAD	418
Producing Arduino-Specific Boards with Fritzing	420
On the Web: Etching Your Own Printed Circuit Board	421
Using Custom Prototyping Boards	422
Making Semipermanent Circuits with Wire Wrapping	422
Effective Use of Plug-in Headers	424
Part 6—Computers and Electronic Control	
Chapter 34—An Overview of Robot “Brains”	425
Brains for the Brawn	427
Igor, Pull the Switch!	428
Brains from Discrete Components	428
Programmed Brains	429
Of Inputs and Outputs	434
Chapter 35—Understanding Microcontrollers	437
All about Microcontroller Categories	437
Microcontroller Shapes and Sizes	440
Under the Hood of the Typical Microcontroller Chip	441
Microcontroller Programmers	444
All about Microcontroller Speed	445
Chapter 36—Programming Concepts: The Fundamentals	446
Important Programming Concepts	446
Understanding Data Types	450
Lucky Seven Most Common Programming Statements	452
Variables, Expressions, and Operators	455
On the Web: More Programming Fundamentals	459

Part 7—Microcontroller Brains

Chapter 37—Using the Arduino

463

Arduino under the Hood	463
Many Variations on a Theme	464
Ready Expansion via Shields	465
USB Connection and Power	466
Arduino Pin Mapping	467
Programming the Arduino	467
Programming for Robots	469
Using Servos	473
Creating Your Own Functions	474
On the Web: Operating Two Servos	475
Flow Control Structures	476
Using the Serial Monitor Window	477
Some Common Robotic Functions	478
Using Switches and Other Digital Inputs	479
Interfacing to DC Motors	479

Chapter 38—Using the PICAXE

482

Understanding the PICAXE Family	482
Programming the PICAXE	487
Core Language Syntax	488
PICAXE Functions for Robotics	492
Example: Controlling an RC Servo with the PICAXE	493
Example: Reading Buttons and Controlling Outputs	494

Chapter 39—Using the BASIC Stamp

496

Inside the BASIC Stamp	496
Stamp Alone or Developer's Kit	498
Physical Layout of the BS2	498
Hooking Up: Connecting the BASIC Stamp to a PC	499
Understanding and Using PBasic	500
Interfacing Switches and Other Digital Inputs	506
Interfacing DC Motors to the BASIC Stamp	507
Interfacing RC Servo Motors to the BASIC Stamp	508
Additions in PBasic 2.5	509

Chapter 40—Interfacing Hardware with Your Microcontroller or Computer	512
Sensors as Inputs	512
Motors and Other Outputs	514
Input and Output Architectures	516
Interfacing Outputs	519
Interfacing Digital Inputs	520
Interfacing Analog Input	522
Connecting with USB	525
Using Analog-to-Digital Conversion	526
Using Digital-to-Analog Conversion	527
Expanding Available I/O Lines	528
Understanding Port Changing	531
On the Web: Understanding Bitwise Port Programming	533
Chapter 41—Remote Control Systems	534
Build a Joystick “Teaching Pendant”	534
Commanding a Robot with Infrared Remote Control	537
On the Web: Control by Radio Signal	543
Broadcasting Video	543
Part 8—Sensors, Navigation, and Feedback	
Chapter 42—Adding the Sense of Touch	547
Understanding Touch	547
Mechanical Switch	548
Using a Button Debounce Circuit	555
Debouncing Switches in Software	556
Programming for Bumper Contacts	557
Mechanical Pressure Sensors	558
Experimenting with Piezoelectric Touch Sensors	563
Experimenting with Piezo Film	565
On the Web: Build a Piezo Bumper Bar	568
Other Types of “Touch” Sensors	568
Chapter 43—Proximity and Distance Sensing	570
Design Overview	570
Simple Infrared Light Proximity Sensor	572

Modulated Infrared Proximity Detector	574
Infrared Distance Measurement	580
On the Web: Passive Infrared Detection	585
Ultrasonic Distance Measurement	585
Chapter 44—Robotic Eyes	590
Simple Sensors for Robotic Eyes	590
Building a One-Cell Cyclops Eye	594
Building a Multiple-Cell Robotic Eye	596
Using Lenses and Filters with Light-Sensitive Sensors	600
Video Vision Systems: An Introduction	600
Chapter 45—Navigating Your Robot	603
Tracing a Predefined Path: Line Following	603
Wall Following	608
Odometry: Calculating Your Robot's Distance of Travel	609
Compass Bearings	617
Experimenting with Tilt and Gravity Sensors	619
More Navigational Systems for Robots	624
Chapter 46—Making and Listening to Sound	625
Preprogrammed Sound Modules	625
Commercial Electronic Sound Effects Kits	627
Making Sirens and Other Warning Sounds	627
Using a Microcontroller to Produce Sound and Music	628
Using Audio Amplifiers	630
Sound and Music Playback with a Microcontroller	631
Speech Synthesis: Getting Your Robot to Talk	632
Listening for Sound	634
On the Web: More Sound Projects	637
Chapter 47—Interacting with Your Creation	639
Using LEDs and LED Displays for Feedback	639
Feedback via Simple Sounds	646
Using LCD Panels	646
Robot-Human Interaction with Lighting Effects	649
Chapter 48—Danger, Will Robinson!	655
Flame Detection	655
Smoke Detection	657

Detecting Dangerous Gas	661
Heat Sensing	664
Robotic Firefighting Contests	665
Finally, Go Out and Do!	665
Appendix A—RBB Online Support	667
You'll Find . . .	667
Backup Support Site	668
Sources for Special Parts, Web Sites	668
Appendix B—Internet Parts Sources	669
Robotics	669
Electronics	670
Hobby	670
Forums and Blogs	671
More on the Web!	671
Appendix C—Mechanical Reference	672
Decimal Fractions	672
Drill Bit and Tap Sizes—Imperial	673
Drill Bit and Tap Sizes—Metric	674
Numbered and Fractional Inch Drill Bit Comparison	675
Fasteners: Standard (Imperial) Threads at a Glance	675
Comparison of Decimal Inch, Fractional Inch, Mil, and Gauge	676
More on the Web!	676
Appendix D—Electronic Reference	677
Formulas	677
Abbreviations	679
Letter Symbols Used in Electronics	681
Numbering Units in Electronics	681
The Six Most Common Units of Measure in Electronics	682
Resistor Color Coding	683
Wire Gauge	684
Index	685

This page intentionally left blank

ACKNOWLEDGMENTS

Once more I've climbed the mountain. And once more I look back to those who have helped me turn my vision into reality.

To my friends at the San Diego robotics group; to John Boisvert and his amazing robotics emporium; to Mike Keesling, Alex Brown, and Tony Ellis; to those I've met over the years on the *comp.robotics.misc* newsgroup for their great ideas, wisdom, support, and advice; and to Frits Lyneborg and the entire *LetsMakeRobots.com* community;

To Russell Cameron, roboteer extraordinaire, and to Jan Malasek of Pololu, Mario Tremblay of Robotshop, Clive Seager of Revolution Education, Jim Frye of Lynxmotion, Nathan Seidle of SparkFun, Gerry Coe of Devantech, and Claudia and the crew at DAGU;

To the makers and keepers of the Arduino, PICAXE, BASIC Stamp, FIRST CAD Library, and Fritzing for great tools, great software, and great art;

To Judy Bass and the editors at McGraw-Hill, who've put up with me all these years; to my agents at Waterside Productions; and to Bill Gladstone who first helped me take on this project way back in 1985 (how time flies!);

And last but certainly not least, to my wife, Jennifer.

I offer my heartfelt thanks, one and all.

This page intentionally left blank

PHOTO AND ILLUSTRATION CREDITS

Adafruit Industries (www.adafruit.com): Figures 37-2, 37-3, 37-10
Christopher Schantz (www.expressionimage.com): Figure 2-14
Cooper Industries (www.cooperhandtools.com): Figure 30-4
Devantech (www.robot-electronics.co.uk): Figure 48-2
General Electric: Figure 2-4
Hitec RCD (www.hitecrcd.com): Figure 23-1
Lynxmotion (www.lynxmotion.com): Figures 1-1, 2-6, 2-7, 20-10, 27-2, 28-13
iRobot Corporation (www.irobot.com): Figures 1-6, 2-3
Maxbotics Inc (www.maxbotics.com): Figure 43-14
Miga Motor Company (www.migamotors.com): Figure 25-7
Parallax Inc (www.parallax.com): Figures 2-10, 34-2, 39-1, 43-3, 48-6
Pitsco Education (www.pitsco.com): Figures 1-7
RoboRealm (www.roborealm.com): Figure 44-12
Pololu (www.pololu.com): Figures 10-4, 10-7, 45-5, 45-19
Russell Cameron/DAGU Hi-Tech Electronic: Figures 1-1, 44-10
Scott Edwards Electronics (www.seetron.com): Figure 27-6
SparkFun Electronics (www.sparkfun.com): Figures 42-14, 46-7, 46-9, 46-10

The author expresses his deepest gratitude for the use of 3D CAD objects developed by Ed Sparks at FirstCadLibrary.com; to the developers and contributors of the Fritzing project (www.fritzing.org); for public domain art developed by Wikipedia user Inductiveload, and others; and to the talented artists of the 3D models used throughout this book.

This publication contains images used under license from Corel Corporation, Hemera Technologies, Shutterstock.com, and other licensors.

This page intentionally left blank

INTRODUCTION

Robotics: Inspired Technology

Which of these fields are involved in robotics? You may choose from the following: engineering, electronics, psychology, sociology, biology, physics, artificial intelligence, math, art, mechanical design, mechanical construction, computer programming, sound synthesis, vision, ultrasonics, linguistics, microelectronics, process control, system automation, musicology.

If you said “all of them,” you’re right.

And if you think I missed some, I have. Robotics is all of these things, and *much* more. Its popularity comes from all the disciplines it embraces. When you build a robot, you can explore everything from mechanical design to computer engineering to behavioral science.

Every robot is different, a reflection of its creator—you. Want to make an art bot that spins around on a piece of paper, drawing pictures using crayons? Why not! It’s your creation. You set the rules.

No Better Time to Play with Robots

New technologies have dramatically driven down the cost of building a robot, not to mention the time it takes to construct one. Now is the ideal time to get into robotics. It’s never been cheaper, and the result far exceeds what was possible even five years ago.

For less than \$75 you can construct a sophisticated, fully autonomous robot that can be programmed from your computer. Easily change its behavior as you experiment with new designs.

Robotics is still a cottage industry. There’s plenty of room for growth, with a lot of discoveries yet to be made. Maybe you’ll be the one to make them?

If so, that’s where this book comes in.

Inside *Robot Builder's Bonanza*

This book is part tutorial and part reference. It tells you what you need to know to build a robot, plus a whole lot more about the art and science of robotics.

Robot Builder's Bonanza, Fourth Edition, is all about having fun while learning how to design, construct, and use small robots. Hands-on plans take you from building basic motorized platforms to giving the machine a brain—and teaching it to walk and talk and obey commands.

This book is about *inspiring you*—in the guise of a how-to book on constructing various robots and components. The modular projects in this book can be combined to create all kinds of highly intelligent and workable robots of all shapes and sizes. Mix and match your projects the way you like.

The projects in this book are a treasure chest of information and ideas on making thinking machines. You'll find what you need to know to construct the essential building blocks of a personal robot.



Absolutely brand-new to all things robotics and electronics? Then be sure to visit the RBB Online Support site for the free *My First Robot* tutorial lessons. See Appendix A, "RBB Online Support," for more details.

ABOUT THE FOURTH EDITION

This book is a fully updated and greatly expanded revision of *Robot Builder's Bonanza*, first published in 1987, and then updated again in 2001 and 2006.

Previous editions of this book have been perennial bestsellers. I'm proud to say this is one of the most widely read books ever published on amateur robotics.

In the pages that follow, you'll find updated coverage of exciting technologies like the latest Arduino and PICAXE microcontrollers; hands-on projects using unique forms of light, vision, and sound sensors; robot construction using high-grade wood, plastics, and metals; advanced servo and DC motor control; and rapid prototyping techniques for building bots in record time.

With this edition is a special emphasis on construction plans that are not only affordable, but easily reproducible through the use of commonly available parts. To make it easier for first-time builders, none of the projects in this book require expensive or elaborate tools.

FREE ONLINE CONTENT, PARTS FINDER, VIDEOS, BONUS GOODIES

This book comes with *free* online content: the *RBB Online Support* site. See Appendix A for the lowdown. On the support site you'll find:

- *My First Robot*—a series of easy-to-follow fully illustrated lessons that teach you the basics of electronics, soldering, robot planning, and construction
- *Project Parts Finder*—where to find all the parts for the projects in this book, including sources and part numbers
- *Animated and interactive learning tools*, including circuit simulators
- *New and updated links* to Web sites and manufacturers
- *Enhanced and updated robot e-plans*
- *How-to videos, bonus articles*, tutorials on robot construction, and more



In order to provide more space for juicy tidbits about robot building, some of the longer programming examples are moved from the printed page and located as free downloads on the RBB Online Support site. When applicable, you'll see a note like this one telling you to go to the support site so you can fetch the code.

What You'll Learn

Robot Builder's Bonanza is divided into eight sections; each one covers a major component of building a robot.

Part 1: The Art and Science of Robot Building. What you need to get started; setting up shop; how and where to get robot parts.

Part 2: Robot Construction. Robots made of plastic, wood, and metal; working with common materials; converting toys into robots; mechanical construction techniques; using rapid prototyping techniques to build fast and cheap robots. Includes three full and complete robot projects: PlyBot, PlastoBot, and TinBot.

Part 3: Power, Motors, and Locomotion. Using batteries; powering the robot; working with different kinds of motors; powering motors from computerized electronics; mounting motors and wheels; using space-age shape memory alloy.

Part 4: Hands-on Robotics Projects. Lots of projects and ideas for building robots with wheels, tracks, and legs; constructing arm systems; building robot hands.

Part 5: Robot Electronics. Circuitry for robots; common components and how they work; constructing circuits on solderless breadboards; making your own soldered circuit boards.

Part 6: Computers and Electronic Control. Smart electronics for your bot; introduction to microcontrollers; programming fundamentals.

Part 7: Microcontroller Brains. All about three popular microcontrollers: Arduino, PICAXE, and BASIC Stamp; interfacing electronics to your microcontroller or computer; operating your robot via wire, infrared, and radio remote control.

Part 8: Sensors, Navigation, and Feedback. Collision detection and avoidance; sensing when objects are nearby; gravity, compass, and other navigation sensors; measuring distance using ultrasonic sound and infrared; eyes for your robot; navigation techniques; making and listening to sound; smoke, flame, and heat detection.



Whenever practical, I've designed the components as discrete building blocks, so that you can combine the blocks in just about any configuration you want. The robot you create will be uniquely yours, and yours alone.

Expertise You Need

Actually . . . you don't need *any* experience to use this book. It tells you what you need to know.

But if you happen to already have some experience—such as in construction, electronics, or programming—you're free to move from chapter to chapter at will. There are plenty of cross-references to help you expand your discovery zone.

Robot Builder's Bonanza doesn't contain hard-to-decipher formulas, unrealistic assumptions about your level of electronic or mechanical expertise, or complex designs that only a seasoned professional can tackle.

I wrote this book so that anyone* can enjoy the thrill and excitement of building a robot. The projects can be duplicated without expensive lab equipment, precision tools, or specialized materials—and at a cost that won't contribute to the national debt!



*If you're under 15 or thereabouts, ask for help from a parent or teacher. Some of the projects in this book involve using tools and techniques that could be dangerous if proper safety precautions are not followed.

The projects in this book have been written to avoid the really bad stuff, but you can still get seriously burned, cut, punctured, or poisoned if you're not careful.

Beyond Cool

Think of *Robot Builder's Bonanza* as an adventure in technology—with a lot of fun thrown in for free.

As you construct your bot, you'll learn the latest technologies firsthand. It puts you way ahead of the curve, whether your interest in robotics is for school, for work, or as a hobby.

Years before the public even heard of a smartphone, bot builders were messing around with electronic compasses, accelerometers, gyroscopes, global positioning satellites, micro-controllers, touch screens, artificial intelligence, voice control, speech synthesis, and more. Robotics is your ticket to many of the exciting technologies of today and tomorrow.

Think of *Robot Builder's Bonanza* as a *treasure map*. The paths provided between these covers lead you on your way to building one or more fully functional robots. So turn the page, and start the adventure.

ROBOT BUILDER'S BONANZA

This page intentionally left blank

The Art and Science of Robot Building

This page intentionally left blank

Welcome to the Wonderful World of Robotics!

There he sits, as he's done countless long nights before, alone and deserted in a dank and musty basement. With each strike of his ball-peen hammer comes an ear-shattering bong and an echo that seems to ring forever. Slowly, his creation takes shape and form—it started as an unrecognizable blob of metal and plastic, soon it was transformed into an eerie silhouette, then . . .

Brilliant and talented, but perhaps a bit crazed, he is before his time: a social outcast, a misfit who belongs neither to science nor to fiction. He is the robot experimenter, and all he wants to do is make a mechanical creature that serves drinks at parties and wakes him up in the morning.

Okay, maybe this is a dark view of the present-day amateur robotics experimenter. Though you may find a dash of the melodramatic in it, the picture isn't unrealistic. It's a view held by many outsiders to the robot-building craft—a view more than 100 years old, from the time when the prospects of building a human-like machine first came within technology's grasp.

Like it or not, if you want to build robots, you're an oddball, an egghead, and—yes, let's get it all out—a little on the *weird* side!

As a robot experimenter, you're not unlike Victor Frankenstein, the old-world doctor from Mary Shelley's immortal 1818 horror thriller. Instead of robbing graves in the still of night, you “rob” electronics stores, flea markets, and surplus outlets in your unrelenting quest—your *thirst*—for all kinds and sizes of motors, batteries, gears, wires, switches, and other gizmos. Like Dr. Frankenstein, you galvanize life from these “dead” parts.

What the Adventure Holds

Just starting out building your first robot? You're in for a wonderful ride! Watching your creation do something as simple as scoot around the floor or table can be exhilarating. Those



Figure 1-1 Amateur robots can take many forms and sizes: mobile (moving) robots use wheels, tracks, or legs for propulsion; arms and grippers allow the robot to manipulate its environment. (Includes photos courtesy Lynxmotion and Russell Cameron)

around you may not immediately share your excitement, but you know that you've built something—however humble—with your own hands and ingenuity.

If you're one of the lucky few who has already assembled a working bot, then you know the excitement I'm talking about. You know how thrilling it is to see your robot obey your commands, as if it were a trusted dog.

You know the time and effort that went into constructing your mechanical marvel, maybe something like the ones in the robo-zoo picture in Figure 1-1—all are hobby robots. And although others may not always appreciate it (especially when it marks up the kitchen floor with its rubber tires), you're satisfied with the accomplishment. You're looking forward to the next challenge.

And if you've built a robot, you also know of the heartache and frustration that come with the process. You know that not every design works, and that even a simple engineering flaw can cost weeks of effort, not to mention ruined parts. This book will help you—beginner and experienced robot creator alike—get the most out of your robotics hobby.

Why Build Robots?

I wanted to build my first robot the moment I saw the sci-fi movie classic *The Day the Earth Stood Still* (the original, mind you, not the remake). Many have gotten the Bot Building Bug from watching movies, like *Star Wars*, *Short Circuit*, or *Terminator*.

No matter where the inspiration comes from, there are many reasons to build your own robot. Here are just a few of them.

ROBOTICS IS A KEystone OF MODERN TECHNOLOGY

I recently purchased a new smartphone. As I went through its feature list I was amazed at all the things it could do—and how many of them were technologies we robot builders were playing with *over a decade ago!*

Robotics is a natural test bed for new ideas. Robot builders have been among the first

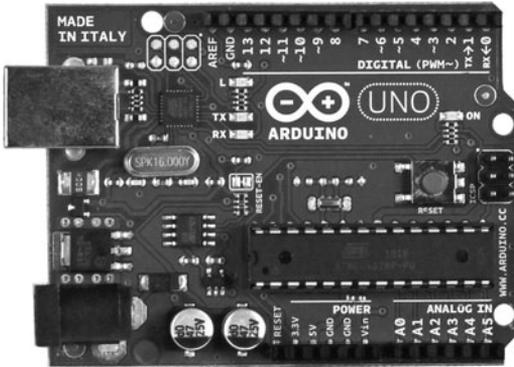


Figure 1-2 The modern microcontroller, like this Arduino, provides an amazing amount of computational power for controlling a robot.

amateurs to play with microcontrollers, accelerometers, digital compasses, voice control, electronic gyroscopes, global positioning satellite modules, speech synthesizers, solid-state imagers, vision recognition, tactile feedback, and many other cutting-edge technologies.

What's more, all of this is available at low cost. The pocket-size microcontroller circuit board in Figure 1-2 costs less than a dinner for two and rivals the thinking power of the computer that put Apollo astronauts on the moon. (You'll be learning lots more about this microcontroller in Chapter 37, "Using the Arduino.")

Whether you're a garage-shop tinkerer, a student, or an engineer working for a Fortune 500 company, experimenting with amateur robotics gives you ample opportunity to discover the technologies the world will be using tomorrow.

ROBOTICS AS A GATEWAY TO A CAREER

Still going to school? Haven't yet decided what you want to do with your life? Believe it or not, building a robot can lead the way.

Robotics involves dozens of interconnected sciences and disciplines—mechanical design and construction, computer programming, psychology, behavioral studies, ecology and the environment, biology, space, micro-miniaturization, underwater research, electronics, and much more.

You don't need to be an expert in these fields just to build a robot. You can concentrate your studies on those things that most interest you, using your robot as a doorway to furthering your interests.

ROBOTICS TO THE RESCUE

Science fiction has long painted the robot as evil—either on its own or as the minion of a mad scientist. Yet it turns out robots may be a way for people to live better, longer lives.

- Robots can venture where people can't, or don't want to, go. Send a bot into a collapsed mine shaft, or to the bottom of the ocean, or to the dusty surface of Mars. It'll get the job done, and it doesn't need air or McDonald's breaks.
- A bomb-sniffing robot can save the lives of many people. It can locate the explosive and defuse it much more safely than humans can.
- Robots can act as nurses and doctors, even to those with highly contagious diseases, half-way around the world.

- Kids respond to robots in ways that can actually help them to develop interpersonal skills. There are even some robots used as therapy for children with certain learning and social disorders

MOST OF ALL, ROBOTICS IS FUN!

It's okay to build robots just for kicks. Really!

Challenge yourself to a new project, and enjoy a hobby shared by many others worldwide. Share your designs on a blog or forum. Enter a competition to see whose robot is fastest or strongest. Post a video of your robot on YouTube, and show it off.

The Building-Block Approach

Robots are made of many individual parts that act as modules or subsystems. So a great way to learn about amateur-made robots is to construct individual components, then combine them to make a finished, fully functional machine.

The robots you create are made from building blocks, so making changes and updates to them is relatively simple. When designed and constructed properly, the building blocks, like those in Figure 1-3, may be shared among a variety of robots. To save time and money, it's not unusual to reuse parts as you experiment with new robot designs.

Most of the building-block designs presented in this book are complete, working subsystems. Some operate without ever being attached to a robot or control computer. The way you connect the modules is up to you. You can experiment with each part, altering it and improving upon it. When it works the way you want, incorporate it into your robot or save it for a future project.

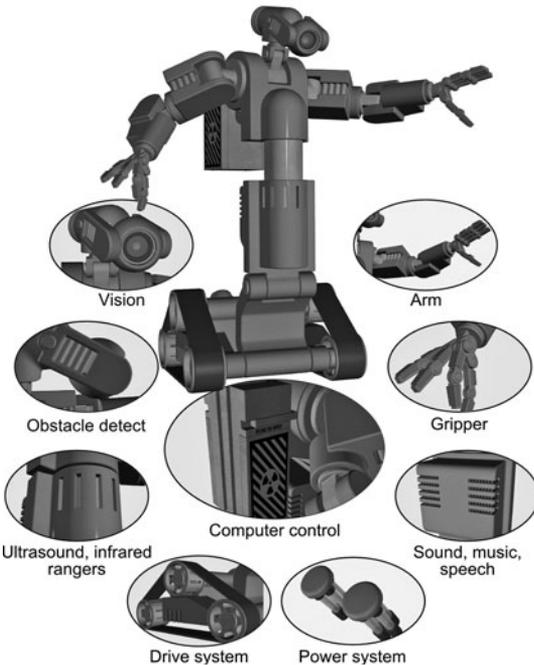


Figure 1-3 The basic building blocks of a fully functional robot include a computer control or other central processor, a drive system for mobility, and various sensors.

Lower Costs, Better Bots

Homebrew robotics used to be expensive and time consuming. Weak electronic brains limited what the finished robot could do. No longer. Amateur robots are now:

- *Lower cost.* It's possible to build a fairly sophisticated autonomous robot for less than \$75, with only ordinary tools. Similar robots used to cost \$500 or \$600 and required specialized gear to make.
- *Simpler.* Thanks to ready-made sensors, specialty electronics, and prefab parts, it's much easier to construct robots by putting together construction blocks.

- *More powerful.* Inexpensive microcontrollers add horsepower and functionality, with more memory, faster processing speeds, and easier interfacing to other components. If you have a PC with a USB port, you can start working with microcontrollers today—many cost just a few dollars and can control an entire robot.

Skills You Need

You don't have to be an expert in electronics and mechanical design to build robots. Far from it. Which of these best describes you?

- *I'm just starting out.* If you're an absolute raw beginner in all things robotics, start with the *My First Robot* lessons found on the RBB Online Support site (see Appendix A for details). The lessons give you step-by-step instructions for building the *RBB Bot*, an inexpensive autonomous (runs on its own) robot. You'll learn about the fundamentals of electronics and robotics.
- *I have some electronics or mechanical background.* Plow straight ahead to the construction guides and how-tos that follow. This book is organized into parts so that you can bone up on your skills and knowledge as you read.
- *I'm an experienced tinkerer.* If you are already versed in electronics and mechanics, you're well on your way to becoming a robot experimenter *extraordinaire*. You can read the chapters in the order you choose. There are plenty of cross-references among chapters to help you connect the dots.

ELECTRONICS BACKGROUND

Electronic circuits are what make your robots “thinking machines.” You don't need extensive knowledge of electronics to enjoy creating robots. You can start with simple circuits with a minimum of parts. As your skills increase, you'll be able to (at the least) customize existing circuits to match your needs.

This book doesn't include much in the way of electronics theory, just practical information as it relates to building bots. If you're looking for detailed college-level instruction on electronics, check out any local library and do a Web search for books and publications.

Many of the circuits in this book are in schematic diagram form, a kind of blueprint for how the parts of the circuit are connected. If you've never seen a schematic, you can read up on them in Part 5 of this book, plus the *My First Robot* lessons (see Appendix A, “RBB Online Support”), which includes a whirlwind introduction to electronics. There, you can see how a schematic road map corresponds to actual components of a circuit you can build yourself—see Figure 1-4 for an example. There are really only about a dozen common schematic symbols, and you can learn what you need to know with just an evening of study.

The parts for the electronic projects in this book are all selected to be widely available and reasonably affordable. I decided not to include vendor part numbers right in the book because these can change quickly.

Instead, you can visit the RBB Online Support site (see Appendix A) for updated lists of parts used in this book and where to get them. You'll also find direct links to many parts—just click and you're there.



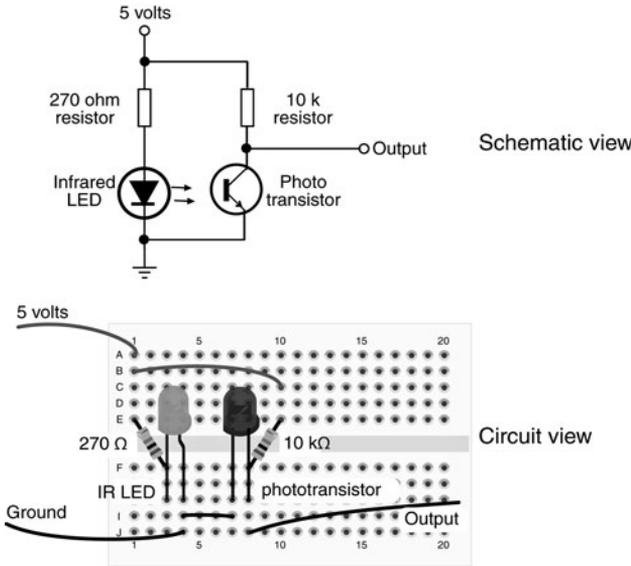


Figure 1-4 Schematics are like road maps for building electronics. You only need to learn about a dozen common symbols, and you’re well on your way to building the typical robot circuit.

PROGRAMMING BACKGROUND

Modern robots use a computer or microcontroller to manage their actions. In this book you’ll find plenty of projects, plans, and solutions for connecting the hardware of your robot to any of several kinds of ready-made robot brains.

Like all computers, the ones for robot control need to be programmed. If you are new or relatively new to computers and programming, start with Chapter 36, “Programming Concepts: The Fundamentals.” None of the projects in this book require expensive or complex programming tools.

MECHANICAL BACKGROUND

Some robot builders are more comfortable with the mechanical side of robot building than the electronic or programming sides—they can see gears meshing and pulleys moving. As with electronics, you don’t need an extensive knowledge of mechanical and engineering theory to build robots.

This book provides several start-to-finish robot designs using a variety of materials, from cardboard to space-age plastic to aluminum. If you’re a workshop beginner, you’ll find helpful tips on what tools to use and the best materials for constructing your robot bodies.

If you’re one of those who just *hate* the idea of cutting a piece of wood, or drilling through plastic, there’s good news: you’ll find plenty of mail-order sources for purchasing bare-bones robot mechanics. You still need to assemble things, but you can get by with just a screwdriver. These sources are listed in Appendix B, as well as throughout the book and on the RBB Online Support site referenced in Appendix A.

THE WORKSHOP APTITUDE

To be a successful robot builder, you should be comfortable working with your hands. Even if you don’t build your own robot bodies start to finish, you’ll need to assemble the parts using basic tools.

If you feel your workshop skills aren't up to par, try one of the basic robot platforms (bodies) in Part 2. You can choose a robot body made with wood, plastic, or metal.

You'll find construction tips and techniques throughout this book, but nothing beats hands-on shop experience. With experience comes confidence, and with both come more professional results.

TWO VERY IMPORTANT SKILLS

So far, I've talked about basic skills that are desirable for the hobby robotics field. There two important skills that you cannot develop from reading books: they are patience and the willingness to learn.

- *Be patient!* Give yourself time to experiment with your projects. Don't rush into things, or else you're bound to make mistakes. If a problem continues to nag at you, put the project aside and let it sit for a few days. Keep a small notebook handy and jot down your ideas so you won't forget them.
- *Be willing to learn new things!* If trouble persists, maybe you need to study more before you can tackle the problem. Be willing to go beyond this book and discover things on your own. *Research is never in vain.*

Do It Yourself, Kits, or Ready-Made?

There's never been a better time to be an amateur robot builder. Not only can you construct robots "from scratch," you can buy any of several dozen robot kits and assemble them using a screwdriver and other common tools.

MAKE YOUR OWN

This book is chock full of robot projects made from wood, plastic, and metal. One of the robots you can build is shown in Figure 1-5. This one is the Hex3Bot, from Chapter 27, "Build Robots with Legs." Construction takes about a day (depending on your shop skills) and requires only basic tools.

Or you can try building a robotic arm, complete with separate motors for each joint. See Chapter 28, "Experimenting with Robotic Arms," for a project you can make in your shop. Then in Chapter 29, "Experiment with Robotic Grippers," give your robo-arm a hand and fingers.

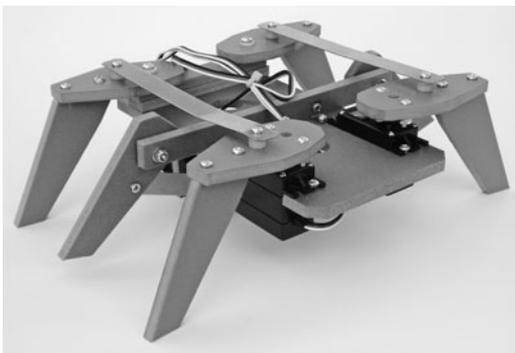


Figure 1-5 Using the right materials and tools, it takes a day or less to construct a homemade robot like this one. The Hex3Bot uses easy-to-cut plastic.



Figure 1-6 If the mechanics of robotics isn't your forte, you can learn by using a ready-made robotic platform. This one is designed to teach the principles of robot programming. (Photo courtesy iRobot Corporation.)

READY-MADE BOTS

If you don't particularly like the construction aspects of robotics, you can purchase ready-made robot bodies—no assembly required. With a ready-made robot you can spend all your time connecting sensors to it and figuring out new and better ways to program it. An example is the iRobot Create, shown in Figure 1-6. It's like the iRobot's Roomba vacuum-cleaning robot, but without the vacuum cleaner. It's meant as a mobile robot platform for educators, students, and developers.

KIT O' PARTS

You know the Erector set—this venerable construction toy provides the hardware for building all kinds of buildings, bridges, cars, and other mechanical things. The traditional Erector Set uses lightweight metal girders of different sizes, plus steel fasteners to put it all together.

Several specialty robotics companies have taken the Erector set one step further, modifying the idea to handle the unique issues of robot building. Their construction kits can be used to create fully functional robots, or you can adapt the bits and pieces to your custom designs. These aren't exactly cheap, but they're convenient and engineered to produce outstanding prototypes.

Here are four of the more popular parts kits specifically created for building robots.

VEX Robotics Design System

Aimed at both the educational and the hobbyist market, the VEX system is based around the Erector set style of predrilled girders and connector pieces. Most parts are fastened using traditional techniques of machine screws and nuts. What sets VEX apart is that its pieces are especially designed for small robotics. Kits come with motors made to fit the girder pieces, gears, wheels, and many other hardware parts.

Lynxmotion Servo Erector Set

The Servo Erector Set from Lynxmotion is composed of various brackets and other hardware for the express purpose of connecting together standard-size R/C servo motors. You can connect these brackets to a traditional robot base to build a rolling or walking machine or attach them to tubes, hubs, and connectors to fashion completely free-form designs.

Bioloid

The Bioloid robotic construction sets are engineered for serious robot building. Using a mixture of fabricated metal and plastic parts, Bioloid pieces can be combined to create both wheeled and walking bots, as well as fully functional grippers and arms. The sets include special high-torque digital motors plus a central microcontroller for operating all electronic components.

Pitsco/FIRST Robotics FTC Kit

FIRST (For Inspiration and Recognition of Science and Technology) is an organization cofounded by inventor Dean Kamen, of Segway fame, to develop ways to inspire young adults in the fields of engineering and technology. The organization hosts national competitions, such as the FIRST Tech Challenge (FTC), where teams of students are required to build, program, and demonstrate a robot that completes a certain task. The task changes for each year, to keep things interesting.

In order to provide a level playing field for all teams, FIRST limits robot construction to a preapproved kit of parts; the latest kit is assembled by educational equipment giant Pitsco. In the case of the FIRST Tech Challenge, the kit incorporates metal and other building parts.



If you like the parts that come with the latest FTC competition kits, you can purchase them separately without going through the FIRST organization. Visit www.pitsco.com, and do a search for *TETRIX*. Their channels and brackets are designed to interface with R/C and DC gear motors (see Figure 1-7) and can be used for all kinds of nifty projects.

PROJECTS TO MATCH YOUR SKILLS

Whether you choose to buy a robot in ready-made or kit form, or build your own from the ground up, be sure to match your skills to the project. This is especially true if you are just

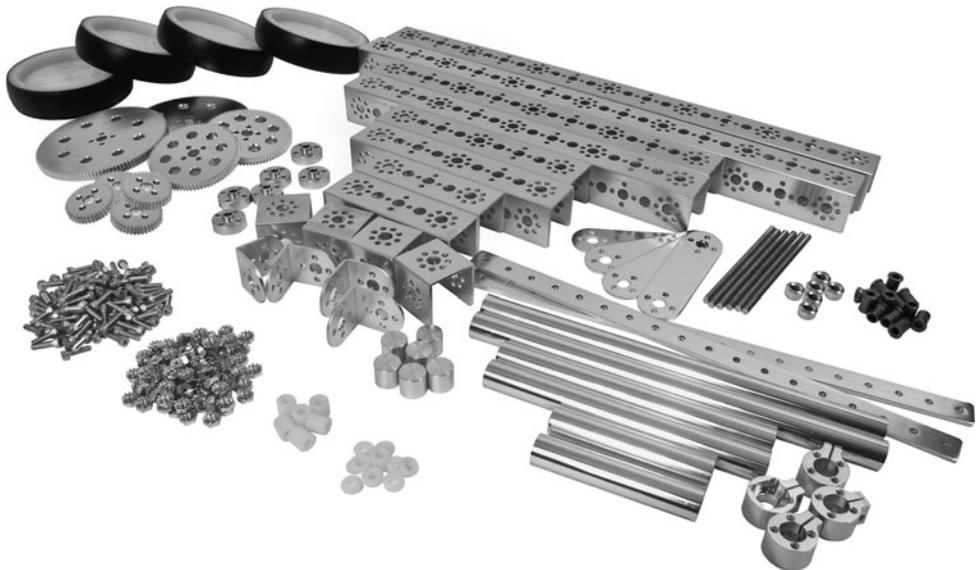


Figure 1-7 Some assembly required: precut and predrilled parts allow you to construct robots by piecing together metal or plastic components. (Photo courtesy Pitsco Education.)

starting out. While you may seek the challenge of a complex project, if it's beyond your present skills and knowledge level you'll likely become frustrated and abandon robotics before you've given it a fair chance.

Thinking Like a Robot Builder

Robot experimenters have a unique way of looking at things. They take nothing for granted:

- At a restaurant, it's the robot experimenter who collects the carcasses of lobsters and crabs to learn how these ocean creatures use articulated joints, in which the muscles and tendons are *inside* the bone. Maybe the lobster leg can be duplicated in the design of a robotic arm . . .
- At a county fair, it's the robot experimenter who studies the way the "eggbeater" ride works, watching the various gears spin in perfect unison. Perhaps the gear train can be duplicated in an unusual robot locomotion system . . .
- While flipping channels on the TV, it's the robot experimenter who thinks that if the remote control can operate a television, the same technique can be applied to a robot . . .
- When washing hands in a restaurant lavatory, it's the robot experimenter who studies the automatic faucet control. The water is automatically turned on and off when the hands come near the sink. Could the same system be adapted to a robot, enabling it to judge distances and see things in front of it? . . .

The list is endless. All around us, from nature's designs to the latest electronic gadgets, are infinite ways to make better and more sophisticated robots. Uncovering these solutions requires *extrapolation*—figuring out how to apply one design and make it work somewhere else.

And that's what the robot builder is especially good at!

Anatomy of a Robot

The human body is, all things considered, a nearly perfect machine: it's (usually) intelligent, it can lift heavy loads, it can move itself around, and it has built-in protective mechanisms to feed itself when hungry and to run away when threatened. Other living creatures possess similar functions, though not always to the same degree of sophistication.

Robots are often modeled after humans, if not in form then at least in function. Nature provides a striking model for robot experimenters to mimic. It's up to us to take the challenge. Some of the mechanisms found in nature can be duplicated in the robot shop. Robots can be built with eyes to see, ears to hear, a mouth to speak—all with the goal of manipulating the environment and exploring the surroundings.

This is fine theory. What about real life? What basic parts must a machine have before it's given the title "robot"? Let's take a close look in this chapter at the anatomy of robots and the kinds of materials hobbyists use to construct them. For the sake of simplicity, not every robot subsystem ever made is covered here, just the ones most often found in amateur robots.

Stationary versus Mobile Robots

When we think of robots most of us envision a machine that walks around on legs or scoots across the floor on wheels.

In fact, the most common robots stay put and manipulate some object placed in front of them. These are often used in manufacturing, and they are *stationary*—they're bolted to the ground. Such stationary robots, like the one in Figure 2-1, assist in making cars, appliances, even *other* robots!

Conversely, *mobile* robots (see Figure 2-2) are designed to move from one place to another. Wheels, tracks, or legs allow the robot to traverse a terrain. Mobile robots may also feature an armlike appendage that allows them to manipulate objects around them. Of the two—stationary or mobile—the mobile robot is probably the more popular project for hobby-

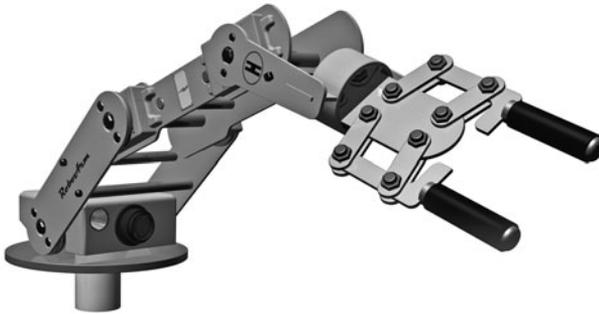


Figure 2-1 Stationary robots don't move. They sit on a tabletop or (for industrial applications) bolt into the ground.

ists to build. There's something endearing about a robot that scampers across the floor, either chasing or being chased by the cat.

As a serious robot experimenter, don't overlook the challenge of building both types of robots. Stationary bots typically require greater precision, power, and balance, since they are designed to grasp and lift things—hopefully not destroying whatever it is they're handling. Likewise, mobile robots present their own difficulties: maneuverability, adequate power supply, and avoiding collisions among them.

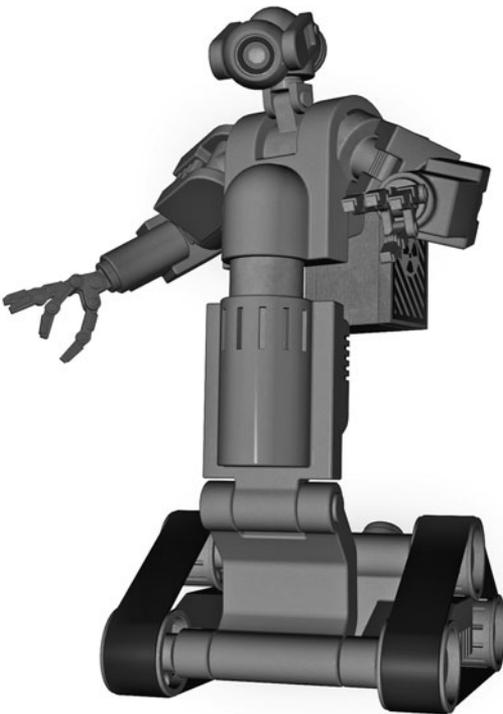


Figure 2-2 Mobile robots move, typically using wheels or tracks but also legs and other forms of propulsion.

Autonomous versus Teleoperated Robots

The first robots ever demonstrated for a live audience were fake; they were actually machines remotely controlled by a person offstage. No matter. People thrilled at the concept of the robot, which many anticipated would be an integral part of their near futures. You know, like flying to work in your own helicopter and colonies on Mars by 1975 . . . yeah, right!

These days, the classic view of the robot is a fully autonomous machine, like Robby from *Forbidden Planet*, Robot B-9 from *Lost in Space*, or that R2-D2 thingie from *Star Wars*. With these robots (or at least the make-believe, fictional versions), there's no human operator, no remote control, no "man behind the curtain."

While many actual robots are indeed fully autonomous, many of the most important robots of the past few decades have been *teleoperated*. A teleoperated robot is one that is commanded by a human and operated by remote control. These are often used in police and combat situations, like the one in Figure 2-3. The typical *telebot* uses a



Figure 2-3 A telerobot requires a human operator to control it. The robot is usually semiautonomous, taking basic instructions via wireless feed and performing intelligent tasks. (Photo courtesy iRobot Corporation.)

video camera that serves as the eyes for the human operator. From some distance—perhaps as near as a few feet to as distant as several million miles—the operator views the scene before the robot and commands it accordingly.

The teleoperated robot of today is a far cry from the radio-controlled robots of the world’s fairs of the 1930s and ’40s. Many telerobots, like the world-famous Mars rover *Sojourner*, the first interplanetary dune buggy, are actually half remote controlled and half autonomous. The low-level functions of the robot are handled by a microprocessor on the machine. The human intervenes to give general-purpose commands, such as “go forward 10 feet” or “hide, here comes a Martian!” The robot carries out basic instructions on its own, freeing the human operator from the need to control every small aspect of the machine’s behavior.



The notion of telerobotics is certainly not new—it goes back to at least the 1940s and the short story “Waldo” by noted science fiction author Robert Heinlein. It was a fantastic idea at the time, but today modern science makes it eminently possible, even for garage-shop tinkerers.

Stereo video cameras give a human operator 3D depth perception. Sensors on motors and robotic arms provide feedback to the human operator, who can actually feel the motion of the machine or the strain caused by some obstacle. Virtual-reality helmets, gloves, and motion platforms literally put the operator “in the driver’s seat.”

Tethered versus Self-Contained Robots

People like to debate what makes a machine a “real” robot. One side says that a robot is a completely *self-contained*, *autonomous* (self-governed) machine that needs only occa-

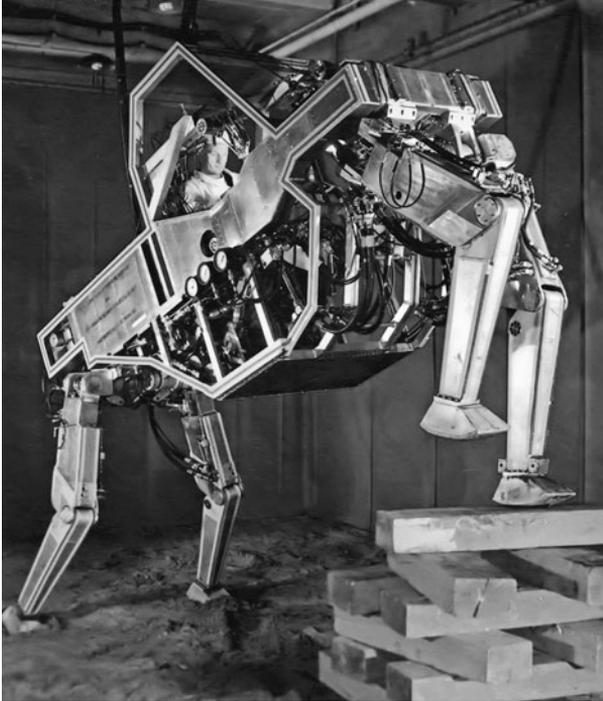


Figure 2-4 This quadruped from General Electric was controlled by a human operator who sat inside it. The robot was developed in the late 1960s under a contract with the U.S. government. (Photo courtesy General Electric.)

1972 novel *Cyborg* (which served as the inspiration for the '70s television series *The Six Million Dollar Man*).

So, What's a Robot, Anyway?

I'm not going to argue robot semantics here—this book is a *treasure map* after all, not a textbook on theory. But it's still necessary to establish some of the basic characteristics of robots.

What makes a robot a robot and not just another machine? For the purposes of this book, let's consider a robot as *any device that—in one way or another—mimics human or animal functions*. The way the robot does this is of no concern; the fact that it does it at all is enough.

The functions that are of interest to us as robot builders run the gamut: from listening to sounds and acting on them, to talking and walking or moving across the floor, to picking up objects and sensing special conditions such as heat, flames, or light. So, when we talk about a robot, it could very well be—

- A self-contained automaton that takes care of itself, perhaps even programming its own brain and learning from its surroundings and environment.

sional instructions from its master to set it about its various tasks. A self-contained robot has its own power system, brain, wheels (or legs or tracks), and manipulating devices such as claws or hands. This robot does not depend on any other mechanism or system to perform its tasks. It's complete in and of itself.

The other side says that a robot is anything that moves under its own motor power *for the purpose of performing tasks that appear to involve intelligence or intent*. The mechanism that does the actual task is the robot itself; the support electronics or components may be separate. The link between the robot and its control components might be a wire, a beam of infrared light, or a radio signal.

In the experimental robot from 1969 shown in Figure 2-4, for example, a man sat inside the mechanism and operated it, almost as if driving a car. The purpose of the four-legged "lorry" was not to create a self-contained robot but to further the development of *cybernetic anthropomorphic machines*. These were otherwise known as *cyborgs*, a concept further popularized by writer Martin Caidin in his

- Or it could be a small motorized cart operated by a strict set of predetermined instructions that repeats the same task over and over again until its batteries wear out.
- Or it could be a radio-controlled arm that you operate manually from a control panel.

Each is no less a robot than the others, though some are more useful and flexible. As you'll discover throughout this book, you're in control in defining how complex your robot creations are.

The Body of the Robot

For most robots, its body holds all its vital parts. The body is the superstructure that prevents its electronic and electromechanical guts from spilling out. Robot bodies go by many names, including *platform*, *frame*, *base*, and *chassis*, but the idea is the same. The bodies of robots differ in size, shape, and style; their type of superstructure; and their overall construction.

ROBOT SIZE, SHAPE, AND STYLE

Robots come in all sizes. Some can fit in the palm of your hand, while others are so big it takes two people to lift them from the worktable.

Homebrew robots are generally the size of a small cat or dog, although some are as compact as an aquarium turtle and a few as large as Arnold Schwarzenegger (if one of these asks you, "Are you Sarah Conner?," answer "No!"). The overall shape of the robot is generally dictated by the internal components that make up the machine, but most designs fall into one of the following "categories":

- Turtle (or desktop)
- Rover
- Walking
- Arms/grippers (also called appendages)
- Android and humanoid

Smaller robots are not only easier to build, but they are also more affordable. Their smaller size means smaller motors, smaller batteries, and smaller chassis—all of which tend to reduce price.

Turtle or Desktop

Turtle or so-called *desktop* robots are simple and compact. As the wording implies, these creations are designed primarily for "tabletop robotics." Turtlebots get their name because their bodies somewhat resemble the shell of a turtle. Researcher W. Grey Walter used the term to describe a series of small robots he envisioned and built in about 1948. In popular usage, turtle robots also borrow their name from a once-popular programming language, Logo turtle graphics, adapted for robotics use in the 1970s.

The turtle category represents the majority of amateur robots (Figure 2-5 shows one of them). It's popular among those involved in noncombat competitions—things like maze following or robotic soccer. Turtle robots are most commonly powered by a rudimentary brain (BEAM robotics, based on simplified electronics, is a good example) or by a small single-chip computer or microcontroller.



Figure 2-5 Small *turtle-sized* robots represent a class that's easy and inexpensive to build. They are ideal for learning about robotic design, construction, and programming.

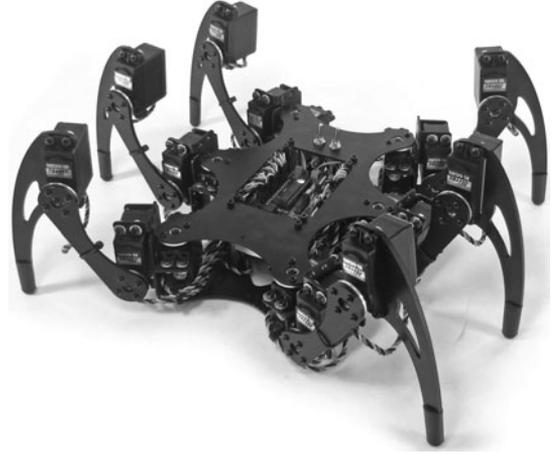


Figure 2-6 A rewarding project is a multilegged walking robot, such as this model that has six legs. It's operated using three individual motors per leg, for a total of 18 motors. (Photo courtesy Lynxmotion.)

Rover

The *rover* category is any of a group of rolling or tracked robots designed for applications that require some horsepower, such as vacuuming the floor, fetching a can of beer or soda, or mowing the lawn. These robots are too big to play with on a desk or tabletop. Sizes range from that of a waffle iron and continue on up. The “death match” combat robots popular on TV are typical robots in the larger end of the rover spectrum, where weight is important to winning.

Because of their larger size, rover robots can be powered—brainwise—by everything from a simple transistor to a desktop computer. Old laptops, particularly the monochrome models, are popular among many robot builders because they can run MS-DOS or early versions of Microsoft Windows, and they can be connected to the robot via standard interface ports.

Walking

A *walking* robot uses legs, not wheels or treads, to move about. Most walker bots have six legs, like an insect, as the six legs provide excellent support and balance. However, gaining in popularity are two- and four-legged walkers, both for “scratch-build” hobby projects as well as for commercial kits.

An example of a six-legged walking robot kit is the Phoenix, from Lynxmotion (Figure 2-6). It's made by combining radio control servo motors and prefabricated plastic and metal parts.

Walking robots require a greater precision in building. The design of the typical robot that rolls on wheels or even tank treads is inherently simpler than that required for the cams, links, levers, and other mechanisms used for walking. For this reason, beginners in the robot-building trade should opt for wheeled designs first to gain experience, even if the walking robot looks cooler.

Note that two-legged walking robots that resemble people are classified (see below) in their own category, considering the technological difficulties in designing and building them. Constructing a small two-legged robot that hobbles along the desk is one thing; creating a C-3PO-like robot is quite another, even if your name is Anakin Skywalker.

Arms and Grippers

The ability to manipulate objects is a trait that has enabled humans, as well as a few other creatures in the animal kingdom, to take command of their environment. Without our arms and hands we wouldn't be able to use tools, and without tools we wouldn't be able to build houses, cars, and—hmmm—robots.

Arms and *grippers* are used by themselves in stationary robots, or they can be attached to a mobile robot. An arm can be considered any appendage of the robot that can be individually and specifically manipulated; grippers (also called *end-effectors*) are the hands and fingers and can be attached either directly to the robot or to an arm.

The human arm has joints that provide various degrees of freedom (*DOF*) for orienting it to most any direction. Likewise, robotic arms also have degrees of freedom. In most designs the number of degrees of freedom is fairly limited, to between one and three *DOF*. In addition to degrees of freedom, robot arms are further classified by the shape of the area that the end of the arm (where the gripper is) can reach. This accessible area is called the *work envelope*. Read more about these in Chapter 28, “Experimenting with Robotic Arms.”

You can duplicate human arms in a robot with just a couple of motors, some metal rods, and a few other parts. Add a gripper to the end of the robot arm and you've created a complete arm-hand module. Of course, not all robot arms are modeled after the human appendage. Some look more like forklifts than arms, and a few use retractable push rods to

move a hand or gripper toward or away from the robot.

Robot grippers come in a variety of styles; few are designed to emulate the human counterpart. A functional robot claw can be built that has just two fingers. The fingers close like a vise and can exert, if desired, a surprising amount of pressure. See Chapter 29, “Experimenting with Robotic Grippers,” for more information.



Figure 2-7 Bipedal (two-legged) robots present special challenges, not only in construction, but in programming. Standardized metal brackets, like those used here, make building easier by not requiring you to have a complete metalworking shop in your garage (though if you do, by all means use it!). (Photo courtesy Lynxmotion.)

Android and Humanoid

Android and *humanoid* robots are specifically modeled after the human form: a head, torso, two legs, and possibly one or two arms. In current usage, the terms *android* and *humanoid* are not the same: an *android* is a robot designed to look as much like a human being as possible, including ears, hair, and even an articulated mouth. A *humanoid* robot is one that shares the basic architecture of a human—bipedal (two legs), head at the top, two arms at the side—but is not meant to be a physiological replica.

Figure 2-7 shows an example humanoid bipedal robot that you can actually build; Figure 2-8 shows a make-believe android bot that for most hobbyists is beyond the reach of pocket-book and technology. Menacing grin optional.

Whether a true android or a humanoid, this category of robots is something of the Holy Grail

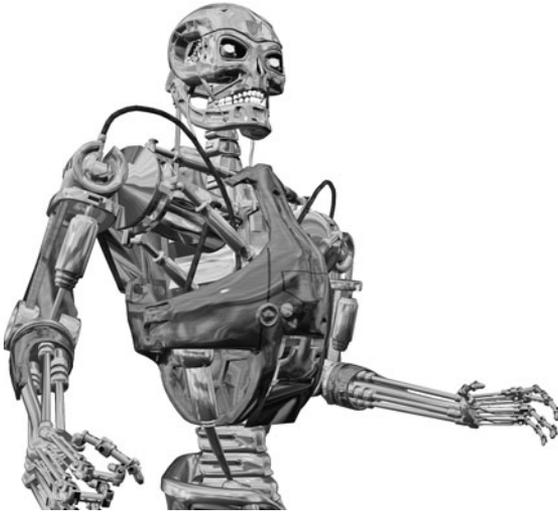


Figure 2-8 Androids take a human form, with head, torso, two arms, and two legs. These designs are the most difficult to achieve, even for companies spending millions of dollars in research and development. The robot shown here, drawn by a 3D modeling program, exists only in the imagination. For now, anyway.

of robot builders and is also the most difficult to create. Because of the added complexities in the design, the cost of construction is much higher than with other forms of robots—spending \$300+ in parts is not uncommon.

The compelling rationale of human-shaped robotics is that since the machine walks on two legs, it can live and work in the same environment as humans. Contrast this with a robot that must roll on wheels or tracks: stairs become difficult, and even clothing discarded on the floor can impede the motion of the robot.

SKELETAL STRUCTURE

In nature and in robotics, there are two general types of support frames: endoskeleton and exoskeleton. Which is better? Neither and both: In nature, the living conditions of the animal and its eating and survival tactics determine which skeleton is best. The same is true of robots.

- *Endoskeleton* support frames are the kind found in many critters—including humans, mammals, reptiles, and most fish. The skeletal structure is on the inside; the organs, muscles, body tissues, and skin are on the outside of the bones. The endoskeleton is a characteristic of vertebrates.
- *Exoskeleton* support frames have the “bones” on the outside of the organs and muscles. Common exoskeletal creatures are spiders, all shellfish such as lobsters and crabs, and an endless variety of insects.

The main structure of the robot is generally a wood, plastic, or metal frame, which is constructed a little like the frame of a house—with a bottom, top, and sides. This gives the automaton a boxy or cylindrical form, though any shape is possible.

Onto the frame of the robot are attached motors, batteries, electronic circuit boards, and other necessary components. In this design, the main support structure of the robot can be considered an exoskeleton because it is outside the major organs.



A shell or covering is sometimes placed over these robots, but the “skin” is for looks only (and sometimes the protection of the internal components), not support. For the most part, the main bodies of your robots will have exoskeleton support structures because they are cheaper to build, stronger, and less prone to problems.

FLESH AND BONE—AND WOOD, PLASTIC, OR METAL

In the 1926 movie classic *Metropolis*, Rotwang—a disgruntled scientist—transforms a cold and calculating robot into the body of a beautiful woman. This film, generally considered to be the first science fiction cinema epic, also set the psychological stage for later movies about robots, particularly those of the 1940s and '50s.

The shallow and stereotypical character of Rotwang and his robot creation, shown in the movie still in Figure 2-9, proved to be a common theme in countless movies since. The shapely robotrix changed form for these other films, but not its evil character. Robots have often been depicted as metal creatures with hearts as cold as their steel bodies.

Which brings us to an interesting question: Are all “real” robots made of heavy-gauge steel, stuff so thick that bullets, disinto-ray guns, even atomic bombs can’t penetrate? Yes, metal is a common part of many kinds of robots, but the list of materials you can use is much larger and more diverse.

- **Wood.** Wood is an excellent material for robot bodies, especially multi-ply hardwoods, like the kind used for model airplanes and model sailboats. Common thicknesses are 1/8” to 1/2”—perfect for most robot projects. Read up on wood robots in Chapter 7, “Working with Wood.”
- **Plastic.** Plastic boasts high strength, but is easier to work with than metal. You can cut it, shape it, drill it, even glue it, with common, everyday tools. My favorite is PCV expanded plastic. These sheets are known by various trade names such as *Sintra*, and they are available at industrial plastics supply outlets. Cheap and easy to work with. See Chapter 9, “Working with Plastic,” for more details.
- **Foamboard.** Art supply stores stock what’s known as *foamboard* (or *Foam Core*), a special construction material typically used for building models. It’s really a sandwich of paper or



Figure 2-9 The misguided Rotwang, and his robot, from the 1927 movie classic *Metropolis*. This is the shape of the robot before it’s transformed to look like a woman, a teacher leading a peaceful worker’s rebellion.

plastic glued to both sides of a layer of densely compressed foam. You can cut it with a knife or small hobby saw. Great stuff for quickie-made bots. I talk about foamboard and similar materials in Chapter 14, “Rapid Prototyping Methods.”

- *Aluminum.* If you want to go metal, aluminum is the best all-around robot-building material, especially for medium and larger machines. It’s exceptionally strong for its weight. Aluminum is fairly easy to cut and drill using ordinary shop tools. There’s more on building robots with aluminum in Chapter 11, “Working with Metal.”
- *Tin, iron, and brass.* Tin and iron are common hardware metals that are used to make angle brackets and sheet metal (various thicknesses, from 1/32” on up), and as nail plates for house framing. Cost: fairly cheap. See Chapter 13, “Assembly Techniques,” for ways to use common angle brackets in your robot construction plans.
- *Steel.* Although sometimes used in the structural frame of a robot because of its strength, steel (and its close cousin stainless steel) is difficult to cut and shape without special tools. Ideal for combat robots. As this book concentrates on small, lightweight, and easy-to-build tabletop bots, we’ll leave the discussion of steel at this.

Locomotion Systems

Locomotion is how a mobile robot gets around. It performs this feat in a variety of ways, typically using wheels, tank tracks, and legs. In each case, the locomotion system is driven by a motor, which turns a shaft, cam, or lever.

WHEELS

Wheels are the most popular method for providing robots with mobility. There may be no animals on this earth that use wheels to get around, but for us robot builders it’s the simple and foolproof choice. Wheels, like those on the robot in Figure 2-10, can be just about any size, limited only by the dimensions of the robot and your outlandish imagination. Turtle-sized robots usually have small wheels, less than 2” or 3” in diameter. Medium-size rover-type robots use wheels with diameters up to 7” or 8”. A few unusual designs call for bicycle wheels, which, despite their size, are lightweight but very sturdy.

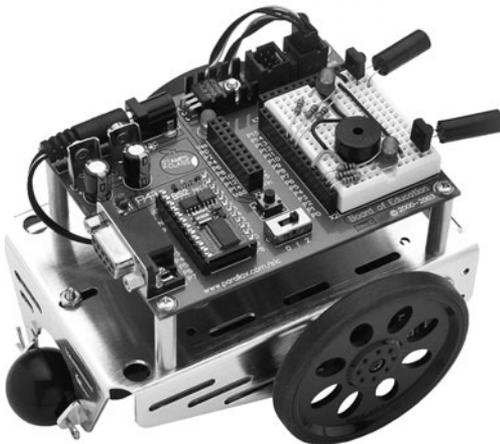


Figure 2-10 Wheels are the most popular way to make a robot move. The size, shape, and even the location of the wheels can vary, though two wheels counterbalanced on one or both ends of the bot is the most common design. (Photo courtesy Parallax Inc.)

Robots can have just about any number of wheels, although two is the most common. The robot is balanced on the two wheels by one or two free-rolling casters, or perhaps even a third swivel wheel or nonmoving skid. Four- and six-wheel robots are also around. You can read more about wheel designs in Chapter 20, “Moving Your Robot,” and the many hands-on projects in Chapter 26, “Build Robots with Wheels and Tracks.”

TRACKS

The basic design of *track-driven* robots is pretty simple. Two tracks, one on each side of the robot, act as giant wheels. The tracks turn, and the robot lurches forward or backward. For most robots, each track is about as long as the robot itself.

Track drive (see Figure 2-11) is practical for many reasons, including the fact that it makes it possible to mow through all sorts of obstacles. Given the right track material, track drive provides excellent traction, even on slippery surfaces like snow, wet concrete, or a clean kitchen floor. See Chapter 26 for some affordable tracked robots you can build yourself.

LEGS

Of all types of mobile robots, those that walk on legs present the greatest challenges. But they're also great conversation pieces! You must overcome many difficulties to design and construct a legged robot. First, there is the question of the number of legs and how the legs provide stability when the robot is in motion or when it's standing still. Then there's the question of how the legs propel the robot forward or backward, and—more difficult still!—the question of how to turn the robot so it can navigate a corner.

Your walking robots can have two or more legs. The fewer the legs, the more challenging the design. Two-legged (*bipedal*) robots use unique balancing methods to keep them from falling over. Four-legged (*quadrupeds*) are easier to balance, but good walking and steering can involve adding extra joints and some sophisticated math to make sure everything moves smoothly.

Robots with six legs (called *hexapods*) are able to walk at brisk speeds without falling and are more than capable of turning corners, bounding over uneven terrain, and making the neighborhood animals run for cover. As a result, the six-legged kind, like the one in Figure 2-12, are the most popular among robot makers. You can read about building your own leg-based robot in Chapter 27, “Build Robots with Legs.”



Figure 2-11 Tracks on a robot provide mobility on uneven or thick terrain like sand, grass, or rocks. Tracks may be made of rubber, plastic, or metal; the plastic ones are among the easiest to use and the least expensive.

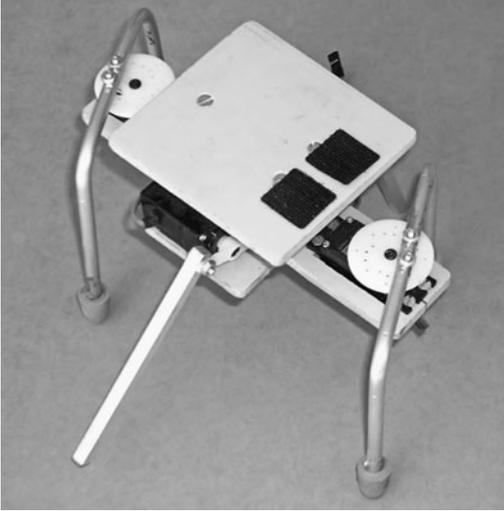


Figure 2-12 Legs allow a robot to navigate where wheels or tracks can't. Plus they look cool. This design uses four motors and six legs. It walks by moving its legs in specific patterns.

Power Systems

We humans eat food that is then processed by our bodies to power muscles. While you could probably design a digestive system for a robot and feed it hamburgers, french fries, and other semi-radioactive foods, an easier way to generate the power to make your robot go is to take a trip to the store and buy a set of batteries. Connect the batteries to the robot's motors, circuits, and other parts, and you're all set.

TYPES OF BATTERIES

There are many kinds of batteries, and Chapter 18, "All About Batteries," goes into more detail about which ones are best for robots. Here are a few quick details to start you off.

Batteries generate voltage and come in two distinct categories: rechargeable and nonrechargeable. *Nonrechargeable* batteries include the standard zinc and alkaline cells you buy at the supermarket, but only the alkaline kind is truly useful in robotics.

Rechargeable batteries include nickel-cadmium (NiCd), nickel metal hydride (NiMH), sealed lead-acid cells, and special rechargeable alkaline. NiCd and NiMH batteries are popular choices because they are relatively easy to find, come in common sizes, and can be recharged many times using an inexpensive charger.

OTHER POWER SOURCES

Small robots can be powered by solar energy when they are equipped with suitable solar cells. Solar-powered robots can tap their motive energy directly from the cells, or the cells can charge up a battery over time. Solar-powered bots are a favorite of those in the *BEAM* camp—a type of robot design that stresses simplicity, including the power supply of the machine.

Hydraulic power uses oil or fluid pressure to move linkages. You've seen hydraulic power at work if you've ever watched a bulldozer move dirt from pile to pile. And while you drive,

you use it every day when you press down on the brake pedal. Similarly, *pneumatic* power uses air pressure to move linkages. Pneumatic systems are cleaner than hydraulic systems, but, all things considered, they aren't as powerful.

Both hydraulic and pneumatic systems must be pressurized to work, and this is commonly done using a pump that's driven by an electric or gas motor. Hydraulic and pneumatic systems are harder and more expensive to implement, but they provide an extra measure of power. With a few hundred dollars in surplus pneumatic cylinders, hoses, fittings, and other parts, you could conceivably build a hobby robot that picks up chairs, bicycles, even people!

These alternative power sources aren't covered here, but if you're interested, check out the local library for books on solar, hydraulic, and pneumatic systems.

Sensing Devices

Imagine a world without sight, sound, touch, smell, or taste. Without these senses, we'd be nothing more than an inanimate machine, like the family car, the living room television, or that guy who hosts the Channel 5 late-night movie. Our senses are an integral part of our lives.

The more senses a robot has, the more it can interact with its environment. That's important for the robot to go about its business on its own. Figure 2-13 shows how sensors form an integral part of the robot.

TOUCH

The sense of *touch* is provided by mounting switches to the robot's frame, so that it can detect when it's run into something—or something has run into it. Another way to register touch is with a pressure sensor, which detects when a certain amount of force is exerted on it. You can build your own inexpensive pressure sensors or buy them ready-made for a few dollars. See Chapter 42, "Adding the Sense of Touch," for details.

LIGHT AND SOUND

It's easy to make a robot sensitive to *light*; in fact, these are among the least-expensive sensors you can add to a robot. Chapter 44, "Robotic Eyes," provides a number of plans for giving sight to a robot. You can even connect a video camera and beam the picture from your spybot to another room.

You can use light as a way to remotely communicate with your robot. Using a standard infrared remote control, you can program your robot to watch for the special coded signals that happen each time you press a button.

The same goes for sensitivity to *sound*. Sound is easy to detect. Find out more in Chapter 46, "Making and Listening to Sound."

SMELL AND TASTE

With some fairly inexpensive sensors, you can equip your bot to effectively smell for dangerous toxic gas, sniff around for smoke, sound the alarm if it detects the flame of a fire, or hit the panic button if things get too hot. See Chapter 48, "Danger, Will Robinson!," for some ideas and hands-on projects on sensing fire, heat, flame, and gas.

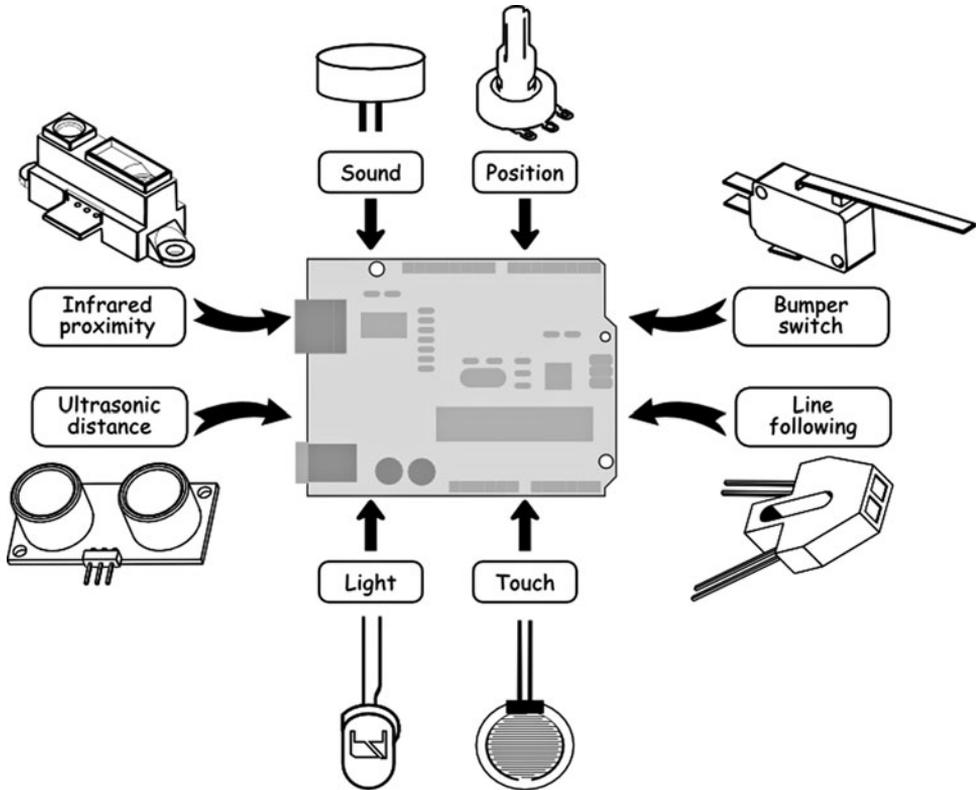


Figure 2-13 Sensors let your robot experience the world around it. Humans may perceive their environment using eyesight and hearing, but most robots use simpler technologies like ultrasonic and infrared distance measuring or light, sound, and touch. The sensors are integrated and processed by a control circuit, which then performs some action based on the sensory input.

A SIXTH SENSE OF THINGS NEARBY

Humans use an elaborate system of sight, sound, and other senses to determine when something is close by. For robots, there are whole families of cheap sensors that do the same thing and are very easy to implement.

One type uses *ultrasound*, just like a bat. The sensor emits a short ping—similar to the sonar of a submarine—then waits for its echo. The time it takes between sending the ping and getting the return echo tells the sensor how far away something is. Sounds complicated, but the whole thing now comes prepackaged in a small device that costs less than \$20.

Another scheme uses invisible *infrared light* and a bit of mathematics to measure distances. The system is more fully explained in Chapter 43, “Proximity and Distance Sensing,” but suffice it to say, these sensors are even smaller and less expensive than ultrasonic detectors.

TILT, MOTION, AND POSITION

Rounding out the sensory systems of today’s amateur robot are things like accelerometers for detecting tilt and motion, gyroscopes for determining speed, and global positioning satellite

receivers for checking where on this earth your robot currently is. Read more on these and other cool devices in Chapter 45, “Navigating Your Robot.”

Output Devices

Output devices are components that relay information from the robot to the outside world. Here are examples of output devices in robots:

- Light-emitting diodes (LEDs) that shine or blink, to indicate what the robot is doing—or trying to do.
- Liquid-crystal displays (LCDs), for showing complete messages.
- Sounds that allow your bot to communicate in nonvisual ways. You might add a speech synthesizer so your robot can talk to you. Or program it to wake you up in the morning by playing your favorite MP3s. You can record any sound and have your robot reproduce it instantly.

Read more on these and other ideas in Chapter 46, “Making and Listening to Sound,” and Chapter 47, “Interacting with Your Creation.”

Where the Word “Robot” Comes From

I close this chapter with a wee bit of pontification: where does the term *robot* come from, and what does it mean to the average robot enthusiast?



Figure 2-14 Robots are made for the drudgery of everyday work. In Czech, Slovak, and Polish languages, the word *robota* (from which *robot* is derived) means “work.” (Rendering courtesy Christopher Schantz.)

The term *robota*, from which the common word *robot* is derived, was first coined by Czech novelist and playwright Karel Capek. He used it first in a short story and then again in a now-classic play titled *R.U.R.*—which stands for “Rossum’s Universal Robots.”

The play was first produced onstage in 1921 and is one of many written by Capek that have a *dystopian* theme—a utopia gone wrong. The basic premise of the play’s “perfect society” is that the society is fatally flawed. In *R.U.R.* the robots are created by humans to take over all labor, including working on farms and in factories. Eventually the automatons conspire against their flesh-and-bone masters and rise up to kill them.

The term *robota* means “work” or “servitude”—like the poor mechanical dude in Figure 2-14, spending its days outdoors, sweeping up other people’s messes. It is the *work* aspect of robotics that is often forgotten, but it defines a “robot” more than anything else.

The work doesn’t have to be complicated. It can be something as simple as a robot that patrols the living room looking for signs of warm-blooded creatures and, when it finds one, sounds off a shrill siren alarm. Annoying, yes, but a robot nonetheless!

Many of us get caught up in trying to make our robot creations do serious work around the house—vacuum the carpet maybe, get a soda can from the refrigerator, or collect dirty socks from the floor. Noble aims, to be sure, but it’s perfectly acceptable to build a robot that’s merely for fun. Build it to amuse you and your family.

Or play tricks with the cat.

Getting Parts

Exactly where do you find robot parts? Your friendly neighborhood robot store would be a good place to start—if only such a store existed! Fortunately, other local and online retail stores are there to fill in the gaps.

The fundamental stuff of bots includes:

- *Basic electronic parts* to control your robot. These include core electronic components such as resistors and capacitors, switches, wire, and relays.
- *Specialized electronic modules* to complete circuits that you connect with the rest of your robot. The options here are nearly limitless, but the main modules you'll be interested in include a microcontroller (holds programs for your robot, operates motors and other parts) and sensors.
- *Motors and wheels* for making your robot go.
- *Construction materials* for building the body, or base, of your robot. You can make the base from found material (old CDs, small plastic containers) or from wood, plastic, metal, foam, even heavy cardboard.
- *Fasteners, adhesives, and other building parts* to put everything together. These might include small screws and nuts, but also cable ties, Velcro, and glue.

In this chapter we review where you can find these and other parts for your robot projects.

FYI

For more information on finding and getting parts, be sure to see Appendix A, "RBB Online Support," which contains updated parts lists for the projects in this book, and Appendix B, "Internet Parts Sources."

Local Electronics Stores

Not long ago most towns had at least one local electronic parts store. Even some automotive outlets carried a full range of tubes, specialty transistors, and other electronic gadgets. Now, RadioShack remains as the only U.S. national electronics store chain. In many cities across the country, it's the only thing going.

The situation is much the same for those in other countries. There's Maplin Electronics and Farnell in the United Kingdom; in Australia and New Zealand you'll find Dick Smith.

Most of the nationwide chains like RadioShack continue to support electronics experimenters. But they stock only the very common components. They're great for getting emergency replacement parts or filling in the odds and ends. If your needs extend beyond common-value components, wire, and switches, you must turn to other sources—and for the best selection that means going online.



Be sure to check your phone company's business directory listings (Yellow Pages or similar) for any parts stores that you might not have heard about. Try the *Electronics Retail* and *Electric Equipment & Supplies* sections. Check out a printed phone directory, or look up your city at www.yip.com.

Online Electronics Outlets

There's no shortage of choices for online electronics stores. While you have to wait for the parts to ship to you, the selection is better, and (often) so are the prices.

Some of the online outlets are so vast you'll be hard-pressed to just "browse the aisles." Their catalogs are extensive, numbering hundreds of pages. You really need to know what you're looking for, and fortunately they all provide a search function on their Web sites. You may look for specific parts or search by category.

Here are some of the key online electronics retailers in North America that you'll want to know about:

All Electronics: www.allelectronics.com (they also have a retail store in the Los Angeles area)

DigiKey: www.digikey.com

Jameco Electronics: www.jameco.com

Mouser Electronics: www.mouser.com

RadioShack: www.radioshack.com

This is only a short list, and there are others, including those outside the United States. See Appendix B, "Internet Parts Sources," for more.

Using FindChips.com to Locate Parts

If you're looking for a particular electronic component, use the specialized search at www.FindChips.com (Figure 3-1). Enter the part number or part reference, and the site will search

FindChips.com

Searching 20 distributors for ATmega328.
This is query number 151,548,015

Feedback? Contact us

Newark - MEDIA Member - Authorized Distributor (1 of 20)

Part #	Mfg.	Description	Newark #	Part Details	Stock	Price	Buy
ATMEGA328P-AU	ATMEL	8-Bit Microcontroller IC, Controller Family/Series AVR-MEGA, Core Size 8-Bit, No. of I/O's 23, Program Memory Size 256Kb, EEPROM Memory Size 1Kb, RAM Memory Size 2Kb, CPU Speed 20MHz, Oscillator Type External, Internal, JTAG Compliant Yes	1458431	Each	0	1.00-4.6, 100-240, 3.75-250, 490-3.47, 500-990, 3.2-1000W, More 2.89	Buy
ATMEGA328P-PU	ATMEL	8-Bit Microcontroller IC, Controller Family/Series AVR-MEGA, Core Size 8-Bit, No. of I/O's 23, Program Memory Size 256Kb, EEPROM Memory Size 1Kb, RAM Memory Size 2Kb, CPU Speed 20MHz, Oscillator Type External, Internal, JTAG Compliant Yes	1592268	Each	0	1.00-4.6, 100-240, 3.75-250, 490-3.47, 500-990	Buy

Figure 3-1 Use Findchips.com to locate electronic components. The free site searches numerous online parts retailers and displays availability, price, and stocking levels.

over a dozen online electronics specialty retailers. You can compare prices and availability. Using the FindChips.com search is free.



Some of the sources on FindChips.com are electronics distributors and wholesalers, which means they require a hefty minimum order. The minimum varies, but is often \$25 or \$50.

Specialty Online Robotics Retailers

With robotics now a hot hobby, a new type of online retailer has sprung up: the specialty robotics site. These sites offer one-stop-shopping for all—or nearly all—of the parts you need to build your robots.

Inventory differs from one site to the next, and some products are exclusive to one company. No doubt you'll develop your favorite haunts. I regularly shop at several of these, looking for what's new and who has the best prices.

Acroname Robotics: www.acroname.com

Lynxmotion: www.lynxmotion.com

Parallax: www.parallax.com

Pololu: www.pololu.com

RobotShop: www.robotshop.com

SparkFun Electronics: www.sparkfun.com

See Appendix B, “Internet Parts Sources,” for more.

Hobby and Model Stores

Hobby and model stores are the ideal sources for small parts, including lightweight plastic, brass rod, servo motors for radio control (R/C) cars and airplanes, gears, and construction hardware. Most of the products available at hobby stores are designed for building specific kinds of models and toys. But that shouldn't stop you from raiding the place with an eye to converting the parts for robot use.

In my experience, many hobby store owners and salespeople are not up to speed on using their products in robotic applications. Your best bet is to browse the store and look for parts that you can put together to build a robot. Some of the parts, particularly those for R/C models, will be behind a counter, but they should still be visible enough for you to conceptualize how you might use them.

If you don't have a well-stocked hobby store in your area, there's always mail order through the Internet. Here are a few of the larger eHobby outlets:

Hobby Lobby: www.hobbylobby.com

Hobby People: www.hobbypeople.net

Horizon Hobby: www.horizonhobby.com

Tower Hobbies: www.towerhobbies.com

Craft Stores

Craft stores sell supplies for home crafts and arts. As a robot builder, you'll be interested in some of the aisles with:

- *Foam rubber sheets.* These come in various colors and thicknesses and can be used for pads, bumpers, nonslip surfaces, tank treads, and lots more. The foam is very dense; use a sharp scissors or knife to cut it (I like to use a rotary paper cutter to get a nice, straight cut).
- *Foamboard.* Constructed of foam sandwiched between two heavy sheets of paper, foamboard can be used for small, lightweight robots. Foamboard can be cut with a hobby knife and glued with paper glue or hot-melt glue. Look for it in different colors and thicknesses.
- *Parts from dolls and teddy bears.* These can often be used in robots. Fancier dolls use something called *armatures*—movable and adjustable joints—that can be applied to your robot creations. Look also for linkages, bendable posing wire, and eyes (great for building robots with personality!).
- *Electronic light and sound buttons.* These are designed to make Christmas ornaments and custom greeting cards, but they work just as well in robots. Electric light kits come with low-voltage LEDs or incandescent lights, often in several bright colors. Some flash at random, some in sequence. Sound buttons have a built-in song that plays when you depress a switch. You could use these buttons as touch sensors, for example, or as a “tummy switch” in an animal-like robot.
- *Plastic crafts construction material.* This can be used in lieu of more expensive building kits. For example, many stores carry the plastic equivalent of that old favorite, the Popsicle stick. (You can also get the original wooden ones, but they aren't as strong.) The plastic sticks have notches in them so they can be assembled to create frames and structures.
- *Model-building supplies.* Many craft stores have these, sometimes at lower prices than the average hobby-model store. Look for assortments of wood and metal pieces, adhesives, and construction tools.

There are, of course, many other interesting products of interest at craft stores. Visit one and take a stroll down its aisles. Don't have a craft store nearby? There are plenty online. See Appendix B, “Internet Parts Sources,” for several of the larger outlets.

Hardware and Home Improvement Stores

Hardware stores and builder's supply outlets are a good source for hand tools for building your robots. They're also good for screws, nuts, and other fasteners, adhesives, and the materials for bot bases.

As you tour your neighborhood hardware store, keep a notebook handy and jot down what they offer. Then, when you find yourself needing a specific item, you have only to refer to your notes. Take an idle stroll through your regular hardware store haunts on a regular basis. You'll always find something new and laughably useful for robot design each time you visit. I know I do.

Samples from Electronics Manufacturers

Shhhh! I'm not supposed to tell you this, but a number of electronics manufacturers will send you free samples, just for the asking.

Okay, it helps if you're an instructor at a school or an engineer working for a company building millions of units of some gizmo. But the truth is, some semiconductor and electronics makers will send free samples to students and hobbyists, too. Just don't be greedy, and observe limits to their kindness.

Availability of freebies is especially useful, as the newest semiconductors may not yet be carried by distributors and wholesalers, making it hard to purchase these products. You can get a jump ahead by asking for free samples and incorporating them in your work.

Sample kits are also occasionally available from wholesalers, distributors, and resellers. Look for kits of parts, fasteners, or various hardware. The kits are limited and are obviously meant to entice you into buying more, but they're free, and you never know when some piece from a kit will come in handy.

If you belong to a school or a robotics club, send your request (preferably on professional-looking letterhead) under the auspices of your organization. Be sure you have permission from the school or club! Most electronics manufacturers realize that today's robotics students and hobbyists are tomorrow's engineers, and they want to actively foster a good working relationship with them (er, you).



Be warned: Many of the latest integrated circuits are available only in surface-mount packages. To use these you may need to solder the integrated circuit to a carrier board. This adds to the cost (you have to buy the board, and they're not always cheap), and you need excellent soldering skills.

Finding What You Need on the Internet

The Internet has given a tremendous boost to the art and science of robot building. Through the Web, you can now search for and find the most elusive part for your robot. Moreover, with the help of Web search engines—such as Google (www.google.com) or Bing (www.bing.com)—you can look for topics of interest from among the millions of Web sites throughout the world.

Remember that the Internet is worldwide. Some of the sites you find may not be located in

your country; many Web businesses ship internationally, but not all do. Check for any specific payment requirements and shipping restrictions. If they accept checks or money orders, the denomination should be in the company's native currency.



If your design requires parts that are no longer widely available or that have you pull the guts out of a certain toy that's long gone from the clearance aisle, try eBay (www.ebay.com) or the classified ads on Craigslist (www.craigslist.org).

Shop Once, Shop Smart

Whether buying locally or through mail order—and that includes the Internet—you'll want to get as many of the parts as you need at once. This saves time, trouble, and expense.

When getting parts locally, there's the hassle of returning to the store for last-minute additions. That costs you time and gas. And when buying through the mail, repeat orders pile up the shipping costs. A transistor may cost only 25 cents, but add order minimums and shipping fees and you could easily be looking at \$5 or \$10 on your credit card.

Some ideas for savvy shopping:

- Keep an inventory of what you have, and think ahead. Plan your next several projects, and get as many of the parts for them at the same time as you can.
- Try to group purchases together when buying from the same store, even if the store doesn't have the lowest price. If the difference is minor, consider the additional costs of driving or shipping from another source.
- For very basic electronic parts try to get an assortment of standard values. Things like resistors and capacitors (discussed in Chapter 31, "Common Electronic Components for Robotics") are just pennies each. A \$10 (or whatever) assortment of the most common values will save you time and money.
- Don't forget what you already have! See "Getting Organized," later in this chapter, on how to keep a good inventory of your stock.

Haunting the Surplus Store

Surplus is a wonderful thing, but most people shy away from it. Why? If it's surplus, the reasoning goes, it must be worthless junk. That's simply not true. Surplus is exactly what its name implies: extra stock. Because the stock is extra, it's generally priced accordingly—to move it out the door.

Surplus stores that specialize in new and used mechanical and electronic parts are a pleasure to find. (Don't confuse these with stores that sell surplus clothing, camping, and government equipment. Completely different animal.) Many urban areas have at least one mechanical/electronics surplus store. Bear in mind that surplus stores don't have mass-market appeal, so finding them is not always easy. Start by looking in the phone company's business directory under *Surplus*.

Surplus parts are also available through the mail. There's a limited number of mail-order surplus outfits that cater to the hobbyist, but you can usually find everything you need if you look carefully enough and are patient. See Appendix B, "Internet Parts Sources," for leads.

While surplus is a great way to stock up on mechanical parts such as big DC motors and gears, you must shop wisely. Just because the company calls the stuff surplus doesn't mean that it's cheap—or even reasonably priced. A popular item in a surplus catalog may sell for top dollar, simply because of high demand.

Getting Parts from Specialty Stores

Specialty stores are outlets open to the general public that sell items you won't find in a regular hardware or electronic parts store. These don't include surplus outlets, which were discussed in the previous section.

What specialty stores are of use to robot builders? Consider:

- *Toy stores.* Look for construction toys like Erector, Meccano, LEGO, and others. Check out their battery-operated toy vehicles, such as motorized tanks. Raid the motors and rubber treads, like the ones in Figure 3-2, for your robot. This is how I get parts for most of my tracked robots.
- *Sewing machine repair shops.* These are ideal for finding small gears, cams, levers, and other precision parts. Some shops will sell broken machines to you. Tear them to shreds and use the parts for your robot.
- *Auto parts stores.* The independent stores tend to stock more goodies than the national chains, but both kinds offer surprises in every aisle. Keep an eye out for small switches, wire, and tools.
- *Junkyards.* If you're into building bigger robots, old cars are good sources for powerful DC motors, which are used to drive windshield wipers, electric windows, and automatic adjustable seats (though take note: such motors tend to be terribly inefficient for battery-based bots).
- *Lawn mower sales-service shops.* Lawn mowers use all sorts of nifty control cables, wheel bearings, and assorted odds and ends. Pick up new or used parts for a current project.
- *Bicycle sales-service shops.* I don't mean the department store that sells bikes, but a *real* professional bicycle shop. Items of interest: control cables, chains, brake calipers, wheels, sprockets, brake linings, and more.

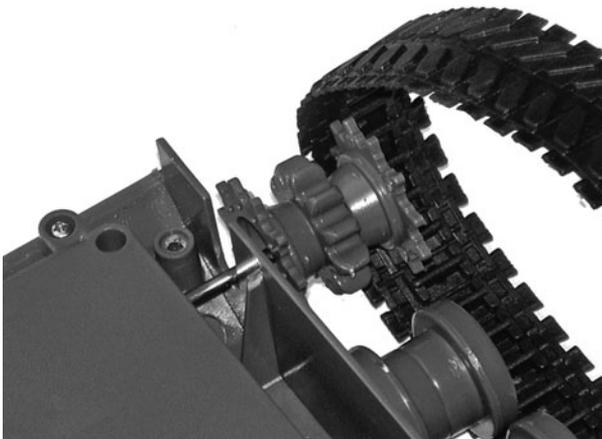


Figure 3-2 Toys are among the best sources for robotic parts, especially motorized military tanks or construction vehicles. These can be turned into robots with tracks.

- *Industrial parts outlets.* Some places sell gears, bearings, shafts, motors, and other industrial hardware on a one-piece-at-a-time basis. The penalty: fairly high prices and often the requirement that you buy a higher quantity of an item than you really need.

Scavenging: Making Do with What You Already Have

You don't need to buy new (or used or surplus) to get worthwhile robot parts. In fact, some of the best parts for hobby robots may already be in your garage or attic. Consider the typical used VCR. It'll contain at least one motor—and possibly as many as five—numerous gears, and other electronic and mechanical odds and ends.

Depending on the brand and when it was made, it could also contain belts and pulleys, miniature push buttons, infrared light emitting diodes and detectors, and even wire harnesses with reusable connectors. Any and all of these can be salvaged to help build your robot. All told, the typical discarded VCR may have over \$30 worth of parts in it.



Never throw away small appliances or mechanical devices without taking them apart and robbing the good stuff. If you don't have time to disassemble that CD player that's been skipping on all of your compact discs, throw it into a pile for a rainy day when you do have a free moment.

Likewise, make a point of visiting garage sales and thrift stores from time to time, and look for parts bonanzas in used—and perhaps nonfunctioning—goods. I regularly scout the local resale stores and for very little money come away with a trunk full of valuable items that I can salvage for parts. Goods that are still in functioning order tend to cost more than the broken stuff, but for robot building the broken stuff is just as good. Be sure to ask the store personnel if they have any nonworking items they will sell you at a reasonable cost.

Here is just a short list of the electronic and mechanical items to be on the lookout for and the primary robot-building components they have inside:

- *VCRs* are perhaps the best single source for parts, and they are in plentiful supply. As discussed, you'll find motors, switches, cable harnesses, and other useful goodies.
- *CD players* have optics you can gut out if your robot uses a specialty vision system. Apart from the laser diode, CD players have focusing lenses and other optics, as well as miniature motors.
- *Old fax machines* contain numerous motors, gears, small switches, and other mechanical parts.
- *Mice, printers, old scanners, disk drives, and other discarded computer add-ons* contain valuable optical and mechanical parts. Mice contain optical encoders that you could use to count the rotations of your robot's wheels; printers contain motors and gears; disk drives contain motors; and so on.
- *Mechanical toys*, especially the motorized variety, can be used either for parts or as a robot base. When looking at motorized vehicles, favor those that use separate motors for each drive wheel.

Getting Organized

Over his 20 years in the military, one of the principal jobs my father had was taking charge of supplies. He had the knack for organizing where everything went—from huge trucks and airplanes to miniscule ball bearings.

If apple trees are organizational traits, this apple (me) fell far from the tree: I inherited virtually none of my dad's mind-set to keep things in their place. All while growing up—and even now as an adult—I'd forget to put things back where they belonged. I've often ended up buying the same thing two and even three times because I'd lost track of my garage inventory.

So, knowing my shortcomings, I now make a conscious effort to maintain order in the shop, or else things rapidly get messy and increasingly expensive. It gets old taking a half hour just to find a part.

While my shop is far from a Robotics Eden, it's now more or less well organized, and this with the help of some small parts cabinets, a few plastic parts bins, and zipper-style plastic bags. My system works for me, but you'll undoubtedly want to improvise your own.

SMALL PARTS DRAWER CABINETS

Plastic drawer cabinets are really the mainstay for any activity that deals with small parts. You can get them in all styles and sizes, from little units with just a half dozen 1" × 2" drawers (these fit on bookshelves quite well) to much larger parts chests, with 20, 30, even 40 drawers of different shapes and sizes. These won't fit on a bookshelf—unless your shelves are for very large books!—but are ideal for workbenches and assembly tables.

I like the cabinets with several drawer sizes, which accommodate parts of different bulks. For example, the cabinet might sport 24 or 30 small drawers of about 1.5" × 2" and 6, 9, or 12 large drawers of about 2" × 4". You might, for instance, place individual values of resistors in the smaller drawers and large capacitors in the large drawers.



Drawer cabinets vary greatly in quality, and I prefer to check them out in person when possible. You can find them at any home improvement, department, or discount store. Look for drawers with thick enough plastic that they won't easily break.

HEAVY-DUTY STORAGE DRAWERS

A step up the size and cost ladder is the plastic heavy-duty storage bin. These are 1- to 3-foot-high units that sport between three and six large, heavy-duty plastic drawers. The drawer sizes may be different or all the same; find a model that suits your needs. The most robust models can readily hold heavy motors, metal gears, and other construction parts.

Larger still (and somewhat more flexible) are stacking drawers, with sizes big enough to hold bulky sweaters and pants. They're made to stack on top of one another, so you can get as few or as many as you want. (Avoid overdoing the height of your Tower of Babel, or else things could topple if you open a heavy drawer near the top of your stack.)

TOOLBOXES AND TOTES

Sometimes you have to build while on the go, and you'll need to bring your tools and supplies with you. For the big stuff, a standard metal or heavy-duty plastic toolbox is the best choice. I



Figure 3-3 A plastic toolbox or tote helps you keep all your robot-building supplies in one convenient location. Get one with several compartments for storing small parts.

still have my Sears Craftsman 24" steel toolbox that I bought over 30 years ago, and I use it regularly.

For lighter jobs, a plastic fishing tackle box or tote (see Figure 3-3) makes it easy to lug your supplies. The typical tackle box has a storage drawer on the top for small parts. When you open the top, the drawer slides up and over, and you can reach into the bottom of the box for larger tools and supplies.

KEEPING TRACK OF YOUR INVENTORY

Unless you have perfect memory you'll need some system to keep track of what has gone where. On the low end of the scale is the old "Magic-Marker-on-the-side-of-the-parts-bin trick." (You can use any type of felt-tip marker; a Sharpie is my favorite.)

Instead of writing directly on the plastic, you can instead tape an index card to the container and write on the card. If you change the contents of the container, just peel the card off and start over. Wide clear packing tape works well, too.

For smaller parts drawers, an electronic labeler is the absolute best way to keep track of parts. (Think of Vanessa Kensington from the movie *Austin Powers*, and how her suitcase was full of labeled bags.) Larger machines can accommodate labels of many different widths.



Need lots of storage bins for your garage? Try corrugated cardboard shelf bins. They lie flat until you need them. Fold a flap here, tuck a flap there, and your bin is ready for use. If you don't need the bin anymore, untuck the flaps and it'll store flat again. Outfits that sell shipping boxes sell these in quantity, usually in packages of 25, 50, and 100.

AD-HOC STORAGE SOLUTIONS

Not everything needs a fancy plastic storage bin. Sometimes simple—and free—is better. Here are some storage ideas for when you don't need a fancy solution.

- *Cardboard shoe boxes* still make great storage containers. Keep the lid so you can stack up the boxes, or just use the box open if you need quick access.
- *Zipper locking food storage bags*, particularly the heavy-duty ones for freezer use, make ideal containers for odd-sized items. Mark the contents with a Sharpie. Tip: The wide-bottom bags stand up on their own.
- *Baby food jars*, plastic or glass, are still an excellent storage solution for very small parts, such as 2–56 size hardware. (If you can get 'em, use the glass jars for electronics parts, as they don't generate static.)
- *Empty egg crates* and egg boxes are also useful for holding small parts, but take care not to overturn the crate or box, as the lid doesn't close over the hollow for the egg. If you're not careful, your parts may spill out or get mixed up.

This page intentionally left blank

Robot Construction

This page intentionally left blank

Safety First (and Always)

Robotics is a safe and sane hobby, but only if you practice it with caution and respect. It can involve working with soldering tools that can burn, saws that can remove bits of flesh, and household current that can electrocute.

Any dangers involved in robot building are easily minimized by taking a few simple steps to practice safe working habits. Take your time and think things through; you may never have to burn yourself with a soldering iron, cut yourself with a knife or saw, or shock yourself with an exposed electrical wire.

Project Safety

If you plan on constructing the bodies and frames of your robots, you'll need to work with saws, drills, and other building tools. Use all tools according to the manufacturer's instructions. Use tools only in well-lighted and well-ventilated areas. Wear proper clothing and shoes.

Power saws are especially dangerous, even those with safety guards and mechanisms. Never defeat them! Don't allow children in the work area. More:



- Wear eye protection at all times, especially when cutting and drilling material. The glasses or goggles should wrap around your temples to prevent stray debris from striking your eyes from the side.
- Make sure that saw blades and drill bits are sharp. If they're dull, sharpen or replace them.
- Some project plans require the use of sharp knives for cutting—making robot bases out of foamboard, for example. Use a sharp knife, and cut on an appropriate surface. Don't use your free hand to hold down the board; if the knife slips, you could get badly cut. Hold down the board with a straightedge.
- Wear ear protection when using saws or any other power tools.



I can't stress enough the importance of using adequate eye protection. I'm fortunate enough to have both of my eyes, but I came close to losing sight in one eye from flying debris. As a result, in my workshop eye protection is now mandatory for myself and all helpers.

Battery Safety

Batteries used in robots may produce only a few volts, but they can generate lots of current—so much current that if the terminals of the battery are shorted, the battery could get *very* hot. If you're lucky, the battery terminals will only melt. But exploding batteries that cause fires are not unheard of.

Never short out the terminals of a battery just to see what'll happen. Store charged batteries so that the terminals will never come into contact with metal objects. Always be sure to recharge batteries in a recharger meant for that type of battery.

Soldering Safety

Soldering electronic circuits requires that you use a *very* hot iron or pencil. Temperatures exceed 600°F, which is enough to give you third-degree burns after only momentary contact with the tool. This temperature is equivalent to an electric stove burner set at medium-high heat, so you can imagine the dangers involved.

If you plan on doing any soldering, keep the following safety tips in mind:

- Always place your soldering tool in a stand designed for the job. Never place a hot soldering tool directly on the table or workbench.
- Mildly caustic and toxic fumes are produced during soldering. Maintain good ventilation to prevent a buildup of these fumes in your workshop. Avoid inhaling the fumes produced during soldering.
- If your soldering tool has an adjustable temperature control, dial the recommended setting for the kind of solder you are using, usually about 650° to 700° for standard 60/40 rosin-core solder.
- Always use rosin-core solder designed for use on electronics. Other kinds of solder could damage the circuit or your soldering tool.
- Do not attempt to solder on a “live” circuit—a circuit that has voltage applied to it. You run the risk of damaging the circuit, the soldering tool, and, most of all, you!

Fire Safety

Fire is a potential hazard during the construction and use of any electrical device. A hot soldering tool can ignite paper, wood, and cloth. A short circuit from a large high-current battery can literally melt wires. Although not common for the type of projects presented in this book, an electric circuit may develop too much heat, and could melt or burn its enclosure and surroundings. Proper construction techniques and careful review of your work will help prevent these kinds of mishaps.

If your project operates under house current, keep an eye on it for the first several hours of operation. Note any unusual behavior, including arcing, overheating, or circuit burnouts. If a circuit breaker trips while your project is on, you can bet that something is amiss with your wiring.



None of the projects in this book involve constructing circuits that are directly powered by household AC current. When AC current is used it's always through a commercially manufactured power supply or certified wall transformer. Whenever possible, avoid AC-powered circuits and use low-cost commercially available power supplies instead.

For obvious reasons, you should always build and operate your projects away from flammable objects, including gasoline, lighter fluid, welding and brazing equipment, and cleaners. Always keep a fire extinguisher near you, and don't hesitate to use it if a fire breaks out.

Melting plastic can release highly toxic chemicals and gases. After you put out the fire be sure to ventilate the area thoroughly. Melted PVC plastic can release hydrogen chloride gas. Seek medical attention if you're not feeling well.

Avoiding Damage by Static Discharge

The ancient Egyptians discovered static electricity when they rubbed animal fur against the smooth surface of amber. When the materials were rubbed together they tended to cling to one another. While the Egyptians didn't comprehend this mysterious unseen force, they knew it existed. Today we fully understand static electricity and know it can cause damage to electronic components.

As a robo-builder, you must take specific precautions against *electrostatic discharge*, otherwise known as *ESD*. Damage from static discharge can be all but eliminated by taking just a few simple steps to protect you, your tools, and your projects from static buildup.

THE PROBLEM OF ELECTROSTATIC DISCHARGE

Electrostatic discharge involves very high voltages at extremely low currents. Combing your hair on a dry day can develop tens of thousands of volts of static electricity. But the current (akin to the *force* of the electricity) is almost negligible. The low current protects you from serious injury.

Many electronic components that are manufactured with semiconductor material are not so forgiving. These include transistors and integrated circuits, especially those that use what's known as a metal oxide substrate. These include:

- MOSFET transistors
- CMOS integrated circuits (ICs)
- Just about any microcontroller
- Any module (digital compass, sensor) that contains one or more of the above

Don't worry if the names of these electronic devices are alien to you. You'll know a static-sensitive device when it comes to you in an antistatic plastic pouch or on antistatic foam. You should keep these devices in their pouch or stuck into their foam until you use them.

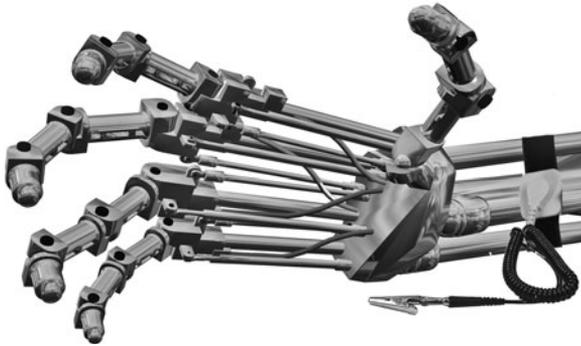


Figure 4-1 An antistatic wrist strap draws static electricity from your body, helping to prevent damage to sensitive electronic components. Be sure to use the wrist strap according to the manufacturer's instructions.

Just to be on the safe side, you should treat all semiconductors as delicate. You don't need to go overboard; things that aren't (usually) sensitive to ordinary static discharge include resistors, capacitors, diodes, motors, or any mechanical device, such as a switch or relay.

USING AN ANTISTATIC WRIST STRAP

If you live in a dry climate or where static is an ongoing problem, consider using an antistatic wrist strap whenever you work with sensitive electronics. This strap, like the one shown in Figure 4-1, grounds you at all times and prevents static buildup.

To use, put the strap around your wrist, then connect the clip to any grounded or large metallic object. A nearby computer (plugged in) or the frame of a metal desk or bookshelf are good choices. If you have an antistatic desk or floor pad it will likely have a metal stub on it that you can connect the wrist strap to. Even if the pad is itself not grounded, the idea here is that it dissipates the static because of its large surface area.

Though you may read otherwise, it's not a good idea to go sticking your wrist strap into the ground hole of an electrical wall socket. It's too easy to accidentally plug yourself into one of the other holes and receive a shock.

STORING STATIC-SENSITIVE DEVICES

Plastic is one of the greatest sources of ESD. Bench-top storage containers are often made of plastic, and it's a great temptation to dump everything into these containers. Avoid that. Invariably, static electricity will develop and a sensitive part could become damaged.

Unfortunately, there's no way to tell if a component has become damaged by ESD just by looking at it. You won't know anything is wrong until you actually try to use the component.

Static-sensitive electronics are best stored using one of the following methods. All work by connecting (grounding) the leads of the component together, thereby diminishing the effect of a strong jolt of static electricity. Note that none of the storage methods is 100 percent foolproof.

- *Antistatic mat.* This mat looks like a black sponge, but it's really conductive foam. Be sure to save your antistatic foam! You can use it to make pressure sensors, as described in Chapter 42, "Adding the Sense of Touch.")
- *Antistatic pouch.* Antistatic pouches are made of a special plastic coated on the inside with a metallic layer. Many are resealable so you can use them again and again.

- *Antistatic tube.* Quantities of integrated circuits are most often shipped and stored in convenient plastic tubes. They're treated with a conductive coating to help reduce static.



Remove the chip or transistor from its antistatic storage protection only when you are installing it in your project. The less time the component is unprotected, the better.

PROPER CLOTHING FOR STATIC REDUCTION

Avoid wearing polyester and acetate clothing, as these can develop static. A cotton lab overcoat can be worn to help reduce static electricity. Overcoats are available at many chemical and industrial supply houses. If nothing else, they'll make you look like an important scientist!

Only thing about the overcoat: Don't wear it when using power tools to cut or drill things. The long sleeves and dangling smock could get caught on spinning parts of the tool.

USE ONLY GROUNDED SOLDERING TOOLS

A common source of ESD damage when building electronic circuits is using an ungrounded soldering iron or pencil. Ungrounded tools have only two prongs on their power cord, instead of three. The third (round) prong is the ground connection.

A grounded tool not only helps prevent damage from electrostatic discharge, but lessens the chance of a bad shock should you accidentally touch a live wire. Be sure to use only a grounded wall socket; if you use an extension cord, be sure it, too, is grounded.

Working with House Current

None of the projects in this book directly use AC house current, but let's cover the safety precautions just the same. A live AC wire can, and does, kill. Exercise caution whenever working with AC circuits. You can greatly minimize the hazards of working with AC circuits by following these basic guidelines:

- Always keep AC circuits fully covered. Always.
- Keep AC circuits physically separate from DC circuits. If necessary, construct a plastic guard within your project to keep the wiring separate.
- All AC power supplies should have fuse protection. The fuse should be adequately rated for the circuit but should allow a fail-safe in case of short circuit.
- Place your AC projects in a plastic box. Don't use a metal project box.
- Double- and triple-check your work before applying power. If you can, have someone else inspect your handiwork before you switch the circuit on for the first time.

First Aid

Despite your best efforts, accidents might happen. With luck, most will be minor, causing little or no injury. If injury does occur, be sure to treat it promptly. If necessary, see a doctor to prevent the condition from getting worse.

Consult a good book on medical first aid treatment for details on how to care for cuts,

abrasions, and other minor injuries. You'll want to keep a first aid kit handy at all times, preferably right there in your shop, or conveniently located in a washroom or lavatory.

Purchase a first aid kit, and keep it in your workshop. If you use an item from the kit, be sure to replace it. Every year inspect your first aid kit and replace anything that is past its expiration or use-by date.

EYE INJURY FIRST AID

Perhaps most serious of all is injury to eyes. Be sure to wear adequate eye protection at all times—when soldering, when using shop tools, the whole bit. Chips of solder and metal leads can fly off when they are cut with nippers. If these get into your eye, not only is it excruciatingly painful, but it could damage your eye, temporarily or even permanently.

Should you get something in your eye, especially a piece of glass, metal, or plastic, see a physician *immediately*. Trying to remove the object yourself can cause further injury.

ELECTRIC SHOCK FIRST AID

Should you get a nasty shock from a circuit or battery, check the area of skin contact for signs of burns. Treat it as you would any other burn.

Immediately after an electrical shock, stop what you are doing and consciously calm yourself down. Monitor your pulse to make sure your heart isn't suffering from any aftereffects of the shock. If you feel anything is amiss, consult a doctor *immediately*.

Use Common Sense—and Enjoy Your Robot Hobby

Common sense is the best shield against accidents and injury, but common sense can't be taught or written about in a book. It's up to you to develop common sense and use it at all times. Never let down your guard. Don't ruin the fun of a wonderful hobby or vocation because you neglected a few safety measures.

Building Robot Bodies—the Basics

So we know robots are a clever combination of mechanics and electronics. The mechanical part constitutes the “body” of the robot—call it a frame, a base, a platform, it’s all the same.

Constructing a robot body comprises three straightforward tasks:

- Choose the material (you can mix and match).
- As needed, cut it to shape, and drill holes for mounting parts.
- Put it all together.

In this chapter you’ll be introduced to the principal materials used in constructing amateur robots, plus the basic tools you need to form the material into the size and shape you want.

Then, the remaining chapters in Part 2 give you explicit and detailed instructions for using these various materials. Chapters include hands-on plans for several robust robot platforms, which you can use as starting points for your robot creations.



Expensive, fancy tools are not required to complete any of the designs in this book. All you need are basic shop tools—you probably already have them in your garage, or you can borrow them from a friend or relative.

Picking the Right Construction Material

Your robots can be constructed out of wood, plastic, metal, even heavy-duty cardboard or foam. But which to choose?

That all depends on the design of the robot: how big and heavy it is and what you intend to use it for. It can be as simple as a piece of wood. Or it can be a fancy amalgam of high-tech plastic covering a body of extruded aluminum and steel support brackets.

Your budget and construction skills, plus the availability of heavier-duty tools, will also influence your choice of materials. It takes a lot less sweat to use a 1/4" sheet of plywood or 1/8" plastic than it does to cut or drill aluminum and steel. And it's easier still to make a robot using foamboard you purchase at the craft store.

- *Wood.* Wood is easy to work with, can be sanded and sawed to any shape, doesn't conduct electricity (avoids short circuits) unless wet, and is available everywhere.
- *Plastic.* Pound for pound, plastic has more strength than many metals, yet is easier to work with. You can cut it, shape it, drill it, even glue it. Some unique plastics might be harder to get unless you live near a well-stocked plastics specialty store; mail order is a great alternative.
- *Metal.* And by metal we're talking mostly aluminum here, though copper and brass sheets and tubes are commonly available in hobby stores. All three are great for building robots.
- *Lightweight composite materials.* Art supply stores stock what's known as "foamboard," also called Foam Core, a popular brand. Foamboard is a sandwich of paper or plastic glued to both sides of a layer of densely compressed foam. Heavy-duty cardboard and lightweight plastic board used to make signs are also contenders.

Let's take a closer look at these. Even more details on these construction materials are found in later chapters.

WOOD

If Noah can build an ark out of wood, it's probably good enough for robots. Wood is reasonably inexpensive and can be worked using ordinary tools.

Avoid soft "plank" woods, like pine and fir; and instead select a hardwood plywood designed for model building. Though more expensive—about \$7 to \$10 for a 12" × 12" square—this material is superior for robot building. These plywoods are less likely to warp, crack, or fleck. They're available at hobby and craft stores.

FYI See Chapter 7, "Working with Wood," for more details on using wood in your robots.

PLASTIC

There are literally thousands of plastics, but don't let that alarm you. For robotics, there are just a small handful of plastic materials that are both affordable and readily available. These same plastics also tend to be the ones most easily worked using standard shop tools.

- *Acrylic* is used primarily for decorative or functional applications, such as picture frames or salad bowls. It's usually clear but also comes in solid and translucent colors. You have to be careful of cracking caused by too much weight.
- *Polycarbonate* is similar in looks to acrylic but is considerably stronger. This plastic is a common substitute for window glass; because of its increased density, it's much harder to work with and is more expensive.
- A special type of PVC that comes in sheets, called expanded PVC, is ideal for making small and medium-size robots. An example robot made with PVC is shown in Figure 5-1. It's easier to work with than either acrylic or polycarbonate plastic.
- *Specialty plastics.* And last, there are a couple of plastics that are a bit harder to find (you get them at specialty plastics outlets; check your local Yellow Pages), but they offer certain

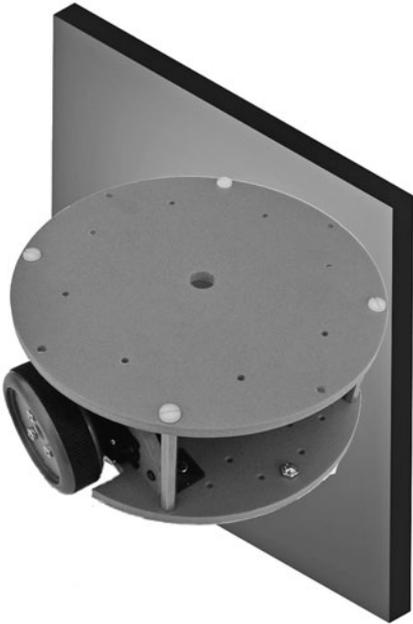


Figure 5-1 Expanded PVC is an ideal construction material for robotics. It's inexpensive and lightweight, and it cuts and drills like wood. It comes in a variety of colors and thicknesses to match the needs of your project.

advantages. These include ABS, acetal resin, and nylon. Their use in robotics is covered in more detail in Chapter 9, "Working with Plastic," for more details on using plastic materials.

METAL

The archetypal material for robots is metal. It's among the most expensive materials for robots—in terms of both cost and weight—and is harder to work unless you have the proper tools and skills. That said, metal is a must if your robot will be bashing other robots to death in combat or if it's made for rugged outdoor use.

For robots, aluminum and steel are the most common metals. Aluminum is a softer metal, so it's easier to work with. But steel is several times stronger, and it's easier to weld if you're planning on building a large robot.

There are three general approaches to metal construction in robots, shown in Figure 5-2:

- A *flat frame* provides the base of the robot and lends it support. A *box-shaped frame* is just what its name implies: a 3D box with six faces. It's well suited for larger robots or those that require extra support for heavy components.
- A *shaped base* is a piece of metal cut in the shape of the robot. The metal must be rigid enough to support the weight of the motors, batteries, and other parts without undue bending or flexing. Some very capable robots are basically a piece of sheet metal on wheels, with a laptop PC resting on top.

Robots made of wood, plastic, or other materials may nevertheless use metal (typically aluminum) in their construction. Common metal parts include brackets, to hold pieces together, and nuts, screws, and other fasteners.

FYI See Chapter 11, "Working with Metal," for more details on using metal materials in your robots.

LIGHTWEIGHT COMPOSITES

Not every robot needs to withstand a winter sandstorm on Mars. A technique known as *rapid prototyping* uses lightweight materials that are cut with basic hand tools, like knives and

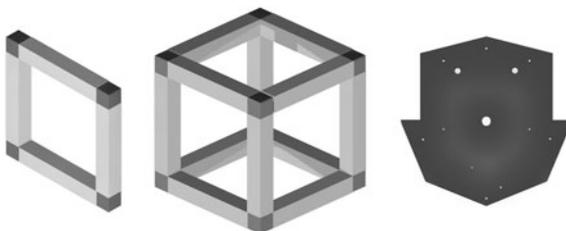


Figure 5-2 The three basic types of robot bases: square frame (it can actually be made of wood, plastic, or metal), a box frame, and a shaped base that provides the structure for the robot's components.

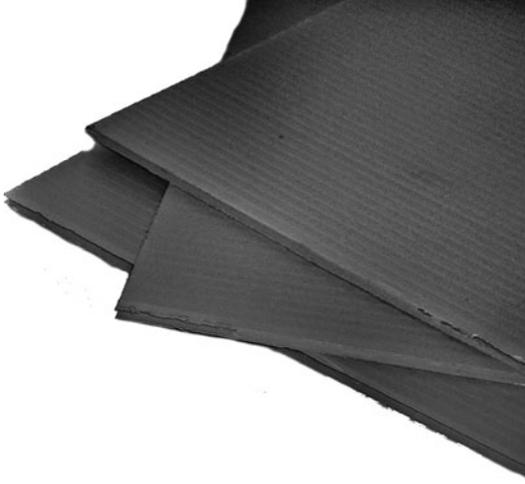


Figure 5-3 Corrugated plastic is like cardboard but made out of plastic. It's well suited for small and lightweight robots that you want to construct in a hurry. The material can be cut with a knife.

hobby saws, and awls or even nails for punching out holes. With rapid prototyping you can make a robot in less time, for less money. It's a good way to test out a design before you build it using stronger materials.

Rapid prototyping is such an important concept in amateur robot building that the subject gets its own chapter (see Chapter 14, "Rapid Prototyping Methods"), but for now, here are some of the typical materials used:

- *Heavy-duty cardboard* is surprisingly strong, yet easy to cut and drill. This stuff is heavier (and thicker) than your average cardboard box, but the concept is the same. It's made by sandwiching paper over a corrugated middle. Heavy automotive parts are often shipped in heavy-duty cardboard boxes, and you can buy the cardboard new.
- *Laminated composite* materials include foamboard, which is a piece of plastic foam inside two sheets of heavy paper. Other kinds of laminated composite sheets may use a combination of wood, paper, plastic, even thin metal.
- *Corrugated plastic* is a favorite among sign makers. They use sheets of it to make lightweight (and very affordable) indoor and outdoor signs. These look like cardboard, as in Figure 5-3, but they're entirely made of plastic.

In Review: Selecting the Right Material

Let's review the four main construction materials for building robots and compare their good and bad sides.

Material	Pros	Cons
Wood	Universally available; reasonably low cost; easy to work with using ordinary shop tools; hardwood plywoods (recommended wood for most robot bases) very sturdy and strong	Not as strong as plastic or metal; can warp with moisture (should be painted or sealed); cracks and splinters under stress

Material	Pros	Cons
Plastic	Strong and durable; comes in many forms, including sheets and extruded shapes; several common types of sheet plastic (acrylic, polycarbonate) readily available at hardware and home improvement stores; other types can be purchased via mail order	Melts or sags at higher temperatures; some types of plastic (e.g., acrylic) can crack or splinter with impact; PVC and many other plastics are not dimensionally stable under stress so they can bend out of shape; exotic types hard to find; better plastics are expensive; some specialty tools required for professional-looking cuts and holes
Metal	Very strong; aluminum available in a variety of convenient shapes (sheet, extruded shapes); dimensionally stable even at higher loads and heats	Heaviest of all materials; requires power tools and sharp saws/bits for proper construction; harder to work with (requires more skill); can be expensive
Composites	Lightweight and very easy to cut and drill using ordinary tools; allow rapid prototyping to test new designs and ideas; very inexpensive; come in many thicknesses	Not as strong as other materials; composites made with paper or wood can be damaged by moisture; some kinds may not be as easy to find except at specialty stores or online

Remember! There is no single “ideal” material for constructing robots. Each project requires a review of:

The robot itself, especially its physical attributes—large, small, heavy, light.

The tasks the robot is expected to do. Robots that do not perform heavy work, such as lifting objects or smashing into other robots, do not require heavy-duty materials.

Your budget. Everyone has a limit on what he or she can spend on robot materials. Tight budgets call for the least expensive materials.

Your construction skills. Robots made from wood, plastic, and composites are easier to build than metal ones.

Your tools. Building robots with metal or thick plastics require heavier-duty tools than when building wood or thick plastic bots.

Robots from “Found” Parts

Before leaving the subject of robot materials, I want to touch on a special construction style known as *found parts*. You’ll read more about the concept in Chapter 16, “Constructing High-Tech Robots from Toys,” and Chapter 17, “Building Bots from Found Parts,” but here’s a quick overview.

With found parts you adapt some ready-made manufactured product that you find (in a store, in your house, on the side of the road) to serve as the base of your robot. Inexpensive housewares, hardware items, and toys can be used in various creative ways to make robot building faster and more economical. Examples of found parts include old CDs and DVDs and plastic container boxes.

Basic Tools for Constructing Robots

Construction tools are the things you use to fashion the frame and other mechanical parts of your robo-buddy. These include such mundane things as a screwdriver, a saw, and a drill.

Take a long look at the tools in your garage or workshop. You probably already have everything you need to build your robot. But if you're short a tool or two, relax in knowing that constructing an amateur robot—at least the ones described in this book—doesn't require anything special.

There are other tools for constructing robot electronics. These tools are detailed in their own chapter; see Chapter 30, "Building Robot Electronics—the Basics."

TAPE MEASURE



You need a way to measure things as you build your robot. A retractable 6- or 10-foot tape measure is most convenient. Nothing fancy; you can get one for a few dollars at a discount store. Graduations in both inches and metric is helpful, but it's not critical.

A paper or fabric tape measure—one yard long, available at yardage stores, often for free—can substitute in a pinch but may not be as accurate. They're handy for measuring in tight places.

SCREWDRIVERS



You need a decent set of screwdrivers, with both flat and Phillips (cross) heads, as shown in Figure 5-4. These come in sizes; get #0 (small) and #1 (medium) Phillips, and small- and medium-tip flat-blade drivers. Magnetic tips are handy, but not necessary. Be sure to purchase a good set. Test the grips for comfort. The plastic of the grip should not dig into your palm. Try soft (rubber) coated grips for extra comfort.

HAMMER



None of the designs in this book call for pounding nails into wood, but you might still use a hammer for tapping parts into alignment or for using a center punch to mark a spot for drilling a hole. A standard-size 16-oz claw hammer is perfect for the job,

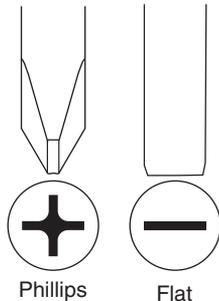


Figure 5-4 The basic assembly tool for making a robot is the screwdriver. There are two common types, Phillips and flat.

but a ball-peen hammer also works, as long as it's not too large. You don't want a sledgehammer when a gentle knock is all you need.

PLIERS



Pliers hold parts while you work with them. A pair each of standard and needle-nose pliers is enough for 94.5 percent of all jobs. Don't use either as a wrench for tightening nuts; they'll slip and round off the corner of the nut, making it harder to remove later on. Instead, use a nut driver, detailed below.

For heavy-duty applications, purchase a larger pair of needle-nose pliers. A set of "lineman's" pliers can be used for the big jobs, and they provide a sharp cutter for clipping nonhardened wire.

HACKSAW



The hacksaw is the mainstay of robot building. Look for a model that allows quick blade changes. Common blade sizes are 10" and 12" in length. The smaller blade length is recommended when working with metal. Purchase an assortment of carbide-tipped blades in 18 and 24 teeth per inch (referred to as *tpi*, or *pitch*).

By convention, hacksaw blades are inserted so that the *teeth face forward*. That means the saw cuts when you push the blade *away from you*. However, there is no strict rule about this. Experiment to see what works best for you.

ELECTRIC DRILL



You use a drill to make holes; an electric drill makes the whole hole process easier. Pick an electric motorized drill with a 1/4" or 3/8" chuck—the chuck is the part where the drill bit is inserted. Chuck size determines maximum diameter for the shank of the bit. The vast majority of work on small robots will require bits of 1/4" or smaller.



Spring for an adjustable-speed, reversible drill. The slight added price is well worth it. Adjusting the speed is important when working with different kinds of materials, as some (like metal) need a slower tool.

DRILL BITS



A drill is what turns a bit; the bit is what actually makes the holes.

So, with that amazingly helpful clarification out of the way, let's get on with the advice: purchase a drill bit set in so-called jobber length. In the United States and other locations where they still use inches, drill bits are measured in fractional sizes. The typical fractional drill bit set contains 29 bits (give or take), in sizes from 1/16" to 1/2", in 64ths-of-an-inch steps. For most robotic creations, you'll use only a third of these, but it's nice to have the full set in case you ever need the others.

FYI

Drill bits under 1/4" are also identified by their numerical size. For example, a #36 bit is the same as 7/64". See Appendix C, "Mechanical Reference," for a handy chart comparing number drill bits with their fractional counterparts.

- The least expensive drill bits for robot building are made of *high-carbon steel*.
- Better drill bits are made with *high-speed steel*, and these keep their sharpness for longer. These are fine for most amateur robotics workshops.
- *Tungsten carbide* bits stay sharp the longest even when cutting metal. They're more expensive than the others.
- *Cobalt* bits are the Terminator of the bunch. They drill into most anything, including hardened steel.

Drill bits have different kinds of coatings, which extend their life. However, when coated, the bit usually can't be resharpened, as the resharpened edge will no longer have the coating. Black *oxide coating* is the least expensive of them all, and is useful for wood, soft plastic, and thin aluminum. Various *titanium* coatings greatly extend the life of your bits, allowing you to use them on thicker aluminum and steel.



Save money! Get a standard fractional drill set in standard high-speed steel, then augment that set with specific sizes of more-expensive longer-lasting bits. The most commonly used bit in my shop is 1/8", so I get extras of those with a titanium coating.

SCREWDRIVER BITS



When you find you're doing a lot of screw assembly, invest a few dollars in a set of Phillips and flathead bits for use in drills and motorized driver tools. These fit into the drill chuck like any other bit but are used to tighten and untighten screws. To be useful, your drill needs a variable-speed motor; you'll surely strip out the screws if you try to use the bits at high speed.

HOBBY KNIFE



Hobby knives include the X-Acto brand, sporting interchangeable blades. They're ideal for cutting cardboard, foamboard, and thin plastics. A word of caution: The blades in these knives are *extremely* sharp. Use with care.

NUT DRIVER



Nut drivers look like screwdrivers, but they're made to tighten hex-head (six-sided) nuts. They come in metric and imperial (inch) sizes. On the imperial front, common driver sizes are:

Driver	For Hex Nut
1/4"	#4
5/16"	#6
11/32"	#8
3/8"	#10
7/16"	1/4"
1/2"	5/16"

Optional Tools

There are a couple more tools that are nice to have but aren't absolutely critical to build a robot. Add them as your budget allows.

- *Miter boxes* help you make straight and angled cuts into tubing, bars, and other “length-wise” material. Attach the miter box to your worktable.
- A vise holds parts while you drill, cut, or otherwise torment them. What size vise to get? One that's large enough for a 2" block of wood, metal, or plastic is about right.
- A *drill press* sits on your workbench or table. It helps you make smoother, more accurate holes. Lower the bit and drill the hole by turning a crank.

Hardware Supplies

A robot is about 70 percent hardware and 30 percent electronic and electromechanical components. Most of your trips for robot parts will be to the local hardware store. The following sections describe some common items you'll want to have around your shop.

SCREWS AND NUTS



Screws and nuts are common fasteners used to keep things together. Screws (called bolts when they're bigger) and nuts come in various sizes, either in metric or in imperial (inch) units. For this book I stick with imperial sizes, because that's still what we use in the United States, and it's what I'm used to.

Anyway, here are the very basics of what you should have to build your robots. You can read more about fasteners in Chapter 13, “Assembly Techniques.”

- Use 4-40 size screws and nuts for the typical tabletop robot. The “4” means it's a #4 fastener; the “40” means there are 40 threads per inch. Screws come in various lengths, with 3/8", 1/2", and 3/4" being the most useful for small robotics. I use 4-40 × 1/2" screws the most.
- For bigger parts and bigger robots, use 6-32, 8-32, and 10-24 screws and nuts. The most commonly used screw lengths are 1/2", 3/4", 1", and 1-1/2". These and other sizes are

available at any hardware or home improvement store, and you can pick up what you need when you need it.

- For very heavy duty work, you want 1/4"-20 or 5/16" hardware. (1/4"-20 means the screw is 1/4" in diameter and has 20 threads per inch; the standard for this size). Get these only when plans call for them.

Locking nuts are a special kind of nut you'll use a lot in robotics. They have a piece of nylon plastic built into them that provides a locking bite when they are threaded onto a screw.

WASHERS



Washers are used with screws and nuts and help to spread out the pressure of the fastener. They're also used when you want to make sure the nut and screw don't come apart. The most common is the *flat washer*, which is typically used to keep the head of the screw (or the nut) from pulling through the material. *Tooth and split lock washers* apply pressure against a nut to keep it from coming loose.

ANGLE BRACKETS



Also ideal for robot construction is an assortment of small steel or plastic brackets. These are used to join two parts together. They come in different sizes, the smaller sizes being perfect for use on desktop robots—smaller brackets weigh less. I often use 1-1/2" × 3/8" flat corner steel brackets when joining two pieces cut at 45° angles to make a frame.

Setting Up Shop

You can build a robot anywhere. But it's more pleasurable when your workspace is well lighted, tidy, and comfortable.

A garage is an ideal location because it affords you the freedom to cut and drill without worrying about getting the pieces in the carpet. Electronic assembly can be done indoors or out, but I've found that when working in a carpeted room, it's best to spread another carpet or some protective cover over the floor. When the throw rug gets filled up with solder bits and little pieces of debris, I take it outside, beat it over a trash can with a broom handle, and it's as good as new.

No matter where you set up your robotics lab, be sure all your tools are within easy reach. Keep special tools and supplies in an inexpensive fishing tackle box. The tackle box provides lots of small compartments for screws and other parts. For the best results, your work space should be an area where the robot-in-progress will not be disturbed if you have to leave it for a time. The worktable should also be off limits or inaccessible to young children.

Good lighting is a must. Both mechanical and electronic assembly require detail work, and you will need good lighting to see everything properly. Supplement overhead lights with a 60-watt (minimum) desk lamp. You'll be crouched over the worktable for extended periods of time, so a comfortable chair or stool is a must.

Mechanical Construction Techniques

More than likely, at one time or another, you've used tools to build or fix something. But even with that experience, you may not know the ins and outs of using tools to work with robot-building materials. If you're a die-hard workshop expert, that's fine; you can skip this chapter and move on to the next. But keep reading if you lack basic construction and tool use know-how, or if you need a refresher course.

This chapter assumes you've read the section on tools in Chapter 5, "Building Robot Bodies—the Basics." It contains a review of all the core tools you need to construct automata in your garage. If you haven't yet read that chapter, do so now, then come back. A few additional specialty tools are covered here.

This chapter also assumes you're working with wood, plastic, or metal. If you're using lightweight rapid prototyping materials like foamboard, you'll also want to check out Chapter 14, "Rapid Prototyping Methods."



This chapter won't teach you *everything* about how to build things. This is a book about robots, after all, not shop class. If you hunger for more, try the Time-Life and Reader's Digest do-it-yourself books. They are particularly well done, with clear and concise illustrations.

First Things First: Eye and Ear Protection

Always wear eye protection when using any tool, power or hand. This helps prevent harmful debris from flying into your eyes, possibly causing injury.

Don't forget your ears when using power tools. High-speed tools, especially power saws and air tools, create a high volume of sound. Wear sound-suppressing ear protection—wraparound earmuffs designed for shop use or even basic earplugs that you can get at the local drugstore.

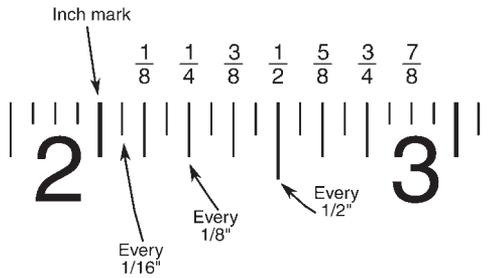


Figure 6-1 Learning to read the graduations of a tape measure is the first step to mastering any construction project. Most tape measures marked off in inches have graduations down to $1/16$ or $1/32$ of an inch.

FYI

Read more about how to safely enjoy the robot-building hobby in Chapter 4, “Safety First (and Always).”

Plan, Sketch, Measure, Mark

Start with a plan, and take a moment to visualize how your robot will look and the parts that will go into it. Then,

- *Work up a quick budget.* If you’re just starting out, you won’t have many parts and supplies. Research what you need (see Appendix A, “RBB Online Support,” and Appendix B, “Internet Parts Sources”), and make a list of their names, part numbers, and prices.
- *Sketch out your plan* on paper or by using a computer vector graphics program. I prefer Inkscape (it’s free).

If you need to cut and drill, mark directly on the materials. For wood, cardboard, and foam-board use a #2 soft lead pencil. For everything else use a black Sharpie or similar marker; a fine tip works best.

Use a measuring tape to ensure accuracy. If you’re using a tape measure marked in inches (see Figure 6-1), the graduations will be every $1/2$ ”, $1/4$ ”, $1/8$ ”, and $1/16$ ”. It’s easy to miscount the subdivisions and get the measurement wrong, so double-check every measurement.

Drilling Holes in Things

Except for rapid prototypes (like those in Chapter 14, “Rapid Prototyping Methods,”), most of your robots will need some holes drilled into them so you can mount things like battery holders, motors, and electronics.

Regardless of the material (wood, plastic, metal), the basic concepts of drilling are the same: you put a bit into the drill (hand or power), mark where you want the hole to be, and drill there. Good drilling involves following some simple procedures, covered here.

PICKING THE RIGHT DRILL BIT

That 29-piece drill bit set you’ve been wanting for your birthday offers a fine selection of the most common drill sizes you’ll need. But in actual practice you’ll probably end up using

only a small handful of them regularly and the rest only very occasionally. That's how it is with me.

I use these five drill bits for the vast majority of the holes I drill for my desktop robots. I keep them in a large block of wood for quick access.

Bit Size	For Drilling:
5/64"	Starter or <i>pilot</i> holes
1/8"	Holes for 4-40 size screws
9/64"	Holes for 6-32 size screws
3/16"	Holes for 8-32 size screws
1/4"	Odds and ends (e.g., holes for feeding wires through)

FYI

Refer to Chapter 5, "Building Robot Bodies—the Basics," for a summary of drill bit types, including special coatings that are used to make the bits last longer.

- When drilling into metal or hard plastic (acrylic, polycarbonate), first use the 5/64" bit to make a pilot hole. Then mount the bit for the hole size you want.
- Unless you're making really large holes, when drilling into wood and soft plastic (PVC or ABS) you can go directly to the bit for the hole size you need.
- When drilling holes larger than 1/4"—and especially when working with metal—start with a smaller bit and work your way up. For example, if drilling a 3/8" hole, begin with a 1/8" pilot, then switch to 1/4", and finish with the 3/8" bit.

SELECTING THE PROPER SPEED

Different materials require different speeds for drilling. High-speed drilling is fine for wood but leads to dull bits when used with metal and cracks when used with plastic.

Material	Drilling Speed
Softwood (pine)	High
Hardwood (birch)	Medium to high
Soft plastic	Medium
Hard plastic	Slow to medium
Metal (aluminum)	Slow
Metal (steel)	Very slow

Most drill motors lack a means to directly measure the speed of the tool, so you just have to guess at what's high, medium, or slow. Go by the sound of the tool. If you're using a portable variable-speed drill, pull the trigger all the way in for full speed. Then let it out and estimate half (medium) and quarter (slow) speeds by listening to the sound of the motor.

If you're using a drill press where you adjust the speed by changing the position of a rubber belt, place the belt into the *High*, *Medium*, and *Low* positions accordingly. As changing the belt is a hassle, you may want to just set it to *Low*, and drill everything that way. It's better to drill wood using a slow bit than metal using a fast bit.

PROPER USE OF THE DRILL CHUCK

The *chuck* is the mechanical "jaws" that hold the bit in the drill motor. While some electric drills use an automatic chuck system, most chucks are operated using a chuck key: insert the key into one of the holes in the side of the chuck, and loosen or tighten the jaws. Keep the following in mind when using the drill chuck:

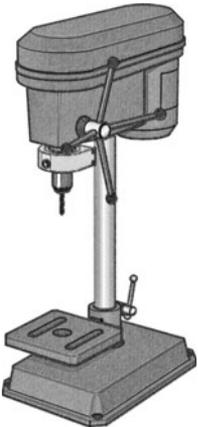
- Insert only the smooth shank** of the bit into the chuck and none of the flutes. Otherwise, the bit might be damaged.
- Be sure the bit is centered** in the jaws of the chuck before tightening. If the bit is even slightly off-center, the hole will come out too large and distorted.
- Don't over tighten the chuck.** Too tight makes it harder to loosen when you want to remove the bit.
- Tighten the chuck using at least two** key holes. This evens out the torque applied to the chuck and makes it easier to loosen the chuck when you're done.

CONTROLLING THE DEPTH OF DRILLING

Most of the holes you drill will be completely through the material, but sometimes you need to drill only partway in, then stop. For example, you might want a limited-depth hole in the jaw of an android robot for setting a pin that's used as a hinge.

There are various methods to control the depth of the hole in thicker materials. My favorite method is to wrap masking tape around the shaft of the drill bit (see Figure 6-2). Place the edge of the tape at the depth you want for the hole.

Another type of depth-limited hole is the counterbore, where the drill hole is in different diameters. Counterbore holes, like that in Figure 6-3, can be created by first drilling with a small bit, then using a larger bit only partway through the material. The technique is often used with tapered screws, when you want the bottom of the hole to be smaller than the top.



ALIGNING HOLES

Drilling with any handheld tool will naturally produce a certain amount of error. Even the most skilled worker cannot always drill a hole that is at exact right angles to the surface of the material.

When precisely aligned holes are a must, use a drill press or drill-alignment jig. The latter can be found at better hardware and tool stores that stock specialized carpentry accessories, but, frankly, I've never found them all that convenient. The

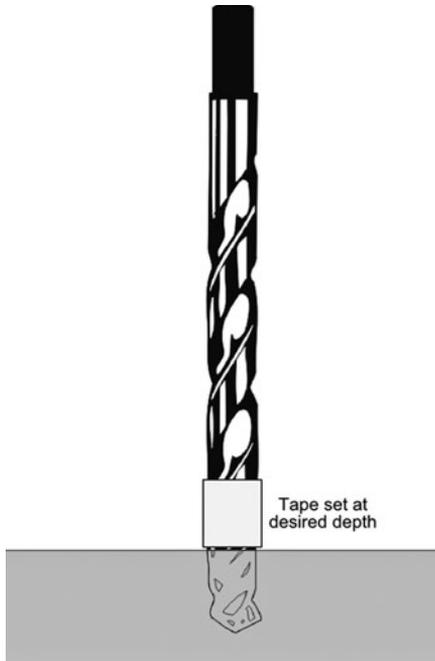


Figure 6-2 A quick and easy way to prepare holes of a specific depth is to wrap a piece of masking tape around the drill bit. Position the bottom of the tape at the depth of the hole you want to make.

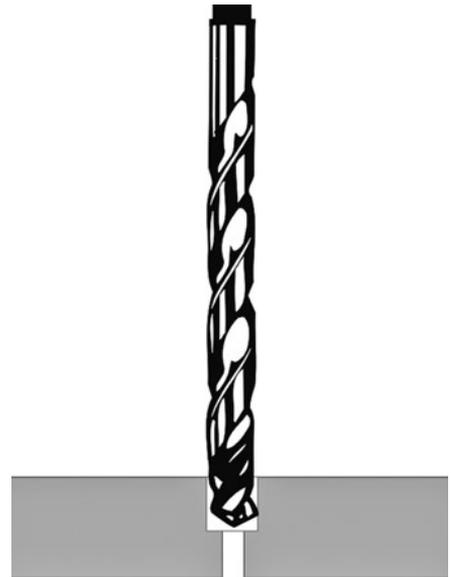


Figure 6-3 A counterbore hole is one that has two sizes. One use is with lag and wood screws, where the top of the screw is not threaded. Make the bottom of the hole smaller to give the threads something to bite.

drill press is the much better tool for ensuring alignment. A basic no-frills bench model is under \$100. I'm still using the one I bought over 30 years ago!

USING CLAMPS AND VISES

When drilling with power tools, exercise care to hold the part in a clamp or vise. Avoid holding the part just with your hands, especially if it's small. Why? As you drill, the bit may “dog” into the material, causing it to get caught. If you're holding the part, it could get violently yanked out of your hands, causing injury.

Hold-down clamps and vises come in various shapes, sizes, and styles. There is no one type that works for every occasion. Spring-loaded clamps (they look like giant tweezers) are useful for very small parts, while C-clamps are handy for larger chunks of material. A vise is required when drilling small parts of any type with a drill press. For very small and lightweight pieces you can use a pair of large lineman's pliers or a pair of Vise-Grips.

TIPS FOR DRILLING

Here are some handy tips for drilling wood, metal, and plastic. All speeds are in RPM.

	Wood	Plastic	Metal
General	<p>Wood is readily drilled using a motorized drill, either handheld or drill press. Speed depends on the size of the bit and the density of the wood. Following are general speed recommendations.</p> <ul style="list-style-type: none"> • Larger than 1/8": 2000 • 1/8" to 1/16": 4500 • Smaller than 1/16": 6000 	<p>For soft plastics (like PVC), speed settings same or slightly slower, as for wood. For harder plastics (acrylic, polycarbonate), reduce drill speed by 50 percent.</p>	<p>Metals should be drilled using a motorized drill. Small parts are more readily drilled using a drill press. Following are general speed recommendations for aluminum and other soft metals. For harder metals, reduce speed by 50 to 70 percent.</p> <ul style="list-style-type: none"> • Larger than 1/8": 500 • 1/8" to 1/16": 1000 • Smaller than 1/16": 1500
Bits	<p>Wood bits should be ground to 118° (pretty much the standard). For cutting all but very dense hardwoods (e.g., oak), regular carbon twist drills are adequate.</p>	<p>Use wood bits for soft plastics. For hard plastics, use a pointed bit designed for acrylic and polycarbonate.</p>	<p>For aluminum and other soft metals, bits should be ground to 118°, the standard. For harder metals like steel (and even hard plastic), use 135° bits. For longer life, consider titanium- and cobalt-coated bits.</p>
Cooling	<p>Air cooling is sufficient. If the wood is very hard and thick, pause every 30 seconds to allow the bit to cool down.</p>	<p>Air cooling is sufficient, but if plastic remelts into the hole, slow down the bit, drill smaller pilot holes first, or splash on some drops of water.</p>	<p>Use cutting oil for metal (thin aluminum can usually be drilled without oil). The idea is to avoid excessive heat, which dulls the bit.</p>

Cutting Things to Size

Wood, metal, and plastic can be cut using hand-operated or power tools. For all but the lightest materials, however, you will find that power tools make short work of the job.

In the realm of hand tools, practical choices are:



For wood, a *backsaw*; for metal and plastic, a standard *hacksaw*. The hacksaw uses a replaceable blade, which is required when working with harder materials.

A *coping saw* allows you cut tight-radius corners in wood, plastic, and softer metals. A coping saw is similar to the hacksaw, except the blades are smaller. Replace the blade when it's dull.



A *razor saw* is used with thin woods and plastics. Its shape is like that of a backsaw, but much smaller. You can find razor saws at the hobby store.

And for power tools, practical choices are:



Jigsaw for wood. You need one with adjustable speed unless you work only with soft wood (not really recommended for robotics anyway).



Circular saws and table saws are useful for cutting long, straight cuts in wood and metal. Be sure to use the proper blade, or else damage to the material and/or blade could result.



A *circular miter saw* is useful when cutting aluminum channel and bar stock. The alternative is a hacksaw, a miter block, and sweat. When cutting metal be sure to use clamps to hold the material in the miter!



There are plenty of other saw types not mentioned here, like the *scroll saw*, which can cut very intricate shapes in wood and metal and some plastics. If you have a preference for one type of tool over another, by all means use it.

REMEMBER SAFETY!

Of all the tools in the shop, saws are the most dangerous—yes, even handsaws. I need not remind you to observe all safety precautions. *NEVER* remove a safety device or guard. Work in a well-lit, unobstructed area. Some additional stern warnings for you:

- Use eye and ear protection when the saw is in operation.
- When using a table saw, feed the last 6" or so of the material through the blade with a flat stick. The woodworking guides show how to do this.
- Don't wear loose clothing. This goes for long-sleeve shirts, ties, baggy pants, everything. The loose material could get caught in the rotating parts of the saw.
- Keep the blade sharp, and always use the right blade for the job. Don't try to cut ferrous (contains iron) metal with a blade meant for thin wood paneling. For one thing, the teeth could break off and fly right at you.

TIPS FOR SAWING

Here are some handy tips for sawing wood, metal, and plastic.

	Wood	Plastic	Metal
General	For hacksaw, bandsaw, or scroll saw, use medium pitch (teeth per inch) blade. For motorized tools, set speed at highest.	Set speed to no more than 50 percent.	For motorized tools, reduce speed to 25 percent.

	Wood	Plastic	Metal
Blade	Match the blade with the thickness and grain of the material. Circular saw blades are often classified by their application (such as “crosscut”). Use this as a guide.	For circular saw: If possible, use a “nonmelt” blade made for plastics; if you can’t find one or it’s too expensive, a high-quality plywood blade will do. The wider the kerf in relation to the thickness of the blade, the better. This avoids remelting. For hacksaw or scroll saw: Use an intermediate pitch blade (18–24 teeth per inch). A wide-kerf is best.	As a general rule, 3–5 teeth should engage the metal. Select the blade according to the thickness of the material you’re cutting. Use an abrasive cutoff tool for heavy-gauge ferrous metals.
Cooling	Air cooling is sufficient, though beeswax can be used if the wood is very dense.	Air cooling is usually sufficient. If remelting occurs, direct 50–75 psi air from a compressor over cutting area.	Use cutting oil or wax for heavy-gauge metals.



In the preceding table, the term *kerf* means the width of the cut made by the blade. Many blades use cutting teeth that protrude to either side (often referred to as *set*). This makes for a wider cut (kerf), but helps keep the blade from binding into the material. When cutting plastic, a wide kerf helps prevent melting.

LIMITING CUTTING DEPTH

When using power saws, you can readily limit the depth of the cut by changing the height of the blade within the tool. This is useful when you want to add channels or grooves in the material but not cut all the way through. The following apply to sawing wood or plastic, and when the thickness of the material is at least 1/4”—any thinner and it doesn’t much matter how deep the cutting is.



Set the blade to route by just grazing into the material. Use this technique to score very hard material, like polycarbonate plastic; once scored, you can snap the pieces apart using a wood dowel placed under the score mark.



Cut grooves into thicker materials by setting the blade depth to about 1/3 to 1/2 thickness. The width of the groove is the kerf (or set) of the blade—the width of the blade itself, plus the right and left offset of the cutting teeth.



With a full cut, the blade penetrates completely through the material.

OTHER WAYS TO CUT MATERIAL

While a saw is the most common means of cutting materials, there are other methods as well. Select the method based on the material you are cutting and the demands of the job.



Figure 6-4 A tubing cutter is easier to use than a saw when sawing a tube in half. It's for so-called thin-wall tube, not thick pipe. Use a small tool (also shown) for tubing under 1/2"; it's available at many hobby stores.

- Very thin (less than 1/8") hard plastics can be cut by scoring with a sharp utility knife. Place the score over a 1/4" dowel, and apply even pressure on both sides to snap apart the material.
- Thin (to about 20 gauge) metal can be cut with hand or air snips. Pneumatic air snips make the work go much faster. Manual snips are available for straight cuts, left-turning cuts, and right-turning cuts.
- For higher gauges of metal, and for long, straight cuts in thinner gauges, cut using a bench shear (also called a metal brake).
- Thin-wall tubing (aluminum, brass, copper) can be cut using a tubing cutter, like those in Figure 6-4. Tubing cutters are easier to use, and they do a better job than using a saw.
- Foam-based materials that are not laminated (e.g., Styrofoam) can be cut using a hot wire. Hot wire kits are available at most craft stores.

Using Portable Power Tools

Your work in the robot shop will go much faster if you use power tools. This applies even if you're constructing your bot out of wood. The use of power tools is fairly straightforward, and the subject needs little additional instruction beyond what you'll get in the instruction manual and maybe a good shop tools book. But here are some of the most important issues to be aware of.

TOOL SAFETY

Sparking in some tool motors is fairly common, but excessive sparking may indicate a damaged motor.

When working outside, use an AC outlet equipped with a ground-fault interrupter circuit. **The grounding lug** on AC-operated tools should *never* be defeated.

TOOL MAINTENANCE

Keep the cooling fan of the tool, if it has one, clean and free of obstruction. Do not use the tool if metal bits have become caught in the cooling vents.

Some power tools must be periodically lubricated, as indicated by the manufacturer. This is especially true of reciprocating saws. Gear-driven tools (e.g., heavy-duty power drills, worm-drive circular saws) should be lubricated with the appropriate grease or high-viscosity oil.

GENERAL STUFF

When using an extension cord, be sure it is rated for the current being passed through it. As a general rule, medium-duty tools (8 to 10 amps) can be used with 14-gauge extension cords, no more than 50 feet. (Gauge refers to the thickness of the wires; the smaller the number, the *larger* the wires.) Check the instruction manual for the tool for guidelines.

Keep the exterior of the tool clean by wiping it off with a dry cloth. Grime can be removed by cleaning with a slightly damp cloth. Never apply water or any other liquid directly to the tool.

Getting Work Done Fast with Air Tools

Electric power tools are by far the most common variety found in any home shop. Yet another kind of power tool is the air (pneumatic) type, which is driven by a blast of high-pressure air. Common air power tools include the drill (standard and reversible models available; see Figure 6-5), air hammers, metal snips (*very handy!*—I use these all the time), and power wrench. Pneumatic tools offer several advantages:

- Air tools tend to be lighter because they lack a bulky motor.
- The tools are generally less expensive than their electrified cousins, because their motor is a fairly simple air pump. Of course, you need a separate compressor to supply the air.
- They're safer to use outdoors, because there's no electricity to the tool. You can keep the compressor indoors and connect the compressor and tool via a hose.
- Properly cared for, air tools can last a lifetime (your lifetime, that is).

Know that water *destroys* air tools. Use pneumatic tool oil to dispel any moisture. Oiling your air tools every day that you use them will greatly prolong their life. To oil an air tool just squirt a dab or two into its air intake the first time you use it that day.

Pneumatic tools require both air pressure and air volume to work. Be sure your compressor can deliver both the pressure and the volume needed by your tools. The specifications that come with each tool will tell you what it needs.

Keep the air hose as short as possible. The longer the hose, the less the pressure to the tool. A length of under 25 feet is ideal. I like using noncoiled hoses, as the “straight” ones tend to provide higher pressures. However, the coiled hoses are more convenient.



Figure 6-5 A pneumatic drill is lighter and smaller than an electric drill, but it needs a constant source of compressed air.

Working with Wood

If billionaire Howard Hughes could build the world's largest powered airplane out of spruce wood, how hard could it be to construct a small robot out of the stuff? Wood may not be high-tech, but it turns out it's an ideal building material for hobby robots. Wood is available just about everywhere, it's relatively cheap, and it's easy to work with.

In this chapter, you'll look at using wood in robots and how you can apply simple wood-working skills to construct wooden robot bodies and platforms.

FYI

Check out Chapter 8, "Build a Motorized Wooden Platform," for a hands-on project making a robot base out of wood. This platform can serve as the foundation for a number of robot designs you may want to explore.

Hardwood Versus Softwood

While there are thousands of types of trees (and, therefore, wood) in the world, only a relatively small handful share the traits that make them ideal for building robots. Wood can be broadly categorized as hardwood or softwood. The difference is not the hardness of the wood, but the kind of tree the wood is from.

Hardwood is produced from trees that bear and lose leaves (deciduous), and *softwood* trees bear needles (coniferous) or do not undergo seasonal change (nondeciduous). In general, deciduous trees produce harder and denser woods, but this is not always the case. A common hardwood that's very light and soft is balsa, often used in craft projects.

Planks or Ply

Unless you're Abraham Lincoln, building a robot from lumber hewn from the forests of Kentucky, you'll most likely purchase milled wood in either plank or laminate (ply) form. (Milling means it's cut and formed to size.)

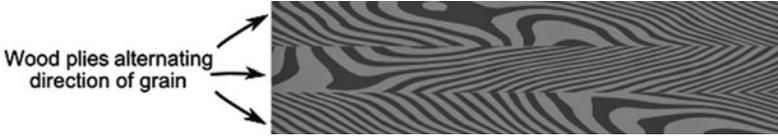


Figure 7-1 Plywood is made using several thin layers of wood, with the grain of each layer alternating to provide added strength.

Planks are lengths of wood milled from the raw lumber stock. They are available in standard widths and thicknesses and come in either precut lengths (usually 4, 6, and 8 feet) or are sold by the linear foot.

Plywood is made by sandwiching one or more types of wood together. The grain—the direction of growth of the original tree—is alternated at each ply for added strength.

Both plank and ply are made from softwoods and hardwoods. Depending on where you live, your local home improvement store is likely to have only softwood ply and just a few types of hardwood plank (mostly oak). For a wider variety you need to shop at a hardwood specialty store or mail order.

USING PLYWOOD

The best overall wood for robotics use, especially for foundation platforms, is *plywood*. Plywood gets its strength by sandwiching two or more pieces of wood together, where the grain (growth pattern) of the wood alternates direction—see Figure 7-1. Plywood is usually stronger than the equivalent thickness of a single slice of wood. Each thin slice strengthens and reinforces the other.

Plywoods are commonly used in residential and commercial construction, but they’re made of softwood and aren’t well suited for use in small robots. A better choice is specialty hardwood plywoods, available at craft and hobby stores. They’re more desirable because they are denser and less likely to chip.

This type of hardwood plywood comes in two “grades”: aircraft and craft. A typical *aircraft plywood* uses birch and consists of from 3 to 24 plies—the more plies, the thicker the wood. It’s rated for use in model airplanes, where structural strength is critical. A less expensive variation is *craft plywood*, which is not intended for use in model planes. It’s made of less expensive woods that aren’t quite as strong. For most robotics use, the craft plywood is perfectly fine. Sheet size for both vary, depending on manufacturer and source. Hobby stores commonly carry plywoods in 12” × 12” squares.

Plywood comes in various thicknesses starting at about 1/4” and going up to over 1”. Thinner sheets are good for making a small robotics platform, but only if the plywood is made from hardwood. Typical thicknesses are:

Thickness		
Metric	Inch	Plies
2.0 mm	0.7874	3 or 5
2.5 mm	0.9843	5
3.0 mm	1.1811	7
4.0 mm	1.5748	8

Metric	Inch	Plies
6.0 mm	2.3622	12
8.0 mm	3.1496	16
12.0 mm	4.7244	24

3.0 mm = approx. 1/8"

6.0 mm = approx. 1/4"

12.0 mm = approx. 1/2"



Don't confuse *hardwood* plywood with *hardboard*. Hardboard is a manufactured product, usually made by munching up sawdust into tiny pieces, then fusing them together under high pressure. See the section "Medium Density Fiberboard" for a kind of hardboard usable in robotics.

USING PLANKING

An alternative to plywood is *planking*. Hardwood planking—stuff like oak or birch—tends to be very heavy and much harder to work with. Opt for Douglas fir or pine. These softer woods are okay for smaller robots, and they are easier to cut and drill.

Plank lumber is available in widths of no more than 12" or 15" wide, so you must take this into consideration when designing your robot platform. Be especially wary of warpage and moisture content when using plank lumber. Borrow a carpenter's square, and check the squareness and levelness of the lumber in every possible direction. Reject any piece that isn't flat and square.

Wood with excessive moisture may bow and bend as it dries, causing cracks and warpage. These can be devastating in a robot you've just completed and perfected. Buy only seasoned lumber stored inside the lumberyard, not outside. Watch for green specks or grains—these indicate trapped moisture.

COMMON PLANKING DIMENSIONS

Plank lumber is milled in common dimensions. Unless otherwise noted, the finished milled dimension is less than the stated size, because of saw kerf. When you buy two-by-four lumber, for example, you're really only getting 1-1/2" × 3-1/2".

Dimension	Thickness	Width
1 × 4	3/4"	3-1/2"
2 × 4	1-1/2"	3-1/2"
1 × 6	3/4"	5-1/2"
2 × 6	1-1/2"	5-1/2"
1 × 8	3/4"	7-1/4"
1 × 10	3/4"	9-1/4"
1 × 12	3/4"	11-1/4"

MEDIUM-DENSITY FIBERBOARD

Medium-density fiberboard (MDF) is a manufactured product, usually available in 4- by 4- or 4- by 8-foot panel sheets, and is made by compressing wood fibers bound with a resin.

There are quite dense MDF panels that are extremely hard and durable, and these are fine for robot platforms. Others, like particleboard, consist of larger fibers and wood chunks, and aren't as strong. Don't use particleboard for making a robot.

Look for cut pieces of MDF at your home improvement store, so that you don't have to purchase an entire sheet. Or find a low-cost paper clipboard at the office supply store. Many use a highly pressed wood in 1/8" thickness. Another option is to use those small hardwood flooring samples they offer at home improvement stores. Great (and cheap!) for small robots!



Any downsides to using MDF panels? A few. A 12" × 12" sheet of 1/4" MDF can easily weigh a couple of pounds. And the corners and edges can chip off more readily than with others kinds of wood. You can keep the corners from chipping if you round them off by sanding or filing.

The Woodcutter's Art

You don't need any special tools or techniques to cut wood for a robot platform. The basic shop cutting tools will suffice: a handsaw, a backsaw, a coping saw—you name it.

- When cutting plywood and wide planking, use a wood handsaw or table saw. You'll get the straightest cuts with a table saw.
- When cutting narrow planking or smaller pieces of wood, use a backsaw (my favorite), a handsaw, or a jigsaw.

Make sure the blade is made for cutting wood. Wood blades have coarser teeth than blades for metal or plastic. If using a power saw, the combination blade that came with your tool isn't the perfect choice for plywood, but it'll do. Consider replacing it with a plywood-paneling blade. These have more teeth per inch and produce a smoother cut.

Handsaws come in two versions: *crosscut* and *ripsaw*; crosscut is the proper tool for plywood and most of the other woods you'll be working with.

CUTTING A BASE

The easiest shape of all is square, and that's what you'll begin with, even if you plan on a more elaborate shape. After all, what's a circle other than a square with lots and lots of corners!

Start with a wooden square or rectangle that's about the size of the finished base. If you have them, you can use power tools to make short work out of cutting the wood to the basic square/rectangle shape.

It's Not Hip to Be Square

Problem is, square is not the ideal shape for a robot base, because the corners can snag on things when the bot is driving around your living room. You can readily turn the square into octagons, hexagons, and pentagons simply by lopping off the corners. Unless you're an expert at the table saw, do these cuts with hand tools; it'll provide extra accuracy. Figures 7-2

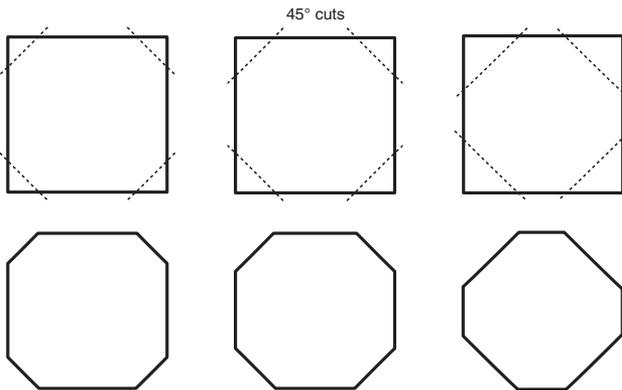


Figure 7-2 Lop off the corners of square wood pieces to streamline the shape of your robot. With all four corners cut, you end up with an octagon. The shape of the octagon depends on how much of each corner you remove.

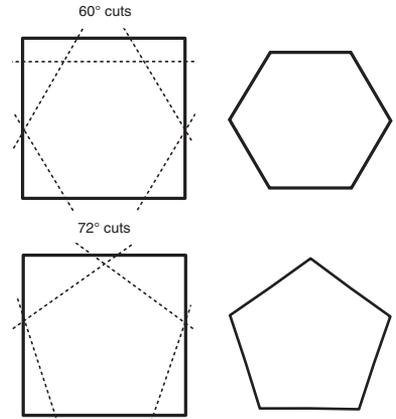


Figure 7-3 By making alternative cuts at the corners you can produce hexagon and pentagon bases. You'll probably want to use a simple protractor (available at any school or office supply store) to measure the angles.

and 7-3 show some variations on a theme. These more elaborate shapes don't take that much longer to produce—just a few minutes per cut, and you'll make a better robot.

- To make an octagon (eight-sided) base, cut the corners off at 45° .
- To make a hexagon (six-sided) base, cut the corners off at 60° .
- To make a pentagon (five-sided) base, cut the corners off at 72° .



If you have a heavy-duty motorized sander, you can lop off the corners by sanding rather than cutting. This is usually a lot quicker. Start with a coarse sandpaper (see the section later on in this chapter about sanding). As a final step, use a fine sandpaper to make the edges smooth.

Lopping Off Even More Corners

You can approximate near circles by cutting off more corners. Eight-sided octagons with their corners lopped off (chamfered) make 16-sided “circlettes,” as shown in Figure 7-4. And as shown in Figure 7-5, chamfered pentagons produce 10-sided shapes; chamfered hexagons make 12 sides.

To make these cuts, mark directly on the wood (you don't have to be precise—to the nearest $1/4$ " is usually fine), or use a piece of graph paper lightly glued to the wood. You can use paper paste—the kind you ate in grade school—or a nonpermanent glue stick.



“But I wanna make a circular base!” That's fine, but you'll need some special tools if you don't want it to look like a very bad shop class reject. Circle cutting jigs are available for power routers and jigsaws. The one for jigsaws is fairly easy to use and is the least expensive. It works just like a beam compass.

Start by drilling a hole in the approximate center of the wood, then adjust the circle cutter to one-half the diameter of the circle you want. For instance, to make a 6" circle, you adjust the

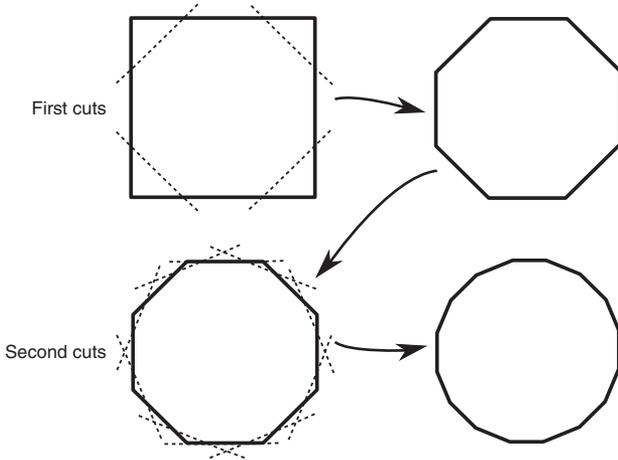


Figure 7-4 For near-circular bases you can cut off the four corners of a square base, then trim off the eight corners you just made. It's a little extra effort, but worth it.

cutter to the 3" mark. You need to also make a starter hole for the jigsaw blade. Position the hole anywhere along the circumference of the circle.

Seat the anchor point of the cutting jig inside the center hole. Cut out the circle starting from the center hole you previously made.

Making Cutout Wells for Wheels

The robot bases we've cut so far don't have special cutouts ("wells") for wheels. Wheel wells are nice to have, because they allow you to place the wheels flush (or nearly so) with the contours of the body or without having to raise the level of the base to clear the wheels. Figure 7-6 demonstrates the basic idea.

Cutting wheel wells is easier when the wheels are placed at one end of the robot. When in the middle of the robot you need to make multiple cuts to literally "carve out" the well. Well cutting works best when you use either a power jigsaw or a coping saw. The coping saw has a small blade for tight corners.

Figure 7-7 shows a simple three-cut approach. Start with a jigsaw or backsaw and make two cuts perpendicular to the side of the base.

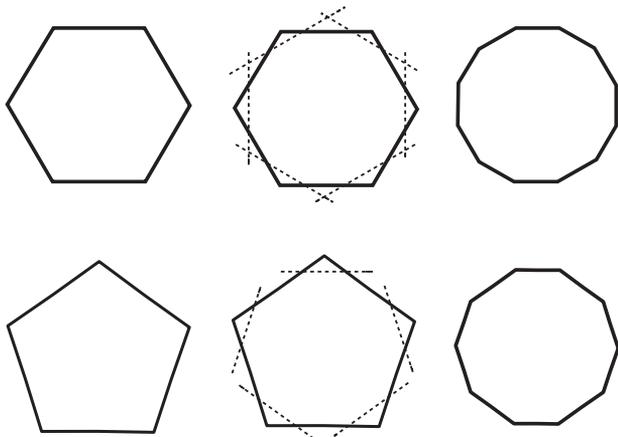


Figure 7-5 Additional corner cuts in hexagon and pentagon shapes make for even sleeker designs.

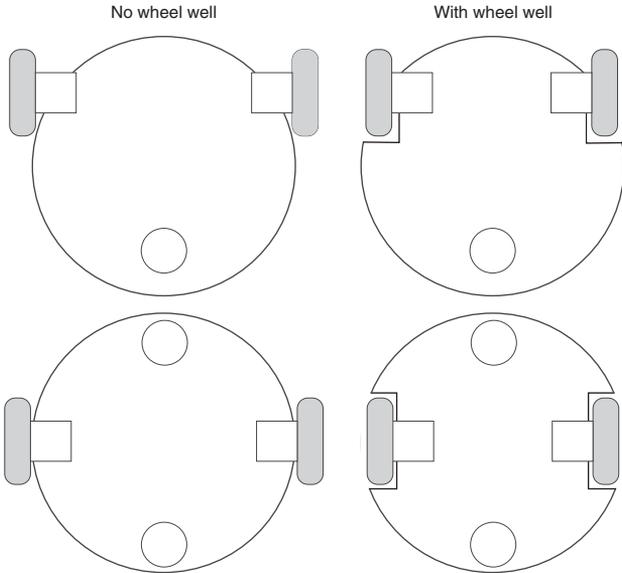


Figure 7-6 Wheel well cutouts allow the robot to have a slimmer profile for any given size of base. The wheels fit into the cutouts, rather than jut out to the sides.

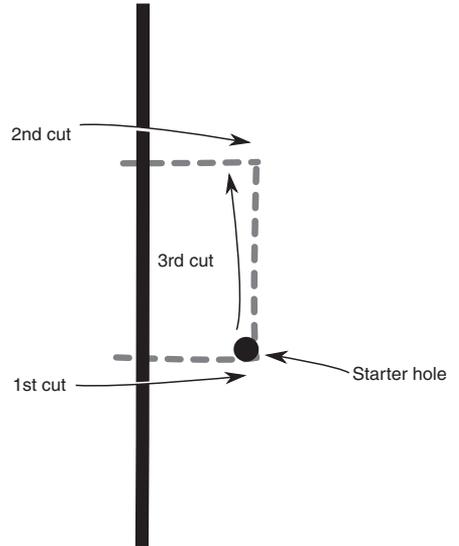


Figure 7-7 The basic wheel well is created by marking a rectangle into the base and cutting as shown. Drill a hole to insert the jigsaw blade.

1. Cut in only as deep as you want to make the wheel—an inch or two is usually enough.
2. Use a 1/4" bit and drill a hole at one of the inside corners of the well. Position the hole so that it's on the inside of the well; that way you won't have the remnants of the hole after the well has been cut out.
3. If using a jigsaw, insert the blade into the hole and cut toward the opposite end of the well. If using a coping saw, remove one end of the blade from the saw frame and pass the blade through the wood. Reattach the blade to the frame and begin cutting.

Figure 7-8 shows another technique that's ideal when using only hand tools.

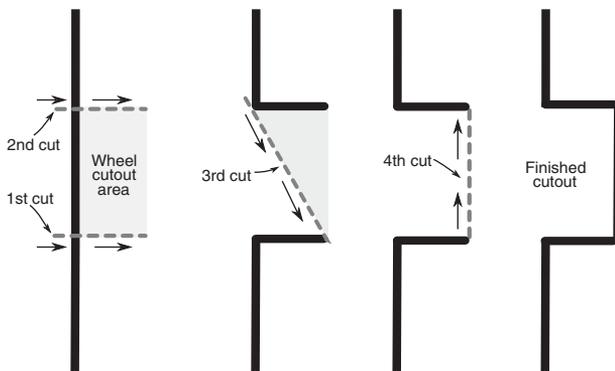


Figure 7-8 A wheel well can be cut using only hand tools, such as a coping saw. Begin with two parallel cuts for the side of the well. The coping saw blade is fine enough to make tight turns.

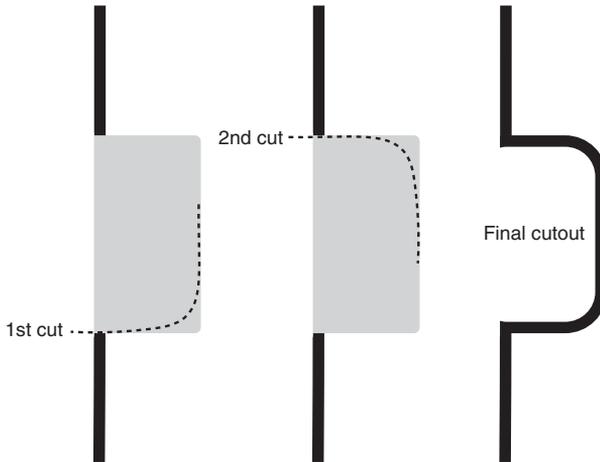


Figure 7-9 Rounded shapes may be cut using a jigsaw, coping saw, or scroll saw. This method requires a bit of practice to get right.

1. Use a backsaw to make cuts 1, 2, and 3. A backsaw is preferred because it will make straighter cuts.
2. Use a coping saw for the fourth and final cut.

And finally, there's no reason why the wheel wells must have right-angle corners. Figure 7-9 shows the rounded contour of the wheel well when using a jigsaw (or a coping saw) to make just two cuts. Go slow, and make a couple of trial cuts first on a piece of scrap. This technique requires a bit of patience and a little skill for good results.

CUTTING A FRAME

Frame construction allows you to make larger but lighter robots. The frame provides the overall skeletal structure of the bot, and over the frame you can place light materials to support the components of your machine. Frames also allow you to build tall robots in the same way they construct multistory buildings. You can stack multiple frames on top of one another, with the equivalent of pillars between them.

Wood frames can be constructed using strips of hardwood. Most hobby stores stock birch and mahogany, both excellent choices. You can buy the strips premade or cut your own if you have a table saw. For frames under 10" square, strips 1/2" to 3/4" wide are adequate; use 1" or wider wood for larger robots.

Wood selection for framing is critical. Stay away from softwoods, such as pine, fir, and redwood. They are not strong enough except for the smallest of bases. Aircraft-grade hardwood plywood is also a good choice.

Frames are best made using a small miter box and a backsaw. You can purchase both of these for under \$20 total. It's a small but wise investment. The miter box helps you make the straight and angled cuts necessary for frame construction.

Accurate Measuring a Must

The miter box will help you to make proper 45° angle cuts, but you still have to be careful that each of the four pieces of the frame is the exact proper length. Otherwise, the frame will not be a perfect square—or a perfect rectangle, whatever the case may be.

Follow these steps to ensure proper measuring and cutting of your frame.

1. Determine the outside dimension of the frame. For example purposes, we'll assume you're making a 9" square robot.
2. Let's start with the *top* frame piece. Using the miter box, cut a "left-hand" miter at the tail end of a wood strip. I refer to it as a left-hand miter because the joint will be on the left side of the frame.
3. With the tape measure placed at the top corner that you just cut, measure exactly 9". Using a pencil, make a mark to the immediate right of the 9" mark on the tape.
4. Again using the miter box, cut a "right-hand" miter, lining up the top of the cut with the mark. When cutting, be sure the blade is just to the right of the mark. This is called "keeping the line," and it ensures the length of the piece will be exactly 9", even considering the width of the saw blade (as you will recall from Chapter 6, this is called the *kerf*).
5. Now for the *bottom* frame piece. Pick up the strip that was left over from Step 4. It'll already have the proper (right-hand) miter cut. Measure exactly 9" and mark.
6. Turn the piece over and place in the miter box. Cut a left-hand miter, being sure to leave the line as you did in Step 4. I'm having you turn the wood strip over so that the mark at 9" is along the top. It's easier to see that way.
7. Now, compare the two frame pieces you've cut. If they are slightly different lengths, carefully sand the longer piece down. When sanding be careful not to round off the cut, or else the mitered corners won't meet well.
8. Repeat steps 2 through 7, this time for the left and right pieces. These are also 9", so the steps are duplicated exactly.

Assembling the Frame

Once the pieces are cut out, they are then assembled to make a frame. The frame pieces are held together using L-shaped angle brackets and steel fasteners.

Start by assembling the upper-left corner, using one side piece and one top piece. With a bracket as a guide, mark a hole for drilling at the corner of one of the frame pieces. Do just one "leg" of the bracket at a time; don't try to mark multiple holes. With the wood marked, move the bracket out of the way and drill the hole (see the next section for details about drilling). Assemble the bracket on the frame piece following the example in Figure 7-10. Use a flat washer next to the nut.



Figure 7-10 An L angle bracket connects the frame pieces together. Use machine screws, nuts, and washers for a solid construction.

Now for the adjoining frame piece. Align it to the corner you just assembled. Using the other “leg” of the bracket, mark a hole in the second piece. As you did before, drill the hole and finish the corner by adding the fasteners.

Put this corner aside, and repeat the steps for the lower-right corner. When done you will have the top left side of the frame as one piece and the bottom right side as the other piece.

Complete the frame the rest of the way by carefully lining up the corners, marking, drilling, and assembling with fasteners.



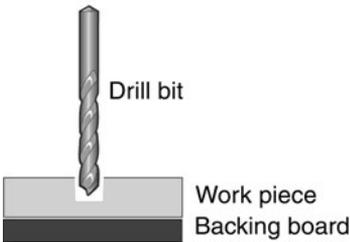
I don't recommend assembly techniques using nails, staples, or glue. Though these are cheaper and may take less time, the finished frame isn't strong enough for robot use. Besides, when using hardware brackets and fasteners you can readily disassemble the frame pieces should you decide to rebuild it.

DRILLING WOOD

Holes are for mounting things to your robot, and holes are made with the drill and bit. You can use a hand or electric power drill to make holes in wood. Electric drills are great and do the job fast, but you can also use a hand drill if you feel uncomfortable with power tools. Either way, it's important that you use only sharp drill bits. If your bits are dull, replace them or have them sharpened.

It's important that you drill straight holes, or your robot may not go together properly. If you have a drill press, and it's large enough, you can use it to drill perfectly straight holes in plywood and other large wood stock.

Using a Backing Board



To prevent splintering, place a piece of scrap wood behind the piece you're drilling. Use a soft wood, like pine, so you can “feel” when the bit has penetrated the wood you're drilling.

If you're still getting splintering on the underside of the wood as you drill, try pushing the bit through more slowly. And be sure you're using a sharp bit. Replace if necessary.

Adjusting the Speed of Drilling

As first noted in Chapter 6, “Mechanical Construction Techniques,” you can use fairly high speeds when drilling through wood. The exact speed depends on the type of wood—hardwoods need a little slower speed because they are denser.

But there is no exact right speed for each wood, and it comes down to experience. My advice: Practice drilling using some scrap pieces of the type of wood you'll be using in your projects. Inspect the quality of the holes you've drilled. You'll soon get the hang of picking the right speed.

FINISHING WOOD

You can extend the life of your wood robot bases and frames, not to mention enhance their looks, with simple finishing. Wood finishing involves sanding, which smoothes down the exposed grain, then painting or sealing. Small pieces can be sanded by hand, but larger bases benefit from a power sander.

Coarse Filing and Shaping

You can shape wood using rasps. A rasp is the same as a file—both are covered with a surface of sharp teeth to grind down the wood—but the teeth of a rasp are much more coarse. You can also shape wood using a heavy-duty drum or disc sander, outfitted with very coarse sandpaper (see the section immediately following, on sandpapers).

Sanding

Sandpapers are used to smooth wood, removing saw marks, chips, and other imperfections. Sandpapers are available in a variety of *grits*—the lower the grit number, the coarser the paper. With a higher coarseness you remove more wood at a time as you use the paper.

The recommended approach is to start with a coarse grit to remove splinters and other rough spots, then finish off with a moderate- or fine-grit paper. For wood, you can select between aluminum oxide and garnet grits. Aluminum oxide lasts a bit longer. Sandpapers for wood are used dry. For hand sanding, wrap the paper around a wood or plastic block to provide even pressure.

Use	Grit			
	F	M	C	EC
Heavy sanding			•	•
Moderate sanding		•		
Finish sanding	•			

Grit Key	Name	Grit
EC	Extra coarse	30-40
C	Coarse	50-60
M	Medium	80-100
F	Fine	120-150

Painting

Wood can be painted with a brush or spray. Brush painting with acrylic paints (available at craft stores) takes longer but produces excellent results with little or no waste. One coat may be sufficient, but two may be necessary. Woods with an open grain may need to be sealed first using a varnish, primer, or sealer, or else the paint will “soak” into the wood. You may also opt to skip the painting step altogether and apply only the sealant.

Spray paints are an alternative to brush painting. Be sure to use the spray can according to the directions on the label. Use only outdoors or in a well-ventilated area.

Build a Motorized Wooden Platform

You read in the preceding chapter how to use wood to create robots. In this chapter you can put that knowledge to good use to build the PlyBot, a simple, affordable, and expandable robot base using just a small sheet of 1/4" plywood. Only simple straight cuts are used. You need to drill just six holes to mount the two motors and the ball caster.

The construction of the PlyBot requires no special tools. You need a saw for cutting the wood and a drill for making holes. Plus: a screwdriver and small pliers for assembling the pieces. Parts for the PlyBot are available online and at better-stocked hobby stores.



In the text that follows I provide the exact model numbers for the pieces you need. But remember that you're free to substitute with something else if you already have it or if you've found substitutes that are cheaper or easier to get. See also the RBB Online Support site (check out Appendix A) for additional parts sources and alternatives.

Making the Base

Refer to Table 8-1 for a list of parts.

Figure 8-1 shows the completed PlyBot, with wooden base, twin gear motors, wheels, and support caster. The robot measures 5" × 7" overall. The base is constructed from 1/4"-thick 5-ply birch plywood and is driven by two Tamiya worm gear motors; the motors come in kit form, and assembly takes 10 to 15 minutes for each motor. The wheels, also made by Tamiya, securely lock onto the axles of the motors using hardware that's included.

The "third wheel" of the PlyBot is a ball caster—more technically called a ball transfer, as they are used on conveyor belts to literally transfer goods (boxes, palettes, machinery) throughout a warehouse or factory. I used a ball caster/transfer from McMaster-Carr (www.mcmaster.com), a large and well-known online industrial supply retailer. The caster costs under \$5, and

Table 8-1 PlyBot Parts List

Robot Base:	
1	5" × 7" 5-ply 1/4"-thick birch plywood*
2	Tamiya Worm Gear kit, model 72004†
1	Tamiya Narrow Tires (pair), 58mm diameter, model 70145‡
1	11/16" diameter ball transfer (i.e., ball caster), McMaster-Carr, item #5674K57‡
6	Assembly hardware: 4-40 × 1/2" machine screws, 4-40 nuts

* Birch plywood is available at hobby and craft stores. Look for 1/4"-thick aircraft-grade plywood.

† Tamiya motors and wheels are available at most online hobby stores, such as Tower Hobbies, as well as many robotics specialty sites.

‡ You may substitute most any ball caster with a 1" to 1.5" flange-to-wheel depth. There is nothing special about this particular caster, so feel free to substitute if you want to use something else.

you can substitute another for it if you like, as long as it's about the same size. Your caster should have a flange-to-ball depth of about 1" to 1.5".

CUTTING AND DRILLING

Begin construction by cutting the wood to 5" × 7". Then, using Figure 8-2 as a guide, cut the sides of the wood to create a narrowing shape. The finished base should look like Figure 8-3.

Once the base is cut, use a sander or small rasp to round off the four corners. This is not mandatory, but it makes for a nicer overall shape and appearance. You should also apply a light sanding to the base now, to remove any rough edges caused by cutting. If you wish, you may apply a coat of sealant or varnish to close the pores of the wood. This will prevent it from absorbing moisture, which could cause warping.

Consult the drilling template in Figure 8-4. Use a 1/8" drill bit for all holes. The location of the holes are not supercritical *except* for the distance between the two holes for each of the



Figure 8-1 The PlyBot uses two Tamiya gear motors and wheels, plus a ball caster (or skid) for balance. It's made using a single piece of quarter-inch-thick 7" × 5" aircraft plywood.

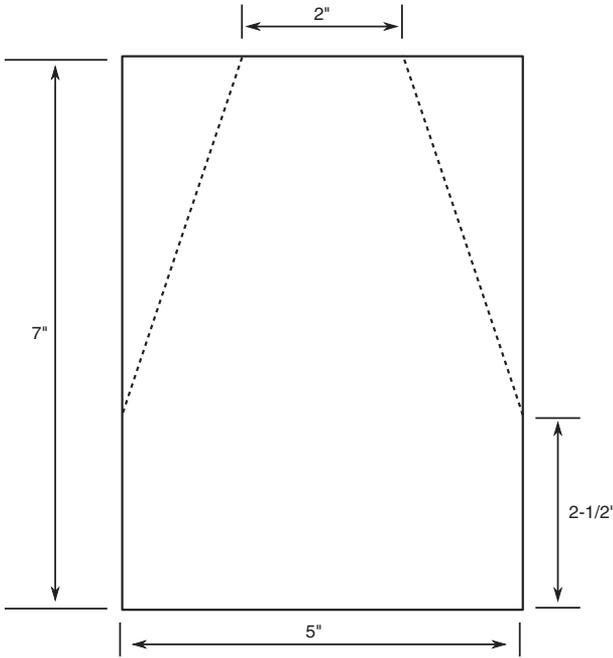


Figure 8-2 Cutting guide for the PlyBot. Dimensions are flexible, and you can scale the robot up in size if you need something a bit bigger.

motors. These need to be fairly accurate, though take note that the holes in the mounting flanges of the Tamiya gear motor are oblong. This allows a few millimeters' leeway in matching the motors to their mounting holes.



Before assembly of the motors, use the motor flange itself to mark the spacing on a piece of paper. You can then transfer the marks to the wooden base using a small punch or a nail.

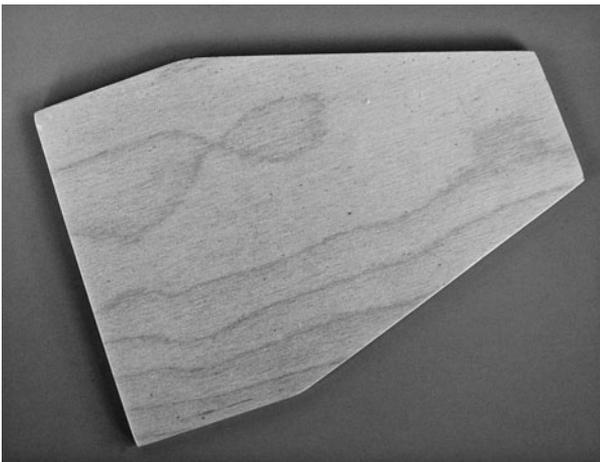


Figure 8-3 Plywood panel cut to shape for the PlyBot. Use a sander or small rasp to round off the corners for a sleeker look.

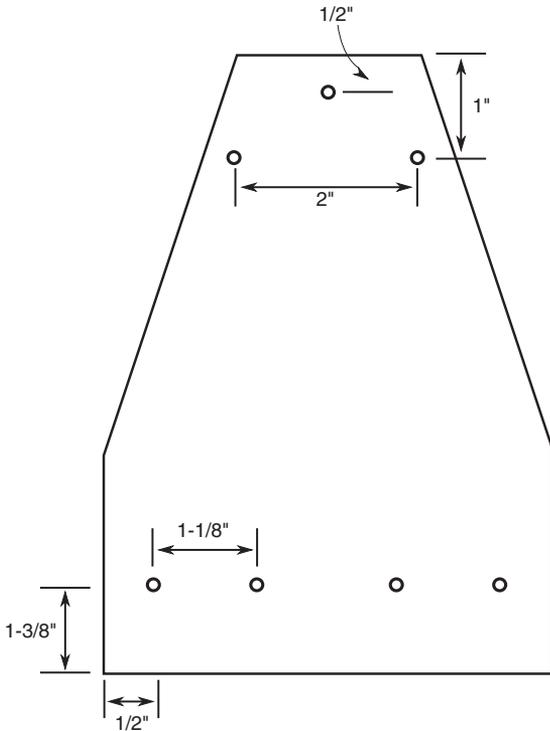


Figure 8-4 Drilling template for the PlyBot. The extra hole at the top is for the optional skid, should you wish to not use a ball caster. Turn the robot over so the clearance between the ground and the base is reduced.

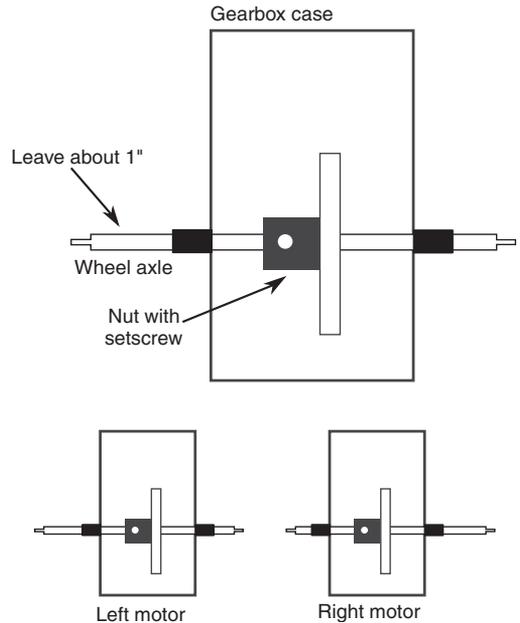


Figure 8-5 Tamiya worm gear motors, showing the alignment of the wheel axle through the output gear. Leave about an inch of the axle showing for mounting the wheel. Make a “left” and a “right” motor, as shown.

Building and Attaching the Motors

The PlyBot uses two Tamiya #72004 worm gear motors. They’re called worm gear because of the type of gearing mechanism that’s used. Even though the gearbox of the motor uses just three gears, it has a high gear reduction, meaning the wheels turn fairly slowly—about the right speed for a small bot.

While assembling the motor you may select either of two ratios: 216:1 or 336:1. I built the prototype PlyBot using the lower, 216:1, ratio, so that the robot scooted around the floor a bit faster. You can opt for either ratio, but make sure both motors are assembled the same way, or your PlyBot will run around in circles! Assembly instructions are included with the motor. You’ll need a small Philips-head screwdriver and pair of needle-nose pliers to build it.

The motor comes with two long wheel axles. You want the axle with the holes drilled into each end. You can save the other axle for some other project.

The axle is secured to the gearbox gears using a set screw. Initially position the axle so that about 1” sticks out of the side of the motor (see Figure 8-5). You’ll be adjusting the position of the axle after you’ve mounted the motors, so for now just lightly tighten the set screw.

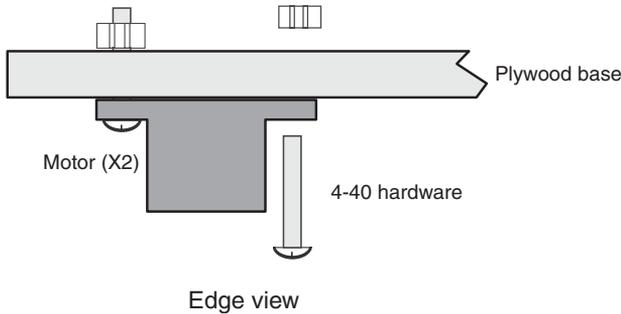


Figure 8-6 Mounting the motors using 4-40 hardware fasteners.



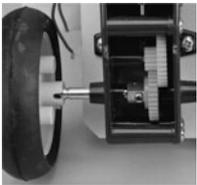
You'll be building one left motor and one right motor. That means the 1" of axle should point to the left on one of the motors and to the right on the other.

Important! Prior to inserting the motor into the gearbox, manually rotate the gears so that the set screw on the wheel axle points upward (away from the mounting flanges). This allows you to fine-tune the position of the axle once the motors have been mounted. If you don't do this now, you might not be able to access the set screw.

Attach each motor to the underside of the plywood base, as shown in Figure 8-6. Use two 4-40 \times 1/2" machine screws and nuts. Feed the screws from the motor side, and tighten the nuts on the top of the base. You may use #4 washers (if you choose) on the nut side.

Building and Mounting the Wheels

The Tamiya wheels specified in the parts list are designed to directly attach to the axle of the worm gear motor. The wheels are in kit form and come with two different hubs. You want the hubs for the 4mm round axles, the kind used in the worm gear motor. These hubs are visually identified with a thin slot that runs through the center. Assemble the wheels according to the instructions that come with the set.



Before mounting the wheels, use a pair of pliers to insert the small spring pin (included with the motor) into the hole at the outside (wheel) end of the axle. The pin engages into the slot in the wheel hub. Without the pin, the wheel will just freely rotate over the axle.

The wheel securely mounts to the axle using the supplied nut. The Narrow Tire wheel set comes with a small plastic wrench for use in tightening this nut over the axle. The wheel set only comes with two nuts, so don't lose these! They can be hard to replace because of their small size.

Attaching the Ball Caster

The ball caster attaches to the front of the robot using two 4-40 \times 1/2" screws and nuts. The screws should be inserted from the top side of the base; tighten the nuts against the mounting flange of the ball caster. Figure 8-7 shows the 4-40 hardware used with the ball caster, and Figure 8-8 shows the caster attached to the bottom of the PlyBot base.

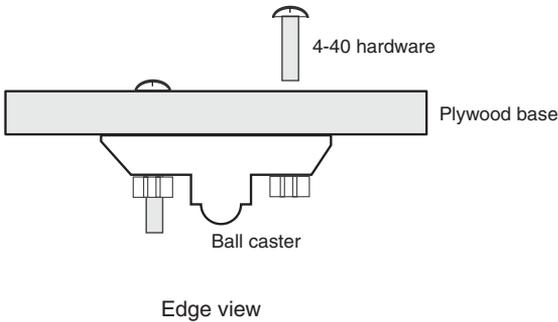


Figure 8-7 Mounting the ball caster using 4-40 hardware fasteners.



Figure 8-8 Ball caster (a.k.a. *ball transfer*) as specified in Table 8-1. You can substitute any other caster mechanism that is the same approximate size.



The PlyBot is “invertible,” meaning you can turn it over, and the side with the motors becomes the top of the bot. If you do this, there’s isn’t enough clearance for a ball caster. Instead, construct a skid using an 8-32 × 3/4” machine screw and matching 8-32 hex nut and acorn (cap) nut. The acorn nut provides a smooth surface for the skid to glide against. You can adjust the height of the skid using the hex nut.

Drill a hole just smaller than the diameter of the screw, so that the screw makes its own threads when you insert it. Be careful not to make the hole too small, or else the wood may chip as the screw is tightened.



Using the PlyBot

The Tamiya motors used in the PlyBot are rated for 3 to 6 volts. You can rig them up to switches to manually control the motor (on/off and direction) or use electronic control. These topics are covered in Chapter 22, “Using DC Motors,” as well as the *My First Robot* lessons found on the RBB Support Site.



We’re getting a bit ahead of ourselves here, but this needs to be mentioned now: When run at 4.5 volts, the motors draw less than 100 milliamps of current. But if the motors stall—meaning they are physically stopped while voltage is still applied to them—current consumption goes up to about 1.5 amps.

Bear this in mind when using electronic control of the motors. Be sure the drive electronics can handle the current. See Chapters 21 and 22 for more details on current consumption of motors and what it means.

The PlyBot has plenty of room on top (and underneath, too) for mounting electronics, batteries, sensors, and other paraphernalia. Because the base is made of wood, it’s easy to drill additional holes for mounting components.

Variations on a Theme

The PlyBot is the basic “T-bone” robot, where there’s a pair of two motors on one end of the bot and a supporting caster or skid on the other. Figure 8-9 shows the concept, whereby you can readily visualize the “T” shape of the bot.

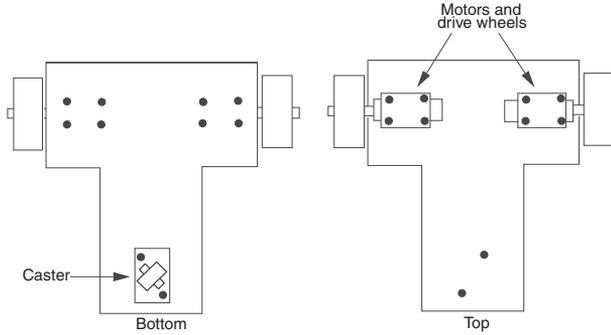
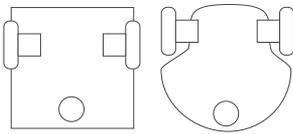


Figure 8-9 Concept of the “T-bone”-shaped robot, where—like the PlyBot—the wheels are located on one end of the base, and a balancing caster or skid at the other.

An advantage of this style of robot is that it’s easy to construct using common materials. The motors attach to the top of the “T,” and the caster or skid to the opposite end. As shown in later chapters (see, for example, Chapter 14, “Rapid Prototyping Methods”), you can often find materials that are already in this shape. Just stick on the caster and the two motors and wheels, and you’ve made yourself a robot.



The exact shape of the base is not relevant to the function of the robot, though for best results smooth or round contours work best. The corners of square robots can snag on furniture and other objects. Though it’s a bit more work, smoothing out the shape of the base provides a more streamlined look, and it allows the robot to navigate more easily, and with less potential damage to its surroundings.

Working with Plastic

It all started with billiard balls. In the old days, billiard balls were made from elephant tusks. By the 1850s, the supply of tusk ivory was drying up and its cost had skyrocketed. So in 1863, Phelan & Collender, a major manufacturer of billiard balls, offered a \$10,000 prize for anyone who could come up with a suitable substitute for ivory. A New York printer named John Wesley Hyatt was among several folks who took up the challenge.

Hyatt didn't get the \$10,000. The material he promoted, celluloid, carried with it too many problems—like occasionally exploding during its manufacture. While Hyatt's name won't go down in the billiard parlor hall of fame, he will be remembered as the man who helped start the plastics revolution. Celluloid was perfect for such things as gentlemen's collars, ladies' combs, containers, and eventually even motion picture film.

Since the introduction of celluloid, plastics have taken over our lives. Plastic is sometimes the object of ridicule—from plastic money to plastic furniture—yet even its critics are quick to point out its many advantages:

- Plastic is cheaper per square inch than wood, metal, and most other construction materials.
- Certain plastics are extremely strong, approaching the tensile strength of such light metals as copper and aluminum.
- Some plastic is “unbreakable.”

So you can imagine that plastic is ideal for use in hobby robotics. Read this chapter to learn more about plastic and how to work with it. In the next chapter I'll show you how to construct an easy-to-build “turtle robot”—the PlastoBot—from inexpensive plastic parts.

Main Kinds of Plastics for Bots

Plastics represent a large family of products. They often carry a fancy trade name, like Plexiglas, Lexan, Acrylite, or Sintra. Some plastics are better suited for certain jobs, and only a relatively small number of them are appropriate for robotics.

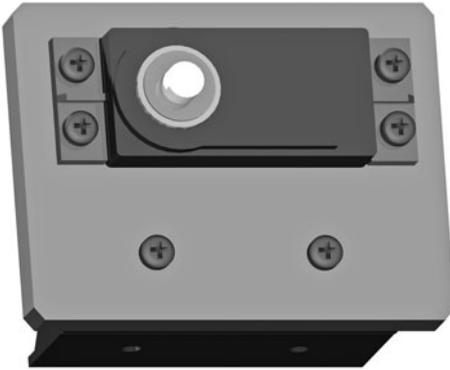


Figure 9-1 Plastic can be used for the entire robot or just parts of it, like this all-plastic bracket for mounting a radio control servo motor to a robot.

You can use plastic for the entire robot or just parts of it—mixing and matching materials is not only allowed in robot construction, it’s encouraged. For example, the inexpensive motor mount in Figure 9-1 is made of a lightweight plastic; you can combine the mount with robots made of wood, any kind of plastic, metal, cardboard, or any other material.

Here’s a short rundown of the plastics that I find most useful in robotics:

ABS is short for “acrylonitrile butadiene styrene,” should you care about this sort of thing.

ABS is most often used in sewer and wastewater plumbing systems; it’s the large black pipes and fittings you see in the home improvement store. But it can be any color, and it can come in any shape; LEGO building blocks are made of ABS plastic. You can get ABS plastic in sheets.

Acetal resin is called an “engineering plastic,” because it is designed to be used for various engineering applications, such as making prototypes. It’s often referred to by one of its brand names, Delrin. Acetal resin comes in sheet and block form.

Acrylic is the mainstay of the decorative plastics industry. It can be easily scratched, but if the scratches aren’t too deep they can be rubbed out. Acrylic is somewhat tough to cut because it tends to crack, and it must be drilled carefully. In the United States, acrylic is often referred to by its most popular trade names, Plexiglas and Lucite. In the United Kingdom, it’s Perspex.

Nylon is tough, slippery, self-lubricating stuff that is more commonly used in robotics for lightweight screws and nuts. Many plastics distributors also supply nylon in rods and sheets. It’s very difficult to glue anything to nylon.

Polycarbonate plastic is a cousin of acrylic but more durable and resistant to breakage. It comes in rods, sheets, and tubes. Often sold as a replacement for glass windows, polycarbonate is fairly hard to cut and drill. Common brand names include Lexan, Hyzod, and Tuffak.

Polyethylene (“polythene” in the United Kingdom and elsewhere) is lightweight and translucent and is often used to make flexible tubing. A variation of this plastic, called high-density polyethylene, or HDPE, is used to make very durable kitchen cutting boards. I like this stuff, but it’s very difficult (darn near impossible) to glue anything to it.

Polystyrene is a mainstay of the toy industry. Although often labeled “high-impact” plastic, polystyrene is brittle and can be damaged by low heat and sunlight. It’s only modestly useful in robotics and is mentioned here only because hobby model stores carry the stuff in sheet form.

PVC is short for polyvinyl chloride, an extremely versatile plastic best known as the material used in freshwater plumbing. Usually processed with white pigment, PVC can be in any color. Great stuff for robots, especially PVC sheets; much more about this stuff later in this chapter.

Best Plastics for Robotics

Let's review the best of these plastics and compare how they are typically used in amateur robots. And while we're at it, let's note how workable—ease of cutting, drilling, and gluing—each plastic is.

Plastic	Typical Robotics Uses	Workability	Gluability
ABS	Base material; small parts cut to size and shape	Easy to drill and cut	Excellent
Acrylic	Base material; small parts cut to size and shape—however, avoid any application where repeated impact or stress might cause cracks or breaks	Medium workability; difficult to cut and drill without cracking	Excellent
Polycarbonate	Base material	Difficult to cut and drill, but not as prone to cracking as acrylic	Good
HDPE—High Density Polyethylene	Base material; structure for framing	Somewhat easy to drill; moderately easy to cut, but requires power tools	Poor
PVC	Base material; small parts cut to size and shape	Very easy to drill and cut	Excellent

Where to Buy Plastic

Some hardware stores carry plastic, but you'll be sorely frustrated at their selection. The best place to look for plastic—in all its styles, shapes, and chemical compositions—is a plastics specialty store or plastics sign-making shop. Most larger cities have at least one plastics supply store or sign-making shop that's open to the public. Look in the Yellow Pages under *Plastics—Retail*.

Another useful source is a plastics fabricator. There are actually more of these than retail plastic stores. They are in business to build merchandise, display racks, and other plastic items. Although they don't usually advertise to the general public, most will sell to you, either full sheets or remnants (ask nicely and they may give you some of their discards). If the fabricator doesn't sell new material, ask to buy the leftover scrap.

The Ins and Outs of Rigid Expanded PVC

Rigid expanded PVC is the robot maker's dream material, and I use it extensively, more than any other type of plastic. Figure 9-2 shows a prototype robot made using 1/4"-thick expanded PVC plastic.

Rigid expanded PVC is commonly used for sign making, so it's relatively cheap, lightweight, and available in a rainbow of colors. It's manufactured by mixing a gas with molten plastic. The plastic is then extruded into various shapes: sheets, rods, tubes, bars, and more. The gas forms tiny microscopic bubbles in the plastic, expanding it. The expansion makes the material bulkier—and therefore lighter. Because of the rough, bubbly appearance of the plastic, expanded PVC is sometimes referred to as "foam PVC" or "foamed PVC."

BENEFITS OF EXPANDED PVC

Rigid expanded PVC comes in all sorts of shapes, sizes, and colors, but it's the sheet form we're most interested in. Because it's been "puffed up" in the expansion process, expanded PVC contains less plastic than ordinary PVC materials. The benefits of the expansion process are:

- *Less plastic = less weight.* That's important for building robots where added weight makes the battery drain faster.
- *Less plastic = less density.* This makes expanded PVC easier to drill, cut, and mill. If you've ever cut acrylic plastic, you know it chips and breaks easily, and its high density makes using hand tools a real chore. The thinner expanded PVC materials can be cut using a knife; the thicker stuff, with an ordinary saw blade.

Rigid expanded PVC (or simply PVC from here on) is often used as a replacement for wood. As robot builders, we're more interested in the PVC sheets used to make signs—sign

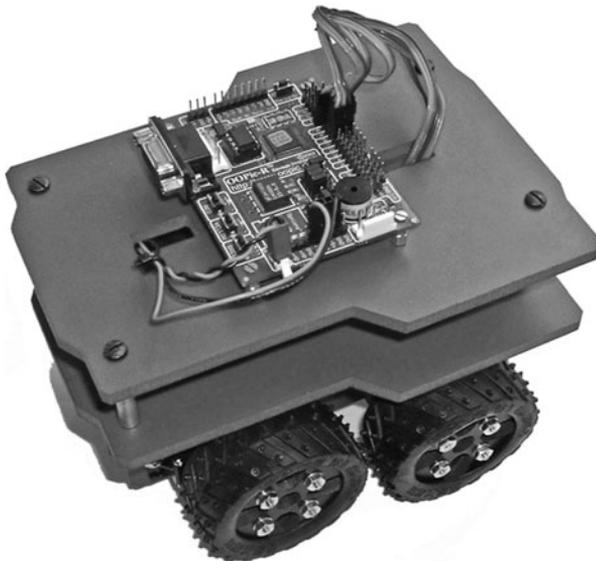


Figure 9-2 With its ease of drilling and cutting, expanded PVC plastic lets you create all sorts of designs for your robots. This one uses two "decks" or levels that are merely rectangles with the corners cut off. The robot is completed using brackets for the four servo motors.

makers refer to this raw material as *substrate*. It's available in a variety of sizes and thicknesses, in many colors: blue, red, orange, tan, black, brown, yellow . . . you name it.

PVC sheet goes by many trade names, such as Sintra, Celtec, Komatex, Trovicel, and Versacel, but it's probably easiest if you just ask for it by its generic *expanded PVC* or *foamed PVC* moniker.

CHOICES IN SHEET THICKNESS

Sheets are commonly available in any of several millimeter sizes. Here are some of the more common thicknesses:

- 3mm, or roughly 1/8"
- 6mm, or roughly 1/4"
- 10mm, or roughly 13/32"

The following table details the weight of a 12" × 12" rigid expanded PVC sheet, at various thicknesses. Comparable weight per square foot for acrylic plastic is also given; all weights are representative, as some brands are lighter or heavier than others.

Thickness	Weight (lb/sq ft)	
	Expanded PVC	Acrylic
.080 (5/64")	.287	.547
.118 (1/8", or 3mm)	.429	.729
.197 (3/16")	.722	1.09
.236 (1/4", or 6mm)	.858	1.46
.393 (3/8")	1.03	2.19
.500 (1/2")	1.30	2.91

How to Cut Plastic

FYI

For more about saws and materials cutting in general, be sure to read Chapter 6, "Mechanical Construction Techniques."

Soft and thin plastics (1/16" or less) may be cut with a sharp utility knife. When cutting, place a sheet of cardboard or art board on the table. This helps keep the knife from cutting into the table, which could ruin the tabletop and dull the knife. Use a carpenter's square or metal rule when you need to cut a straight line. Prolong the blade's life by using the rule against the knife holder, and not by the blade.

CUTTING BY SCORING

Harder plastics can be cut in a variety of ways. When cutting acrylic plastic less than 3/16" inch thick, one way is to use the score method (see Figure 9-3):

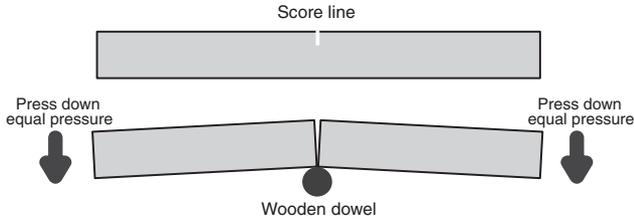


Figure 9-3 Cut acrylic, polycarbonate, or other hard plastic by scoring a line with a sharp knife. Then break at the line over a round wooden dowel.

1. Use a sharp utility knife and metal carpenter's square to "score" a cutting line. By scoring you're cutting in only part of the way—making a deep scratch. If necessary, use clamps to hold down the square. Most sheet plastic comes with a protective peel-off plastic on both sides. Keep it on when scoring.
2. Carefully repeat the scoring from 5 to 10 times, depending on the thickness of the plastic. The thicker the plastic, the more scoring lines you should make.
3. Place a 1/2"- or 1"-diameter dowel under the plastic so the score line is on the top of the dowel. With your fingers or the palms of your hands, carefully push down on both sides of the score line. If the sheet is wide, use a piece of 1-by-2 or 2-by-4 lumber to exert even pressure.



Cracks are most likely to occur on the edges, so press on the edges first, then work your way toward the center. Don't force the break. If you can't get the plastic to break off cleanly, deepen the score line with the utility knife.

CUTTING BY SAWING

Thicker sheet plastic, as well as extruded tubes, pipes, and bars, must be cut with a saw. If you have a table saw, outfit it with a fine-tooth blade designed for nonferrous metals when cutting acrylic or other hard plastic, a plywood-paneling blade when cutting PVC. When cutting acrylic and other hard plastic, slow down the feed rate—the speed at which the material is sawed in two. Forcing the plastic or using a dull blade heats the plastic, causing it to deform and melt.

When working with a power saw, use fences or pieces of wood held in place by C-clamps to ensure a straight cut.

You can use a handsaw to cut smaller pieces of plastic. A hacksaw (Figure 9-4) with a medium- or fine-tooth blade (24 or 32 teeth per inch) is a good choice. You can also use a coping saw (with a fine-tooth blade) or a razor saw. These are good choices when cutting angles and corners as well as when doing detail work.

You can use a motorized scroll saw to cut plastic, but you must take care to ensure a straight cut. If possible, use a piece of plywood held in place by C-clamps as a guide fence. Be sure the material is held down firmly. Otherwise, the plastic will vibrate against the cutting tool, making for a very rough edge and an uneven cut.



Slow down the speed of the scroll saw to prevent the plastic from melting at the cut as the blade gets hot. If the plastic melts back into the cut even with the saw at its slowest setting, choose a coarser blade.



Figure 9-4 Use a hacksaw with a medium- or fine-tooth blade to cut plastic. For cuts with curves use a coping saw.

How to Drill Plastic

Wood drill bits can be used with expanded PVC and ABS, but for acrylic and polycarbonate plastic the best bet is to use a drill bit that's specially made for the job. What happens with regular bits is that the flutes may suddenly get caught in the plastic, causing the tool to grab the plastic and crack or break it.

Needless to say, not only does this wreck the piece, it's dangerous, as tiny bits of plastic fragments may fly through the air. That's why you should always wear eye protection, even when doing something as simple as drilling a hole.

START WITH A PILOT HOLE

If you can't afford a specialty bit made for drilling in plastic, try starting the hole with a small bit, say 5/64", then gradually increasing the bit size until you get the hole you want. This minimizes the "dogging" that can occur when the flutes of the bit get caught in the plastic. You can use an ordinary high-speed steel bit when drilling into PVC.

RIGHT SPEED WITH POWER TOOLS

When making holes in plastic with a power tool, the drill should have a variable-speed control. Reduce the speed of the drill to about 500 to 1000 RPM. When drilling acrylic and polycarbonate, always back the plastic with a wooden block. Without the block, the plastic is almost guaranteed to crack. As with cutting, don't force the hole and always use sharp bits. Too much friction causes the plastic to melt.

Expanded PVC doesn't usually require backing with wood, but be sure not to force the drill bit through.

FYI

For more about drill tools and drilling in general be sure to read Chapter 6, "Mechanical Construction Techniques."

Making Plastic Bases

I don't hide that my unabashed favoritism in building robots is for plastic. For its size, plastic is stronger than most woods, and it's easier to work and cheaper than metal. My preferred

types of plastics are expanded PVC, ABS, high-density polyethylene (HDPE), and polycarbonate, in that order. Expanded PVC is used in industry as an alternative to wood (example: wood molding), and it cuts and drills much like wood.

FOR STARTERS, SEE WOOD BASE DESIGNS

Many of the techniques that you can use with wood to make handsome robot bases also apply to plastics, especially sheet PVC. Refer to the “Cutting a Base” section in Chapter 7, “Working with Wood,” for details on how to cut out useful shapes for robot bases.

One exception is the use of a motorized jigsaw. Unless you’re experienced, the jigsaw may produce too much vibration as it cuts, especially when working with the thinner 1/8” plastic sheets. At best, the vibration will cause rough edges in the cut pieces; at worst, the plastic may crack or even break.

Here are some additional ideas for making bases using plastic sheet.

BASES FROM STRAIGHT CUT PIECES

Chapter 7 detailed the benefit of cutouts for the wheels of your robot—so-called wheel wells (see Figure 9-5). Because plastic is a bit stronger by thickness than wood, it’s possible to construct bases with wheel wells without fancy cutting. It can be done just with squares and rectangles. Figure 9-6 shows the idea, using three separate rectangles of 3mm expanded PVC plastic.

1. Begin by cutting the center piece. This piece is the full length of the robot, but only as wide as the inside edges of the wheel wells.
2. Cut two end pieces. These measure the total width of the base, but their length is from the wheel well to end of the base.
3. Since avoiding square edges is a good idea (when possible), chamfer two corners of each end piece as shown. You can cut the chamfers with a saw or use a rasp or coarse sandpaper. If you use a motorized sander, such as a drum or disc sander, push the plastic in slowly; otherwise, it’ll melt.
4. Assemble the three pieces using glue or fasteners. Ordinary household glue will work, or you can use contact cement or solvent cement (as discussed later in this chapter) designed for the plastic you’re using.



Using two 3mm sheets of plastic, the main center portion of the assembled base is 6mm (about 1/4”) thick. That’s usually sufficient for bases under about 10” in size. Note that the corners of the base are only 1/8”; that’s okay, as the corners aren’t structurally relevant, and you are unlikely to mount heavy components there.

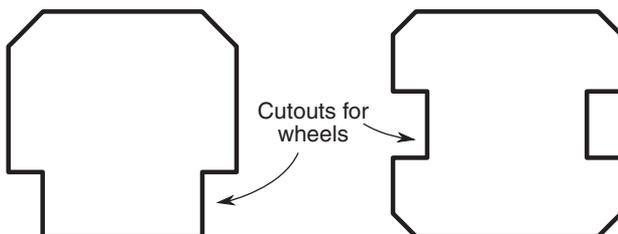


Figure 9-5 Wheel wells are chunks removed from the base of the robot to make room for the wheels.

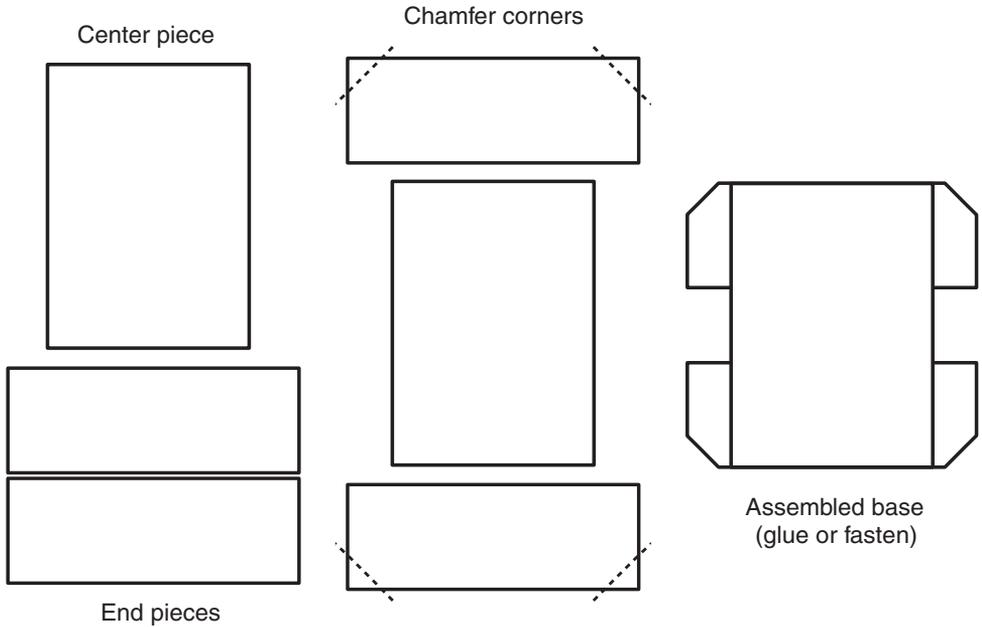


Figure 9-6 If cutting out wheel wells proves difficult with the material or tools you have, you can achieve the same result by assembling individual rectangular pieces together.

Making Plastic Frames

You read about making full-bodied frames out of wood in Chapter 7, “Working with Wood.” The majority of plastics available to consumers are rather flexible—this is especially true of expanded PVC. This makes them less than adequate for the job of creating robot frames.

However, given a thick enough material (I’d say 3/8” or larger), the flexing is minimized. You can combine strips of 6mm PVC to make 1/2” framing material, like that shown in Figure 9-7. See Chapter 7 on how to use a miter box and backsaw to cut the ends of the frame pieces to 45°. Assemble using the same kind of flat angle brackets and steel fasteners.

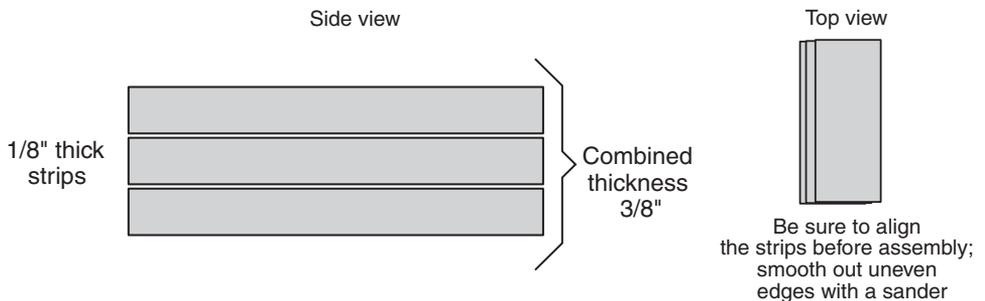


Figure 9-7 Pieces of thinner plastic sheets can be rigidized by stacking them several layers deep. You can make solid bars of plastic, useful for frames, with just a few strips of 1/4” material.

How to Bend and Form Plastic

Many kinds of plastics can be formed by applying low localized heat. A sure way to bend sheet plastic like PVC and acrylic is to use a strip heater, available at ready-made plastics supply outlets. Look through your Yellow Pages for local stores, or see Appendix B, “Internet Parts Sources,” for leads on online retailers that carry plastics and plastic tools.

When you plug in the tool, a narrow electric element applies a regulated amount of heat to the plastic. When the plastic is soft enough, you can bend it into just about any angle you want.

Plastic heaters are easy to use, but getting good results takes lots of practice. Try with some scrap pieces until you get the hang of it. None of the projects in this book require plastic bending and forming, but feel free to try it on your own if you want to experiment. A couple of tips to get you started:

- Be sure that the plastic is pliable before you try to bend it. Otherwise, you might break it or cause excessive stress at the joint (a stressed joint will look cracked or crazed).
- Bend the plastic *past* the angle that you want. The plastic will “relax” a bit when it cools off, so you must anticipate this. Knowing how much to overbend will come with experience, and the amount will vary depending on the type of plastic and the size of the piece you’re working with.
- Use the right heat setting for the plastic. Don’t apply too much heat, or you’ll be sorry. Expanded PVC has a very low melting point—about 165 to 175°F. This stuff emits a very noxious and corrosive gas (hydrogen chloride) when it burns, so treat it with care!

How to Smooth the Edges of Plastic

After cutting, your plastic parts may need a bit of smoothing to remove any rough edges. As with wood, you can apply a light sanding with a fine-grit aluminum oxide (not garnet) sandpaper. For soft plastic like PVC, use the sandpaper dry. For hard plastic, you can use the paper dry or get it wet with water.

You can also shape the plastic—to remove sharp corners in square bases, for example—by using a very coarse sandpaper.

Recommended Grits: Aluminum Oxide

Use	Very Fine (160–200 grit)	Fine (120–150 grit)	Coarse (50–60 grit)	Extra Coarse (30–40 grit)
Shaping			•	•
Smoothing	•	•		

Grit represents the coarseness of the surface of the sandpaper. The higher the number, the finer the grit.

How to Glue Plastic

When building bots, my preference is always to use mechanical fasteners . . . nuts, screws, that type of thing. The reason: It’s not uncommon to want to disassemble a robot, either to

reuse parts or to allow someone else to rebuild it, and learn from the experience. When parts are glued, disassembly—and, of course, reassembly—is much harder.

Still, there are plenty of reasons to cement pieces together, and most plastics (especially PVC, ABS, and acrylic) make gluing pretty easy. The kind of glue you use depends on the kind of plastic you're working with and the type of bond you want.

- PVC, ABS, acrylic, and polycarbonate plastics can be bonded using a solvent-based cement. The cement contains chemicals that actually melt the plastic at the joint. The pieces become fused together—that is, they shall be whole, the two made one, like the *Dark Crystal*.
- Household adhesives can be used for gluing plastic together, with varying degrees of success, depending on the glue and the plastic. Experiment. Table 9-1 lists adhesives I recommend for bonding popular plastics with themselves and with other materials. When all else fails, try epoxy cement.

APPLYING SOLVENT CEMENT

There are different solvent mixtures for the different plastics. The best is to use a solvent-based cement specifically made for the kind of plastic you're gluing—PVC solvent for PVC plastic, and so on. Otherwise you can try an “all-purpose” or “universal” solvent cement, though these may not provide as strong of a joint. Most home improvement stores stock at least one type.

You don't need some fancy-schmancy solvent cement when working with expanded PVC sheets. The same cement made for PVC irrigation pipes can be used with expanded PVC. It's clear and has a good consistency for brush-on application. You can find it at any home improvement store.

When using a solvent-based cement to PVC or ABS plastic, either brush it on the surfaces to be bonded or squirt it into the joint by using a bottle applicator.

You must be sure that the surfaces at the joint of the two pieces are perfectly flat and that there are no voids where the cement may not make ample contact. After applying the cement, wait several minutes for the plastic to re-fuse and the joint to harden. Disturbing the joint before it has time to set will permanently weaken it.

Table 9-1 Plastic Bonding Guide

Plastic	Cemented to:		
	... itself, use	... other plastic, use	... metal or wood, use
ABS	ABS-ABS solvent	Rubber adhesive	Epoxy cement
Acrylic	Acrylic solvent	Epoxy cement	Contact cement
Polystyrene	Model glue	Epoxy	CA glue*
Polystyrene foam	White glue	Contact cement	Contact cement
Polyurethane	Rubber adhesive	Epoxy, contact cement	Contact cement
PVC	PVC-PVC solvent	PVC-ABS (to ABS)	Contact cement

* CA stands for cyanoacrylate ester, sometimes known as “Super Glue,” after a popular brand name.

APPLYING HOUSEHOLD ADHESIVE

Solvent cements work well when bonding together similar types of plastics, but they do little or nothing when trying to glue plastic to wood, metal, and other materials.

For these, you can try a household adhesive. As noted in Table 9-1, contact cement works well when bonding plastic pieces to metal or wood. You can get contact cement at the home improvement store. Two-part epoxy (you mix liquid from two tubes) is used for the same things, and is good when you need a very strong bond.

If you are joining pieces whose edges you cannot make flush, apply a thicker type of glue—contact cement, epoxy, and household glue are good contenders. You may find that you can achieve a better bond by first roughing up the joints to be mated. You can use coarse sandpaper or a file for this purpose.



Some plastics don't like to be glued. You can just about forget bonding nylon and high-density polyethylene (HDPE), unless you use an industrial cement that you can't get anyway because they're expensive and require special applicators. Instead, use mechanical fasteners to hold these pieces together.

Using Hot Glue with Plastics

Perhaps the fastest way to glue plastic pieces together is with hot glue. You heat up the glue in a glue gun, and when the glue is all melty, squeeze the trigger to spread it out over the area to be bonded.

Hot-melt glue and glue guns are available at most hardware, craft, and hobby stores in several different sizes. The glue is available in a “normal” and a low-temperature form. Low-temperature glue is generally better with most plastics because it avoids the “sagging” or softening of the plastic sometimes caused by the heat of the glue.

How to Paint Plastics

Sheet plastic is available in transparent or opaque colors, and this is the best way to add color to your robot projects. The colors are impregnated in the plastic and can't be scraped or sanded off. But you can also add a coat of paint to the plastic to add color or to make it opaque. Most all plastics accept brush or spray painting.

Spray painting is the preferred method for all jobs that don't require extra-fine detail. Carefully select the paint before you use it, and always apply a small amount to a scrap piece of plastic before painting the entire project. Some paints contain solvents that may soften and ruin the plastic.

Among the best all-around paints for plastics are the model and hobby spray cans made by Tamiya. These are specially formulated for styrene model plastic, but work with many other plastics, too. You can purchase this paint in a variety of colors.

Household Plastics for Bot Constructions

You need not purchase all the plastic for your robot at a hardware or specialty store. With a bit of digging, you might find some of the plastic you really need right in your own home. Here are a few good places to look:

- *Used compact discs (CDs)*. These are the denizens of the modern-day landfill. CDs, made from polycarbonate plastic, are usually just thrown away and not recycled. Exercise caution when working with CDs: they can *shatter* when you drill and cut them, and the pieces are very sharp and dangerous.
- *Used LaserVision discs*. These are “grown-up” versions of CDs. With the advent of DVD, 12-inch-diameter laser discs are now relics for the collectors. But you can sometimes find them online and at resale stores for just a dollar or two each. As with CDs, use care to avoid shattering the plastic.
- *Old phonograph records*. Found in local thrift stores, vinyl records can be used in much the same way as CDs and laser discs. Resale stores are your best bet for old records no one seems to want anymore (who is that Mantovani guy, anyway?). Note that some old records, like the *V-Discs* made during the 1940s, are collector’s items. Don’t wantonly destroy a record unless you’re sure it has no value!
- *Salad bowls, serving bowls, and plastic knickknacks*. They can all be revived as robot parts. I regularly prowl garage sales and thrift stores looking for such plastic material.
- *PVC irrigation pipe*. This can be used to construct the frame of a robot. Use the short lengths of pipe left over from a weekend project. You can secure the pieces with glue or hardware or use PVC connector pieces (Ts, “ells,” etc.)

Build a Motorized Plastic Platform

This chapter details construction of the PlastoBot, the base for a small but peppy robot that's constructed using 1/8"-thick plastic—most any plastic will do, so if you already have a piece in your garage feel free to use it. I built the prototype PlastoBot using 3mm (about 1/8") expanded PVC.

PlastoBot is basically a square, and making it involves only straight cuts. The corners of the plastic are chamfered—lopped off at a 45° angle—to enhance the looks and to prevent the base from snagging on things. You can build the robot with or without cutouts for the wheels. The cutouts can be tricky to make, but they allow the robot to retain its sleek design by keeping the wheels inside the profile of the base.

The PlastoBot as described here is 4" square, but the design is scalable. As desired, you can make it larger . . . or smaller, if you have teeny-tiny motors and wheels. The practical maximum is about 10" square. For any base larger than 5" or 6" you should double the thickness of the plastic—from 1/8" to 1/4".



I provide the exact model number for each of the pieces you need, but remember that you're free to substitute parts for others if you already have them or if you've found substitutes that are cheaper or easier to get.

Making the Base

Refer to Table 10-1 for a list of parts.

Figure 10-1 shows the completed PlastoBot. It measures 4" square and is balanced on one end with a small plastic ball caster. (If you don't want to use a ball caster, you can provide the bot with a static skid, using an 8-32" × 1" machine screw and acorn nut. See the end of Chapter 8 for a quick description of the idea.)

Table 10-1 PlastoBot Parts List

1	4" × 4" × 1/8"-thick plastic*
2	Pololu micro gear motors, 100:1 gear ratio, item 992†
1	Pololu mini motor brackets (pair), item 989†
1	Pololu wheels with rubber tires (pair), 32mm diameter, item 1087†
1	Pololu ball caster, 1/2" ball, item 952†
Misc.	Assembly hardware for these parts comes with the parts; if you substitute, you may need an assortment of 2-56 or 4-40 × 1/2" machine screws and nuts.

* Use expanded PVC, ABS, polycarbonate, acrylic, or most any other plastic. Acrylic and polycarbonate may be found at better-stocked home improvement stores; PVC and ABS plastic is available at specialty plastic outlets and online.

† Motors, motor mounts, wheels, and ball caster are available at Pololu.com, and many of its distributors. You may also use most any other miniature motors and wheels if you have another source for them. The motors are approximately 5/8" square by 1" long. The wheels are 32mm (about 1-1/4") in diameter.

While the plans call for 1/8" plastic, you may use another thickness if that's what you have. For reasons of weight, you'll want to avoid using a dense plastic (acrylic or polycarbonate) if it's over 3/16" or so. The thicker plastic is also harder to cut and drill.

Begin with a 4" by 4" square, and lop off the corners. How much material you remove is up to you, but a 1/2" chamfer is sufficient—to measure this amount, just make a set of marks 1/2" from each corner, and cut a 45° diagonal across the marks.

The PlastoBot design calls for rectangular cutouts for the wheels—what I refer to as *wheel wells*. For the wheels specified (see Table 10-1), the cutouts measure 1-3/8" by 1/2"; both

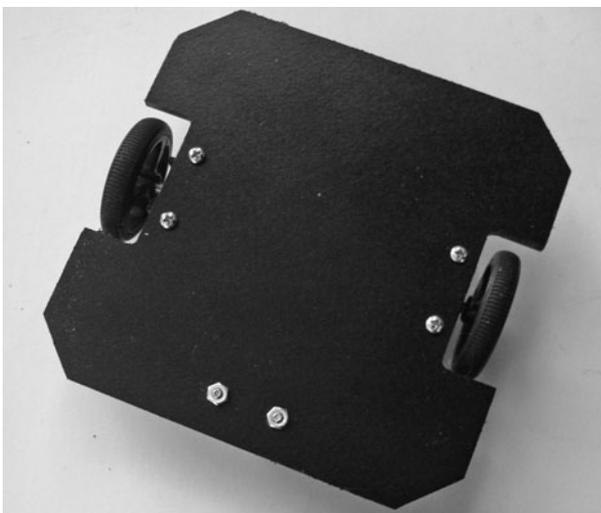


Figure 10-1 The finished PlastoBot, using two micro motors, wheels, and a small plastic ball caster for balance. The robot measures 4" square.

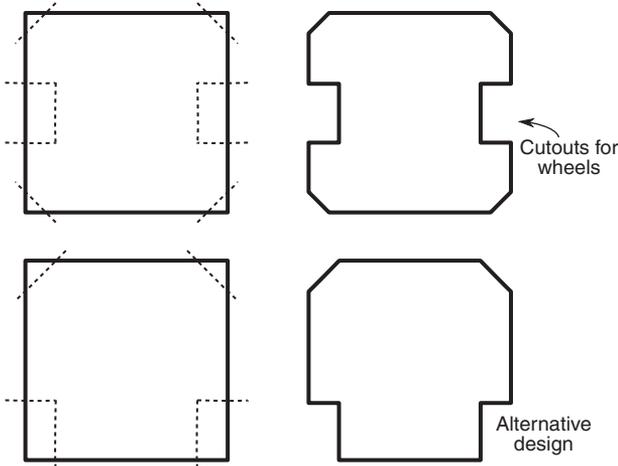


Figure 10-2 The base of the PlastoBot uses chamfered corners and cutouts for the wheels. You may elect to place the wheels centerline (preferred) or at one end of the base.

are placed centerline in the base. Refer to Figure 10-2 for a guide. You can see the benefit of the cutouts in Figure 10-3, which shows the differences in the profile of the robot, given wheels on the outside of the base versus wheels within the area of the base.

Also shown in Figure 10-2 is an alternative design for the PlastoBot, where the wheels are located at one end of the base. This style is a bit easier to make, because it doesn't require an elaborate internal cut for the wheel wells. As with the centerline wheel wells, make these cutouts 1-3/8" by 1/2".

Attaching the Motors

The PlastoBot uses a pair of micro-miniature motors, shown in Figure 10-4. These are highly precise motors that use metal gears, yet their price isn't much more than most other small motors for robotics.

The motors I selected are sold online through Pololu.com, as are the plastic mounts for attaching the motors to the base. Also provided by them are the wheels and the plastic ball caster described below. The same or similar motors and wheels are available at other robotics specialty stores; see Appendix B, "Internet Parts Sources" for more information. (In many

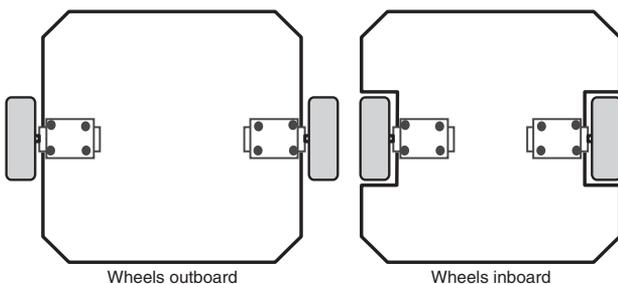


Figure 10-3 The effect of having wheels outboard of the base and inboard. When inboard, the shape of the robot is more streamlined, and it's less likely to snare on objects it encounters.

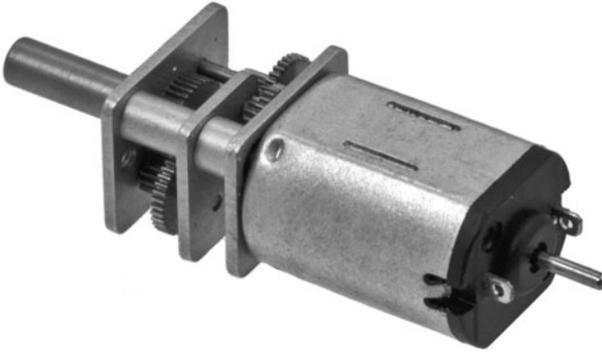


Figure 10-4 Metal gear micro motor. Two are used in the PlastoBot. (Photo courtesy Pololu.)

cases these other stores resell the Pololu products, and in other cases, they offer competing merchandise.)

The micro motors measure 0.94" by 0.39" by 0.47", and have a 3/8"-long 3mm D-shaped shaft that directly couples to the specified wheels. The motors are too small to have a mounting flange to directly attach them to the base, but brackets are available that make the job a cinch.

Follow the drilling guide in Figure 10-5. The spacing of the holes requires a modest amount of accuracy, so measure twice. The brackets have two holes spaced 18mm (about 11/16") apart, and they come with their own miniature 2-56" steel machine screws and nuts. The nut fits into a shaped recess inside the bracket; the screw comes up through the base to secure the motor in place (see Figure 10-6). For now just finger-tighten the screws.



Both the motor brackets and the ball caster use millimeter sizes. I've included the spacings in both millimeters and the closest fractional inch, accurate to 1/16".

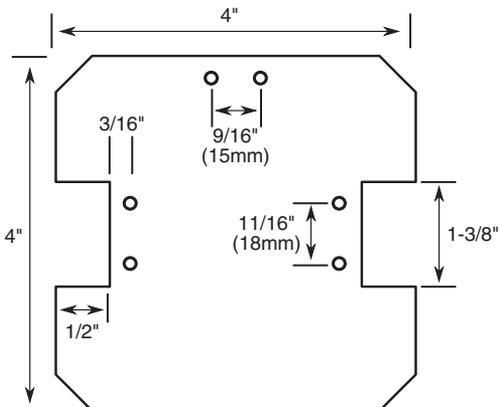


Figure 10-5 Drilling layout for the PlastoBot. The horizontal position of the two sets of holes for mounting the motors is not super critical, but the vertical spacing of the holes must be reasonably accurate. They must match the hole spacing of the motor mounts.

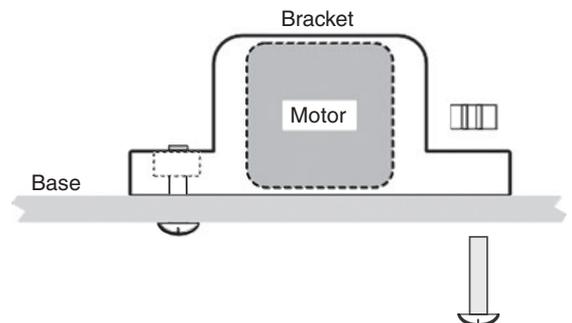


Figure 10-6 Mounting the motors using the specialty mounts. Miniature 2-56 fasteners come with the mounts and are inserted as shown.

Fitting the Wheels

PlastoBot's wheels are molded plastic with a removable rubber tire. The wheels are available in a variety of sizes, and I've picked one of the smallest, because of the diminutive stature of the robot. If you scale up the PlastoBot to bigger dimensions you'll probably want to opt for a larger wheel—the 80mm or 90mm wheels can be used for bases three and four times the size of PlastoBot.

To fit the wheels, merely press them into place over the motor shaft. The wheels have a D-shaped hub, matching the flatted shaft of the motors.

Once the wheels are in place, slide the motor into the bracket, adjust the position of each motor so the wheels are the same distance from the base, and tighten the two screws until the motor is snug.

Attaching the Ball Caster

The support caster uses a 1/2" plastic ball (see Figure 10-7), which smoothly rotates inside a cavity. Like the motor brackets, the ball caster comes with its own 2-56 fasteners. It also comes with several plastic spacers; use the thicker of the two spacers to increase the height of the caster to better match the diameter of the PlastoBot's wheels. The screws must be inserted so that the head fits into the body of the case. Position the nuts on the top side of the base.

Figure 10-8 shows the underside of the PlastoBot, with caster, motors, and wheels attached.

Using the PlastoBot

The PlastoBot uses high-efficiency micro motors that operate at between 3 and 9 volts, with 6 volts being nominal (normal). You can rig them up to switches to manually control the motor



Figure 10-7 Miniature ball caster, with 1/2" plastic ball. The ball rotates against small rollers inside the caster body. (Photo courtesy Pololu.)

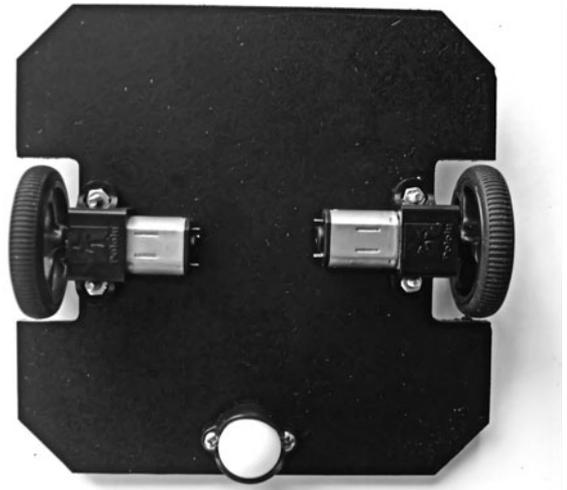


Figure 10-8 Underside of the PlastoBot, showing motors in their mounts and the ball caster.

(on/off and direction) or use electronic control. These topics are covered in Chapter 22, “Using DC Motors.”

These motors draw low current for the torque (turning power) they provide, allowing you to use most any kind of electronic control. When free-running (no load), the motors consume only 40 milliamps; at stall (the motors are physically prevented from turning), current rises to a still-respectable 360 milliamps. The low current demand of the motors lets you pick from a wide variety of motor drive circuits.

FYI

See Chapter 21, “Choosing the Right Motor,” for more on motor torque, milliamps, current measurements, voltage, and other motorific specifications.

If you’ve built the centerline wheels version of the PlastoBot, be mindful of maintaining a weight balance that slightly favors the end with the ball caster. This prevents the robot from tipping over on the end without the caster.

Altering the PlastoBot Design

Feel free to experiment with the design of the PlastoBot base. For example, by increasing the chamfer, you can make an octagonal base. Just cut off more of each corner.

You can also fashion your own motor mounts using standard fasteners and strips of flexible plastic. Figure 10-9 shows an octagonal PlastoBot base with holes for mounting motors up to about 1” wide. The idea for the mounts is shown in Figure 10-10: drill two holes just wider than the width of the motor. Using machine screws and nuts, attach a strip of flexible plastic to the top. Tighten the screw against the bottom nut to hold the motor in place.

Candidate plastic includes pieces stolen from construction toys, clamps for cable and wire management, and household odds and ends. The basic requirement (besides being the right

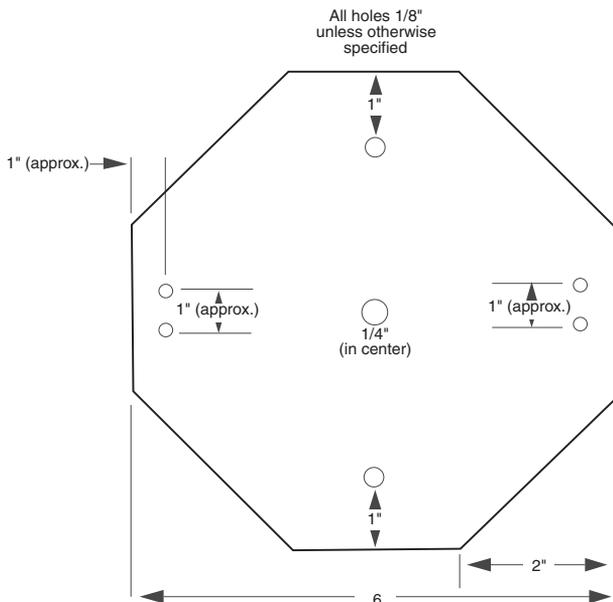


Figure 10-9 One of many alternative base designs for a robot made of plastic. A hexagonal base may be constructed by cutting off the four corners of a square.

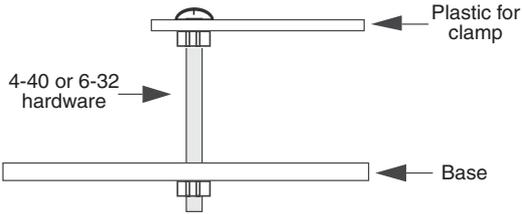


Figure 10-10 Ways to mount a motor using simple hardware and a flexible plastic strap. The technique works for motors with square or round cases.

approximate size and shape) is that the plastic be flexible. As you tighten the screw against the body of the motor, you want the plastic to deform, in order to hold the motor in place.

Another approach to homemade motor brackets is to use small cable management clamps. The Pololu micro motors used in the PlastoBot fit snugly in 1/2" clamps. You need two, positioned opposite one another to hold the motor in place. The clamp around the gearbox secures the motor in place, and the clamp around the body keeps the motor from shifting around on the base.



Working with Metal

Before the modern dependency on plastic, metal was the mainstay of the construction material world. Toys were not made of plastic, as they are today, but of tin.

Metal is a good material for building robots because it offers extra strength that other materials cannot. In this chapter you'll learn how to construct robots out of readily available metal stock, without resorting to welding or custom machining.

All about Metal for Robots

Metal is routinely broken into two broad categories: ferrous and nonferrous.

- *Ferrous* metals are made from iron (*Fe*, from which “ferrous” is derived, is the symbol for iron on the Periodic table of elements).
- Metals other than iron are *nonferrous*. This includes copper, tin, and aluminum.

When you buy a piece of metal, you're seldom buying the stuff in its pure form. Instead, metal is almost always processed with other metals. The resulting material is called an *alloy*. Different alloys provide different properties for the metal. For example, there are aluminum alloys specifically designed for casting, and others intended for machining parts.

ALUMINUM

Aluminum is the most common metal used in robot construction projects, partly because of cost and partly because it is strong yet lightweight. It's also one of the easier metals to cut and drill, and it requires only a modest assortment of tools. The aluminum you buy at the hardware store is actually an alloy; raw aluminum (which is manufactured from bauxite ore) has little commercial value as a finished metal. Rather, the alumina metal is alloyed with other metals.

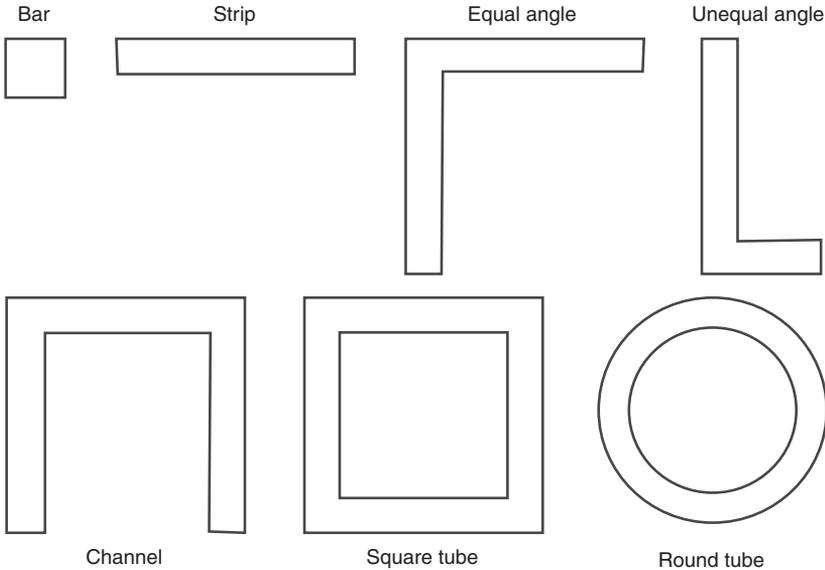


Figure 11-1 Extruded aluminum is available in a variety of handy shapes for building robots. Strips, angles, and channels are among the most useful.

Aluminum alloys are identified by number. A common aluminum alloy is 6061, which boasts good machinability (it's not difficult to drill or saw), yet is still lightweight and strong. Common forms of aluminum useful for robotics include thin sheets and extruded shapes, such as squares, tubes, angle Ls, and U-channel (Figure 11-1).

STEEL

Iron is used to make *steel*, which is further classified by the amount of carbon added to it during processing. The kind of steel you can routinely find at the local welder's shop or home improvement store has low amounts of carbon and is called *mild steel*.

Stainless steel is a special formulation of steel with small amounts of chromium added in. The chromium develops a microscopic film on the metal that helps it to resist rust and corrosion. Because stainless steel is very difficult to work with, is hard to weld, and is more expensive than most other metals, it's seldom used in amateur robots (exception: combat robots).

COPPER

Copper is available as just copper metal, but it's also often alloyed with other metals to make something else. When combined with zinc, copper makes *brass*; when combined with tin, copper makes *bronze*. All three are soft metals and are relatively easy to cut and drill.

Copper and its alloys can be readily *annealed*, which is the process of heating the metal to high (but not melting point) temperatures, then allowing it to cool very slowly. Annealing is used to change such properties as the softness of the metal. Annealing is commonly used to make metal springs and metallic spring strips.

ZINC AND TIN

Zinc and tin are often employed as an alloy ingredient or as a coating. *Tin*, which resists corrosion and inhibits rust, is also a common material in some crafts and home decorating. Tin

is a soft metal, and if the sheet is thin enough, it can be cut with a pair of large scissors. It's easy to make hold-down straps and other parts using tin sheets designed for "punch" crafts.

Measuring the Thickness of Metal

There are many ways to denote the thickness of metal. The most common are:

- *Fractional inch/millimeters.* Metal primarily sold to consumers may be specified using inches or metric units. Thickness is expressed in fractions, such as $3/64"$ or $1/16"$. When in metric, the thickness is in millimeters.
- *Decimal inch.* There's more precision in measuring thickness as a decimal value, down to the thousandths of an inch. A thickness of .032 means the metal is 0.032" thick—or "thirty-two thousandths."
- *Mil.* In this case "mil" doesn't mean millimeters or millionths or military, but a unit of measure equal to one thousandth of an inch. A decimal thickness of 0.032" is equivalent to 32 mils. By comparison, a plastic trash can liner is usually 2 to 4 mils.
- *Gauge.* This is a pseudo-standard used to specify the thickness of various materials, especially sheet metal and wire. Gauges of different types of materials aren't always the same (e.g., between plastic and metal), so you can't readily compare one to another. Purely as an example, 20-gauge aluminum is 0.032" thick.

Complicating matters: When used with metal sheet, gauge varies depending on the metal. That hunk of 20-gauge aluminum is equivalent to something between 21 and 22 gauge for steel, and 13 gauge for zinc.

You need a machinist's micrometer (Figure 11-2) to accurately measure the thickness of metal. Digital micrometers can be switched between decimal inch and other units of measure (typically millimeters), but less expensive models just have a mechanical scale, which can take a bit of getting used to.

They're not hard to operate, but you'll want to read the instructions that come with the tool for general guidelines on how to read the markings. When using micrometers marked in inches, the measurement will be in thousandths of an inch, .001". Most metric micrometers are marked in half-a-millimeter steps.

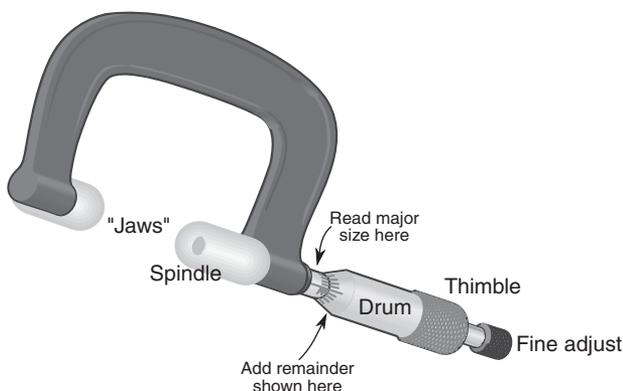


Figure 11-2 Use a micrometer to measure the size or thickness of small parts.

What's This about Heat Treatments?

When shopping for metal bits and pieces, you may come across references to how the material was heat treated. Heat treating is used to enhance certain physical properties of the metal. As heat treatments can affect the price of the metal, there's no reason to pay for something you don't need. So you can be an informed shopper, here are the most common treatments in a nutshell:

- *Hardening* strengthens the metal and literally makes it harder. The process also makes the metal more brittle. Tools are commonly made of hardened steel, and the hardening process makes it very difficult to cut or drill the metal.
- *Annealing* softens the metal and makes it more workable—a metal that is *ductile* or *malleable* is easier to work with, usually because it's been annealed. Copper is routinely annealed, but many other metals, like aluminum, can be annealed, too.
- *Tempering* removes some of the hardness and brittleness of steel and in doing so makes it even tougher. An example of a tempered aluminum alloy that you may encounter is 6061-T6, which has several times the strength of the same alloy untempered.
- *Case hardening* is a coating process for soft steels and allows relatively low-carbon steels, such as wrought iron, to be hardened. It's frequently used with steel to make tools.



Hardening and tempering can be accomplished in the home shop, useful for the die-hard combat robot enthusiast. The subject is beyond the scope of this book, but if you're interested in the concept, check your local library for a good tutorial on home metalworking.

Where to Get Metal for Robots

You'll find most metals for robot building at these local sources. If they don't provide the materials you need, try a Web search to locate mail-order suppliers of the metals you want:

Hardware and home improvement stores carry some aluminum and steel sheets, but look for angle brackets, rods, and other shapes. See the section "Metal from Your Home Improvement Store," below, for more details.

Hobby stores sell aluminum, brass, and copper, in small sheets, rods, tubes, and strips. A common brand sold by stores in North America is K&S Engineering. The metal is sold in small quantities, which makes it more expensive, but more convenient. Read more in "Metal from Craft and Hobby Stores."

Metal supply shops that cater to welders are usually open to the public and offer all kinds of useful metal. Many sell stock in large pieces, which you can have cut so you can get it home in your car. Tip: Check out the "remnant" bin for odds-and-ends sizes.

Restaurant supply stores, most of which are open to the public, sell many aluminum and steel materials. Look for spun bowls, cookie and baking sheets, unusually shaped utensils, strainers, and other items that you can adapt to your robot creations. Metal is metal.

Recap of Metals for Robotics

See Table 11-1 for a review of metals that are particularly well suited for the construction of robotics. Each metal is noted with its common use, main benefits, and principle drawbacks.

Table 11-1 Summary of Metals for Robot Building

Metal	Common Robotics Applications	Main Benefits	Main Drawbacks
Aluminum	Bases; arms; all structural parts	Reasonably priced; lightweight but strong; easy to cut and drill using proper tools	Plethora of alloys makes picking the right one difficult; can be hard to weld
Brass	Brackets and straps; structural parts for small robots; nonstructural parts for larger bots	Commonly available at hobby and craft retail stores	Relatively soft; low tensile strength
Copper	Brackets and straps; nonstructural parts	Easy to cut, bend, and shape; readily machined	Somewhat expensive; tarnishes easily
Steel	Heavy-duty frames	Very strong; inexpensive	Rusts if not protected; can be hard to drill and cut
Stainless steel	Heavy-duty frames	Resists rust and corrosion	Can be hard to cut and drill; more expensive than other metal choices
Tin*	Sheet metal bodies; easy-to-make parts like motor mounting straps	Soft and malleable; thin sheets ideal for robot bodies; relatively low melting point (450–725°F)	Can be hard to find locally; look online for “punch tin” sheets used for craft projects

* Some sources sell thin sheets of tin-plated steel and call it “tin.” Though not the same, the materials may be interchangeable depending on the application.

Metal from Your Home Improvement Store

Your local home improvement or hardware store is the best place to begin your search for metal. Depending on the store, here’s what you may find:

EXTRUDED ALUMINUM

Aluminum comes in all kinds of forms, including sheets, bars, rods, and something called *extrusions*. It’s the last form that’s really—and I mean *really*—useful in robot making. I started using it for my bots back in the ’80s, when most folks were still cutting out sheet metal to form robot bodies. Ugh! Extruded aluminum has now become the number one metal for making homebrew robots.

Extruded aluminum is made by pushing molten metal out of a shaped orifice, as those Play-Doh Fun Factory play sets do. As the metal cools it retains the exact shape of the orifice. Extruded aluminum generally comes in various lengths, from 12 inches to 12 feet. My local stores routinely sell it in 4-foot lengths. It’s often used in home improvement jobs to trim the edge of wood or tile and to make channels for shower doors.

There's lots of variety to choose from, allowing you to pick the extrusion to match your project. Common shapes include equal and unequal L-angles, U-channels, and flat bars.

I've found the following extrusions among the most useful in my robot building. Try these for starters, but don't be afraid to select other shapes and sizes for your projects. For easy comparison, several of the most popular extrusion variations are shown in Figure 11-3.

- *U-channel—equal and unequal sizes.* A popular size is 1/2" × 1/2" × 1/16" U-channel, which is sold as trim edge for 3/8" plywood.
- *Angle—equal and unequal.* The 1/2" × 1/2" × 1/16" (equal) angle stock is especially useful for use in robots up to about 15" in size.
- *Bar and rod.* Example: 3/4"-wide × 1/16"-thick flat bar.

TIE PLATES

Tie plates are designed to strengthen the joint of two or more pieces of lumber that are nailed together. The plates are made of steel but are *galvanized* to resist rust. The galvanizing gives the metal a noticeable mottled look.

Check for this stuff in the lumber area of your home improvement store. Much of what they'll offer is preformed shaped for specific jobs, such as attaching 2-by-4 lumber to a ceiling beam. What you want is the flat plates (or flat plates with a flanged edge; those are useful, too), available in several widths and lengths. Common widths are 3" to 5" and lengths from 3" to 7". See Figure 11-4 for a size comparison of popular 3" × 5", 3" × 7", and 3" × 9" plates.

You can use the plates as is or cut them to size. The plates have numerous predrilled holes to help you hammer in nails, but the metal is thin enough (usually 20 gauge or so) that you can drill new holes. Just be sure to use sharp drill bits.

In North America the most common brand of tie plate is the Simpson Strong-Tie. Visit the company's Web site (www.strongtie.com) for sizes and specifications of their various prod-

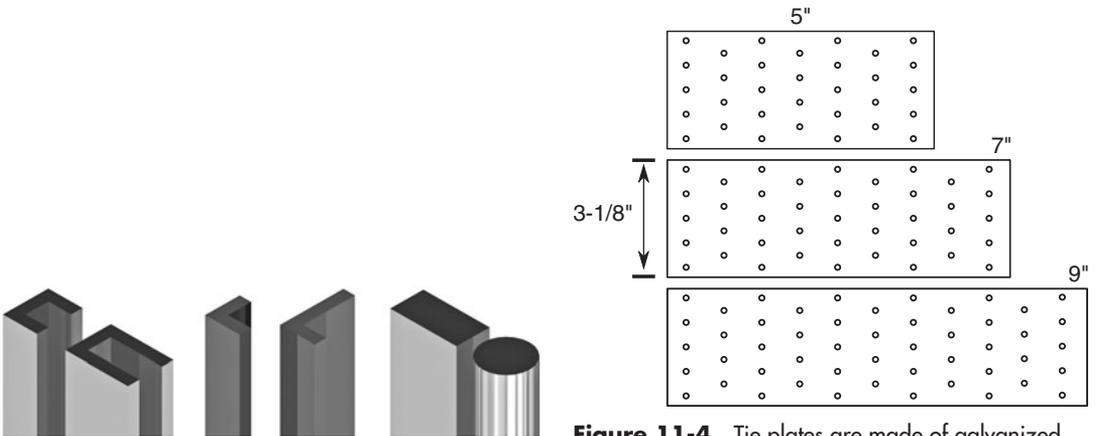


Figure 11-3 Look for extruded aluminum at your neighborhood hardware or home improvement store. It comes in various lengths and may be cut to size using a hacksaw.

Figure 11-4 Tie plates are made of galvanized steel and are useful as base or construction metal for your robots. They come in different sizes; avoid unnecessary cutting by matching the size to best fit your intended use.

ucts. In Chapter 12 you'll learn how to use some of these common and inexpensive tie plates to make a sturdy metal robotic base.

STEEL TUBES AND ANGLES

Most hardware stores carry a limited quantity of extruded round and square tubes, as well as angles and other shapes. These are solid and fairly heavy items made with 14-gauge steel. Use them when you need lots and lots of support. Many are predrilled with holes along the sides, saving you the effort. You still have to cut them, though. For cutting, use a hacksaw with a new fine-toothed blade, as detailed later in this chapter.

EMT PIPE

EMT means electrical metallic tubing, often referred to as *metal conduit* or *thinwall*. It's used to enclose electrical wiring for a building—and sometimes to make robots. You'll find it in the electrical section of the hardware or home improvement store.

Conduit comes in a variety of sizes, starting with 1/4 and going up. The 1/2 and 3/4 sizes are more common. Electrical conduit pipe can be used to make unusual robot frames. You can bend it with a conduit-bending tool.



The sizes of EMT conduit is not in fractional inches, though it's often listed that way in catalogs and stores. Depending on the metal used, EMT 1/2 (the 1/2 is called the *trade size*) is actually 0.622" on the inside and 0.706" on the outside. When shopping, bring along a tape measure so you can be sure you're getting the size you need.

In addition to the EMT pipe itself, other conduit-related materials are useful in robot construction. Topping the list is the conduit hanger and clamp, like that in Figure 11-5. These make good mounts for things like motors. You can almost always find a conduit hanger or clamp that will fit a round-shape motor—give it a bit of a squeeze or stretch if the size isn't perfect right off the bat.



Figure 11-5 EMT pipe clamps make for useful motor mounts on your robots. They come in plastic or metal, in various sizes and styles.

Metal from Craft and Hobby Stores

Hobby and craft stores are another handy outpost for your metal needs. Sheets of stainless steel, copper, and even bronze are common. Some of the metal products are meant to be used for decorative purposes, so they're very thin and not as useful as structural parts in a robot. These are more like thick aluminum foil than metal sheets. But other, heavier pieces (22 gauge and above) are suitable for making bases and frames.

Besides sheet metal there are metal strips, bars, and tubes, both round and square. The tubes are sized so that they "telescope"—they fit within one another. You might

use these to make ersatz shaft couplers, for example, to match a small shaft to a larger wheel hole. In fact, Chapter 24, “Mounting Motors and Wheels,” discusses this very topic.

The Metalsmith’s Art

One reason to use metal for your robot is that it’s stronger. It also looks pretty cool—though to be fair, you can get plastic that looks like metal, but that’s beside the point. Another key benefit is the longevity of a metal or frame; it simply lasts longer. Even if your robot falls off the table or gets mauled by the dog, at least its body will remain intact.

But while metal provides a resiliency that wood and plastic cannot match, it’s harder to work, costs more, and weighs more. Choices, choices.

As first discussed in Chapter 6, “Mechanical Construction Techniques,” to cut metal you should use a hacksaw outfitted with a fine-tooth blade—24 or 32 teeth per inch. Coping saws, keyhole saws, and other handsaws are generally engineered for cutting wood, and their blades aren’t fine enough for metal work.

You’ll probably do most of your cutting by hand. When cutting pieces for a frame you’ll want a miter box. You don’t need anything fancy. Be sure to get a miter box that lets you cut at 45° both vertically and horizontally. Firmly attach the miter box to your workbench using hardware or a large clamp.

You’ll always have better-than-average results if you use sharpened, well-made tools. Dull, bargain-basement tools can’t effectively cut through aluminum or steel stock. Instead of the tool doing most of the work, *you* do. That’s no fun, and robotics is supposed to be fun.



Cutting, drilling, and finishing metal are hazardous! Be sure to always wear eye and ear protection. Don’t disengage the safety device on any tool. Follow all manufacturer instructions. When using power tools be sure to hold smaller pieces in a vise or clamp while you work with them. **DO NOT** ever hold the work with your bare hands!

CUTTING A BASE

Use sheet metal to make robot bases. Mild steel and aluminum are the most common for use in bots, and they are relatively inexpensive. For both, select a thickness that will support the size and weight of the robot, but without adding undue weight of its own. Consider the following starting points in your design.

For small robots, under 8”:

- Aluminum: 1/32” to 1/16” (0.03125” to 0.0625”)
- Steel: 22–20 gauge

For medium robots, 9” to 14”:

- Aluminum: 1/16” to 1/8” (0.0625” to 0.125”)
- Steel: 20–18 gauge

For large robots, 14” and over:

- Aluminum: 1/8” to 1/4” (0.125” to 0.250”)
- Steel: 18–16 gauge

Cutting sheet metal by hand is a chore, but it can be done with a bit of work. These are some specialty hand tools for cutting sheet metal:

Aviation snips are suitable for aluminum under 1/8" (0.125") thick, or 18-gauge thickness for steel. When cutting out circular bases, consider using a right- or left-hand tool; it'll make better cuts. "Right-hand" and "left-hand" in this case refer to the direction of the cut, not whether you are right- or left-handed.

A nibbler tool takes out little bites to cut and shape the metal.

Pneumatic shears make quick work of cutting sheet metal. I don't know what I'd do without mine! The shears themselves are relatively inexpensive (most under \$40), but you need a suitable air compressor to power them. An alternative is all-electric sheet metal shears, but these cost more. With both, the maximum practical thickness is 18-gauge steel.

The basic steps are the same whether you use snips, nibbler, or shears:

1. Use a scribe or construction pencil (anything that writes on metal) to mark the cut you want to make.
2. Hold the metal sheet in one hand—use a work glove if you need to—and the cutting tool in the other, or clamp the metal in a vise.
3. Cut along the marked path. If you need to make sharp turns, you may have to cut (or nibble) away from the scrap portion of the work, then approach the path from another angle.
4. Smooth the edges with a metal file or fine-grit sandpaper, as detailed later in this chapter.

Figure 11-6 shows an example of galvanized 20-gauge tie plate being cut with an air shear. Tie plates are very inexpensive (this one cost under a dollar), and they're commonly used in home construction.

Cutting Thick Sheet Metal

For thicker stocks, specialized metalworking tools are needed. On large pieces, a metal brake is used to make straight cuts. You probably don't own one of these, but if you attend school, ask the shop teacher if you can use the school's for a few minutes.

Very thick steel (1/4" or more) can be cut with a torch, then ground down as needed with an electric grinder. Obviously, this requires a cutting torch and the experience that goes with using it.

Cutting Thin Sheet Metal

On the other side of the spectrum, the thinner aluminum stocks (1/32" and under) can be cut to shape using aviation snips (see above) or even a pair of heavy-duty scissors. You can also use a scroll saw, as long as it is a variable-speed model (use the slower speeds) and you've equipped it with a metal cutting blade.



Figure 11-6 Air shears make quick work out of cutting sheet metal. Use a pencil to mark the line you want to cut, then slowly work the tool around the contours of the line.

CUTTING A FRAME

Robot frames can be constructed using aluminum extrusions, described above in "Metal from Your Home

Improvement Store.” These pieces are meant to be used for such things as trim for a shower door in the bathroom, but it won’t mind if it’s used for making a robot.

- *U-channel (equal and unequal)*. U-channel is available with equal and unequal dimensions. Pick the style best suited for your project. It’s available in thicknesses starting at 1/32”.
- *Angle (equal and unequal)*. Available with equal and unequal sides, frames can be constructed using standard bracket hardware, pop rivets, and machine screw fasteners. Thicknesses start at 1/32”.
- *Flat bar*. Available in different widths and thicknesses, use this stock to make your own brackets and support columns. It’s also useful as a reinforcement strap.
- *Tubing*. Available in square, rectangle, or round shapes. These tend to be the most expensive, and are harder to build frames from.

Sources for these structural shapes include:

Larger dimensions: Your nearby hardware and home improvement store is likely to have a good selection. Online is another choice if your local store is out of stock.

Smaller dimensions: Hobby and craft stores are your best bet. Some hardware and home improvement stores also carry a limited assortment of the smaller pieces. As noted earlier in the chapter, a popular maker of metal structural components is K&S Engineering. Check their Web site at www.ksmetals.com for a description of their products.

Using a Backsaw and Miter Box

The process of cutting a frame from aluminum (or other) stock is the same as it is when making a frame out of wood. So instead of repeating those steps here, please refer to Chapter 7, “Working with Wood,” for the lowdown details.

A small, lightweight but sturdy frame can be constructed using 1/2” × 1/2” × 1/16” U-channel stock, cut to length with miters. The 1/2” dimensions are for the outside of the stock; inside it’s 3/8”, so you can connect the pieces using 3/8”-wide L angle brackets (see Figure 11-7), also available at your hardware store.



Figure 11-7 Solid metal frames can be constructed out of U-shaped aluminum channel, 3/8” wide L angle brackets, and metal fasteners. Cut the aluminum channel with a miter box and hacksaw.

To assemble, use 4-40 machine screws and nuts. The screws can be 3/8” or 1/2” in length. You need only one screw per “leg” of the angle bracket—this makes it easier for you to assemble the frame and saves on weight. In a pinch, you can use 6-32 machine screws and nuts, but these are bigger and heavier.

Creating a Box Frame

Box frames are three-dimensional bodies for your robots. They can be constructed using two (or more) square frames, anchored together with metal or plastic “pillars,” as shown in Figure 11-8.

For good strength but less weight, I like to use 6mm PVC for the pillars. These are cut to about 3” wide. Then,

1. Drill a pair of holes near both the top and the bottom of the pillar. Space the holes no closer than 1/4” from the top (and bottom), and no closer than 1/2” from the sides.

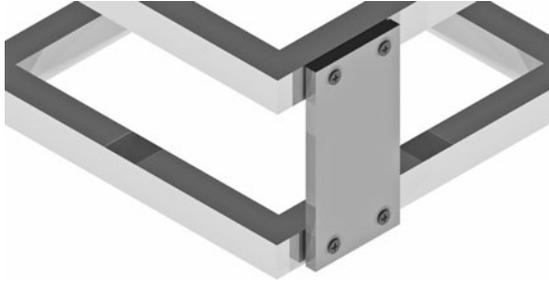


Figure 11-8 By using pillars (also called risers or columns), you can literally stack frames together to make boxlike body shapes for your robots.

2. Drill corresponding holes at the approximate center of each side of the frame. (See “Drilling Metal,” next section.)
3. Assemble, using 4-40 × 1/2” machine screws and nuts. Use flat washers on the screw-head side.

When building large frames—say, over 14” square—use two pillars per side; that means eight total. Position each pillar about 1” to 2” from the corners of each frame. This gives you room in the middle of the box to work.

DRILLING METAL

Metal requires a slow drilling speed. That means, when using a drill, you should set it to no more than about 25 percent of full speed. Variable-speed power drills are available for under \$30 these days, and they’re a good investment. Use only sharp drill bits. If your bits are dull, replace them or have them sharpened. Quite often, buying a new set is cheaper than professional resharpening. It’s up to you.

Punching a Starter Hole

You’ll find that when you cut metal, the bit will skate over the surface until the hole is started. You can eliminate this skating by using a center punch tool prior to drilling. (In a pinch, a nail works, too.) Use a hammer to gently tap a small indentation into the metal with the punch.

Dabbing On Some Oil

When drilling aluminum thicker than 1/16”, or most any thickness of steel, first add a drop of oil over the spot for the hole. If you’re drilling a very thick piece, you may need to stop periodically and add more oil. There’s no rule of thumb, but one drop per 1/16” or 3/32” of thickness seems about right. The purpose of the oil is to keep the bit cool, which makes it last longer.

Using a Drill Press

When it comes to working with metal, particularly channel and pipe stock, a drill press is a godsend. It improves accuracy, and you’ll find the work goes much faster.

Always use a proper vise when working with a drill press. *Never* hold the work with your hands. Especially with metal, the bit can snag as it’s drilling and yank the piece out of your hands. For best results the vise should be physically attached to the base of the drill press. Anchoring the vise may not be critical if it’s heavy, and it will hold the pieces you drill in place just by its sheer weight.

If you can’t place the work in the vise, use a pair of Vise-Grips or other suitable locking pliers. The pliers allow you to adjust the jaw size and “lock down” on the material for a sure

hold. Once locked, you can hold the pliers normally, without continually exerting pressure. To unlock, squeeze or pull the release lever.

If using a drill press, your model may require you to change the position of belts in order to alter the speed. Set the speed to the lowest available.

Tapping Holes

When using thicker metal (1/16" or more), you can tap the holes you drill so they have threads for machine screws. To use a tap you must first drill the appropriate-size hole. You then manually "screw" the tap into the hole; this conforms the hole with the threads needed for the screw.

You don't need a fancy 10-in-1 tap-and-die set. You can get by with just a couple of tap sizes, along with their corresponding drill bits. I use 4-40 and 6-32 screws almost exclusively in my robot designs, so I really only need the 4-40 and 6-32 taps. If you require holes for other screw sizes, you can purchase taps for these on an as-needed basis.

In order to thread the tap into the hole, you need a *T-wrench* for the tap (see Figure 11-9). The tap is secured by a chuck, as shown, or a setscrew in the wrench. Follow these basic steps to tap holes for threading with machine screws:

1. Drill the hole with the appropriate-size bit. The hole is always smaller than the tap. For example, to tap a hole for a 4-40 screw, use a 3/32 (or #43) drill bit.
2. Add a drop of oil at the location of the hole, and proceed to drill.
3. After drilling is complete, check the hole for burrs and remove them as detailed in the next section, "Finishing Metal."
4. Mount the tap into the T-wrench. Add a few drops of cutting oil to the entire length of the tap.
5. Keeping the tap as perpendicular to the surface of the metal as possible, slowly thread the tap into the hole. Apply steady but firm downward pressure on the wrench. You should soon feel the tap "bite" into the metal.
6. Continue threading the tap into the metal until about 1/4" of the tap is fully through the other side.
7. Clean off any metal debris, add a drop or two more of cutting oil, and carefully back the tap out of the hole. Clean the newly threaded hole when done.

FYI

See Appendix C, "Mechanical Reference," for a handy chart comparing tap sizes and appropriate drill bits, for both imperial (inch) and metric.



Figure 11-9 A T-wrench and tap are used to thread holes for mechanical fasteners. Be sure to drill the hole to the proper size for the tap you are using.

BENDING METAL

Most metals can be bent to another shape. Sheet metal is typically transformed into curves and angles using a tool called a metal brake. Hollow pipe, such as metal conduit for electrical wiring, is bent using (get this!) a pipe bender.

Not all kinds of metals are as easily bent, at least not without specialty tools. The thickness of the metal and its overall size, shape, and tempering qualities greatly affect how easy (or hard) it is to bend. Strips of softer metal like copper and brass bend easily. The thinnest pieces can be bent by hand; otherwise, you can use a bench vise and rubber mallet to bang it into shape.

Mild steel up to about 1/16" thick that has not been heat treated (tempered) to make it harder can be bent using a shop vise and brute force. But you get better results with a metal brake.



Tube bending, especially thin-wall aluminum or brass, requires either skill or a special tool to keep the tube from collapsing. The idea is to prevent a sharp kink at the bend. Benders are available for different sizes of tube. Smaller tubing available at hobby stores can be worked using a tightly wound spring. With practice, you can also bend lightweight tubing by first filling it with sand.

Bending causes stress in the metal, which can weaken it. Some aluminum alloys are engineered to be bent, but most of the stuff you find at hardware and home improvement stores has been hardened. Bending it to more than 20° or 30° can seriously degrade the structure of the metal. You especially don't want to bend it in a 90° angle, then back out straight, as the metal will likely just snap apart.

Metals that have been annealed using a heat treatment are easier to bend. The annealing makes the metal softer and more ductile. If you need to bend aluminum (up to about 1/8" thick), find an alloy that has been pretreated using an annealing process. Or you can anneal the piece yourself if you have a gas welding torch. I'm not going to get into the steps here (check metalworking books at your local library), but in general you heat up the metal to about 700°F, then allow it to cool slowly back to room temperature. Once cooled, the spot that was annealed is now softer, and it can be more readily bent to shape.

FINISHING METAL

Cutting and drilling often leave rough edges, called *flashing* and *burrs*, in the metal. These edges should be finished using a metal file or very fine sandpaper. Otherwise, the pieces may not fit together properly, and the rough edges can scratch skin and snag on carpet. Aluminum flash comes off quickly and easily; you need to work a little harder when removing the flash from steel or zinc stock.



If there is a lot of material to remove, use a small grinding wheel attached to a drill motor or hobby tool, such as the Dremel. Use the standard Dremel mandrel. It's composed of two pieces: a shaft and a screw. Then purchase an aluminum oxide grinding wheel in the shape you need. Example: Dremel item 541 is a set of two 7/8"-diameter flat wheels made to be mounted with the mandrel.

Using Metal Files

A metal file is the same as any other file, excepts its teeth—the part that removes the material—are finer. You should always use a metal file for metal; never use a wood file,

because the teeth on a wood file are far too coarse. Unless you plan on doing lots and lots of metalworking, you can purchase an inexpensive set with a variety of files in it or get just what you need. I'd start with just one or both of the following:

- For aluminum, get a single-cut mill, half-flat, half-round file; that is, on one side the file is flat, and on the other it's rounded. You don't need a handle on the file unless you want one.
- For steel, get the same, except make it double-cut.

The number of teeth per inch defines the smoothness of the finished work. For general-purpose deflashing and deburring, opt for files with 30 to 40 teeth per inch. This equates roughly to "bastard" and "second cut" files, if the tools you're looking at are marked that way instead.

For small pieces, you might want to invest in a set of needle files, so-called because they're small like knitting needles. The set has different sizes and shapes.



Most files have teeth that face forward, away from the handle. So they do their work as you push it into the material, not as you draw it back. Use this fact to make your work easier. Don't try to use the file like you're cutting a loaf of bread with a knife. Make even strokes, and bear down only on the forward stroke.

Using Sandpaper

Use sandpaper if you want the smoothest edge possible. This is not just for looks, but for function, too: the surfaces need to be like glass on pieces that slide against one another.

You need aluminum oxide or silicon carbide sandpapers for working with metal. For general deburring and cleaning, use a fine or medium aluminum oxide paper; for finishing/polishing, use a fine emery cloth (doesn't use a paper backing), and dip it in water as you work. The higher the grit, the finer the finish.

Grit Key	Name	Grit
M	Medium	80-100
F	Fine	120-150

For a final smooth finish, buff the metal using 00 or 000 steel wool. You can get this at a hardware or home improvement store. Look in the paint section. Which brings us to . . .

Painting Metal

Bases and frames of mild steel should be painted, to prevent rust. No painting or other treatment is needed if the aluminum is already anodized—it'll have a silvered, black, or colored satin appearance. For bare aluminum, the metal can be left as is, but you may prefer to paint it. There are several alternatives for painting aluminum and steel:

- *Brush painting* is simple, but brush marks are hard to avoid in metal. And the paint tends to be applied too thickly.
- *Spray painting* with a paint designed for metal yields better results, but only if you apply the paint in *several light coats*. Apply a primer coat first. Enamel paints work the best. Paints for automobile engines are especially durable. Let dry *completely* before handling.

- *Powder coating* is a process that applies a layer of paint to the metal using an electrostatic charge. The pigment penetrates the surface layer of the metal for a more durable finish. Powder coating can be performed at home if you have the right gear, but this is usually something for specialty paint shops.

In all cases, before painting it is extremely important that the metal be completely clean and free of grease, oils (including skin oil), and dirt. Use denatured alcohol to clean the metal, including all the nooks and crannies.

Build a Motorized Metal Platform

Metal offers strength and ruggedness that wood and plastic cannot. While constructing robots out of metal requires more effort (and sharper tools for cutting and drilling), a properly made metallic bot will last years and years and is suitable for both indoor and outdoor use.

This chapter details the construction of a small but flexible TinBot, using metal for its principal body parts. Its design uses commonly found hardware, which keeps down its cost and limits the amount of hacking pieces of metal down to the size and shape you want. In fact, construction of the TinBot requires only two cuts through thin-gauge steel. The cuts are relatively easy to make using an ordinary hacksaw.



Before you build the TinBot, review Chapter 23 on how to use radio control (R/C) servo motors with your robots. This type of motor *cannot* be simply attached to a battery to make the bot operate.

R/C motors require a special control signal. In robotics the most common way of providing this control signal is with a microcontroller. These specialty electronic circuits are designed in-depth in Part 7 of this book.

Making the Base

Refer to Table 12-1 for a list of parts.

Figure 12-1 shows the completed TinBot, which measures 7" × 6-1/4", and stands about 2-1/4" high. It's powered by two radio-controlled servo motors. Ordinarily, these kinds of motors operate in a confined arc of about 90° to 180°. But the motors on the TinBot have been modified for continuous rotation, so they function like ordinary gear motors.

You can either modify the motors yourself, following instructions provided in Chapter 23, "Using Servo Motors," or buy them already modified. You can find the already modified ones

Table 12-1 TinBot Parts List

1	Simpson Strong-tie TP35 nail plate (measures 3-1/8" × 5")*
1	Simpson Strong-tie TP37 nail plate (measures 3-1/8" × 7")*
2	Simpson Strong-tie LSTA9 straps (measures 1-1/4" × 9")*
2	Standard-size R/C servo motors, modified for continuous rotation†
2	2-1/2"- or 2-5/8"-diameter wheels, with hubs for attaching to the servo motor†
2	Plastic servo motor mounts‡
4	Metal corner angle ("L") brackets, approximately 3/4" × 1/2" wide§
1	1-1/4" swivel caster¶
1	1/4" plate for swivel caster
6	6-32 × 3/8" machine screws
2	6-32 × 1/2" machine screws
8	6-32 nuts
14	#6 washers
16	4-40 × 1/2" machine screws and nuts
8	#4 washers

* These items are commonly available at most hardware and home improvement stores.

† Servo motors may be found at any hobby store selling radio control parts. You will need to modify the motor for continuous rotation, or purchase the motor already modified for continuous. The wheels must have hubs that match the spline type of the servo motor, either Futaba or Hitec. See Chapter 23 for details.

‡ Directions for making these is found in this chapter. Or you may use most any ready-made plastic or metal servo mounting bracket. These are available from specialty robotics Web sites; see Appendix B.

§ First choice is a Keystone model #619 tin-plated bracket (available through a variety of online sources, such as Mouser and Allied; see Appendix B). Each "leg" measures 0.687" × 0.687" and the bracket is 3/8" wide. Or you may substitute for the slightly larger 3/4" × 1/2" wide corner angle bracket, commonly found at hardware and home improvement stores.

¶ As necessary you can substitute the 1-1/4" caster for a 1-1/2" model.

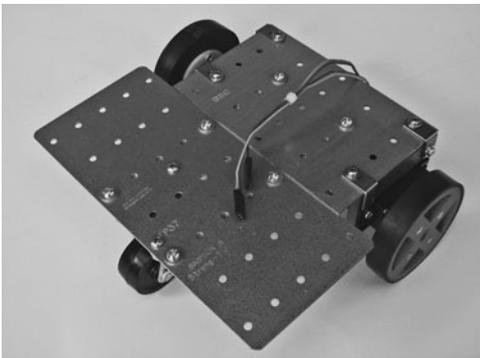


Figure 12-1 The completed TinBot, made with common metal plates found at the hardware or home improvement store.

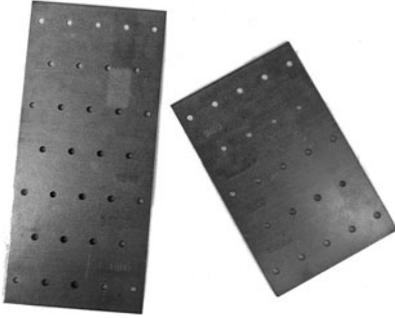


Figure 12-2 Simpson Strong-tie TP35 and TP37 steel nail plates used for the body of the robot. Note the holes already drilled into the plates.

at many online specialty robotics Web sites, such as Parallax and Pololu. Cost is only slightly more than the typical standard servo motor, but they save you the hassles of modifying the servo yourself.

TinBot's metal body consists of 20-gauge galvanized steel *nail plates* (see Figure 12-2), intended for home construction and repair. The Simpson Strong-tie hardware I've listed is a common find at many home improvement stores, as well as online hardware retailers. If this exact brand is not available, you can substitute most anything else that matches the approximate dimensions.

PREPARING THE METAL PIECES

As noted, the only cutting you need to do is to strip off a portion from the two LSTA9 straps. Using a hacksaw outfitted with a metal-cutting blade, cut the straps to a length of 5-1/2". Use a miter box (see Figure 12-3) to ensure a straight cut. Plastic miter boxes are inexpensive and are very handy for all kinds of robot construction chores. The cuts don't need to be perfect, but file down any rough edges that may remain. Figure 12-4 shows the straps before and after cutting.

With the file still in your hands, use it to round off the edges of the TP37 plate, to remove the sharp corners.



If you have a pair of metal shears, either hand or power operated, these will make the chore of cutting the straps even easier. For best results, first use a hacksaw to score a shallow cut in the metal, then cut with the shears. Apply a dab of oil on the cutting surfaces to prolong the life of the tool.

DRILLING HOLES

Though the nailing plates and straps have holes in them, they don't always match up where we want them to. Using some holes already in the plates, matching holes are drilled into the

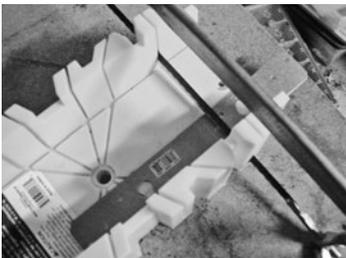


Figure 12-3 You need to cut off a few inches from the nail straps. Use a hacksaw (with a sharp blade for metalwork) and, for best results, a miter box.

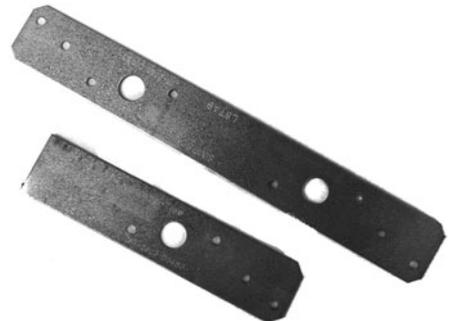


Figure 12-4 Original and shortened nail strap pieces. After cutting, file off the metal for a smooth edge.

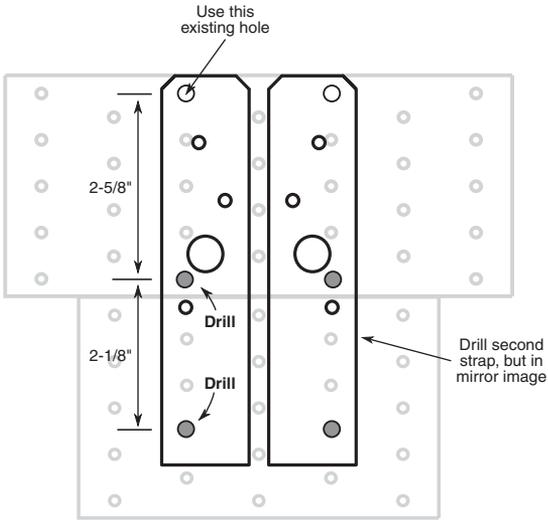


Figure 12-5 New holes need to be drilled into the nail straps to conform to the holes already in the TP35 and TP37 plates. For each strap you need to drill two new holes.

straps, as shown in Figure 12-5. The TinBot design uses one of the strap holes to get things started; you need to drill two additional holes in each strap.

Use a new or recently sharpened 9/64" bit to drill the holes. As covered in Chapter 11, start each hole using a center punch or a sharp nail to make a slight indentation in the metal. Apply a small drop of cutting oil, then set your drill motor to low. Allow the tool to do the work for you. Don't press down too hard, or you'll dull the bit.

ASSEMBLING THE PLATES AND STRAPS

Assemble the metal plates and straps as shown in Figure 12-6. Use 6-32 hardware, with washers top and bottom. Finger-tighten only until all six fasteners have been assembled. Then go back and retighten everything.

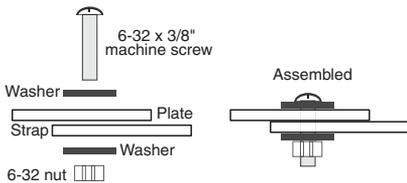
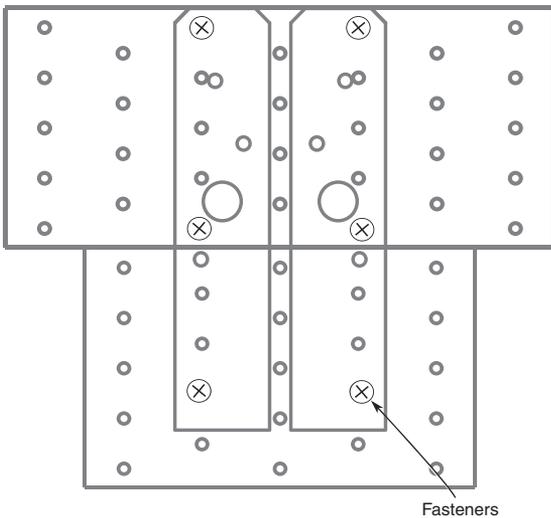


Figure 12-6 Use 6-32 fasteners to assemble the plates to the straps.

MOUNTING THE SERVO MOTORS

Refer to Figure 12-7 for a layout diagram for the servo mounts. You need two of these. They may be constructed out of 1/4" aircraft-grade plywood or expanded PVC, or 1/8" polycarbonate or acrylic plastic (plywood and PVC are much easier to work with). The dotted lines along the bottom of the mount indicate optional "cut-throughs." Cut through at the dotted lines if you have trouble cutting out the inside pocket for the servo. This creates a slightly weaker mount, but they're easier to make.

You may also purchase ready-made servo plastic or metal mounts, though you may need to drill new holes to match the hole pattern in the nail plates. The hole spacings in the layout are designed to match the TP35 nail plate used on the TinBot.

The servo mounts are attached to the robot base using metal corner angle brackets, as shown in Figure 12-8. Assemble using 4-40 hardware. Be sure to tighten the screws securely, or they'll work loose over time. The brackets should be about 3/4" on each side of the angle.

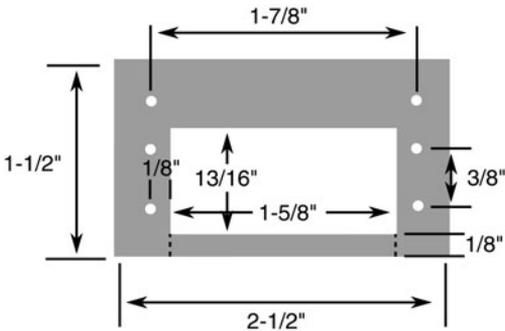


Figure 12-7 Cutting and drilling guide for the servo motor mounts. The dotted lines along the bottom indicate optional cuts, for making it easier to create the inside “pocket” for the body of the servo motor.

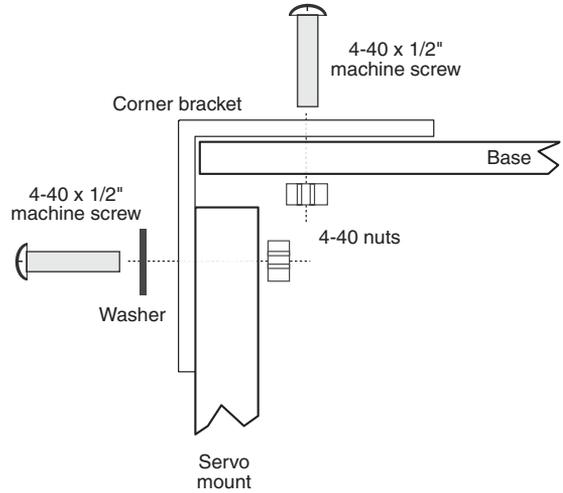


Figure 12-8 Assembly detail for attaching the servo motors to the TinBot. Use 3/4” metal corner angle brackets and 4-40 fasteners.

Alternatively, you can do away with the servo mounts completely by attaching the servos directly to the corner angle brackets. This will require that you drill new holes in the TP35 plate to match the spacing of the mounting holes in the servo.

It also reduces the height of the robot by about 1/2”; this is not a problem per se, but it means the size of the caster must likewise be reduced to match. While there are 3/4” and 1” swivel casters available, they tend to be cheaply made and don’t turn well. You may want to swap the swivel caster for a ball caster, like the one used in the PlyBot in Chapter 8. These are more expensive, though.

MOUNTING THE CASTER

You also need to drill two holes for mounting the caster, which is used to balance the robot. The exact placement of the holes will depend on the exact caster you use. To mark the holes, place the baseplate of the caster against the metal, positioning it as shown in Figure 12-9. The caster is mounted right at the back edge of the TinBot.

Mount the caster using 6-32 hardware (see Figure 12-10). Note the spacer between the caster baseplate and the body of the robot. You can construct the spacer using a scrap piece of 1/4” plywood or PVC plastic, or you can simply use a stack of three or four #6 washers. The spacer/washers increase the height of the 1-1/4” caster to better match the drive motors and wheels. The added spacer is not needed if using a 1-1/2” caster.

ATTACHING THE SERVO MOTORS AND WHEELS

As a final step, attach the servo motors into the servo mounts using 4-40 machine screws and nuts, then secure the wheels to the motors (a screw for mounting the wheels comes with the servos). See Figure 12-11 for an underside view of the TinBot.

Though standard-size R/C servos have four mounting holes, you need only use fasteners in two, as shown in Figure 12-12. This makes construction a little easier. Insert the fasteners

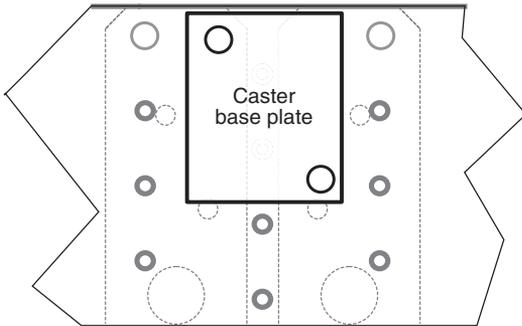


Figure 12-9 Mounting location for the balancing caster. Drill holes in the TinBot base according to the specific spacing of the caster you use.

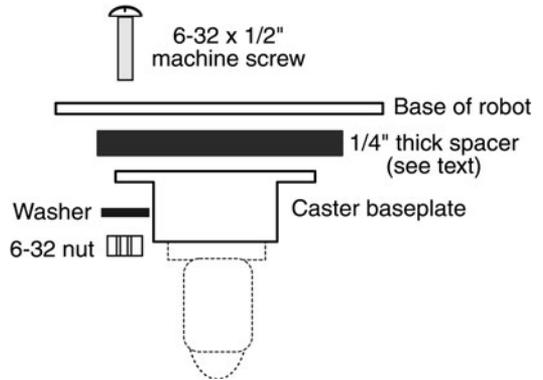


Figure 12-10 Assembly detail for attaching the caster to the base. The spacer (you may also use a stack of three or four washers) is recommended if you use a caster with a wheel diameter under 1-1/2”.

in opposite corners of the servo. Note that this does not apply if you’ve cut through the bottom of the mount at the dotted lines, as described under “Mounting the Servo Motors.” In this case, you’ll need all four screws and nuts to help reinforce the integrity of the mount.

Using the TinBot

As noted at the start of this chapter, R/C servo motors require a unique control signal in order to power them. They will not work simply by connecting them to a battery. See Chapter 23, “Using Servo Motors,” for more details on their operation.

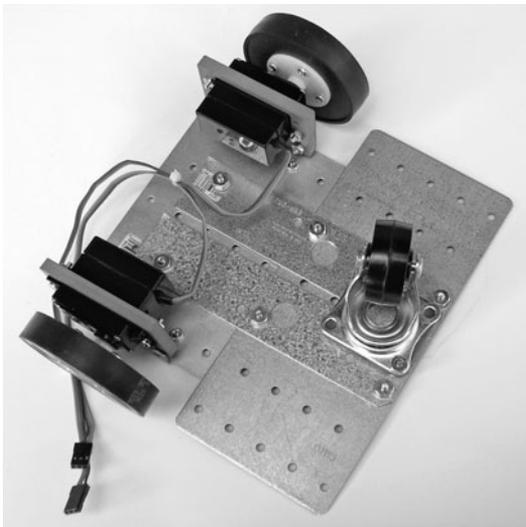


Figure 12-11 The underbelly of the TinBot, showing mounted motors and caster.



Figure 12-12 Close-up of the servo in its mount.

Also see the chapters in Part 7 for working examples of using servo motors with several popular microcontrollers. These are miniature programmable computers that, among other things, can be adapted to run servos.

The TinBot has adequate room on both the top and the bottom of its metal plates for mounting batteries, electronics, and other components. The plates already have holes in them, which you can use to mount things. At worst, you can use one hole already there and drill a second hole if the existing spacing doesn't match the component you're adding.

Because the TinBot is made of metal, and metal conducts electricity, be especially careful how you mount your electronics. Never ever allow bare wires, component leads, or the underside of circuit boards to touch the metal, or your robot parts may be irreparably damaged.



As necessary, use plastic washers (and plastic fasteners) to keep things physically separate from the metal base. Or lay down a thin sheet of plastic to act as an insulator. Every time you use your TinBot, visually inspect that there are no short circuits.

Do this prior to connecting your TinBot electronics to the battery and turning things on. Or else . . . poof! "Poof" is bad, so avoid it at all costs.

Assembly Techniques

A robot is an amalgam of parts both small and large, important and seemingly inconsequential. We don't forget the big stuff: motors, wheels, batteries, bases. But it's easy to overlook how the robot is put together—the way it's assembled.

How your robot is connected is no less important than its motors, wheels, batteries, and bases. So in this chapter, you'll learn about assembling the raw materials of your robots into full-fledged creations. After all, even the Frankenstein monster used bolts to keep his head from falling off.

Screws, Nuts, and Other Fasteners

Mechanical fasteners are the most elementary of all assembly hardware. They are the nuts, screws, and washers used to hold pieces together. Fasteners are favored because they are cheap, easy to get, and simple to use. Most fasteners can also be undone, so you can disassemble the robot if need be. This last trait is actually quite handy in academic environments, where one or two robots must be shared among many students. After one group has built and experimented with the robot, it can be disassembled and is ready for the next.

There are dozens of fastener types but just a few that are practical for amateur robotics. These are nuts, screws, and washers, shown in Figure 13-1.



Figure 13-1 Nuts, screws, and washers are the fundamental pieces used in mechanical fastening. Nuts and screws are threaded to match; washer size is based on the diameter of the screw.

- *Screws* are designed for fastening together the parts of machinery, hence the name. *Wood screws* and *sheet metal screws* have a pointed end and drive right into two (or more) pieces of material, to cinch and hold them together. *Machine screws* do not have a pointed end; they're designed to be secured by a nut or other threaded retainer on the other end.
- *Nuts* are used with machine screws. The most common is the hex nut. The nut is fastened using a wrench, pliers, or hex nut driver. Also handy in many robotics applications is the *locking nut*, which is like a standard hex nut but with a nylon plastic insert. The nylon helps prevent the nut from working itself loose.
- *Washers* act to spread out the compression force of a screw head or nut. The washer doubles or even triples the surface area. Washers are available in sizes to complement the screw. Variations on the washer theme include *tooth* and *split lock washers*; these provide a locking action to help prevent the fastener from coming loose.

FASTENER SIZES

Fasteners are available in common sizes, either in metric or imperial.

Imperial

Imperial (also referred to as American, standard, or customary) fasteners are denoted by the diameter, either as a reference number or in fractions of an inch. For machine screws and nuts, the number of threads per inch is also given. For example, a machine screw with a size of

$$6-32 \times 1/2''$$

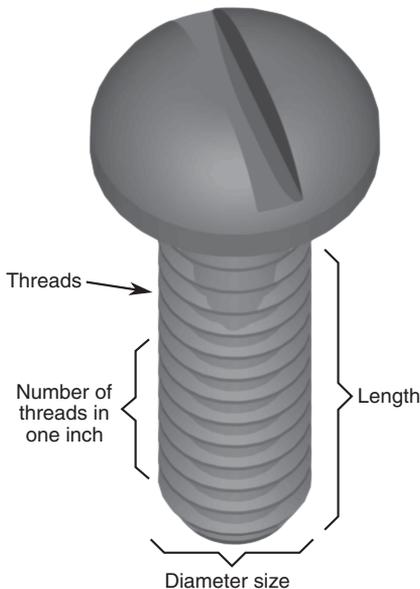


Figure 13-2 The size of a machine screw is specified by its diameter (in inches, millimeters, or numbered scale), threads per inch (or millimeter), and length.

has a diameter referred to as #6, with 32 threads per inch and a length of 1/2". Diameters under 1/4" are indicated as a # (number) size; diameters 1/4" and larger can be denoted by number but are more commonly indicated as a fractional inch measurement—3/8", 7/16", and so on. See Figure 13-2 for the sizing parameters of the typical machine screw fastener.

The number of threads per inch can be either *coarse* or *fine*. With few exceptions, your local hardware store carries just coarse thread fasteners. The one major exception is the #10 machine screw, which is routinely available in either coarse (24 threads per inch) or fine (32 threads per inch). Be careful which machine screws you buy, because nuts for one won't fit the other.

Metric

Metric fasteners don't use the same sizing nomenclature as their imperial cousins. Screw sizes are defined by diameter; the *thread pitch* is the number of threads per millimeter. This is followed by the length of the fastener. All in millimeters. For example:

$$M2-0.40 \times 5\text{mm}$$

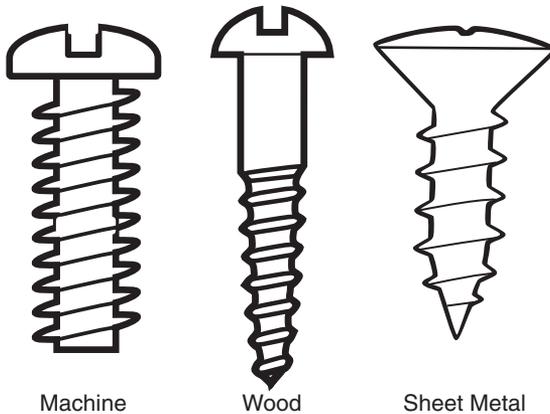


Figure 13-3 The three main types of screws: machine, wood, and sheet metal. Machine screws are used with nuts (or threaded holes); wood and sheet metal screws are intended to be fastened directly into the material.

means the screw is 2mm in diameter, has a pitch of 0.40 threads per millimeter, and has a length of 5mm.

ABOUT WOOD AND SHEET METAL SCREWS

Machine screws and nuts are intended to fasten together. I like them because they let you rebuild your robot an unlimited number of times—it's easy to take them off and put them back on.

But wood and sheet metal screws have their place, too. These are made to hold together two or more pieces of wood or metal. Both types of screws are made to be self-tapping; that is, they form their own threads in the material as they are being screwed in. See Figure 13-3 for a comparison.

- *Wood screws* are intended for wood and have a smooth part at the top of their shank. This smooth part should be about the same thickness as the first piece of wood that's been assembled. You can also use wood screws with soft plastics, such as expanded PVC.
- *Sheet metal screws* are meant to hold together thin sheets of metal. With these, the threads go all the way up to the head of the screw.

For most projects (and materials) you can often mix and match wood and sheet metal screws, and things will still work. It might not be the “official” way of using these fasteners, but as long as they don't fall out or cause the materials to crack or break, there's no harm. Use what you have available.

SCREW HEAD STYLES

When buying screws (machine, wood, or metal), you have a choice of a variety of *heads*. The head greatly contributes to the amount of torque that can be applied to the screw when tightening it. Additionally, certain head styles are designed to have a lower profile, so they stick out less than the others.

	Pan	Good general-purpose fastener. However, the head is fairly shallow, so it provides less grip for the screwdriver.
	Round	Taller head protrudes more than pan head, so it provides greater depth for the screwdriver. Good for higher-torque applications when it doesn't matter if the head sticks out.
	Flat (or countersunk)	Used when the head must be flush with the surface of the material. Requires that you drill a countersunk hole (or if the material is soft, like PVC plastic, drive the head down into the material).
	Fillister	Extra-deep head for very high torque. The top of the head is rounded. Often used in model airplanes and cars for miniature mechanical parts.
	Hex bolt	Doesn't have a slot for a screwdriver, and requires a wrench to tighten. For high-torque applications.

These are the most common, but there are at least a dozen others. They don't see heavy use in most robotics applications, so I won't list them here.

SCREW DRIVE STYLES

Most screws available at the hardware store come made for different types of drivers (that is, the tool you used to tighten or untighten the screw). For all types, different sizes of drivers are used to accommodate small and large fasteners. In general, the larger the fastener, the larger the driver.

	Slotted	Made for general fastening and low-torque drive; screwdriver may slip from the slot.
	Phillips	Cross-point drive resists drive slippage, but the head is easily stripped out when using an improperly sized driver.
	Socket	Hexagonal-shaped wrench resists slippage. Can be made very small for ultra-miniature screws.
	Hex	Uses a wrench or nut driver to tighten or untighten. Nut drivers look like screwdrivers, but with a hex socket at the end. You can use these tools to tighten nuts, too.

Which to use? I personally prefer Phillips. They're particularly well behaved when using motorized screwdrivers. Downside: The head of a Phillips screw can be greatly mangled by using a too-small or too-large screwdriver bit, so choose your tools carefully.

GOING NUTS OVER NUTS

When using machine screws you need something to tighten the screw against. That's the job of the machine screw nut. (You can also use tapped threads in the material itself; that's covered later in this chapter.) Nuts must be of the same size and thread pitch as the machine screw they are used with. That is, if you're using a 4-40 screw, you need a 4-40 nut. Makes sense.

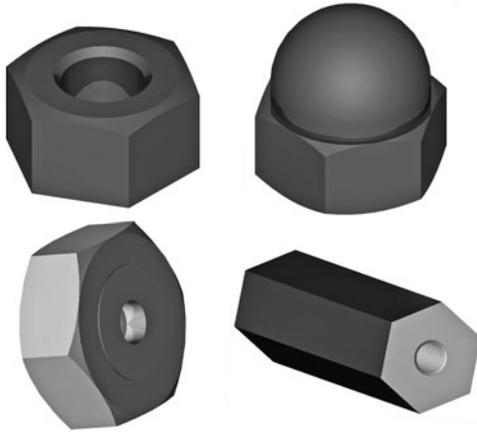


Figure 13-4 Four common nuts for use with machine screws: hex, acorn (or cap), threaded insert, and threaded coupler.

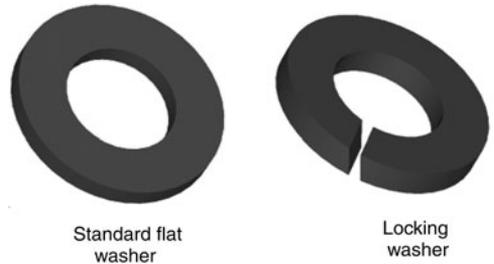


Figure 13-5 Use a standard flat washer as a spacer or to spread out the surface area of the fastener. Split washers are used to help “lock” the fastener, preventing it from coming loose.

Figure 13-4 shows several popular types of nuts for machine screws—standard hex nut, acorn (cap), nylon insert (locking), and threaded coupler. There are many more than those shown here; I’ve limited the list to just those that are of most use in robotics construction.

- *Standard hex nuts* are the most common, so named because they have six sides (hexagonal = six sides).
- *Acorn or cap nuts* are like hex nuts, but with no hole on the other side. You can use them for decorative purposes, but a more common application is as a balancing “skid” for a two-wheeled robot.
- *Nylon insert* nuts have a nylon plastic core. They’re used to create self-locking mechanisms. To use, tighten the nut into place. The plastic inside keeps the nut from working loose.
- *Threaded couplers* are like very long nuts. Use them as spacers or, if very long, as “risers” for the separate levels of your robot base.

For all, you may use a wrench or a nut driver to tighten.

WASHERS AND WHEN TO USE THEM

Washers are disc-shaped metal or plastic, used with fasteners. They aren’t fasteners themselves, but they augment the job of screws and nuts.

Washers come in two general forms: flat and locking (see Figure 13-5). *Flat washers* are used as spacers and to spread out surface area. Each size screw has a corresponding “standard-size” washer. Specialty washers are available with larger diameters and thicknesses. For example, a *fender* (or *mudguard*) washer has a very large diameter in comparison to the screw it’s used with.

Locking washers, or simply lock washers, come in two basic styles: tooth and split. Tooth-style washers dig into the material and/or fastener to keep things in place; split lock washers use compression to keep pressure against the fastener.



Which type of lock washer should you use? Split washers provide the highest locking power. But to do their job, you have to tighten the fastener enough that the split in the washer is compressed. That means they're best suited for metal.

Toothed lock washers are best used with softer materials like wood, plastic, or even aluminum (the teeth dig into the material to hold it), and for those applications when you must use nylon fasteners or can't apply a lot of force when tightening.

FASTENER MATERIALS

The most common metal fastener is steel plated with zinc, and these are the kind you'll use for almost everything. But so you know, other fastener materials may include brass (typically used for decorative purposes), stainless steel, and nylon. Nylon fasteners are lighter than steel fasteners, but can't be used where high strength is required.

SHOPPING FOR FASTENERS

You can save considerable money by purchasing your fasteners in quantity. If you think you'll make heavy use of a certain size of fastener in your robots, invest in the bigger box and pocket the savings. Of course, buy in bulk only when it's warranted. As you build your robots, you'll discover which sizes you use the most. These should be the ones you purchase in bulk.

Robot builders gravitate toward favorite materials, and fasteners are no exception. I can't tell you which sizes of fasteners to buy, because your design choices may be different from mine. But I can tell you what is used the most in my robot workshop. Perhaps that'll give you a starting point if you're just now stocking up.

For small tabletop robots I try to use 4-40 screws and nuts whenever possible, because they're about half the weight of 6-32 screws and, of course, they're smaller. I use 4-40 \times 1/2" screws and nuts to mount servos on brackets, and 4-40 \times 3/4" screws for mounting small motors. Larger motors (up to about a few pounds) may be fastened using 6-32 or 8-32 hardware.

I try to keep a few dozen of the following fasteners in stock at all times:

- 4-40 steel machine screws in lengths 1/2", 3/4", and 1"
- 6-32 steel machine screws in lengths of 1/2" and 3/4"
- 8-32 steel machine screws in lengths 1/2" and 1"
- #6 \times 1/2" and #6 \times 3/4" wood screws, for fastening together panels of rigid expanded PVC

And, of course, I keep around corresponding stock of nuts, plus some flat washers and lock washers in #4, #6, and #8 sizes. For all other sizes I buy them when needed.

TAPPING THREADS

You don't absolutely need a nut to hold a machine screw in place. If the materials you're assembling are thick enough, you might be able to use self-tapping machine screws or tap the hole with threads so that the screw will firmly anchor inside it.

Self-tapping machine screws look a lot like regular screws, except they're pointed at the very end. The threads are much coarser, as well. They're best used in soft plastics and metal (like aluminum). To use, you drill a hole just smaller than the diameter of the screw. You then tighten the screw into the hole, making threads as it goes.

Tapping threads involves using a *tap*, a special tool that looks like a drill bit. It works much like a self-tapping machine screw, except it makes standard threads, allowing you to then use regular machine screws.

For tapping to work when using metal, the material must be at least 1/16" thick, or roughly the equivalent of two or three threads of the screw—the more threads the better. For example, when using 4-40 screws, there are 40 threads per inch, which is slightly more than two threads when using materials of 1/16" thickness.

When using plastic, the minimum thickness should be no less than four or five threads, and 8 to 10 threads if using expanded PVC. Any fewer and it's likely the screw will simply strip the threads out of the hole when the fastener is tightened.

FYI

See Chapter 11, "Working with Metal," for step-by-step instructions on how to use a tap to make threaded holes. The technique works for most plastic, too.

Brackets

Brackets are used to hold two or more pieces together, usually—but not always—at right angles. You might use a bracket to mount a servo motor to a robot base, for example. Brackets come in a variety of shapes and materials. Here's what you need to know.

ZINC-PLATED STEEL BRACKETS

Though intended to lash two pieces of wood together, steel zinc-plated hardware brackets are ideal for general robotics construction. These brackets are available in a variety of sizes and styles at any hardware store. You can use the brackets to build the frame of a robot constructed with various stock. See the "Cutting a Frame" section in Chapter 7 for ways brackets are used in robot construction.

The most common brackets are made of 14- to 18-gauge steel; the *lower* the gauge number, the thicker the metal. In order to resist corrosion and rust, the steel is zinc plated, giving the brackets their common "metallic" look. (Some brackets are plated with brass and are intended for decorative uses.)

Common sizes and types of steel brackets, shown in Figure 13-6, are:

- 1-1/2" × 3/8" flat corner (or L) brackets. Use these with wood, plastic, or metal pieces cut at 45° miters to make a frame.

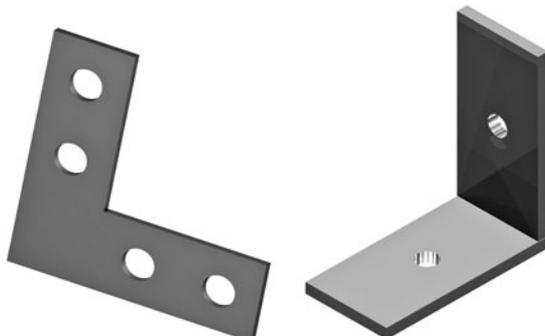


Figure 13-6 Angle brackets, both flat corner (L-shaped) and 90°. They come in a variety of sizes, thicknesses, and materials. See text for common sizes for robots.

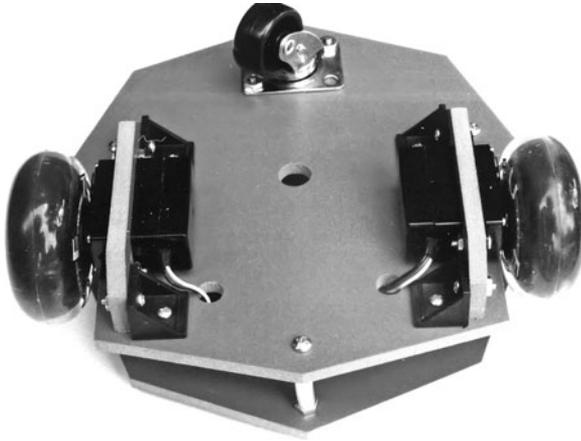


Figure 13-7 Plastic brackets may be used to secure parts to robot bases, like this pair of brackets used to secure radio control servo motors to the underside of a robot.

- $1" \times 3/8"$ or $1" \times 1/2"$ corner angle brackets. Typical uses are for attaching pieces at right angles to base plates and for securing various components (such as motors) to the robot.

PLASTIC BRACKETS



Metal brackets can add a lot of extra weight to a robot. Plastic brackets add only a little weight and for small robots are just as good. The bracket is made of a durable plastic, such as high-density polyethylene (HDPE).

To add strength, the bracket uses molded-in gussets that reinforce the plastic at its critical stress points. The result is a bracket that is about as strong as a steel bracket but lighter. Figure 13-7 shows a pair of plastic brackets used to secure servo motors in their mounts. The brackets make it easy to assemble and disassemble the parts of the robot.

Alas, plastic gusset brackets are not easy to find. They are available from some furniture-building outlets, as well as select online resources. See the RBB Online Support site (refer to Appendix A) for leads. Sizes and styles are fairly limited, but those sizes tend to be quite adequate for most jobs.

Selecting and Using Adhesives

Glues have been around for thousands of years—stuff like tree sap, food gluten, and insect secretions. Maybe the first Post-it was made with a piece of papyrus soaked in camel spit. You never know!

Modern glues—more accurately called adhesives—are chemical concoctions designed to bond two surfaces together. While there are bazillions of adhesives out there, only the ones available to consumers are covered here. Intentionally left out are the glues that need expensive tools to apply, are very dangerous to use, or are only available in 55-gallon drums.

SETTING AND CURING

All glues bond by going through a number of phases. The main phases are setting (also called fixturing), then curing. During the first phase, *setting*, the adhesive transforms from a liquid

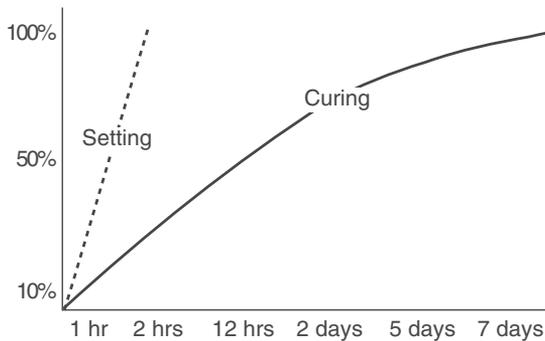


Figure 13-8 Adhesives have both a setting time and a curing time. So-called instant glues have a short setting time, but their curing time can be minutes, hours, even days.

or paste to a gel or solid. Though the adhesive may look to be “hard” when set, it is not yet very strong. This requires *curing*. Setting times for most adhesives are measured in minutes or even seconds. But curing takes a lot longer—typically 12 to 24 hours (see Figure 13-8), and often more.

For most adhesives, curing time is greatly dependent on several factors:

- *Surface temperature.* Warm surfaces tend to promote faster curing. This is most notable when gluing metal.
- *Adhesive volume.* The more adhesive that is applied to a joint, the longer it takes to cure. That’s why you shouldn’t apply too much glue.
- *Air temperature.* The warmer the air, the faster the curing (and setting time, for that matter).
- *Air humidity.* Adhesives differ in their affection for moisture in the air. Some, like Super Glue, cure faster when the air is moderately humid. Others, like epoxy, cure faster when it’s dry.

ALL ABOUT “HOUSEHOLD” GLUE

The term “household glue” is a large, diverse, and not very accurate way to describe glues that you’d use for normal household chores, like fixing broken plates or mending a busted chair. They’re also good for most robotics chores because they’re easy to get, inexpensive, and most won’t kill you the moment you uncap the bottle.

PVAc

PVAc-based adhesives are among the most popular general-purpose glues now available, and they are often sold as white and yellow “woodworking” glues. They are water-based, easy to clean up, and inexpensive. They’re best with porous material, like wood.

Silicone

Silicone-based adhesives are used for both gluing and sealing. They can bond most any non-porous surface to another, such as metal to hard plastic. A common trait of silicone adhesives is that they remain elastic. Use only in well-ventilated areas. After use, be sure the bottle or tube is recapped tightly so that no moisture can enter.

Contact Cement

Contact cement is based on various volatile organic compounds, which I personally can’t tolerate without getting a major headache. So I tend to stay away from it. Still, it’s great stuff for cementing just about anything to anything else.

As its name implies, this adhesive is designed to bond more or less instantly on contact. This is accomplished by applying a thin layer of the cement on one or both surfaces to be joined, then briefly waiting for the cement to partially set up. Applying pressure to the joint aids in creating a strong bond. Popular contact cements include Weldwood and Fastbond.

Solvent Cement

Solvent cement uses a chemical that dissolves the material it is bonding. It can be tricky to use because if the solvent isn't precisely matched with the material, nothing happens!

The most common solvent-based adhesive is for bonding PVC irrigation pipe, which can also be used with expanded PVC sheets, detailed in Chapter 9, "Working with Plastic." Other solvent-based cements are available for ABS plastic, polycarbonate, acrylic, and styrene.

APPLYING HOUSEHOLD ADHESIVES

With very few exceptions, household adhesives (those grouped above, at any rate) use single-part chemistries, so there is nothing to mix. Just open the tube, can, or jar, and apply the adhesive to the surfaces to be joined.

- Use all adhesives sparingly. A common mistake is to think that if a little bit of adhesive will do the job, a lot must be better. In fact, the reverse is true.
- For adhesives with a watery consistency, apply with a small brush or cotton-tipped swab. For thicker consistencies, apply directly from the tube or with a wooden toothpick or a manicure (orange) stick.
- Very few adhesives will stick to grease and dirt, so be sure to always clean the surfaces to be joined. Household-grade rubbing (isopropyl) alcohol does the trick.
- Avoid moving the glued joint until the adhesive has had a chance to set. As needed, clamp the or tape the bonded parts together.

ALL ABOUT TWO-PART EPOXY ADHESIVES

When normal household adhesives aren't enough, you need to turn to the "big guns": two-part epoxy adhesives. They're called two-part because at the time of application you combine a separate *resin* and a *hardener* (also called a catalyst). Separately, these materials remain in a liquid form. But when combined at the time of use, the hardener reacts with the resin and the mixture sets quickly. During this process, a bond is created as the epoxy liquid fills pores, cracks, and crazes in the surfaces of the material.

Why Is Epoxy So Special?

Epoxy sets up relatively quickly, generally in from 5 to 30 minutes. Epoxy tends to be a thick and goopy adhesive, so it has good gap-filling qualities. That's great for assembling parts that don't quite fit together. And epoxies can bond to many surfaces, including paper, wood, metal, fiberglass, most plastics, and fabric.

The typical package of epoxy adhesive consists of two tubes or bottles: one is the resin, and the other is the hardener. The tubes are separate in some products, and in others they are joined as one unit, with a single "plunger" in the center for accurately metering the resin and hardener. Figure 13-9 shows a typical plunger-style tube applicator. These are a lot easier to use than the two-tube products.



Figure 13-9 Two-part epoxy can be quickly and accurately applied when using a two-tube plunger.



So-called 5- or 30-minute epoxies represent the setting time, not the curing time. It takes 6 to 24 hours for most epoxies to cure to 60 percent to 80 percent, then the remainder over a period of *several days*. Once cured, the bond achieves its maximum strength.

Using Two-Part Epoxy

To use a two-part epoxy, it's necessary to first mix the materials together, or else use an applicator that thoroughly premixes them. The latter is not common for consumer use, so we'll concentrate on the manual mixing method. With most epoxies for consumer use, the liquids are mixed in a 1:1 ratio.

1. Apply short (1" to 2") but equal-length parallel beads of resin and hardener to a piece of index card paper. It's always better to mix too little than too much, and you can always mix in more as you need it.
2. Use a wooden (not metal or plastic) toothpick to stir the liquids together. Mixing must be thorough. Try this: mix parallel beads of resin and hardener with a zigzag action (see Figure 13-10), then scoop the material toward a common center. Stir this center "dolph" for 15 to 20 additional seconds.
3. Apply the mixed epoxy to one or both surfaces to be joined.
4. Most epoxied joints should be taped or clamped to prevent movement of the joint during the setting time. If the joint moves while the epoxy is setting up, the bond will be greatly weakened.

Unused mixed epoxy *must* be discarded. It cannot be reused. Allow the unused liquid epoxy to harden on the paper card before throwing it into the trash.

YOU, YOUR ROBOT, AND SUPER GLUE

Super Glue is a trade name, but it's often used as a generic term for a family of adhesives known as ethyl cyanoacrylates, or CAs. It's well known as being able to bond to most anything within seconds. But with the good also comes some bad. If used incorrectly, CA glues may provide only a weak and temporary bond. Keep the following in mind when using cyanoacrylate adhesives:

No gaps, please. The most common cyanoacrylate is water-thin and unable to fill in any gaps between the materials to be joined (if you need that, get the thicker "gap-filling" kind).

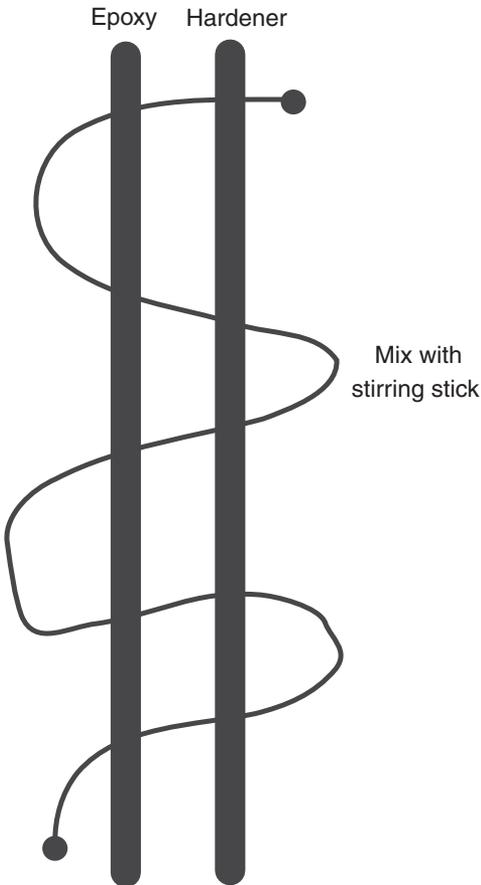


Figure 13-10 Always be sure to thoroughly mix the two parts of the epoxy before applying it to the surfaces to be bonded. After spreading equal amounts of both liquids onto an index card, use a toothpick to stir for 30 seconds.

Keep it clean. CA glues are very susceptible to ruined bonds from dirt and oil. Prior to gluing, clean surfaces with isopropyl alcohol.

Don't use too much. Applying too much CA glue is far worse than not using enough. Try a few drops on the bonded surface only.

Keep bonded joints away from heat and sunlight. Otherwise, the bond will become weak, and it may even spontaneously come apart.

Don't use with natural fibers. CA glues can produce a reaction when in contact with wood or cotton. Don't wear cotton gloves when using CA glue.

Check the date. Cyanoacrylate glue has a relatively short shelf life. Toss product that's more than a year old. Keep the unused portion in a cool, dry place.

Don't get any glue on your skin, or you could wind up cementing your fingers together. If you have an accident, acetone is a good solvent for CA glues. And for heaven's sake, keep this stuff away from your face! Seek medical help *immediately* if any CA glue gets in your eyes.

USING HOT-MELT GLUE

Hot-melt glue comes in stick form (at least the kind we're interested in) and is heated by a special gun-shaped applicator. Depending on the type of glue, melting temperatures range from 250° to 400°F. As it turns out, hot-melt glue isn't glue at all, but plastic. Adhesion occurs when the molecules of the plastic contract and harden.

The main benefit of hot-melt glue is that it sets quickly—in about a minute—yet yields a strong bond.

While using the gun is easy, the following method

helps ensure the strongest bonds. The main drawback is that the bond may not be permanent, especially when using consumer-grade glue sticks. Avoid using hot melt in high-stress situations or when the bond may be subjected to sharp impacts.

1. While the glue gun is heating up, prepare the surfaces to be bonded. Surfaces must be clean and dry.
2. If the surfaces are smooth, rough up one or both with 100–150 grit sandpaper.
3. Clean the tip of the gun with a paper towel, as needed. Careful! The tip and any liquefied glue can be quite hot.
4. Test the glue application on a paper towel. The glue should come out without delay.
5. When you're sure the gun is ready, apply a bead of hot melt to one surface to be joined. Do not overapply—less is more. If you need to apply glue to a large surface area, use a zigzag or spiral pattern to spread out the glue. Avoid applying the bead closer than about 1/4" from the edge of the surface to be joined.

6. As quickly as possible (within five seconds, no more), bring the opposite surface into contact and apply pressure to spread the glue. If you can, immediately upon initial contact gently rotate the joint 5° to 10° , then realign as needed. This helps to spread the glue.

If any excess glue oozes out from the joint, wipe it up promptly with a paper towel. Don't try to remove it with your bare fingers . . . the glue is still very hot!

CLAMPING AND TAPING GLUED JOINTS

It takes time (minutes or even hours) before glue has set to the point where it holds the pieces together on its own. For very quick bonds—on the order of a minute or two—it's acceptable to manually hold the pieces until they are set. Longer setting times may require clamping or taping. This ensures:

- Adequate pressure to “seal” the bond. The pressure of the clamp promotes full integration of the adhesive into the material. This applies mostly to porous materials, but it also affects some nonporous (e.g., plastics, metals) materials as well.
- No movement until the joint is set. If movement occurs, the adhesion may be greatly weakened.

Woodworking clamps, like those in Figure 13-11, are adequate for larger parts. But for smaller pieces, taping the joint is the most effective. After applying adhesive and mating the joint, tape is applied to keep the joint together. Masking tape works well in most situations, but if you need something stronger, white bandage tape can also be used. It's available in widths of $1/2$ " and wider.

USING JOINT REINFORCEMENTS

Critical to the strength of any bond is the way the pieces are aligned and positioned. The weakest are “buted” joints (no jokes please), where two materials are bonded end to end. The reason: There is little surface area for the joint. As a rule, the larger the surface area, the more material the adhesive can join to, and therefore the stronger the bond.

For a stronger joint, you will want to apply any of a variety of reinforcements that increase the surface area of the bond. Joint types and reinforcement techniques are explained below (see Figure 13-12).



Figure 13-11 Clamping may be necessary to hold parts together while the adhesive sets and cures.

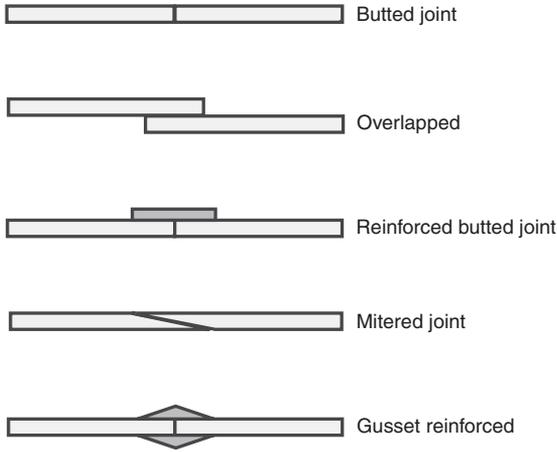


Figure 13-12 Use a joint reinforcement to add strength to bonded pieces. The weakest is the simple butted joint. Overlap or add reinforcing pieces to make the joint stronger.

Table 13-1 Selecting an Adhesive

Adhesive Type	Pros	Cons	Best Used For
Contact cement	Very fast adhesion	Careful assembly of parts required; toxic fumes	Laminating flat substrates
Cyanoacrylate (e.g., Super Glue)	Good adhesion to rubber or plastics	Poor gap filling; poor impact resistance; hard to accurately dispense with consumer tubes and bottles	Bonding porous and nonporous materials that are not subject to impact
Epoxy	Strong bond when prepared and applied correctly	Toxic fumes; curing sensitive to temperature and moisture; must be mixed correctly	Bonding all materials except silicone, Teflon, and other “slippery” materials
Hot melt	Readily available, fast setting, no harmful fumes	Can weaken under heat; poor impact resistance; accurate metering of adhesive difficult with consumer guns	Bonding wood, plastics, and light metals; use a high-temperature glue stick for a better bond with heavy materials
PVAc	Commonly available household adhesive	Not for use when both materials to join are nonporous	Bonding porous to porous and nonporous bonds (e.g., wood to metal, foam to plastic, etc.)
Solvent cement	Extremely strong bond in plastics and rubber	Requires matching the solvent to the material	Bonding plastic and rubber
Silicone	Remains flexible after curing	Toxic fumes; low strength	Bonding rubber and plastics; creating semiflexible seals

Butted joint. The typical butted joint provides minimal surface area for a strong bond and is the weakest of all. Avoid it when you can.

Overlapped. Overlap the pieces themselves, instead of butting them end to end. This is not always feasible, but it is a quick-and-easy method when the option is available. You can readily adjust the amount of overlap as needed.

Reinforced butted joint. Overlap an extra piece of material along the seam of the joint. Use the widest overlap piece you can, in order to increase the surface area. Apply adhesive to this extra piece, and clamp or tape until set. You can also reinforce with small fasteners.

Mitered. Increase the surface area of the join by mitering the ends. This is most practical with materials that are 1/4" or thicker. The technique is particularly helpful when joining wood.

Gusset reinforced. Use gusset pieces on the top and/or bottom.

IN SUMMARY: SELECTING A GOOD GLUE

With so many types of adhesives to choose from, it can be hard to select the right one. Table 13-1 summarizes the most common adhesive families, along with their pros and cons, and the bonds they are best used for. Table 13-2 provides various bonding recommendations for each major adhesive family.

Table 13-2 Recommended Adhesives, by Bonding Material

Adhesive	Bonding to:			
	Metal	Plastic/Foam	Rubber	Wood
Contact cement	OK	Recommended	OK	Recommended
Cyanoacrylate	OK	Recommended	OK	OK
Epoxy	Recommended	Recommended	OK	OK
Hot melt	OK	Recommended	—	OK
PVAc	—	OK	—	Recommended
Solvent cement	—	Recommended	OK	—
Silicone	OK	OK	OK	OK

— means not applicable or noneffective.

Rapid Prototyping Methods

Not every robot needs heavy-duty construction. Sometimes, all you're after is a general idea that your design is workable. Rather than use traditional construction with wood, plastic, or metal, you build a "rough-cut" prototype of the robot and use it for testing purposes.

This concept is known as *rapid prototyping*, and it borrows from a wealth of other technologies to provide you with a fast-track method of building your bots. Construction takes less time, and it's less expensive. While rapid prototyping is most useful for testing the merit of a design, it can also be used to build finished robots that don't require sturdy long-life construction.

In this chapter you'll learn about mechanical hardware prototyping using materials that are lightweight, cheap, and easy to cut and drill. Keep in mind, the resulting bot may not win any beauty contests, and it may not last long—fast prototypes can be rather delicate. But you won't have spent much on constructing it, and at the end you'll know for sure your idea has merit.

Selecting Lightweight Robot Materials

At the core of rapid prototyping is lightweight yet reasonably strong materials for the body of the robot. To reduce the time required to produce the prototype, you want to select a material that is easy to cut and drill, perhaps even with hand tools. So we're looking for stuff that can be cut with a knife, a razor saw, or even a pair of scissors. Candidate materials include heavy-duty cardboard, corrugated plastic, laminated paper and foamboard, and others.

Collectively, these are often referred to as "substrates," because they're used as an underlayment for things like indoor and outdoor signs, walls for temporary booths at trade shows, and posters for hanging up on your wall.

Substrates often (but not always) have multiple layers of complementary materials—each



Figure 14-1 Substrates are constructed of sandwiching layers of materials together, each layer reinforcing the other. For very lightweight substrates, a foam core is layered top and bottom with a thick paper or plastic sheet.

layer contributes to adding strength to the substrate. An example is shown in Figure 14-1. The “sandwich” construction places a top and bottom layer (say, of thin paper or plastic) with a very lightweight core, such as Styrofoam. Individually each layer is quite flimsy, but when combined the material is surprisingly strong and sturdy.

CARDBOARD—HEAVY-DUTY, THAT IS

Cardboard is the most basic of all rapid prototyping substrates. While you could build a robot base out of the cardboard of an ordinary shipping box, it’s a little too thin for the job (though in a pinch you could always layer the cardboard to make it double-thickness).

Heavy-duty cardboard is available in thicknesses from 1/8” to over 1/2”. You can find it in larger sheets or simply cut up a used heavy-duty shipping box. Laminate several pieces of cardboard to make it thicker and stiffer. “Criss-cross” the corrugation of the inner layers of the cardboard for greater strength. Use a good paper glue or contact cement for a solid bond. Cut with a sharp knife (be careful of your fingers!) or with any small fine-toothed saw.

An even heavier-duty cardboard uses a stiff honeycomb-like inner layer. It’s much more expensive than ordinary cardboard, but when used properly it can hold over 50 pounds. You can often find this type in packing materials for shipping very heavy objects, such as automobile engines. For this, you want a hand or power saw; it’s too thick to be safely cut with a knife.

CORRUGATED PLASTIC

Corrugated plastic is a common staple in the sign-making biz. It’s used for temporary outdoor signage, restaurant menu boards, that sort of thing. The plastic is composed of several layers, all bonded together during manufacture. To give the material its strength, the inner core is corrugated, like cardboard. Corrugated plastic comes in a variety of thicknesses, with 1/4” being common. You can cut it with a knife or even a heavy-duty scissors. A quick mock-up or prototype can be roughed out in minutes, and with simple tools.

Corrugated plastic gets its rigidity from its “fanfold” design. It’s meant to be used as a backing for temporary outdoor signs, so it’s not particularly hardy. If you need a stiffer substrate, you can use several layers of the plastic, sandwiching the layers at 90°. This orientation increases the rigidity of the material.

FOAMBOARD

Foamboard (aka Foam Core, a brand name) is likewise a good candidate for quick prototypes. This material is available at most craft and art supply stores and is constructed out of a foam laminated on both sides with stiff paper. Most foamboard sheets are about 1/4” thick, but other thicknesses are available, too, from 2mm to half an inch. You can find foamboard in colors at any art or craft supply store. Colored boards are more expensive, but you really only need white.

Cut with a knife or small hobby saw. Make holes with a hand drill. Because the board is laminated with paper, you can use any of a number of paper glues to try out different designs.

CONSTRUCTION FOAM

I'm using the term "construction foam" for the material used in buildings for subflooring, insulation, and sound deadening. This material is commonly referred to as "blue foam," and represents a rather diverse family of molded and expanded polystyrene plastic.



Though called "blue foam," its color may be either blue or pink. The foam is available in different densities, with many of the pink variety foams having the lowest density. You may find the heavier blue foam easier to work with because it's not as floppy.

Blue foam is from 1" to 3" thick, though specialty versions, such as floor sound-deadening foam, are available in thinner sheets. This latter material can be found at flooring stores.

Rather than as a base all on its own, blue foam is most useful as a substrate or "rigidizer" for your robot. It weighs very little for its size and bulk, yet offers remarkable rigidity. It's best used when physically cemented to another substrate, such as corrugated plastic or cardboard. The two materials together provide a strong yet lightweight building platform for your robot.

PICTURE FRAME MAT BOARD

Pictures are often framed using a heavy paper mat. Mat board is available at any art supply and picture frame store, and, depending on composition, is less than a dollar per square foot. A common size is 16" × 20", but it's also available in larger sheets (not practical for mail order), and precut into smaller pieces, such as 5" × 7" or 8" × 10". You can cut mat board with a utility knife or with a specialty mat cutter. You can also get it custom cut for you at a picture frame store; you can select squares, circles, and ovals.



Ask the sales assistants at the picture frame shop if they have any mat discards. When mat board is cut to make a picture frame, only the *outside* parts are used; the inner parts (the stuff of most use to us robot builders) is thrown away. Ask nicely, and they may give you these pieces at no cost.

While mat board may look like a piece of really thick paper, it's actually composed of many layers—plies—glued to one another. The typical quality 1/8"-thick mat has 8 to 12 plies. While thicker mat board is available as a specialty item, it's usually easier and cheaper to simply stack several 1/8" boards together, cementing the stack using ordinary paper glue.

Cutting and Drilling Substrate Sheets

The idea with all of these materials is that they are easy to cut and drill. In most cases, ordinary hand tools are all you need. Holes can be drilled with a hand drill, making these materials better suited for robot projects involving young learners. (Give younger children pieces already cut to size, to avoid having them handle a sharp knife.)

When cutting cardboard, foamboard, or corrugated plastic, bear in mind that small pieces are inherently stiffer than larger ones. A 2- × 4-foot sheet of corrugated plastic looks awfully flimsy when you hold it, but cut down to the sizes you'll most often use—4" to 8" round or square—and suddenly the stuff is remarkably more rigid than you thought.

If, after cutting to size, you think the material is too thin to do its job, consider doubling up

the thickness or gluing on reinforcement strips out of thin wood or metal from the hobby store.

CUTTING WITH A KNIFE

When cutting substrate with a knife, always use a sharp blade. Dull tools make you press down too hard, which ruins the cut and increases the possibility that the knife will slip and cause you injury.

Use a metal straightedge that you can hold down with one hand. I made mine out of 2"-wide aluminum I purchased at a hardware store for \$5. Put a plastic handle in the middle. (Be sure to countersink the holes on the bottom, and use flat-headed screws.) You can also glue on a block of wood or plastic as the handle. You don't need anything fancy.

To cut:

1. Use a #2 pencil to draw the line you wish to cut, even if you're using a straightedge. The line helps you know you're cutting in the right spot.
2. Place the substrate material on a flat surface, covered with some kind of "sacrificial" backing board—a piece of discarded paperboard or cardboard will do. You'll cut into this back instead of into the table.
3. Position the straightedge just to the side of the line, and hold down firmly.
4. Make an initial score (shallow) cut with the knife. Press the knife down just enough to break through the surface of the substrate.
5. Repeat again for a thorough cut with the hobby knife. For deep cuts (1/8" or more), draw the knife over the substrate several times, being sure to retrace the same route each time.

USING A MAT CUTTER

A mat cutter, designed for cutting out picture frame mats, also works well with softer substrates like foamboard. Most mat cutters are designed with a built-in straightedge. The cutting tool is enclosed in a heavy-duty handle, which slides along the length of the straightedge.

The advantages of mat cutters are that they're generally safer to use than hobby knives (though they can still cause injury if misused) and they're more accurate. Be sure to use a mat cutter with which you can adjust the depth of the cut—at least equal to the thickness of the substrate material you're using. Many penetrate up to 1/4".



Not all mat cutters cut straight lines. Some are designed for curves and even circles. A circle cutter lets you make small, round bases—up to about 20" diameter in some. Adjust the cutter to a smaller size and you can even pop out your own wheels! If you don't want to buy your own circle cutter, see if your neighborhood picture framing store will make the cuts for you.

Rapid Construction with Semipermanent Fasteners

Tape, ties, hold-downs, cable clamps, and hook-and-loop are used to produce semipermanent rapid prototypes. The level of permanence depends on the product, the surface area (larger tape = higher bond), and the material of your robot.

What follows applies to consumer-grade tape, Velcro, and other products. If you can get your hands on the industrial-grade stuff, it'll work the same but will give you greater holding power. Naturally, it's harder to find, more expensive, and often available only in bulk. Try specialty industrial suppliers, such as McMaster-Carr.



Don't forget you can use regular machine screw and nut fasteners with rapid prototypes. Assembly can still be fast, especially when you use a motorized screwdriver. A word of advice: When using cardboard, foamboard, and other soft substrates, add flat washers on both sides of the fasteners. This helps prevent the screw and nut from popping through the material.

HOOK-AND-LOOP FASTENERS

Velcro was discovered when its inventor noticed how burrs from weeds stuck to the fur of his dog. The construction of Velcro is a two-part fabric: one part is stiff (the burrs) and the other soft (the dog). Attach them together and they stick. The term "Velcro" is a combination of the French words "velour" and "crochet."

Velcro is a trade name for a kind of *hook-and-loop fastener*, and the Velcro company probably sells more of it than any other. It's available in a variety of sizes and types, from ordinary household Velcro you already know about to heavy-duty industrial strips that can support over 100 pounds. Figure 14-2 shows some Velcro in action on a robot. It's being used to mount a wheel caster.

Among the most useful hook-and-loop products is the continuous strip, which you can cut to the desired length. The strip comes in packages of one foot to several yards, in any of a number of widths, with 1/2" and 1" wide being the most common. The strips come with a peel-off adhesive backing. If the adhesive is not strong enough (which is sometimes the case), you can reinforce the material with a heavy-duty epoxy, screws, staples, or the like.

While Velcro may be the best-known hook-and-loop material, it's not the only kind. A great alternative—sold in the tool department of many department stores—is 3M Dual Lock (see Figure 14-3), a unique all-plastic strip that is composed of tiny plastic tendrils. Dual Lock has no separate "hook" and "loop" components. It sticks to itself.

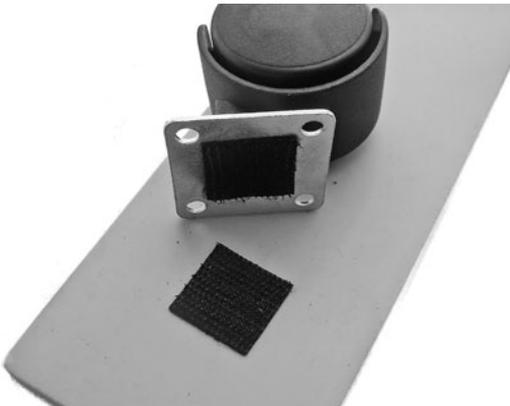


Figure 14-2 Temporarily attach parts to your robots using hook-and-loop (Velcro) fabric. Get the heavy-duty kind for holding larger components.

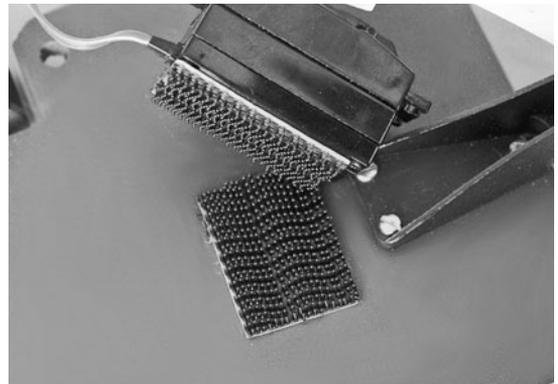


Figure 14-3 3M Dual Lock is an all-plastic version of the venerable hook-and-loop material. It's available in different thicknesses, with or without self-adhesive backing.

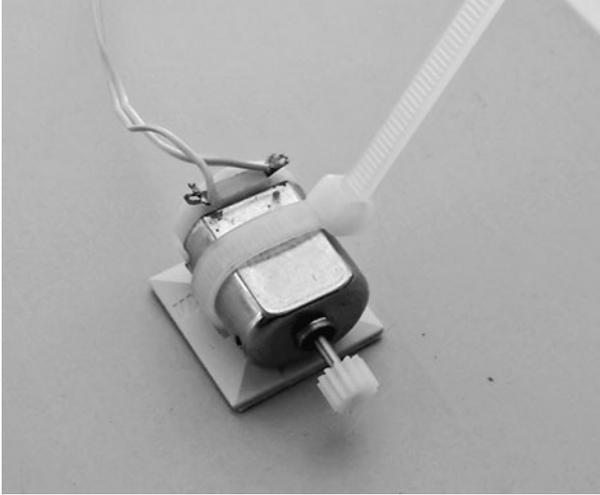


Figure 14-4 Plastic cable ties and mounts can be used to secure pieces to your robot. Use several ties for larger components. The mounts may be stuck onto the robot with self-adhesive tape or mounted using small fasteners.

PLASTIC TIES

Intended to hold bundles of wire and other loose items, *plastic ties* can also be used to hold things to your robot. The tie is composed of a ratcheted strip and a locking mechanism. Loop the strip into the mechanism, and pull the strip through. The locking mechanism is one-way: you can tighten the strip, but you can't loosen it (this applies to most plastic ties; some have a releasable lock).

Plastic ties are made of nylon and are very strong and durable. They're available in a variety of lengths, starting at 100mm (a little under 4") to well over 12". Anchor the tie into a hole you've drilled in your robot, or use one of several mounts specifically designed for use with plastic ties (see Figure 14-4). I prefer mounts designed for use with hardware fasteners; they provide a stronger hold.

STICKY TAPE

Sticky tape is a broad family of products that have an adhesive on one or both sides of the tape. Sticky tape is cheap, easy to use, and bonds nearly instantly to the surfaces you apply it to. While sticky tape makes for handy construction material, remember that the tape adhesive is gummy and can leave residue on the parts. Use denatured alcohol to remove the residue. (But test first to ensure that the alcohol doesn't dissolve the parts of your robot you want to keep!)



Most sticky tapes are not dimensionally stable; that is, under stress and load—like a drive motor—parts may shift under the adhesive. This can cause a “creep” that will result in parts becoming misaligned over time. Use sticky tape for noncritical applications only, or when you provide another way (such as mechanical stops) to keep things aligned.

DOUBLE-SIDED FOAM TAPE

A common staple in any robot builder's workshop is a roll of *double-sided foam tape*. This is like the aforementioned sticky tape, but thicker. This tape is composed of a layer of springy

foam, usually either 1/32" or 1/8" thick, and from 1/4" to over 1" wide. The tape is coated with an aggressive adhesive on both sides. To use, peel off the protective paper and apply the tape between the parts to be joined. The adhesive is pressure-sensitive and cures to a strong bond within 24 hours.

Many consumer-grade foam tapes are engineered with an adhesive that never fully cures. It stays gummy so that the tape can be more readily removed from walls. Maybe this is what you want, or maybe not. For a more permanent bond, look for industrial-grade double-sided foam tape; it's available at better hardware stores, as well as industrial supply mail-order outlets, such as McMaster-Carr. One such product is 3M VHB self-adhesive tape.

CABLE CLAMPS

Motors need to be fastened to the base of the robot in such a way that they won't easily come off or go crooked. Motor shafts akimbo result in misaligned wheels, which make your bot harder to control and steer.

When using round motors (the most common kind), look for suitably sized plastic cable clamps, available at hardware stores and online at computer accessory outlets. These clamps can accommodate cable thicknesses from 1/4" to over 1", and are secured to a surface using screws—see Figure 14-5 for an example. Use one or two clamps as a motor mount; if the motor is a bit too small for the clamp, wrap electrical tape around it to thicken things up a bit.



When using just a single clamp per motor, you'll need a way to keep the clamp from pivoting at its fastener hole. You can try tightening the fastener as far as it'll go, but a better method is to put "stops" in front of and behind the clamp. The stops—which can be something as simple as a screw head sticking out of the robot's base—prevent the clamp from moving. For larger motors you can use two clamps, with the mounting holes on either side of the motor.

ALTERNATIVE ADHESIVE DISPENSERS FOR RAPID PROTOS

Typical adhesives are dispensed from a tube or bottle. There are other methods, too, and many of the following ones are easier to use for making rapid prototypes. Give 'em a shot.

Glue Dots, a trade name, are representative of a method of applying premeasured adhesive. The dots are provided on a long roll and can be applied by hand or by machine. The dots come in various "tacks": high-tack provides permanent bonding, and low-tack, a temporary sticking place. The dots cure upon pressure.

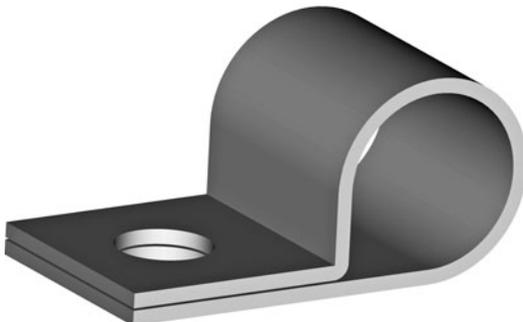


Figure 14-5 Cable clamps, designed to hold bundles of wires or cables together, can be used to attach small motors (and other things) to a robot.

Glue Stick, yet another trade name, is representative of polymer-based products that smear on a jelly-like adhesive from a self-contained applicator. Most consumer products are intended to be used to join paper to paper, but industrial stick adhesives are available for bonding metals, plastics, rubber, and other materials.

Adhesive transfer tapes are like ordinary adhesive tape, except only the adhesive portion—and not the tape backing—is left on the parts to be joined. 3M is a major manufacturer of adhesive transfer tapes. Their product, and others like it, are best applied using a special applicator that separates the protective backing of the tape while laying the adhesive over the material to be joined.

Aerosol adhesives are sprayed from a can, airbrush, or compressed air canister. The major advantage of aerosol adhesives is that they can be applied quickly in rather thin coats to a large area. 3M Photo Mount is a good example of this stuff.

Drafting Bots with Computer-Aided Design

Your robot may operate under its own power, but you probably designed and made it by hand. You can streamline the construction process by using computer-aided design, or CAD. Using nothing more than your own computer, a free or low-cost CAD program, and a printer, you can try out different designs before you ever plug in that saw.

In this chapter you'll learn about creating designs and construction layouts, first by hand (so you can see the process), and then by computer. And with a computer layout, you can even ship off your design files to a service, where they'll make the parts for you. Sounds expensive? You'd be surprised. Anyway, let's get started.

FYI

This chapter concentrates on mechanical design. A form of CAD is also used for making printed circuit boards, used in constructing robot electronics. That topic is covered separately, in Chapter 33, "Making Circuit Boards."

Making Drilling and Cutting Layouts

Everything goes better when you have a plan.

Producing drilling and cutting layouts by hand takes less time, but using a computer graphics program makes changes a snap. For example, with a computer it's easy to make the layout slightly smaller or larger, in case you want to adjust the size.

CREATING LAYOUTS BY HAND

The straightforward method of producing robot layouts is to hand draw them. The drawing can be directly on the part itself or onto paper. You can then attach the paper to the material.

Direct Layout

Mark the material with the layout you want using a soft pencil. A *construction pencil*, which you can find in the tool section of your local hardware store, has a very soft lead and writes on most any surface, including plastic, metal, and even tile (TileBot anyone?). You can also use a fine-tipped black marker such as a Sharpie.

Paper Layout

Drawing the layout on paper then using the paper as a template is a lot more forgiving. If you make a mistake it's easy just to start again with a new piece of paper. Use a sheet of ordinary unlined white paper. If the sheet is too small, you can use white craft paper, available in rolls at a craft or discount store.

With a pencil, draw the layout on the paper. A ruler or other drafting aid will help in making straight, accurate lines. If you wish, you can use graph paper (1/4" grid) to help with the layout.

When done, fix the paper template directly to the material, like that in Figure 15-1. Use tape, a glue stick, or other temporary adhesive to hold the paper in place. Use the layout to punch pilot marks prior to drilling.

After cutting and drilling is completed, peel the paper away from the material. For plastic and metal, any adhesive residue that is left can be cleaned off using denatured alcohol. For wood, the adhesive can be removed by a light sanding.

Making Multiple Parts

Paper templates make it easy to make more than one copy of the same part. Here are two methods you can try, depending on the type and thickness of the material you're using:

- Draw the layout once, then have it copied on a plain paper copier. Attach each copy to the piece of wood, plastic, or metal you're using for your robot. Be sure the copier reproduces the images at 100 percent by holding up the original and copy to a light and noting any misalignment. Some copiers automatically apply a 2 percent (or so) reduction, and this can be compensated for on better copiers by slightly enlarging the image.
- Make just a single copy, and use it to cut out multiple pieces at a time. Stack the material like layers on a cake. Drill and cut through all of them at once. This method works best when you're using thin materials, such as 1/8" expanded PVC or hardwood plywood.

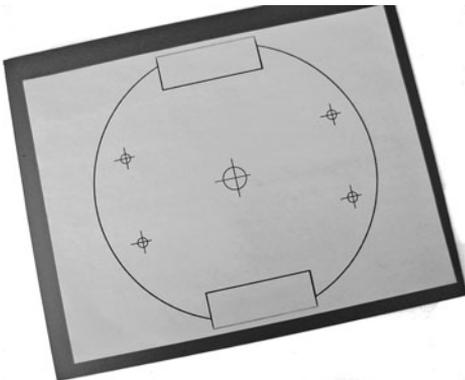


Figure 15-1 A paper template serves as a cutting and drilling guide for making any kind of robot base. Just be sure to print the template to exact size. Attach the paper using Glue Dots or other nonpermanent adhesive.

Using Transfer Paper and Scribes

Sometimes you can't (or don't want to) physically stick the paper to the material. You can use *transfer paper* instead. It works just like old-fashioned carbon paper, but it isn't as messy. Trace the design on the paper using a ballpoint pen. The tracing will appear through the transfer paper. Unwanted transfer lines can be removed with a soft pencil eraser. Find transfer paper at art supply stores.

For materials with a rough or irregular surface, the pattern can be transferred using a *scribe*. A machinist's scribe is the appropriate tool for the job, but these can be expensive. Most any sharp metal implement, such as a scratch awl or nail (with its tip sharpened) will work with plastics, aluminum, and other soft metals.

CREATING LAYOUTS WITH COMPUTER GRAPHICS PROGRAMS

Whatever you can do with a paper layout by hand, you can do better with a computer graphics program. You can create and store your layouts for future use, share them with others, and, given the right kind of program, make tweaks and changes for quick updates.

There are two general types of graphics programs: *bitmap* and *vector*. The difference is how the program stores the shapes you draw.

- *Bitmap graphics* is composed of a series of dots, like the dots in a newspaper picture. Windows Paint is a good example of a bitmap graphics program. Pick a drawing tool, and it creates a swath of dots in some distinct shape, size, and color.
- *Vector graphics* is composed of lines and other shapes. You make drawings by combining different shapes—squares, rectangles, lines, and so on—together. See Figure 15-2 for an example.

Changes are harder with bitmap graphics, because once the bits for a shape are laid down on the digital canvas, the shape itself can no longer be edited. On the other hand, with a vector graphics program you can directly change any of the shapes, even delete them. Changes are much easier.

Vector Graphics Best Choice

Of the two, vector graphics programs are by far the most useful in drafting your robots. You can use the program to create the overall design—basically a drawn picture of how your robot will look when finished. Or you can use the program to create drill and cutout templates—the same idea but better execution to layouts drawn by hand.

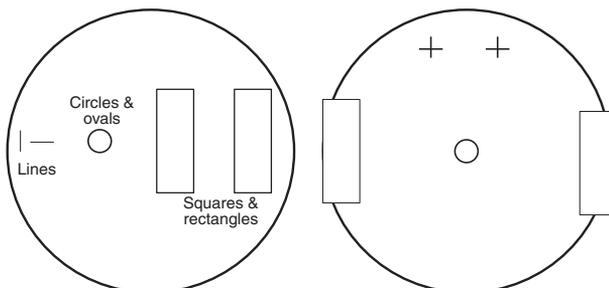


Figure 15-2 Prepare a construction template using simple shapes by combining circles, squares, rectangles, and lines.

You might already have a vector graphics program handy—even Microsoft Word comes with vector drawing tools. But there are plenty of free and open-source vector graphics applications you can try, and some are better suited for robot building. These include Inkscape, which has become something of an industry standard, and Google Sketch (available free or paid; the free version is good enough for most tasks.)

Using Inkscape for Robot Design

Inkscape serves as a good reference for using a vector graphics program to design robots. Figure 15-3 shows the Inkscape program window. Here’s a rundown of the important parts of the program interface.

- Menu bar and tool bars:** These control the program using commands in menus and on various toolbars.
- Canvas:** Your drawing goes in the middle portion, the canvas. Zoom controls let you see your drawing from a distance or up close.
- Drawing tools:** You create or edit the drawing using this small selection of tools on the left side of the screen. Vector graphics programs like Inkscape are based on what’s known as *Bezier curves*, whereby any shape is composed of one or more lines. Each line can be bent into sharp or smooth curves—a circle is really a line that bends back to itself.
- Color palette:** Solid shapes and their lines can be filled with color using a handy palette of preset shades, located at the bottom of the screen. Millions of other colors can be set using specialized tools.



Inkscape also supports writing text on your drawings. Use the text tool to label parts or provide dimensions. Inkscape lacks features for automatically inserting dimensions (this is more the domain of a CAD program; see the next section), but you can readily add the text yourself.

Figure 15-4 shows a drill and cutting layout created in Inkscape using just simple shapes. A circle is used to define the shape of the robot’s base, and rectangles indicate cutouts for

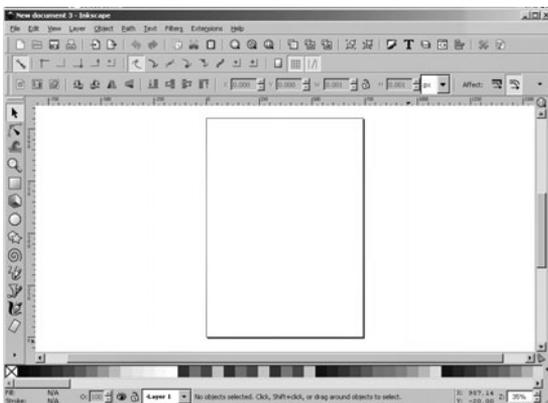


Figure 15-3 Inkscape (available for Windows, Macintosh, and Linux) is free software for creating and printing vector graphics. Its interface is simple and easy to learn.

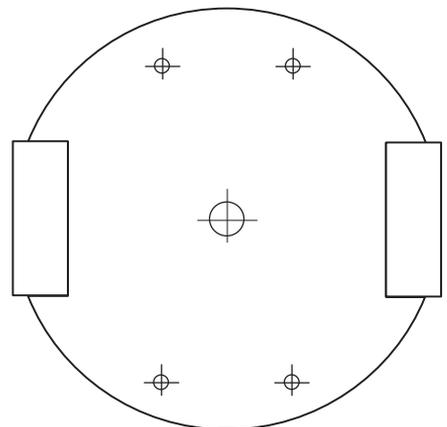


Figure 15-4 The cutting and drill template shown in Figure 15-1 created by Inkscape and then printed to scale.

motors. Small circles with thin crosshair marks show where holes go, and the size of the circles indicates the approximate diameter of the holes.

To use the template, just print it out, then tape or transfer it to the material used for your base. Use the same techniques detailed previously under “Creating Layouts by Hand.”

MAKING LAYOUTS WITH LOW-COST CAD PROGRAMS

Another way to produce layouts for your robot projects is with a *computer-aided design* (CAD) program. CAD is like a vector graphics program, but it also combines mechanical drafting features. The idea behind CAD is that not only can you draw a square, you can draw a square that is precisely 1” by 2”. Absolute measurements are stored with the CAD file and, when used with the appropriate printer, produce highly accurate renditions of your drawings.

CAD programs are often referred to as 2D or 3D. A 2D CAD program can create a two-dimensional drawing. The layout on the drawing has height and width, but no depth. This is the kind you use to produce layouts for cutting and drilling.

A 3D program can create a three-dimensional drawing that has height, width, and depth. Most 3D CAD programs can “render” 3D shapes using complex lighting and shading options. 3D CAD is not required for producing basic robot layouts, but can be used to visualize or document its construction.

AutoCad, from AutoDesk, is perhaps the best-known CAD program. As with most commercial CAD software, AutoCad is frightfully expensive. If you’re a student, you may qualify for a discount.

An alternative is a free or low-cost CAD program. Several are available for download from the Internet. While they may not compare with high-end commercial products like AutoCad, they are more than sufficient for our application.

A leader in low-cost but capable CAD programs is TurboCAD. It’s available in different editions, with both low-end and high-end versions—pick the consumer-oriented Deluxe version, as it’s a lot less expensive than the Pro version. Figure 15-5 shows a simple drill and cut template designed in TurboCAD. The template is meant to be printed as is and pasted to a sheet of wood, plastic, or metal.

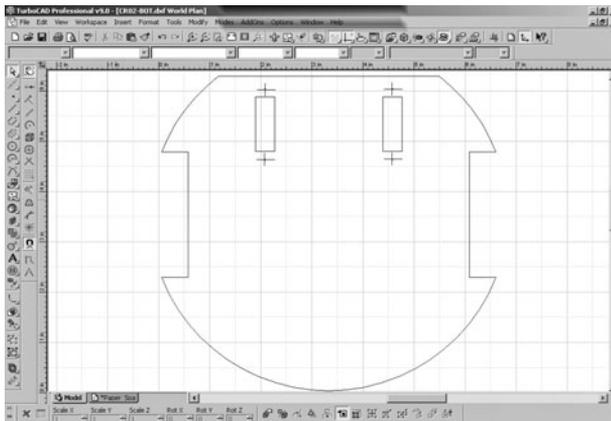


Figure 15-5 Sample cutting and drilling template created in TurboCAD, one of many CAD (computer-aided design) programs for precision drawing. It’s available in low-cost consumer versions.

Benefits of CAD

There are several benefits of using a CAD program to create cutting and drilling layouts.

Accuracy: With CAD, it's relatively easy to draw shapes with the *exact* size you want. No more guessing. You can precisely control both the size of lines, circles, and other objects and their spacing relative to one another. This can be done using the sizing and dimensioning tools, and with *snaps*, where drawn objects conform to known sizes and boundaries.

Drawing automation: If you need to produce a series of 20 holes around the circumference of a 6" circle, for example, tools provided by the CAD program make this easy.

Editability: Designs can be readily and quickly altered, in case you need to make adjustments. While some refinements can be made "on the fly" as you work with the finished robot, you may also wish to go back to the original design, make changes, and start over again.

Automated construction: A fourth benefit applies when using a computer-controlled mill or other machine to construct the pieces you design. The drawing you produce with the CAD program can be used directly to build the finished part. Most software for computer-controlled manufacturing can read DXF files, which is a commonly supported file format of 2D CAD programs. See "File Formats for Vector Graphics" for more details on graphic formats.



Not many people own their own computer-controlled mill, but you can always send your drawings to a service bureau and have them use their machines to make your parts. See "Using Laser Cutting Services," later in this chapter. Unless you plan to market your robot product, it's more affordable to send out parts to be manufactured.

Basic CAD Functionality

Most CAD programs require something of a steep learning curve to discover how to use its features. But basically, and for our purposes, the functionality of the program can be narrowed down to the following:

- **Drawing setup.** Here you define the drawing size and drawing scales (e.g., 1:1, 1:12, etc.), unit of measurement, as well as grid size and drawing resolution. For most robotics projects, you'll want a 1:1 scale, a grid of 1/4" or 1/8", and a resolution of 2- or 3-decimal places—that is, down to the hundredths or the thousandths of an inch.
- **Drawing tools.** Only a few shapes are used for typical robot layout drawings: line (and/or polyline), circle, and rectangle. Lines are used to mark cutting layouts. A polyline is a set of lines that share at least two vertices (corners), and it is used whenever you want to cut out more complex shapes. Circles are typically used to denote holes for drilling. A rectangle or square is a closed polyline shape and can be produced using the line, polyline, or rectangle tool.
- **Editing/sizing tools.** You can adjust the size and look of the shapes by using the mouse or by entering values at a command-line prompt. The mouse is good for "eyeballing" the design, but the command-line entry is handy when you need accurate placement.
- **File saving/printing.** Once done with the drawing, you can save it for future use or print it out. Any supported printer will do, such as a laser or ink-jet printer. CAD programs don't have to be used only with pen plotters anymore.
- Drawings are placed on a *workplane*. With 2D CAD, a simple *X* and *Y* coordinate system

is used to denote the origin (the start point in virtual space) of the drawing. With most CAD programs the origin is the lower-left corner of the drawing and is denoted as **0,0**. The first digit is the X axis; the second digit is the Y axis.

File Formats for Vector Graphics

You're probably familiar with GIF, JPG, and PNG graphics files. These are all bitmap file formats and typical of images you see on Web sites. Each is a different data format and, by convention, the specific format is used as the filename extension—a file named *myrobot.jpg* is a JPG bitmap of someone's robot.

Likewise, vector graphics have their own file formats. As many were created for a special purpose or graphics program, they tend to be unique to themselves and often are not compatible with one another (you can sometimes use a converter program to exchange one format for another, but the results can leave much to be desired).

Of the varied vector graphics formats, these that follow are the most common, and the ones you'll likely work with and share with others.

- **SVG**—Scalable Vector Graphics, now the uber-standard for vector-based images, and supported by Wikipedia. This is Inkscape's default file format.
- **EPS**—Encapsulated PostScript is an interchange format promoted by Adobe. It contains definitions of the graphic elements as PostScript command codes, plus (usually) a medium-resolution bitmap for use by those programs that can only import a bitmap. Programs have varying support for EPS, and some—like Inkscape—need additional plugins to understand the format.
- **AI**—The native file format for Adobe Illustrator, AI is one of the premier commercial vector graphics applications. Saving and sharing this format is acceptable as long as everyone has Illustrator and, better yet, similar versions. Otherwise, use SVG.
- **DXF**—The Drawing Exchange Format is most commonly found on CAD programs and was originally developed by AutoDesk to allow AutoCad to share its 2D files with other applications. It's now become a de facto standard for all CAD apps.
- **DWG**—Like DXG, the Drawing format is used for interchange with CAD programs and is primarily intended for 3D graphics. It's not as widely supported as DXF.

File formats common in some 3D CAD programs include SAT, STEP, IGES, and INV. Not all CAD programs can open them. If you share your designs with others, be sure everyone has the same CAD program or can open the files without significant loss of information.

Using Laser-Cutting Services

See that cutting and drilling template in Figure 15-4? Because it's made to exact scale, you could use the drawing to have your robot base professionally produced with a laser cutter. Laser cutters use a high-intensity pencil-thin light beam to make precision holes and cuts in wood, plastics, and even some metals.

Cost varies depending on the amount being cut and the type and thickness of the material, but it's less expensive than most people think. It's a good alternative if you have a complex design and need the precision that hand-cut parts can't provide.

Not all services cut every material. Most will accept cardboard, plywood, acrylic plastic, polystyrene, and polycarbonate plastic. Laser cutters generally won't accept jobs involving aluminum or expanded PVC, as working with these materials can damage their machines.

Custom laser-cutting services can be found locally as well as online. Do a Web search for *laser cutting*. If you're only interested in local services, online business directories (Yellow Pages, Yellow Book, and so on) will help you narrow your search to those near you.

Some tips:

- Be sure to read and understand the instructions for submitting files. Most laser cutters prefer files using the *DXF* format (these have a *.dxf* extension). Any self-respecting CAD or vector graphics program can save files in this format.
- Use just one line thickness. Don't use a "fill" on any of the shapes. Holes should be circles; the hole will be the diameter of the circle.
- The lower left of your drawing should start at 0,0. No part of your drawing should go below the 0 marks, or it may not be cut.
- In all CAD and vector graphics programs, the shapes (*objects*) of the drawing are stacked one on top of the other on the canvas workplane. The stacking order of these objects matters: the cutter will start with the objects at the bottom and finish with those on the top. When cutting out a base, you want the outline of the base to be cut last. See the manual for your CAD/vector program to learn how to manipulate the order of objects on the workplane.

Producing “Quick-Turn” Metal and Plastic Prototypes

Thanks to computer automation, you can now quickly and affordably design and produce parts for your robot out of metal or plastic. The concept is called *quick-turn prototyping*, and it takes the idea of laser cutting to the next level. Using approved CAD software, or a graphics program that the prototyping service provides, you design your part, then submit it to them for manufacture. They then produce the part—usually within a few days—on their automated equipment.

Common parts for quick-turn prototyping include your own motor or servo brackets, custom gripper components, and even complete robot bodies. Aluminum is widely used in quick-turn manufacturing, because it can be machined, cut, and bent on automated (and partly automated) machinery.

A variation on the theme is using *3D printers*, where liquid ABS, acrylic, or polycarbonate plastic is squirted in precise, measured amounts as a “print head” zigzags back and forth. On each zig and zag, a 3D shape is formed bottom to top. 3D printing is also known as *additive manufacturing*, *RepRap* (after a popular open-source project by that name), and *FDM*, which stands for Fused Deposition Modeling.

To turn your ideas into 3D shapes, you need to start with 3D CAD software (SolidWorks is popular with many quick-turn manufacturers) or some other mainstream program. Or you may be expected to use the proprietary modeling software provided by the quick-turn service shop.

Constructing High-Tech Robots from Toys

Ready-made toys can be used as the basis for surprisingly complex homebrew hobby robots. Snap or screw-together kits, such as the venerable Erector or Meccano set, let you use premachined parts for your own creations. Not to mention that the toy industry is robot crazy, and you can buy a basic motorized or unmotorized robot for parts, building on it and adding sophistication and features.

Some kits, like LEGO and K'NEX, are even designed to create futuristic motorized robots and vehicles. You can use the parts in the kits as is or cannibalize them. Because the parts already come in the shape you need, the construction of your own robots is greatly simplified.

Let's take a closer look at using toys in your robot designs in this chapter and examine several simple, cost-effective designs using readily available toy construction kits.

Erector Sets

(Note: Everything in this section also applies to the Meccano brand and similar beams-and-girders construction sets sold by others.)

Several Erector sets come with wheels, construction beams, and other assorted parts that you can use to construct a robot base. Motors are typically not included in these kits, but you can readily supply your own.



I've found the general-purpose sets to be the best bets. Among the useful components of the kits are prepunched metal girders, plastic and metal plates, tires, wheels, shafts, and plastic mounting panels. You can use any as you see fit, assembling your robots with the hardware supplied with the kit or with your own 4-40 or 6-32 machine screws and nuts.

The prepunched metal girders included in the typical Erector set make excellent motor mounts. They are lightweight enough that they can be bent, using a vise, into a U-shaped

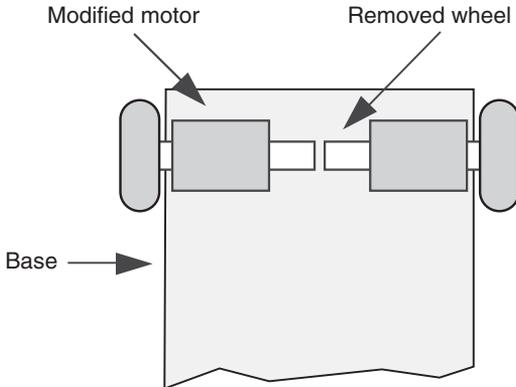


Figure 16-1 Robot bases powered by two motors and wheels on either side may be constructed using a pair of inexpensive salvaged motorized toys. For each toy, remove one of the wheels from the motor mechanism.



Figure 16-2 Sampling of parts from a general-purpose Fischertechnik kit.

motor holder. Bend the girder at the ends to create tabs for the machine screws, or use the angle stock provided in an Erector construction set.

Most Erector vehicles use four wheels, but that wheel arrangement makes it difficult to steer the robot. Instead, use a two-wheel design like those depicted in Chapter 20, “Moving Your Robot.” Mount a battery holder on the top of the platform for power.

One method of motorizing a nonmotored Erector set is to rob the motor drives from two inexpensive powered toy vehicles. These use a single motor to power two wheels; the wheels are on either side of the motor. To convert these motor drives for use on a robot platform, remove one wheel from each side of the motor, as shown in Figure 16-1.

- For the right motor, pull off the left wheel.
- For the left motor, pull off the right wheel.

You now have two independent motor drives. Use girders or other parts from the Erector set to mount the motors to the base.

Fischertechnik

The Fischertechnik kits, made in Germany and imported into North America by a few educational companies, are the Rolls-Royces of construction toys. Actually, “toy” isn’t the proper term because the Fischertechnik kits are not just designed for use by small children. In fact, many of the kits are meant for high school and college industrial engineering students, and they offer a snap-together approach to making working electromagnetic, hydraulic, pneumatic, static, and robotic mechanisms.

All the Fischertechnik parts (see Figure 16-2) are interchangeable and attach to a common plastic baseplate. You can extend the lengths of the baseplate to just about any size you want, and the baseplate can serve as the foundation for your robot. You can use the motors supplied with the kits or use your own motors with the parts provided.

K'NEX

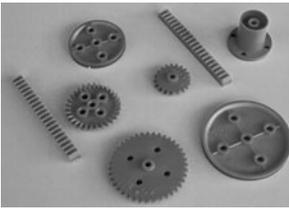
K'NEX uses unusual half-round plastic spokes and connector rods to build everything from bridges to Ferris wheels to robots. You can build a robot with just K'NEX parts or use the parts in a larger, mixed-component robot. For example, the base of a walking robot may be made from a thin sheet of aluminum, but the legs might be constructed from various K'NEX pieces.

A number of K'NEX kits are available, from simple starter sets to rather massive special-purpose collections (many of which are designed to build robots, dinosaurs, or robot-dinosaurs). Several of the kits come with small gear motors so you can motorize your creation. The motors are also available separately.

Other Construction Sets to Try

Toy stores are full of plastic put-together kits and ready-made robot toys that seem to beg you to use them in your robot designs. Here are some toys you may want to consider for your next project.

INVENTA



United Kingdom-based Valiant Technologies offers the Inventa system, a reasonably priced construction system aimed at the educational market. Inventa is a good source for gears, tracks, wheels, axles, and many other mechanical parts. The beams used for construction are semiflexible and can be cut to size. Angles and brackets allow the beams to be connected in a variety of ways. Inventa isn't the kind of thing you'll find at the neighborhood Toys "R" Us. It's available via mail order and through the Internet; see the Inventa Web site at www.valiant-technology.com.

ZOOB

Zoob is a truly unique form of construction toy. A Zoob piece consists of a stem with a ball or socket on either end. You can create a wide variety of construction projects by linking the balls and sockets together. The balls are dimpled so they connect securely within their sockets. One practical application of Zoob is to create armatures for human- or animal-like robots. The Zoob pieces work in a way similar to bone joints.

ZOMETOOL

For kids, parents, experimenters, and educators, Zometool sets are composed of ball-shaped connectors (called nodes) and a variety of interconnecting rods they refer to as struts. The sets can be used to create different physical models of things like DNA, molecules, plane and solid geometric shapes, and much more. For robotics you can use the parts as construction girders and brackets.

The struts solidly latch into sockets within the nodes, but it's probably not secure enough for a robot that careens across the floor. As needed, use glue (a dab of hot glue works) to help secure the pieces together for a temporary fit; for a more permanent bond use ABS solvent cement (see Chapter 13, "Assembly Techniques," for more information about using various types of adhesives.)

GONE BUT NOT FORGOTTEN

As of this writing, these most excellent construction toy sets are not in production—though that could change if some company buys the rights and production molds, and re-releases the product. In any case, you may be able find bits and pieces at garage sales, resale shops, and online traders (try Craigslist and eBay):

- Milton Bradley Robotix
- Capsela
- Construx

Construction with Snap-Together Components

I'm far from the purist. I don't mind reusing things like LEGO bricks (see Figure 16-3), MEGA Bloks, and K'NEX (see above) in my robots. "Parts is parts," as they say. Unless you're going after an unusual design, there is no cutting or drilling involved—just pick the piece you want to use, and snap it into place.

MAKING JOINTS (MORE OR LESS) PERMANENT

Snap-together components are by their nature temporary. They are made to be taken apart and reused. This may be your aim with your latest robot creation. Also bear in mind that temporary constructions can come apart when you don't want them to, especially if the robot is mishandled, takes a fall from the workbench, or bangs into objects or other robots.

Though snap-together parts are most often used in robotic constructions with or without adhesives, it is also perfectly acceptable to use other binding techniques with them, including double-sided foam tape and nylon tie-wraps. By no means are you limited in any way in how you lash the goodies together. As the variations are endless, I'll just leave the discussion at that, and let your creativity come up with interesting alternatives.

If you decide gluing the parts together is the method, pick the glue to match the kind of plastic used for molding the pieces.

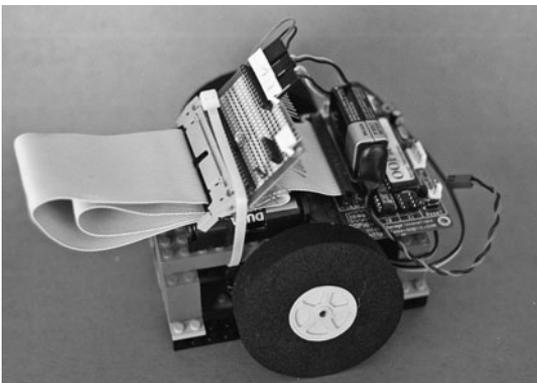


Figure 16-3 A small desktop robot made with surplus LEGO bricks and beams. The LEGO pieces are glued together.

Construction Parts	Temporary	Permanent
LEGO, K'NEX, Zometool	White glue	ABS plastic solvent cement; 2-part epoxy
MEGA Bloks, plastic models (e.g., model cars or airplanes)	White glue, low-temperature hot-melt glue	Plastic model-building (polystyrene) solvent cement; 2-part epoxy
Fischertechnik, most other plastic construction toys	Low-temperature hot-melt glue	ABS-PVC solvent cement; 2-part epoxy

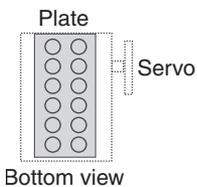
- For a strong but less permanent bond, use only very small amounts of solvent cement or epoxy.
- You can also make nonpermanent bonds by using very small amounts of cyanoacrylate (CA) adhesive. Super Glue is a common brand of CA adhesive. Apply to one side only. Remove the parts by giving them a good twist.
- High-temperature hot-melt glue provides a good middle ground between temporary and permanent constructions. Use sparingly if you wish to disassemble the parts later. The glue can usually be peeled off.
- Flexible adhesives, such as Shoe Goo or any silicone-based RTV adhesive also make for strong yet temporary bonds.

Depending on the design of the construction toy, you can also use mechanical fasteners to hold things together. Drill holes in your LEGO, MEGA Bloks, or K'NEX parts, for instance, and secure them with miniature 2-56 or 4-40 machine screws and nuts. The plastic is easy to drill through, and the fasteners can be readily removed if you need to take things apart.

USING SNAP-TOGETHER PARTS TO MAKE MODULES

You're not limited to using snap-together construction parts just for the body or structure of your robot. You can use extra or discarded LEGOs and similar construction pieces for making customized accessories. The snap-on nature of these parts allows you to easily reuse these accessories for different prototype projects.

For example, you might glue the flat bottom of a standard R/C servo motor to some LEGO blocks. (It's okay to use glue here because you're able to reuse the motor as much as you'd like. Simply pull it off the LEGO plate when you're done.) Use the servo motor to quickly and effortlessly attach a sensor turret to your base. Just add a bracket and ultrasonic or infrared sensor on top of the servo, snap the servo into place on the base, and you're done.



The same concept works for other components as well, such as compasses, accelerometers, microcontrollers, speakers, and lights. The idea is to mount the part to one or more accommodating LEGO pieces, ensuring that everything is aligned. This allows you to press one LEGO part into the next.

The idea doesn't stop with LEGO beams, plates, and other parts. You can also use plastic or metal construction set pieces from an Erector set. One benefit here is that if you make accessories using these parts, they are easily transferred to full robots once you have successfully prototyped the design. For instance, you can permanently mount a short LEGO Technic

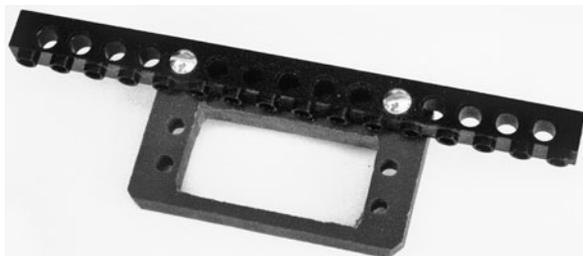


Figure 16-4 You can combine LEGO pieces with homemade plastic parts for such things as making mounts for radio control servo motors.

beam to a bracket for a servo motor (see Figure 16-4), and use it as a mounting bracket for a cardboard-based prototype.

Specialty Toys for Robot Hacking

Some toys and kits are just made for *hardware hacking* (retrofitting, remodeling) into robots. Some are already robots, but you may design them to be controlled manually instead of interfacing to control electronics. Here are just a couple of ways you can use inexpensive toys to make homebrew robots.

TAMIYA

Tamiya is a manufacturer of a wide range of radio-controlled models. They also sell a good selection of gearboxes in kit form that you can use for your robot creations. One of the most useful for starter bots is the Twin Motor Gearbox (item number 70097), which consists of two small motors and independent drive trains. You can connect the long output shafts to wheels, legs, or tracks. This gearbox is used for the *My First Robot* lessons; refer to Appendix A, “RBB Online Support.”



One big disadvantage of the Twin Motor Gearbox is that the DC motors it uses are quite inefficient. Under heavy load or stall (powered but stopped) conditions, the motors can draw over 2 amps each. That’s a lot for this size motor. (Amps and other terms are covered in more detail in Chapter 22, “Using DC Motors.”)

The Twin Motor Gearbox is acceptable when controlled by switch or relay, but when using transistor control you should replace the DC motors it uses with more efficient versions. Pololu and several other online retailers offer these replacements; cost is under \$3 per motor.

Other Tamiya kits include single motors with selectable gear ratios, basic walking and rolling robots, wheels, and more.

OWIKIT AND MOVITS

The OWIKIT and MOVITS robots are precision-made miniature robots in kit form. A variety of models are available, including manual (switch) and microprocessor-based versions. The robots can be used as is, or they can be modified for use with your own electronic systems. For example, the OWIKIT Robot Arm Trainer (model OWI-007) is normally operated by pressing switches on a wired control pad. With just a bit of work, you can connect the wires

from the arm to a computer interface (with relays, for example) and operate the arm via software control.

Most of the OWIKIT/MOVITS robots come with preassembled circuit boards; you are expected to assemble the mechanical parts. Some of the robots use extremely small parts and require a keen eye and steady hand. The kits are available in three skill levels: beginner, intermediate, and advanced. If you're just starting out, try a beginner-level kit.

Once (properly) constructed, the OWIKIT and MOVITS robots last a long, long time. I have several models—no longer available—I built in the mid-1980s, and, with just the occasional nut tightening here and a dab of grease there, they have continued to operate flawlessly.

Making Robots from Converted Toy Vehicles

Toy cars, trucks, tractors, and tanks can make ideal robot platforms. With some motorized vehicles, you can directly convert them to robot service by hacking into their motor connections. Quick and simple! With others, you may need to do a bit of disassembly and rebuilding, especially those toys that have only one drive motor—it's best when they have two.

And let's not forget that you can rob parts from nonmotorized toy vehicles. I've gotten some of the best stuff off of cheapo "dollar store" toys! Push-around toys with rubber tank treads are an especially nice find. Rob the treads and put 'em on your own robot base.

MOTORIZED VEHICLES

Let's start with inexpensive radio-controlled cars. These have a single drive motor and a separate steering servo or mechanism; the setup doesn't lend itself well to robot conversion. In many cases, the steering mechanism is not separately controlled; you "steer" the car by making it go in reverse. The car drives forward in a straight line but turns in long arcs when reversed. These are impractical for use as a robot base and you should spend your attention elsewhere (they're okay for stripping off parts).

On the other hand, most radio- and wire-controlled tractor (farm, military tank, construction) vehicles are perfectly suited for conversion into a robot. Remove the extra tractor stuff to leave the basic chassis, drive motors, and tracks.

You can keep the remote control system as is or remove the remote receiver (or wires, if it's a wired remote) and replace it with new control circuitry. In the case of a wired remote, you can substitute relays or an electronic circuit for the switches in the remote. Of course, each toy is a little different, so you'll need to adapt this wiring diagram to suit the construction of the vehicle you are using.

FYI

The most common electronic circuit used as a substitute for switches is the transistorized H-bridge, discussed more fully in Chapter 22, "Using DC Motors." When replacing manual switches with an H-bridge, be sure the motors don't draw more current than the H-bridge can handle, or damage to the electronics could occur.

Chapter 21 details the method of testing the amount of current consumed by a motor. You'll need a digital multimeter to complete the test.

Another option is to use two small motorized vehicles (mini "4-wheel-drive" trucks are perfect), remove the wheels on opposite sides, and mount them on a robot platform. Your robot uses the remaining wheels for traction. Each of the vehicles is driven by a single motor,

but since you have two vehicles (see Figure 16-5), you still gain independent control of both wheel sides.

The drives in the picture were taken from a pair of Tamiya Monster Beetle Jr. toys (catalog # 17001); the same 4WD drive is used in a number of Tamiya toy products that differ only in the car body used on top—examples include the Juggernaut Jr. (catalog # 17014), and the Toyota Hi-Lux Monster Racer (catalog # 17009). The removed wheels are replaced with 1/8" Dura collars; this prevents the axles from coming loose in the chassis. The collars are locked in place with a miniature set screw. The set screw comes with the collars.

Whatever vehicles you use, be sure they are the same exact type. Variations in design (motor, wheel, and so on) will cause your robot to “crab” to one side as it attempts to travel a straight line. The reason: The motor in one vehicle will undoubtedly run a little slower or faster than the other, and the difference in speed will cause your robot to veer off course.

USING PARTS FROM VEHICLES

It's called *repurposing*. Toys can be a terrific source of parts that would otherwise cost a lot more if purchased as honest-to-goodness “robot accessories.” This is especially true of wheels and tank treads.

Because of the economies of volume production, a \$10 toy may contain four wheels that would otherwise sell for \$5 each—a savings of 50 percent. The same is true of rubber, plastic, and even metal tank treads, which are hard to find in any case. A pair of rugged rubber treads specifically for robotics could retail for between \$30 and \$50, yet a toy tank with the same treads might sell for \$19.95.

Figure 16-6 shows a motorized remote control tank outfitted with rubber treads, drive sprocket, and “idler wheels” that keep the tread in place. It's operated by two motors, one for each tread—there's even a third motor on this toy, used to swivel the cannon turret back and forth.

The majority of these toys are imported from China, where stock comes and goes, so you never know what will be available, or for how long. That can be frustrating, but if you're on your toes, you can snatch a bargain when you least expect it.

I bought four of these tanks for experimenting, and within six months the source for them was dry, the item replaced with some other toy vehicle (that unfortunately wasn't as good—that's



Figure 16-5 This 4WD robot base was constructed from two motorized toy models. The models used a single motor to power four wheels. For each motor the wheels on opposite sides were removed; a metal collar (fastened in place with a set screw) prevents the axle from coming out.



Figure 16-6 A store-bought tank toy contains motors, rubber tracks, drive sprockets, and idler wheels—all can be repurposed for a desktop tracked robot.

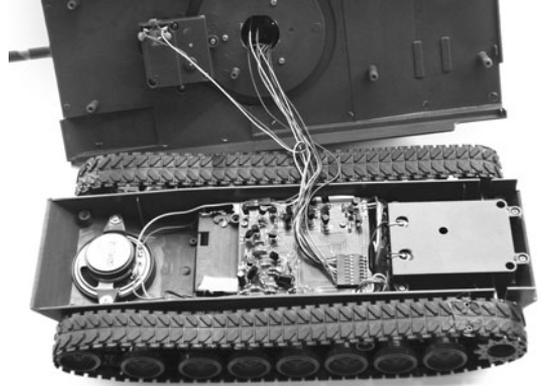


Figure 16-7 Inside the tank depicted in Figure 16-6, showing the modular construction. Not all tank toys are like this model, but when you find one like this, it's a pleasure to tear apart.

the way it goes). The moral: Always be on the lookout for motorized toys that could make for good robot platforms.

Toy vehicles such as this one are ideal starter bases for your robots. Remove the remote control electronics and attach your own to the motors already in the tank. The innards are shown in Figure 16-7; you can see the motor in its case, a speaker for sound effects, and the circuit board that contains the remote control and motor drive electronics. This board is readily removed and replaced with your own microcontroller and motor drive.

REUSING ALL THE PARTS

When removing treads be sure to also collect the drive sprocket and any idler wheels used to keep the tread in place. You'll want to reuse these with your robot, too. Figures 16-8 and 6-9

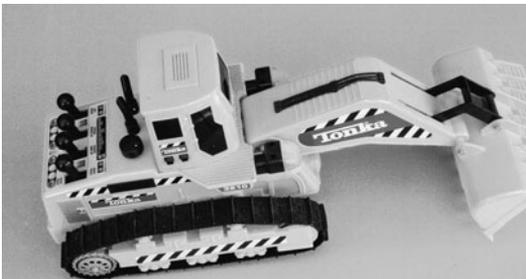


Figure 16-8 Tank toy as it came out of the box.

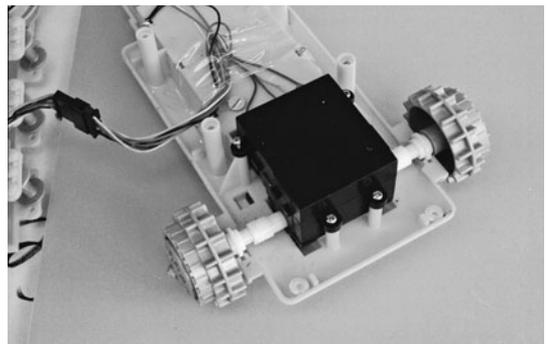


Figure 16-9 Tank toy showing the motor drive unit, with drive sprockets for the rubber tracks.

show a flexible robot base using treads and related parts from a \$20 tank toy. The pictures show you the kind of mechanics you might find in a motorized toy.

FYI

See Chapter 22, "Using DC Motors," for various ways to operate the motors from hacked toys. Options include switches for manual control, relays, and motor bridge circuits. The latter two allow you to operate the vehicle from a microcontroller or other electronics.

Building Bots from Found Parts

Found parts” are things you find around the house—or garage or hardware store or anywhere else—that are just *begging* to be used in your next robot. Or used as your next robot! Found parts can help reduce the costs of building a robot. And if the found part can be used as is, without any special cutting, it makes building the robot easier because you don’t need as many tools for construction.

Toys are among the most popular form of found parts, and these are covered in their own chapter; see Chapter 16, “Constructing High-Tech Robots from Toys” for more details. In this chapter you’ll learn how to adapt common household articles and retrofit them for use in everyday robots.

There’s practically no limit to the number and type of found items you can press into using with your robotics projects—either as the body of the robot itself, or as part of a subsystem. Rather than even attempt to cover them all, this chapter explores the concepts of using found parts to stimulate your creativity.

A Dozen Ideas to Get You Started

There are plenty of everyday objects you can use for robot building—all it takes is looking at them a bit differently than the objects’ manufacturers intended. Some examples to whet your appetite (all of these have been turned into robots, either by me or by someone I know):

Plastic storage containers: Available in square, round, and other shapes, these durable plastic boxes—available in the housewares section of any department store—can be used with or without their press-on lids. Plastic boxes are available from small snack size to big shoe boxes.

Small “dorm-size” trash cans: Just large enough to hold a Big Gulp, these trash cans have a convenient cylindrical shape and removable top. Great for building miniature

R2-D2 bots. The plastic trash cans are easy to drill through and cut, for mounting motors and other parts.

Computer mice: A discarded computer mouse makes a great body for a micro-miniature robot. Almost all mice can be disassembled by removing one or two screws on the bottom. After removing the circuit board, mouse ball, cable, and switches, you can install small motors, a small battery, and a one-chip brain.

Compact discs and DVDs: Save the world’s landfill and use these 4.7”-diameter discs for robot bases. Use care when drilling holes in the plastic: the material can shatter into very sharp pieces. If you need added strength, sandwich two discs together.

Solderless breadboards: Solderless breadboards are used to experiment with circuits before using more permanent solder and wire-wrap construction. Mount motors and wheels on the underside of your solderless breadboard, and you create a versatile and ever-changeable mobile robot.

Plastic project boxes: These boxes, sold by RadioShack and other electronics stores, are made to hold custom electronics projects. The boxes come with removable metal or plastic lids to allow access to the inside. The plastic is easily drilled for mounting motors and other parts.

Clear or colored display domes: Also called hemisphere or half-round domes, display domes can be purchased in sizes from about 2” to over 12” in diameter. The dome can be used as the body of the robot or as a cover to protect its electronics. A “robotic ball” can be made by gluing two domes together. The wheels of the robot spin the ball, which in turn rolls on the floor.

Metal hardware parts: These include T-braces used for lumber framing in houses. Sizes and shapes vary greatly; take a stroll down the aisles at the hardware store and you’re sure to find plenty of candidates. There are lots of sizes to choose from, for making palm-sized robots to large 50- to 75-pound rovers. More about this idea later in the chapter.

Wide-mouth beverage bottle caps: Looking for cheap and easy wheels for your robot? Try the plastic cap of that beverage drink you just finished. Aim for the wide-mouth bottles, the ones with caps measuring 1-1/2” in diameter. These wheels are just about the perfect size for use with modified radio control servos. Mount a round servo horn to the inside of the cap. Hint: Steal the fat rubber band off a broccoli stalk for the tire.

PVC irrigation pipe: All forms of polygonal frames can be constructed using PVC irrigation pipe. Most hardware and plumber supply stores carry PVC pipe in various sizes and wall thicknesses. Select the pipe based on the size and weight of the robot. Obviously, you’ll need larger and thicker pipe for the big and heavy robots.

Experimenting with “No-Cut” Metal Platform Designs

Of all the aspects of robot building, cutting stuff up is my least favorite, especially if it involves metal. Most designs use stock metal of some kind: U-channel, tubing, strips, or large plates that must be cut down to size.

But what if you could find metal already in the size and shape you need for building robots? You can, but this stuff is found in a different part of the hardware store than the stock metal bins. And with it, you can construct “no-cut” metal platforms that require no (or very little) cutting to form into usable sizes and shapes.

The basic idea behind the no-cut is to use base materials that are already the proper size and shape. The parts of the robot—the motors, sensors, batteries, and so forth—can then be attached using fasteners, glue, hook-and-loop, double-sided foam tape, tie-wraps, or other techniques.

A prime source for materials for no-cut bases is the hardware store, but other outlets shouldn't be ignored. Keep your eyes open, and you'll note many ready-made components that can be used, without any additional sawing or sanding, for a robot base. Following is an example of a no-cut mobile robot design using commonly available (and inexpensively priced) metal pieces.

INTRODUCING THE MINI T-BOT

The Mini T-bot is made from a 6" strapping T (or *tee*), commonly used in lashing together pieces of lumber in a home. Strapping Ts are available in numerous sizes; the 6" size is the smallest that I've been able to locate, but they are also available up to 16". The size measures the top of the T; the vertical portion of the T is in various lengths, depending on the design.

One popular strapping T is the Simpson Strong-Tie T Strap. The brand doesn't matter; anything similar will do. The Mini T-bot uses the Simpson 66T, made of 14-gauge galvanized steel, and it measures 6" × 5", with a strap width of 1-1/2". The 66T, like most strapping Ts, has holes in it for nailing. The holes are offset and most will not line up with hardware you want to hang on your robot. You'll need to drill new holes. A power drill or, better yet, a drill press is recommended for drilling the holes.

MAKING THE MINI T-BOT

The basic layout template for the Mini T-bot is shown in Figure 17-1. The robot uses the following parts, in addition to the strapping T and assorted fastening hardware. You are, of course, free to substitute others you may have on hand or like better.

- Tamiya worm gear motors, #72004 (two)
- Tamiya narrow tires, 58mm diameter, #70145 (one set of two tires)
- Tamiya ball caster, #70144 (comes in sets of two, only one used)

The Tamiya parts can be purchased from most any online hobby retailer (see Appendix B, "Internet Parts Sources"). The motors are mounted on the ends of the cross, and the caster is

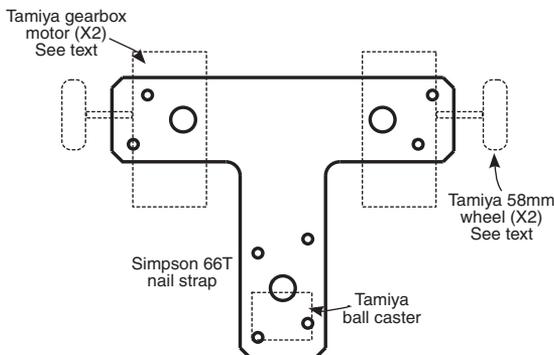


Figure 17-1 Design layout for the Mini T-bot. It uses a 6" T-shaped galvanized steel strap, found in the lumber section of your nearby hardware or home improvement store.

mounted at the base. The Tamiya caster offers the option of two heights; I selected the taller height to better match the wheelbase afforded by the motors and tires.

Only a few holes need be drilled in the strapping T. I used a 5/32” drill to make holes for 4-40 × 1/4” machine screws. The small fasteners and the somewhat larger holes provide some “slop” in mounting. With some wiggle room, you can better align the caster (not critical) and the two motors (critical).

Total weight of the Mini T-bot prototype, with 66T strapping T, motors, wheels, caster, battery holder, battery, 25-column breadboard, and assorted small switches, is 17.5 ounces (that’s 496 grams for you metric folks). Note that the four AA batteries alone contribute 3.5 ounces (about 100 grams) to the weight of the robot.

For your reference, here are the specifications of the most commonly available sizes of Simpson Strong-Tie strapping Ts, and their weight in ounces and grams. Larger robots can be built using bigger strapping Ts. The 1212T strap weighs almost a pound, so you need bigger motors (and batteries) to haul around that kind of weight.

Model	Material	L	H	W*	Weight
66T	14-gauge galvanized	6”	5”	1-1/2”	5 oz; 142 g
128T	14-gauge galvanized	12”	8”	2”	11 oz; 312 g
1212T	14-gauge galvanized	12”	12”	2”	14 oz; 397 g

* W is the width of the strapping metal.

USING LARGER Ts FOR LARGER BOTS

The robot brute in Figure 17-2 uses a pair of 1212T straps, separated by 5”-long aluminum tubing used as “risers.” In this particular prototype, the motors were mounted at an angle, with the metal of the lower T bent at 45°. This was partly done to accommodate the motor itself, as its mounting holes were on the side opposite the drive shaft and wheels.

As shown in Figure 17-3, you can use more conventional mounting, where the motor

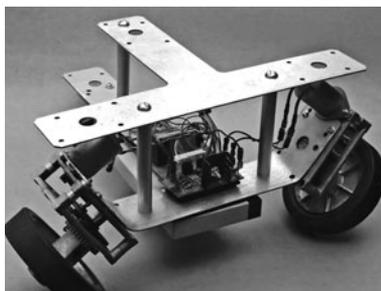


Figure 17-2 Larger version of the T-bot, made with a pair of 12” T straps. Because of the weight of the straps, motor, and battery, the bottom straps were bent upward to provide wheel camber, as well as a means to mount the motors.

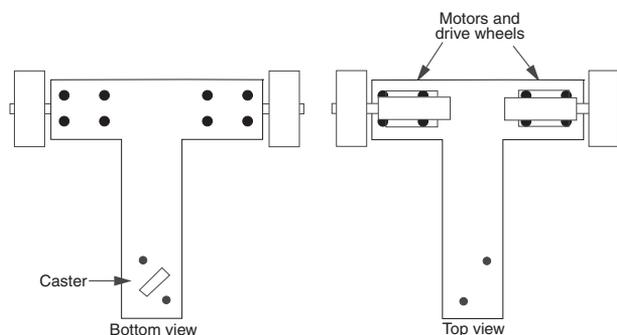


Figure 17-3 T-bot layout with conventional motor mounting.

frames are attached in parallel to the T-strap. Drill holes in the metal of the T strap to match those in the flange or mounting bracket of the motor you are using.

SHEET METAL FOR LUMBER STRAPPING

Don't stop with just T-shaped straps for building robot bodies. The same lumber section of your local home improvement store has plenty of other choices. Some of it is specially formed and bent pieces made for things like hanging 2-by-4 joists in an attic or garage. These are somewhat less useful than flat metal.

Figure 17-4 shows the outline drawing of three commonly available nail plates—so called because they're used to nail together two lengths of wood. As with the T-strap for the Mini T-bot, these are made by Simpson, but if you can't find this brand, there are other, similar products out there.

LSTA9 strap tie: Measures 9" × 1-1/4". Example uses: center rail in a walking robot; connecting strap for wood, metal, or plastic bases; side angle bracket for tracked bases.

66L L strap: L-shaped plate measures 6" on each side. If you need bigger, there's the 88L, which is 8" on each side. Example uses: mounting brackets for larger robots; out-riggers for motors.

TP37 and similar tie plate: Flat plate with different lengths to suit various applications. Width for all is 3-1/8": TP35 length: 5"; TP37 length: 7"; TP39 length: 9". Example uses: robot base; mounting plate for heavy parts (large motors, batteries) on framed robots; side panels.

Figure 17-5 shows how you might combine some of these sheet metal pieces to make various kinds of robotic bases. Or you could use the LSTA9 as a crosspiece for mounting motors or legs, or use it as a long side bracket for a tank-style robot.

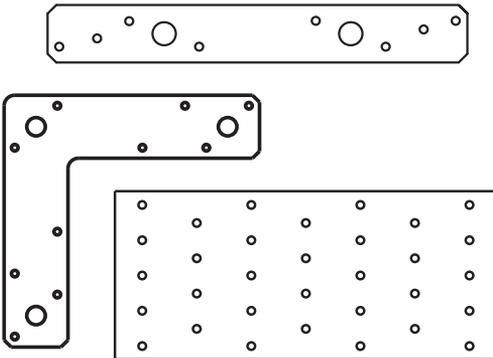


Figure 17-4 Outline shapes of three common nail plates. Like T straps, these are found in the lumber section of local hardware or home improvement stores.

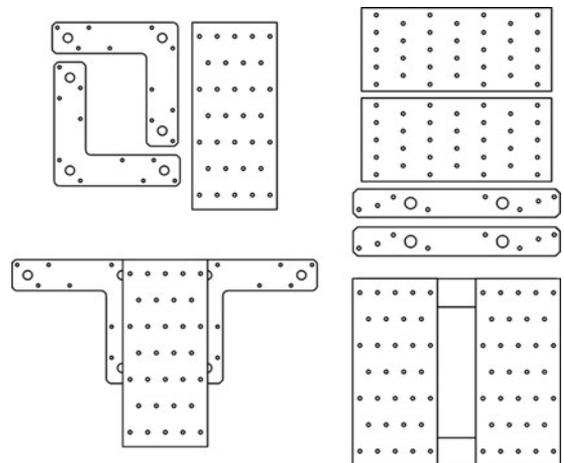


Figure 17-5 Two of an almost unlimited number of ways to combine nail plates to construct robot bases of all shapes and sizes.

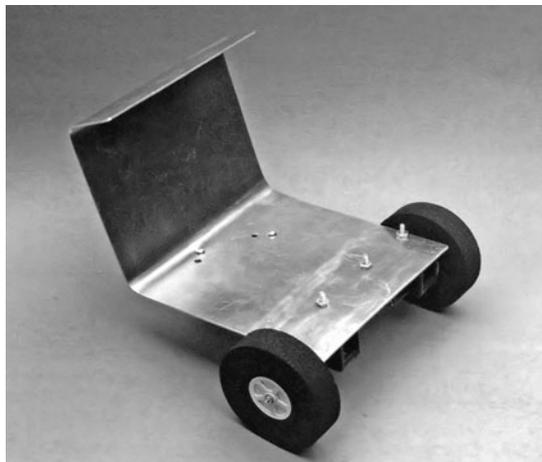


Figure 17-6 BuggyBot, made with a single sheet of metal hand-bent into a dune buggy-like shape. See the construction plans for this design at the RBB Online Support site (see Appendix A).



You may elect to cut or trim some of the pieces, but since they're already in the basic shape you need, there is less work required overall. Sheet metal for lumber strapping is typically 18 or 20 gauge and can be cut with a hacksaw, metal snips, or motorized (electrical or air-powered) shears.

GOING EVEN FURTHER

Of course, the concept of the no-cut extends beyond the Mini T-bot or the other strapping products detailed here. You can use the same idea for other robot designs made out of different metal materials, no matter where you find them. The key points to keep in mind are:

- The material should already be in the size you need, so no cutting is required.
- Drilling may be needed. Avoid materials that already have lots of holes. The holes may not line up with the motors and other components you wish to add, and the existing holes can cause trouble when drilling new ones so close by.
- Avoid very thick materials for small robots, as they add unnecessary weight.
- Consider sheet materials that can be bent to create unusual robot base shapes. An example is shown in Figure 17-6. This BuggyBot (featured in the second edition of this book) is made from an uncut 6" × 12" aluminum sheet purchased at the local hobby shop.

FYI

Check out the the RBB Online Support site (see Appendix A) for building instructions for the BuggyBot.

Using Wood and Plastic Samples

Walk through a well-stocked home improvement store and you're bound to find free or low-cost samples of wood and plastic products that you can reuse in your robots. For example, hardwood flooring samples are about the right size for a small robot. If not free then the cost for samples is quite low, maybe a dollar or so for a piece of wood that measures about 4" × 8" (dimensions vary depending on the manufacturer).

Most hardwood flooring is a laminate of a thin veneer over a sheet of high-density board. Thickness: 1/4". The sample usually includes the tongue-and-groove edges used to assemble the wood to make flooring; you'll want to cut or sand this part off. You'll also want to round off the corners to keep them from chipping.



If the board samples are too small, you can lash several together using something like the LSTA9 strap tie discussed in the previous section. Use short wood screws to hold things together or, for a less permanent construction, heavy-duty double-sided foam tape.

Other small-piece samples (usually available free or for a very small charge) can often be found in the kitchen cabinet department of the home improvement store. Look for 2" × 3" or larger samples of countertops made with Formica, resin, or other materials. These may be too small for building a robot base, but they're useful as housings for small sensors, backing material for touch switches, and other routine requirements.

Keep Your Eyes Peeled and Your Tape Measure Out

Before leaving the home improvement store, be sure to take one last stroll down the aisles. You'd be surprised what you'll find when you look at things from a robot builder's perspective. Be sure to check out the plumbing section—plastic pipe, pipe hangers, drainage pipe hole covers (they're large, they're round, they're funny looking, but they work), and other inexpensive knickknacks you probably don't know exist unless you're a plumber.

Though examining ersatz robotic parts firsthand is always preferred, if you can't get out to the home improvement store just now, take an online stroll to see what they offer. All the major home improvement and hardware chains have Web sites where you can browse by department. Bookmark items you think might be useful and check them out next time you're able to see them in person.



Found parts aren't just the domain of hardware and home improvement stores. Be on the lookout for unusual items that you can reapply in your next robotics project at the craft store (a natural), yardage and sewing outlets, resale stores (old junky VCRs and more), toy and housewares aisles of dollar and discount centers, sporting goods outlets and departments, and many, many others.

Power, Motors, and Locomotion

This page intentionally left blank

All about Batteries

Forget miniature atomic piles. Forget dilithium crystals. The robots in science fiction are seldom like the robots in real life. With few exceptions, today's robots run on batteries—the same batteries that power a flashlight, portable CD player, or cell phone. To robots, batteries are the elixir of life, and without them, robots cease to function.

To be sure, batteries may not represent the most exciting technology you'll incorporate into your robot. But selecting the right battery for your bot will go a long way toward enhancing the other parts that *are* more interesting. Here's what you need to know.

An Overview of Power Sources

Before getting waist-deep in the big muddy of battery selection, let's first review the practical power sources available for use with mobile robots. Note the word *practical*, m'kay? There are plenty of potential power sources available in the world. Some forms of power are not suitable because of their size, safety, or cost.

- *Windup mechanisms* provide power using tension that is slowly released. A common type is based on the idea of a clock mainspring. These use a metal coil as a tension spring. The coil powers a shaft or other movement as its tension is relieved. For robotics, the typical windup mechanism is confined to small toys, particularly older collectable toys.
- *Solar cells* get their power from the sun and other light sources. A disadvantage of solar cells is that power is directly related to the intensity of the light. Robots that use solar power are often equipped with a rechargeable battery or a large capacitor; both store the energy collected by the solar cell for later use.
- *Fuel cells* are gaining in popularity as an alternative energy source. Most use hydrogen in a complex chemical reaction that produces heat, as well as a flow of electricity between two electrodes.

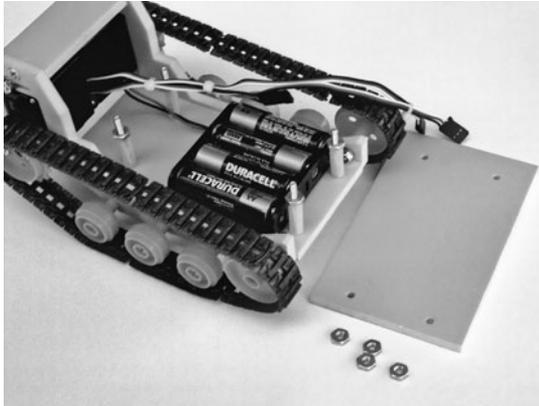


Figure 18-1 Desktop robot (this one uses tracks) hides its battery supply under an expansion panel. There's room for additional batteries (as needed) or a larger battery holder.

- *Batteries* are by far the most common and among the least expensive methods of powering any mobile device. Batteries can be grouped into two broad categories: nonrechargeable and rechargeable. Both have their place in robotics, and cost and convenience are the primary factors dictating which to use. These issues are discussed throughout the chapter. Figure 18-1 shows a robot and its power source—ordinary household batteries in a convenient holder.

Batteries for Your Robots

While there are hundreds of battery compositions, only a small few are ideal for amateur robots.

CARBON-ZINC

Carbon-zinc batteries are also known as garden-variety “flashlight” cells: because that’s the best application for them—operating a flashlight. They’re an old technology and not up to the task of running a robot. Let’s move on.

ALKALINE

Alkaline batteries (not to be confused with Hall-of-Famer Al Kaline of the Detroit Tigers) offer several times the operating capacity of carbon-zinc and are the most popular nonrechargeable battery used today. Robotics applications tend to discharge even alkaline batteries rather quickly, so a bot that gets played with a lot will run through its fair share of cells. Good performance, but at a price.



Alkalines are also available in a super-duper form; these go by a variety of self-descriptive names, like *Monster* and *Ultra*. High-capacity alkalines are made for loads demanding higher power. They’re pretty expensive, though, making them best suited as emergency backup power, in case your regular robo batteries get unexpectedly worn out.

RECHARGEABLE ALKALINE

Rechargeable alkaline batteries are the mass-merchandizing answer to the high cost of regular alkaline batteries used in high-demand applications—robotics is certainly one such application, though battery makers had things like portable CD players in mind when they designed these puppies. Rechargeable alkalines require a recharger designed for them and can be revived dozens or hundreds of times before discarding.

Rechargeable alkalines are probably the best choice as direct replacements for regular alkaline cells. The reason: Most rechargeable batteries put out about 1.2 volts per cell; both rechargeable and nonrechargeable alkalines are rated at 1.5 volts per cell. See “Understanding Battery Ratings,” later in this chapter, for more details about cell voltage.

NICKEL-CADMIUM

Nickel-cadmium rechargeable batteries are an old technology and, unfortunately, one that’s unfriendly to the environment—cadmium is extremely toxic. Lately, battery companies have been favoring the “greener” nickel-metal hydride formulation that follows. Nickel-cadmium batteries are still plentiful, and you’ll likely use them for at least some of your projects.

Nickel-cadmium (NiCd for short) cells are available in all standard sizes, plus special-purpose “sub” sizes for use in sealed battery packs for consumer products—things like rechargeable handheld vacuum cleaners, cordless phones, and so forth. Most battery manufacturers claim their NiCd cells last for a thousand or more recharges.

A new, higher-capacity NiCd battery is available that offers two to three times the service life of regular nickel-cadmium cells. More important, these high-capacity cells provide considerably more power and are ideally suited for robotics work. Of course, they cost more.

Earlier NiCd batteries suffered from “memory effect,” whereby the useful capacity of the battery was reduced the more times it was recharged. The newer NiCd batteries—those made within the past 10 years or so—are said not to exhibit this memory effect, or at least not as much as the older variety.

NICKEL METAL HYDRIDE

Nickel metal hydride rechargeable batteries not only offer better performance than NiCds, they don’t make fish, animals, and people (as) sick when they are discarded in landfills. They are the premier choice in rechargeable batteries today, including robotics, but they’re not cheap. They require a recharger made for them (Figure 18-2). Many of the latest rechargers will work with rechargeable alkalines, NiCds, and NiMHs; just don’t use a NiCd-only recharger with NiMH.

NiMH batteries can be recharged 400 to 600 times and have what’s known as a low internal resistance. That means they can deliver high amounts of juice in a short period of time. Unlike NiCds, NiMH batteries of any type don’t exhibit a memory effect. NiMH cells can’t be recharged as many times as NiCd batteries: about 400 full charge cycles for NiMH, as opposed to 2000 cycles for NiCd.

LITHIUM-ION

Lithium-ion cells are frequently used in the rechargeable battery packs for laptop computers and high-end camcorders. They are the Mercedes-Benz of batteries and are surprisingly light-

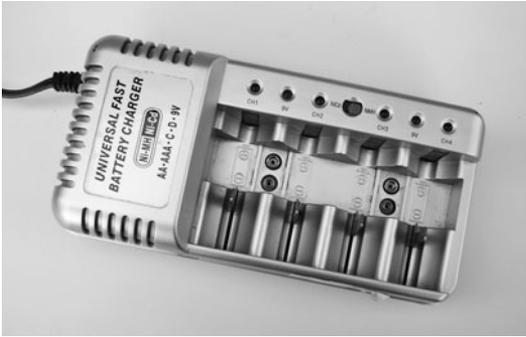


Figure 18-2 This universal charger works with both NiCd and NiMH batteries. You flip a switch depending on which kind you are recharging.



Figure 18-3 Sealed lead-acid (SLA) batteries are the largest and heaviest of the bunch, but they provide a lot of current, useful for larger robots.

weight for the current output they provide. However, Li-ion cells require specialized rechargers; both battery and recharger can be frightfully expensive.

SEALED LEAD-ACID (SLA)

Sealed lead-acid batteries are similar to the battery in your car, and they work in much the same way. SLAs are “sealed” to prevent most leaks, but in reality the battery contains tiny pores to allow oxygen into the cells. SLA batteries, which are rechargeable using simple circuits, are the ideal choice for very high current demands, such as battle bots or very large robots.

They’re also pretty inexpensive for the amount of juice they pack. However, they are among the heaviest of all the rechargeables, so use them only in robots that can support the weight.

SLA batteries are most often sold in 6- and 12-volt packs, like the one in Figure 18-3 (24-volt and higher packs are also available, but not quite as common). Inside the battery are multiple 2-volt cells—the cells are combined to create the desired voltage. Three cells are used to make a 6-volt pack, for example.

SO WHICH ONE SHOULD I PICK?

Most experienced builders select from a small palette of battery types based on the size and application of their robot.

- Alkaline, NiCd, and NiMH batteries are the most common among small tabletop robots. When using alkalines, you may choose between the rechargeable and nonrechargeable types. Nonrechargeable alkalines are a convenience, but using them can be expensive if you need to replace them often. Opt for rechargeable alkalines to save some money, or switch to NiMH or NiCd cells.
- Midsize “rover” robots use larger NiMH or Li-ion cells; bigger robots still are ideally suited for sealed lead-acid batteries. SLA batteries are available in a wide variety of capacities, and the capacity largely determines the size and weight of the battery. More about battery capacity in a bit.

- Because of the power demands, larger robots, such as those for machine combat, use sealed lead-acid batteries whenever possible. The biggest and most brutish robots can use liquid electrolyte batteries originally intended for use in motorcycles, small boats, or cars. These batteries are the heaviest of the bunch, but they pack a lot of wallop.



Which is better for the environment: nickel-cadmium or nickel-metal hydride? Actually, both contain poisonous materials, and both pose a threat. Never throw away your batteries, no matter what they're made of. Recycle instead. If you have a choice, NiMH batteries are probably the best bet, but remember to recycle those, too.

Understanding Battery Ratings

Batteries carry all sorts of ratings and specifications. The two most critical are voltage and capacity.

VOLTAGE

The importance of *voltage* (or *V* for short) is obvious: the battery must deliver enough volts to operate whatever circuit it's connected to. A 12-volt system is best powered by a 12-volt battery. Lower voltages won't adequately power the circuit, and higher voltages may require voltage reduction or regulation, both of which entail some loss of efficiency.

Nominal ("Normal") Voltage Level

Battery voltage is not absolute. The voltage of a battery may—and usually does—diminish as it is used.

Take a battery that's rated at 1.5 volts. It puts out 1.5 volts, give or take. That "give or take" is important; the rated voltage of a battery may vary as much as 10 to 30 percent. When fully charged, the typical 1.5-volt cell may deliver 1.65 volts. When fully discharged, the voltage may drop to 1.2 volts.

Batteries are rated at a *nominal* voltage (see Figure 18-4). Nominal simply means "normal." Only for a certain period during the battery's discharge does it actually deliver this specific voltage.

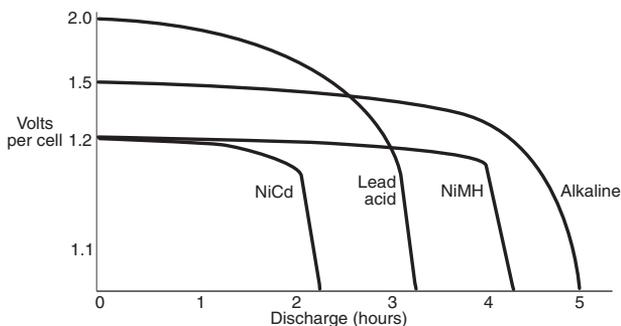


Figure 18-4 Simplified but representative discharge curves for several popular types of batteries used in robotics: NiCd, NiMH, alkaline, and lead-acid.

Battery Type*	Nominal Voltage
Alkaline	1.5 volts
NiCd, NiMH	1.2 volts
Sealed lead-acid	2 volts

* Individual cells.

To achieve higher voltages, you can link cells together, like lights on a Christmas tree. See “Increasing Battery Ratings,” later in this chapter, for more details. By linking together six 1.5-volt cells, for example, your battery “pack” will provide 9 volts.

When Decreasing Voltage Becomes a Problem

The varying voltage of a battery as it discharges doesn’t usually present a problem. That is, unless the voltage falls below a certain critical threshold. That depends on the design of your robot, but it usually affects the electronic subsystems the most.

Batteries are considered dead when their power level reaches about 70 to 80 percent of their rated voltage. That is, if the cell is rated at 6 volts, it’s considered “dead” when it puts out only 4.8 volts. Some equipment may still function below this level, but the efficiency of the battery is greatly diminished.

Most electronics systems in robots use a voltage regulator of some type, and this regulator requires some overhead . . . usually a volt or two. As the battery voltage drops below that needed for the regulator, the electronics go into a “brownout” mode, where they still receive power but not enough for reliable operation.

Brownouts are a common and damnable problem in robotics. You want to avoid this scourge at all costs. See Chapter 19, “Robot Power Systems,” for some ideas. If your robot has an onboard computer, you want to avoid running out of juice midway through some task. At best, this is a nuisance; at worst, damage to the robot or its surroundings could occur. See Chapter 19 for ways to add a simple battery voltage monitor to your bot.

CAPACITY

A common analogy for explaining electricity is to compare it to water going through a pipe. If voltage is pressure, then *current* is the amount of water that flows through every second.

Current in a battery determines the ability of the circuit it’s connected with to do heavy work. Higher currents can illuminate brighter lamps, move bigger motors, propel larger robots across the floor, and at higher speeds.

Because batteries cannot hold an infinite amount of energy, the current of a battery is most often referred to as an *energy store*, and is also referred to as *capacity*, abbreviated *C*.

Capacity Expressed in Amp-Hours

Battery capacity is rated in *amp-hours*, or roughly the amount of amperage (a measure of current) that can be delivered by the battery in a hypothetical one-hour period. In actuality, the amp-hour rating is an idealized specification: it’s really determined by discharging the battery over a 5- to 20-hour period, as shown in Figure 18-5.

What exactly does the term *amp-hour* mean? Basically, the battery will be able to (again, theoretically) provide the rated current for 1 hour. This current is expressed in *amps*—short

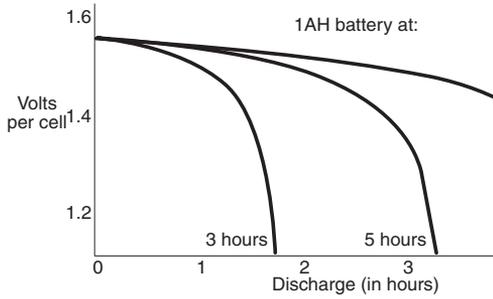


Figure 18-5 While batteries carry an “amp-hour” rating, it is actually tested under a lighter current load, for a longer period of time. Battery discharge at different current rates allows the battery to last longer.

for *amperes*—the common unit of expressing current flow from one part of an electrical circuit to another. (Technically, 1 amp is equal to about 6.24×10^{18} electrons passing by in 1 second. Pity the poor guy who had to count all those electrons.)

If a battery has a rating of 5 amp-hours (expressed as “Ah”), the battery can—*theoretically*—provide up to 5 amps continuously for 1 hour, 1 amp for 5 hours, and so forth.

The battery is used for 10 or 20 hours, at a fairly low discharge rate. After the specified time, the battery is tested to see how much capacity it has left. The rating of the battery is then calculated by taking the difference between the discharge rate and the reserve capacity and multiplying it by the number of hours under test.

Plan for Extra Capacity

When choosing a battery, select one that has a capacity of at least 40 percent (preferably more) than the highest current demand of your robot. Design the robot with the largest battery you think practical. If you find that the battery is way too large for the application, you can always swap it out for a smaller one. It’s harder to do the reverse.

Some components in your robot may draw excessive current when they are first switched on, then settle to a more reasonable level. Motors are a good example of this. A motor that draws 1 amp under load may actually require several amps at start-up, as shown in Figure 18-6. The period is very brief, on the order of 100 to 200 milliseconds.

The Dangers of Overdischarging

As a battery discharges, it produces heat. Not only is heat destructive to batteries (and, therefore, heat production may be intentionally limited by the design of the battery), but it alters the electrical characteristics of them.

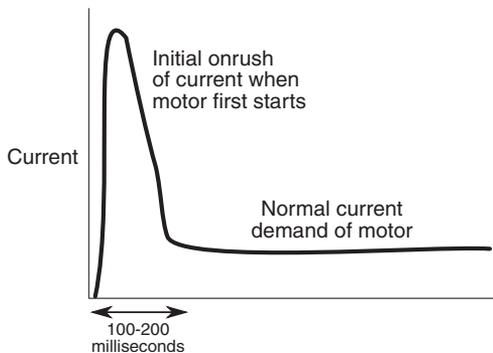


Figure 18-6 Current demand is the highest when electronic components, especially motors, are first switched on. The high onrush of current lasts a fraction of a second but can cause problems for the robot’s electronics.

The faster the discharge, the higher the heat that is generated. With few exceptions, batteries are not engineered to dump all their current in a short period of time. So manufacturers provide an idealized specification that more accurately represents the typical use of their wares.

Capacity Ratings for Smaller Batteries

Smaller batteries are not capable of producing high currents, and their specifications are listed in *milliamp-hours*. There are 1000 milliamps in 1 amp. So a battery that delivers half an amp is listed with a capacity of 500 milliamp-hours, abbreviated *mAh* (or less accurately as *mA*).

Even larger batteries might be rated in milliamp-hours, as is the case with rechargeable NiCd and NiMH cells, where it's not uncommon to see them listed as 2000 mAh—that's the equivalent of 2 amp-hours. Amp-hour ratings are typical in sealed lead-acid batteries. Amp-hours is commonly abbreviated as *Ah* or, less accurately, as *A*. For example: 500 mAh (500 milliamp-hours) or 3.5 Ah (3.5 amp-hours, or 3500 milliamp-hours).

Very occasionally, and for some applications, batteries may be rated in watts, though this is an imperfect measure. Technically, wattage is calculated at voltage times current, or $V * I$ (*I* stands for current—for the time being, don't worry about why; that's just the way it is). So a battery operating at a nominal 12 volts, delivering 2 amp-hours, is rated at 24 watts.

UNDERSTANDING INTERNAL RESISTANCE

Batteries, like humans fighting the Borg, resist—in this case, they resist giving up their charge. This phenomenon is called *internal resistance*; the higher it is, the less current a battery can deliver at any moment. The internal resistance of a battery determines the maximum rate at which current can be drawn from the cells.

- Alkaline and lithium-ion batteries have a *high internal resistance*. These can still deliver current, but they cannot “dump” all of it within a very short period. Think of them as long-distance runners.
- Lead-acid, NiCd, and NiMH batteries have *low internal resistance*. If necessary, these batteries will empty their charge within minutes, depending on the demands of the load. They're the sprinters of the track-and-field team.

Bigger batteries have more surface area inside them, which also affects internal resistance. That's why big batteries of any given type can power higher loads.

For most robotics applications, internal resistance isn't a major issue. When it really matters is for those tasks where extreme (and short-lived) current draw may be required. Examples are combat robots and the battery used to power the electric propeller on a self-guided drone plane.



Rapid, high current discharge of any battery can result in the cell overheating, fire, even explosion! That's why you never want to intentionally short out the terminals of a battery. This causes the battery to discharge all of its current as quickly as it can.

UNDERSTANDING BATTERY RECHARGE RATE

Most batteries are recharged more slowly than they are discharged. A good rule of thumb when recharging any battery is to limit the recharging level to one-half to one-tenth the

amp-hour rating of the cell. For a 5-amp-hour battery, a safe recharge level might be 500 to 2500 milliamps.

Limiting current is extremely important when recharging NiCd and NiMH cells, which can be permanently damaged if charged too quickly. Lead-acid batteries can take an occasional “fast-charge.” However, repeated quick-charging will lessen the life of the battery.

The recharge period, the number of hours the battery is recharged, varies depending on the type of cell. A recharge interval of 2 to 10 times the discharge rate is recommended.

Recharging Batteries

Nickel metal hydride, rechargeable alkalines, and rechargeable lithium-ion batteries all require special rechargers. Avoid substituting the wrong charger for the battery type you are using, or you run the risk of damaging the charger and/or the battery—and perhaps causing a fire.

Batteries are recharged by applying voltage and current to their power terminals. Exactly how much voltage, and how much current, depends on the type of battery. Some general tips and observations:

- Most lead-acid batteries can be recharged using a simple 200- to 1000-mA battery charger. The charger can even be a DC adapter for a video game, as long as the output voltage of the DC adapter is slightly higher than the voltage of the battery. Remove the battery from the recharger after 24 hours.
- Standard NiCd batteries like to be recharged slowly, typically with currents under 100 or 200 mA. Use a charger that supplies too much current, and you will destroy the cell.
- Rechargeable alkaline, NiMH, and lithium-ion must use a battery charger designed for them. High-capacity NiCd batteries can be charged at higher rates, and there are rechargers designed especially for them.
- Always observe polarity when recharging batteries. Inserting the cells backward in the recharger will destroy the batteries and possibly damage the recharger.
- Lithium-ion batteries can *catch fire* if they are incorrectly recharged. *Only* use a recharger specifically designed for the cell or battery pack you are using.

Robot Batteries at a Glance

As you’ve seen, batteries can be rechargeable or nonrechargeable. And different battery types also vary by the volts per cell. Table 18-1 shows, in a nutshell, the common battery types most often used in robotics, the nominal voltage they deliver per cell (when fully charged), and other selection criteria.

Common Battery Sizes

Battery sizes have been standardized for decades (see Figure 18-7), though most consumers are familiar with just a few of the more common types: N, AAA, AA, C, A, and 9-volt. There are many other “in-between” sizes as well.

For the most part, the size of the battery directly affects its capacity—assuming the same types of batteries are compared. For example, because a C battery provides roughly double

Table 18-1 Batteries and Their Ratings

Battery	Volts per Cell*	Application	Recharge†	Internal Resistance	Notes
Carbon-zinc	1.5	Low demand, flashlights—not robots	No	Moderate	Cheap, but not suitable for robotics or other high-current applications
Alkaline	1.5	Small appliance motors and electric circuits	No	High	Available everywhere; can get expensive (replacement costs) when used in a high-current application like robotics
Rechargeable alkaline	1.5	Substitute for nonrechargeable variety	Yes	High	Good alternative to nonrechargeable alkalines
High-capacity alkaline	1.5	Same as other alkaline cells, but can handle larger current demands	No	Low	More expensive than standard alkaline; keep for emergencies
NiCd	1.2	Medium and high current demand, including motors	Yes	Low	Slowly being phased out because of their toxicity
NiMH	1.2	High current demand, including motors	Yes	Low‡	High capacity; still a bit pricey
Li-ion	3.6§	High current demand, including motors	Yes	High	Expensive, but lightweight for their current capacity
Lead-acid	2.0	Very high current demand	Yes	Low	Heavy for their size, but very high capacities available

* Nominal volts per cell for typical batteries of that group. Higher voltages can be obtained by combining cells.

† Some nonrechargeable batteries can be “revitalized” by zapping them with a few volts over several hours. However, such batteries are not fully recharged with this method, and are recharged very quickly.

‡ Internal resistance of NiMH batteries starts out low when the cell is new, but increases significantly as it is recharged over many times.

§ Li-ion cells have different voltage characteristics, depending on manufacturer; 3.6 volts per cell is common but is not considered a standard. Li-ion batteries are almost always used in “smart” battery packs (they contain control circuitry). Packs designed to provide 7.2, 10.8, or 14.4 volts are typical.

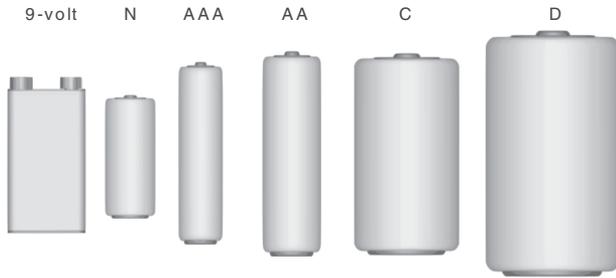


Figure 18-7 Comparison of consumer battery sizes, from N to D cell, plus 9-volt.

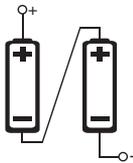
the internal area of an AA battery, it stands to reason that the capacity of a C battery is about twice that of the AA cell. (In actual practice, size versus capacity is more complicated than this, but it'll do for a basic comparison.)



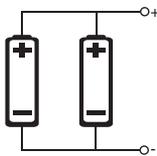
There's more about batteries, battery sizes, and selection on the RBB Online Support site (see Appendix A for more details).

Increasing Battery Ratings

Through the magic of electricity, you can get higher voltages and current by connecting two or more cells together. Increasing volts or current depends on how you wire the batteries together:



To increase voltage, connect the batteries in series. The resulting voltage is the sum of the voltage outputs of all the cells combined. This is the most common way to connect batteries.



To increase current, connect the batteries in parallel. The current is the sum of the current capacities of all the cells combined.

Take note: When you connect cells together, not all cells may be discharged or recharged at the same rate. This is particularly true if you combine two half-used batteries with two new ones. The new ones will do the lion's share of the work and won't last as long. Therefore, you should always replace or recharge all the cells at once. Similarly, if one or more of the cells in a battery pack are permanently damaged and can't deliver or take on a charge like the others, you should replace it.

Robot Power Systems

You learned about batteries in the preceding chapter. Now you can go about using them in your robot designs. That takes attaching the battery (or battery pack) to the robot and stringing wires from the battery to the motors and electronics of your robot.

Easily enough said, but the specifics are a bit more than this. This chapter cracks what you need to know to apply power systems to your robots.

FYI

This chapter makes reference to common electronic components, such as resistors, capacitors, and diodes. If you're new to these subjects, be sure to check out Chapter 30, "Building Robot Electronics—the Basics," and Chapter 31, "Common Electronic Components for Robotics." You may also want to check out the newbie lessons in *My First Robot*, located on the RBB Online Support site. See Appendix A.

Power and Battery Circuit Symbols

Let's start with the symbols used in electronic circuit designs to denote power and battery sources. Schematics are the road maps of electronic circuits. And in most circuits, it all starts with the power connection. For robotics, that power typically comes from batteries. Figure 19-1 shows the most commonly used symbols for power and batteries.

- When power comes from an arbitrary source (batteries, connection from wall transformer, whatever), it's often shown as a small circle, or sometimes an upward-pointing arrow. To complement the power connection, the circuit will also show another connection for *ground*. The exact form of the ground symbol is varied, but the set of three lines tapering to a point is among the most common.
- When power comes from a battery or battery pack, the symbol indicates the positive connection with a + (plus) sign. The - (negative) connection is inferred.

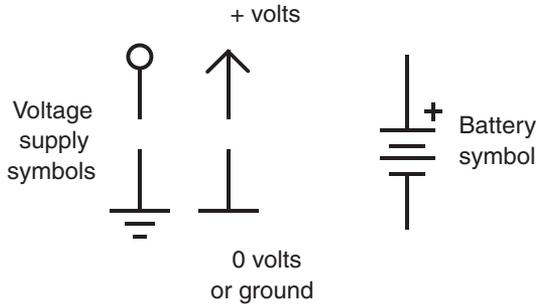


Figure 19-1 Various common battery symbols used in electronic diagrams. This book uses the ones on the left and right.



There are lots of different names for the *ground* connection. You may also encounter *common*, *negative*, *0 volts*, *0V*, *return*, *chassis*, and *earth*. These terms don't always mean the same thing, depending on the application. But for general electronics and robotics work, unless otherwise noted in the description of the circuit you can assume the terms are interchangeable.

Using a Premade Battery Pack

Many of today's consumer products use rechargeable battery packs. So can your robot. The battery pack can use cells that fit into all shapes and sizes of battery compartments. Wires and a quick-disconnect plug allow easy removal of the batteries, for things like recharging or replacement.

You can rob the battery pack out of a discarded consumer electronics device, such as a cordless phone you no longer need. You'll then have batteries for your robot, as well as the charger that came with the phone.

Or you can purchase replacement packs, though these can carry a premium price. This approach is worthwhile only if you already have a recharger for the pack. Otherwise, you're better off using battery packs designed for radio control (R/C) applications, as detailed next.

USING PREMADE R/C BATTERY PACKS



From
battery
pack

Radio control (R/C) applications, like model airplanes and model cars, are power hungry. High-capacity rechargeable battery packs are the norm. The packs are available in a variety of voltages and capacities from any local or online store specializing in R/C cars and planes. Common packs voltages are:

- 4.8 volt
- 7.2 volt
- 9.6 volt

The 7.2-volt packs are perhaps the most useful for robotics. Note the fractional voltages in this list. These are the result of the 1.2-volts-per-cell batteries used inside the packs. Current capacities range from about 350 milliamp-hours (mAh) to over 1500 mAh. The higher the current capacity, the longer the battery can provide juice to your robot. Unfortunately, higher-capacity batteries also tend to be larger and heavier.



You should always pick the capacity based on the estimated needs of your robot, rather than just selecting the biggest brute of a battery that you can find. Bigger batteries weigh more and they cost more. As you work more with robotics you'll get a sixth sense for the sizes of the batteries you need to power your creations.

PACKS WITH NICD OR NIMH BATTERIES

As you read in Chapter 18, “All About Batteries,” the two types of rechargeable batteries that are both easy to find and affordable are nickel-cadmium (NiCd) and nickel metal hydride (NiMH). Both are frequently used in R/C applications, and both can be recharged many times. Of the two, NiCd batteries are the least expensive because they've been around the longest. NiMH batteries provide for high capacities, with ratings of 900 to 3000 mAh, and over.

Both NiCd and NiMH battery packs require rechargers designed for them. The better battery rechargers work with a variety of pack voltages. I recommend these so you don't have to keep multiple chargers around.

Making Your Own Rechargeable Battery Pack

The shape and layout of your robot may make it difficult for you to use a standard-size battery pack or one of the battery holders detailed in the next section. You can always make your own battery pack, using rechargeable cells designed to be soldered together. These are special battery cells with solder terminals on them. They also come in so-called fractional sizes—a 2/3 AA cell is the same diameter as a standard AA cell, but roughly (about, kinda sorta) 2/3 the length.

You can arrange the cells in whatever layout best suits the space constraints of your robot. Remember that when batteries are connected one after the other in a string (series), the volts of each cell are added together. When the batteries are connected side by side (parallel; see Figure 19-2), the current-carrying capacity of the cells are added together.

Most battery packs use cells in series. Start with one cell, and wire its positive terminal to another cell's negative terminal, like that in Figure 19-3. Continue until all the cells are connected.



If you need to attach the cells using wire, match the gauge of the wire with the current demand from the battery. Thicker wire can handle more current. A good starting size is 14- or 16-gauge stranded (not solid) wire. See Appendix D, “Electronic Reference,” for a comparison of wire gauges. Just remember: the *lower* the number, the *larger* the wire.

After you've soldered the batteries together, use “battery shrink-wrap,” a PVC plastic tube to enclose everything into one nice package. Use a hair dryer on high to shrink up the plastic so it makes a snug fit around the batteries. But be sure the PVC doesn't get so hot it burns. Battery shrink-wrap is available at many hobby stores catering to builders of R/C airplanes and cars, as well as online battery outlets.

To complement the pack, you'll need a compatible battery charger that is designed not only for the total voltage of all the cells, but for the type of cells and their amp-hour capacity. Rechargers are available for either NiCd or NiMH (some are switchable), with common voltage ratings as follows:

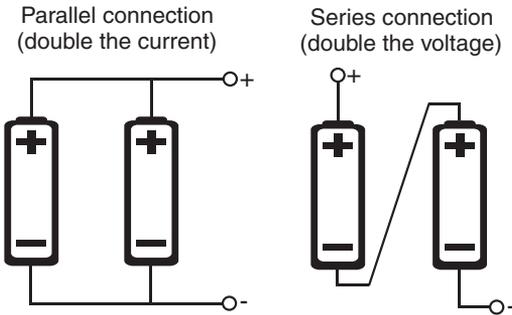


Figure 19-2 Batteries can be connected in series or in parallel. Parallel connection is seldom used, as current from the strongest battery is drained even when the pack is not connected to a circuit.

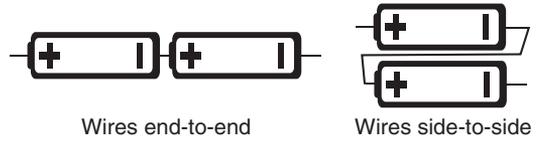


Figure 19-3 Batteries with solder tabs may be wired side to side or end to end, whatever makes the best shape for the intended use.

Voltage	Number of Cells	Voltage	Number of Cells
2.4	2	9.6	8
4.8	4	12	10
7.2	6	14.4	12

In-between voltages are also possible, using an odd number of cells. Seven cells of either NiCd or NiMH batteries is 8.4 volts, for example. You just need to make sure your battery charger is adjustable for that voltage. Some chargers will work with packs within a range—say, 9.6 to 18 volts. The recharger automatically adjusts to the proper voltage.

Use a polarized connector so you can easily hook up your battery to the rest of your robot. *Never* leave the wires bare, as this increases the chance of a short circuit. Shorting freshly charged batteries can cause fire or burns and can permanently damage the battery.

Using Battery Cells in a Battery Holder

Perhaps the most convenient method of using batteries with your robot is with a battery holder. Electrical contacts in the holder form the proper connections from the cell-to-cell; the voltage at the holder terminals is the sum of all the cells.

Holders are available for all the common battery sizes and even for some of the not-so-common ones. (However, if you plan on using the fractional AA size, or one of the subsizes, you are better off building a battery pack, as detailed previously.) Holders for two and four cells are among the most common. Battery holders come in either plastic or metal. The plastic holders are a tad more bulky, but they are lighter, cheaper, and easier to mount.

Battery holders often must conform to the shape of the object they are used in, so there are plenty of variations in how the cells are laid out—for example, a four-cell holder may orient the cells all in a single row, or it might pack them side by side. See Figure 19-4 for some examples of single-side battery holder layouts. There are even more variations for battery holders with cells on both sides.

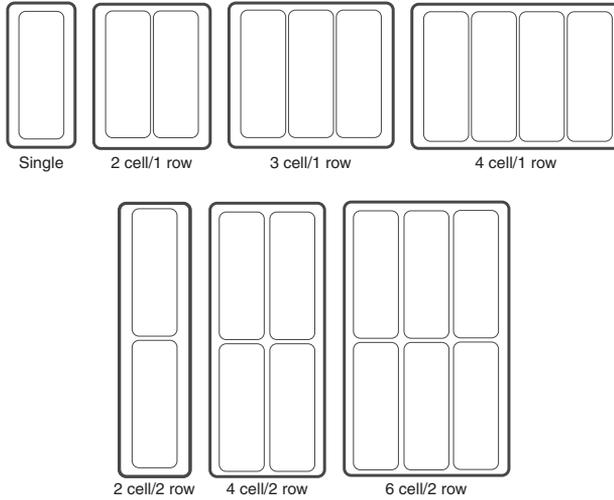


Figure 19-4 Sampling of battery holder layouts. Only a single side of the holder is shown here; the same layouts are available in double-sided holders, which carry twice the number of cells.

MOUNTING BATTERY HOLDERS TO YOUR ROBOT

Single- and multicell battery holders can be readily secured to your robot using any of three basic methods:

- *Fasteners provide a solid mounting.* Most holders have mounting holes for use with small machine screw hardware. Use 2-56 hardware, or drill out the holes for larger screws. Opt for flat-headed screws, and insert the screws through the holder first—this way the screw hardware will not interfere with the batteries.
- *Velcro, Dual Lock, or similar two-part hook-and-loop* allows you to mount the holder to your robot but still pull it off when you need to. The two parts of the hook-and-loop can be readily separated. Simply remove the holder, exchange the batteries, then press the holder back into place.
- *Double-sided adhesive or foam tape* provides a quick and easy method of mounting a battery holder. Like hook-and-loop, the mounting is semipermanent; even the strongest foam tape can often be dislodged with some effort. However, reserve this method for when you don't need to often remove the holder from its place in your robot.

Try to mount the holder in a location that allows for ready access to the battery cells, so you can easily get to them. As necessary, consider the underbelly of the robot. Most holders secure the cells with a tight fit, so you can mount the holder upside down and the batteries should not (normally) fall out.

SNAPS AND CLIPS FOR 9-VOLT BATTERIES

Use a polarized battery snap for 9-volt batteries. Secure the battery to your robot with a 9-volt battery clip; these are available for either front or side mount. I like the metal holders, as you can manually squeeze the tangs together to make for a tighter fit against the battery.

Mount the clip to the robot using double-sided foam tape, hook-and-loop, or fasteners. I

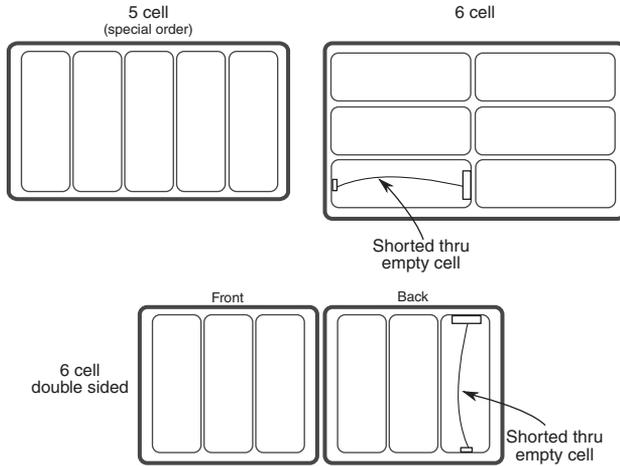


Figure 19-5 Create your own “odd-number cell” battery holders by bridging across the terminals of one of the cell pockets. This maintains electrical continuity for the holder but lets you use one fewer batteries.

prefer the fastener approach when a permanent installation is required and I want to make sure the battery stays put.

BATTERY HOLDERS FOR “IN-BETWEEN” VOLTAGES

Sometimes you may want an in-between voltage. Just use an odd number of cells. For instance, five rechargeable cells provides 6 volts; five nonrechargeable cells, 7.5 volts. While they do make battery holders for odd numbers of cells, they’re not always easy to find, and they tend to be more expensive.

One alternative is to use a standard even-cell holder and “bridge” one of its cell “pockets” to produce an odd-cell voltage. A four-cell AA holder provides 6 volts—too much for electronics designed for 5 volts. By removing one cell from the holder, the voltage is reduced to 4.5 volts, which is within the normal operating range.

Figure 19-5 shows the concept of bridging a cell pocket in a six-cell holder, which is commonly available in both single- and double-sided versions. You can bridge the pocket temporarily by using a jumper clip. For a permanent job you’ll want to solder a length of wire between the battery posts in the last cell pocket.



There’s more about batteries, battery holders, and battery packs on the RBB Online Support site (see Appendix A for more details).

Best Battery Placement Practices

For obvious reasons, the battery in your car is intentionally mounted for easy access. The same obvious reasons apply to robots. When possible, the battery holder or pack for your bot should be located where it affords quick and reliable access.

One ideal location for the battery pack or holder is on the underside of the robot. This assumes there is enough ground clearance between the robot and the floor. Mounting on the bottom allows for quick access to the cells, either for replacement or recharging: just turn the

robot over. And it saves space for electronics and other parts. Avoid any location that requires you to dismantle lots of parts of the robot just to access the batteries.

Wiring Batteries to Your Robot

There are lots and lots of ways to wire batteries to the rest of your robot. Picking the right method depends on the design and complexity of your bot.

In all cases, it's important to use a wiring system that eliminates the chance of short-circuiting the terminals of the batteries. It's also a good idea to use a connection scheme to prevent damage if the battery pack is connected to the robot circuitry in reverse, as detailed in the following section

Simple bots, replaceable cells. For the most basic robots, using battery holders with individual cells, you can merely solder the battery pack to the electronics and motors.

To replace or recharge the batteries, just remove them from the holder.

Solderless breadboards. Rather than solder to the electronics, insert the wires from the pack into the + and - rails of the breadboard. You'll need to solder a short (half-inch) length of 22-gauge solid conductor wire to the end of the battery pack leads, so you can plug into the breadboard.

Rechargeable packs with a standard 0.100" two-prong female plug. Use corresponding male header pins on your circuit or breadboard so you can readily connect and disconnect the batteries. Danger! This poses a chance of reverse polarity, so consider one of the solutions under "Preventing Reverse Battery Polarity" (later in this chapter).

Battery packs with a polarized plug. Use the corresponding socket on your circuit board or breadboard. A popular polarized connector found on DC wall transformers (see Figure 19-6) is the coaxial barrel plug, discussed in the next section. It can also be used on battery holders; use a six-cell (7.2- or 9-volt) holder and a 2.1mm barrel plug to power an Arduino board with batteries.

Battery packs with a customized polarized plug. You can make your own polarized connector systems or adapt one from the battery packs used in R/C model airplanes and cars. These are also discussed in the next section.



Figure 19-6 Wall transformer with a barrel plug. If purchasing a wall transformer for your robotics projects, make sure it provides the correct voltage and that the plug has the proper polarity to match your components.

Preventing Reverse Battery Polarity

At all costs, avoid reversing the connection of your batteries when you plug them into your robot. At best, your bot won't function; at worst (more likely), you'll instantaneously blow out some or all of the electronics. There are two general choices: mechanical interlock and electronic polarity protection.

MECHANICAL POLARIZED CONNECTIONS

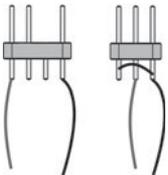
You can reduce and even eliminate the possibility of hooking up your batteries the wrong way simply by using a “polarized” connector. A coaxial barrel connector used with many low-voltage DC wall transformers is by far the simplest and most common. You need a mating socket on your circuit or breadboard.

Barrel plugs and sockets come in different diameters, sized in millimeters. You need to make sure the plug on your battery pack is the right size. The Arduino microcontroller boards, for example, use a 2.1mm barrel plug. Many sellers of the Arduino offer battery packs and 9-volt battery connectors with the 2.1mm barrel plug already on them.

The center connector on the plug can be + or -. When soldering a barrel plug to your battery pack (or using a wall transformer with a barrel plug already on it), *be absolutely sure to observe the correct polarity!* Center positive is the most common, but never assume that's the case. Always double-check.



Most wall transformers with barrel plugs indicate the polarity of the plug, using a pictogram like this one.  Double-check the polarity with your digital multimeter. If you're not sure how to check voltages see Chapter 30, “Building Robot Electronics—the Basics” for details.



You can make your own bare-bones polarized connections using ordinary 0.100” header pins and sockets. Some soldering is required. These also work with solderless breadboard circuits. The idea is to use three or four pins—rather than just two—and wire up the pins in a way that it's impossible to connect the battery incorrectly.

Three-wire polarized plug. Use a block of three 0.100” header pins. Solder the + lead from the battery pack to the center pin. Solder the - lead to the other pins.

When using a solderless breadboard, it's still possible to carelessly misalign the pins and cross up the polarity. Help reduce this by positioning the incoming power connectors to one end of the breadboard. Insert “dummy” (do-nothing) jumper wires immediately around it to prevent you from putting the battery plug in the wrong place (see Figure 19-7).

Four-wire polarized plug. Use a block of four 0.100” header pins. Solder the + and - leads from the battery pack to the two outside pins. Cut off one of the inside pins, and insert the cut pin into the corresponding female connector on your circuit or breadboard. The cut pin will prevent you from reversing the plug.

See Chapters 30 and 32 for more ideas on working with 0.100” header pin connectors. You can also buy polarized connectors that restrict or disallow you mating them backward.

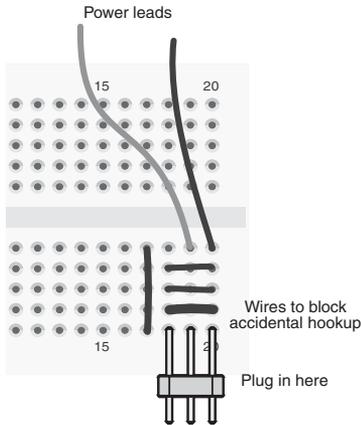


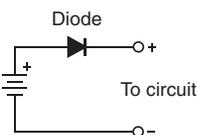
Figure 19-7 Use guard wires in a solderless breadboard to help prevent accidentally connecting the power plug to the wrong column of tie points.

Polarized connectors and wiring harnesses with polarized connections are available at local and online R/C model stores.

ELECTRONIC POLARITY PROTECTION

When you can't use mechanical means to avoid physically connecting the batteries backward, you can turn to some simple electronics that will protect the rest of your circuitry.

The simplest method is to put a diode inline with the incoming + connection from the battery. Diodes restrict current flow to one direction only. If you reverse the power leads from the battery, the diode will stop current from entering the circuit. Nothing will work, but your circuit won't be damaged.



Choose a diode with the current-carrying capacity for the circuit. The 1N4001 silicon diode handles up to 1 amp; the 1N5401 diode, up to 3 amps.

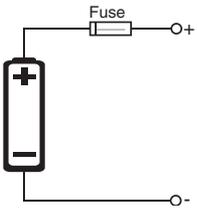
The downside to this method is that there is a voltage drop across the diode. For silicon diodes the drop is about 0.7 volts. For Schottky diodes the drop is only about 0.3 volts. See Chapter 31, "Common Electronic Components for Robotics," for information about different kinds of diodes.

On the Web: How to Solder a Barrel Plug onto a Battery Holder or DC Wall Transformer

Check out the RBB Online Support site (see Appendix A) for a step-by-step soldering guide on how to solder a barrel plug onto a battery holder or DC wall transformer. Make your own polarized power connectors.

Adding Fuse Protection

Big lead-acid, NiMH, and NiCd batteries can deliver a most shocking amount of current. In fact, if the leads of the battery accidentally touch each other, or there is a short in the circuit, the wires may melt and a fire could erupt.



That's why fuses were invented. Fuse protection helps eliminate the calamity of a short circuit or power overload in your robot. Connect the fuse in line with the positive terminal of the battery, as near to the battery as possible. You can purchase fuse holders that connect directly to the battery wire.

Choosing the right value of fuse can be a little tricky. It requires that you know *approximately* how much current your robot draws from the battery during normal and stalled motor operation. You can determine the value of the fuse by adding up the current draw of the various parts of your robot, then tacking on 20 to 25 percent overhead.



Assuming you're not building a big-brute combat bot, you can use your digital multimeter to determine the power draw of your robot. It's easiest if your meter has a 10-A (10-amp) current setting. See Chapter 21, "Choosing the Right Motor," for more details on how to connect your multimeter to read current draw.

Let's say that the two drive motors in the robot draw 2 amps each, the main circuit board draws 500 milliamps (0.5 amp), and other parts draw less than 1 amp. Add all these up and you get 5.5 amps. Installing a fuse with a rating of at least 6 amps will help ensure that the fuse won't burn out prematurely during normal operation. Adding that 20 to 25 percent margin calls for a 7.5- or 8-amp fuse. You may have to get the next-highest standard value.

Recall that motors draw lots of current when they are started. To compensate for the sudden inrush of current, use a "slow-blow" glass-type fuse. Otherwise, the fuse may burn out prematurely.

Fuses don't come in every conceivable size. For the sake of standardization, choose the 3AG fuse size—these fuses measure $1\text{-}1/4" \times 1/4"$. Holders for them are easy to find at any electronic parts seller.



An alternative to glass fuses is the *resettable PPTC fuse*. PPTC stands for "polymer positive temperature coefficient"—try saying that fast three times! These fuses are miniature electronic components that react to the heat caused by high currents. If too much current flows through the fuse, it "trips" momentarily, causing a break in the circuit. When the circuit fault (like a short circuit) is removed, the device cools back down, and it reconnects the circuit. PPTC (also referred to as PTC) fuses are smaller than standard glass fuses and are used in the same way.

Like standard glass fuses, you need to match the current rating of the resettable fuse to the highest acceptable current draw for your circuit. Select the fuse based on its *trip* current, the maximum current you want to allow. Devices are available with trip currents as low as 100 milliamps (0.1 amp) to over 50 amps.

Providing Multiple Voltages

Sometimes your robot needs more than one voltage. The electronics may require 5 volts, for instance, but the motors may need 10 or 12 volts.

Providing the proper voltages to the various subsystems in your robot requires some careful planning. Here are four approaches to powering the various components in your robot. Each one has its place, depending on the overall design of your robot and the power demands of each of its subsystems.

SINGLE BATTERY; SINGLE VOLTAGE

Depending on your robot, you may be able to use just a single battery and single voltage for everything. An example is if your electronics might be okay using more than 5 volts—possible for very simple electronics, such as the LM555 timer chip, which can operate at up to 18 volts. Since applying excessive voltage to electronics can damage them, always check the specifications first.

One disadvantage of using a single battery for both electronics and motor is that DC motors—especially the bigger ones—produce a lot of electrical noise that can disrupt the operation of microcontrollers and computers.

If you plan on operating your robot from a single, nonregulated battery pack, be sure to add noise suppression to the motors. It's easy: solder 0.1 μF nonpolarized disc capacitors across the terminals of the motor. Then, for good measure, solder a 0.1 μF nonpolarized disc capacitor from each terminal to the metal case of the motor. See Chapter 21, "Choosing the Right Motor," for information on this topic.

SINGLE BATTERY; MULTIPLE VOLTAGES

Most of the electrical equipment in your home or office is operated from one voltage, provided by the sockets in the wall. Each piece of equipment, in turn, uses this voltage as is (as in the case of an electrically powered fan), or it converts the incoming voltage to whatever level it needs.

This same approach can be used in your robot. A single battery—delivering, say, 12 volts—powers the different subsystems. A voltage regulator is used to provide each subsystem with the precise voltage it requires. There's more about voltage regulation in the section "Regulating Voltage," later in this chapter.

MULTIPLE BATTERIES; MULTIPLE VOLTAGES

Another method, and my personal favorite, is to use separate batteries and battery packs, each one delivering the right voltage to its robot subsystem. One battery pack may power the main electronics of the robot; another may power the motors. This works out very well because the electronics often need regulation and the motors do not. The pack for the electronics can be 6 or 9 volts (regulated to 5 volts), and the pack for the motors can deliver 12 volts.

This arrangement is more commonly referred to as a "split supply." The main advantage of this approach is that the two battery sources provide isolated power to each subsystem of the robot, eliminating or reducing potential issues. The trick to making this work is to connect all the ground (negative terminal) wires of the battery packs together. Each subsystem receives the proper voltage from its battery pack, but the shared grounds ensure that the various parts of your robot work together.

MULTIPLE BATTERIES; SPLIT VOLTAGES

And finally, you can "tap" voltages from a set of batteries connected in series, as shown in Figure 19-8. This figure illustrates using a 1.5-volt battery cell, but the idea works for any other battery voltage.

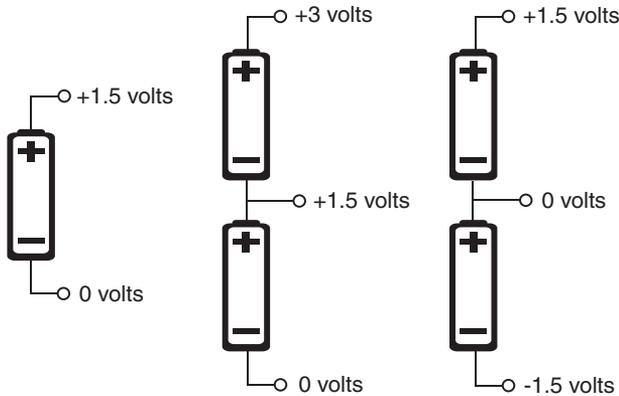


Figure 19-8 Separate cells may be tapped to derive several different voltages. Note that the ground connection is relative: when ground is the connection between batteries, you get both positive and negative voltages.

- With one cell you get the normal 1.5 volts.
- With two cells, you get 1.5 volts when tapping off the first cell, and 3 volts when tapping off the second cell.
- The polarity of the voltage taps is relative. If you have a two-cell power supply, like that shown, the center tap can be made ground, or 0 volts. That means connecting to the negative terminal of the bottom cell actually provides a negative voltage cell, or -1.5 volts. The positive terminal of the top cell provides $+1.5$ volts.

Of course, you are not limited to tapping between single cells. If you have eight cells, for instance, you could tap at every other cell, and get voltages of 3, 6, 9, and 12 volts. Or you can combine two 6-volt battery packs to produce 6- and 12-volt supplies, or -6 - and $+6$ -volt supplies. These types of *split* supplies are useful in some kinds of motor connection schemes. See Chapter 22, “Using DC Motors,” for ideas.

Regulating Voltage

The parts of your robot may need specific voltages to operate properly. This is most often the case with electronics, which typically require 5 or 3.3 volts. The exact voltage can vary a bit, but not much. For example, on a 5-volt system, the acceptable voltage levels may be from 4.5 to 5.5 volts. No higher, no lower.



DC motors typically don't require voltage regulation. Most run fine over a wide range of voltages. Increasing the voltage has the effect of making the motor run faster, and vice versa. R/C servo motors are an exception. Unless otherwise specified, they need to be operated at between around 4.5 and 7.2 volts.

Voltage regulation is accomplished in a number of ways. Here are the five most common; the first four are explained in more detail in this chapter.

- *Silicon diodes* can be used to drop the voltage in increments. This is not true regulation, but it can be used when all you need is a simple means to achieve a lower voltage. When put into a circuit, the typical silicon diode will reduce the voltage by about 0.7 volts.

- *Zener diodes* clamp the voltage to a specific level and won't let it get any higher.
- *Linear voltage regulators*, the most common variety, are cheap but relatively inefficient. In effect, they "step down" voltage from one level to another; the difference in voltage is dissipated as heat.
- *Switching voltage regulators* are more efficient—some boast up to 80 percent. They're recommended over linear voltage regulators, but they may require more external components to implement in your designs. Many switching regulators can increase voltage—boost 3 volts to 5 volts, for example—as well as produce negative voltages from a positive voltage source.
- *Modular DC-DC converters* are self-contained voltage changers. Internally they use one or more switching regulators, and they also include all the additional components required. They're more expensive.

DROPPING VOLTAGE WITH SILICON DIODES

Diodes are the simplest of all semiconductors. A common use of diodes is to prevent current from flowing a certain direction in a circuit. Current will flow only in the "allowed" direction, but in doing this, there's an inherent drop of voltage through the diode. For silicon diodes, the cheapest and most plentiful diode type, the voltage is reduced by about 0.7 volts.

Diodes don't really "regulate" voltage; they only drop it by approximate amounts. Not all circuits need exact voltage regulation, but if yours does, one of the other methods is recommended instead.

What's more, the actual voltage drop needs to be measured when the diode is connected in line with the circuit. The drop increases as the current load increases. The highest drop is at the rated current for the diode.

You can use this effect to easily drop a voltage in increments of *about* 0.7 volts. Using one diode, a 6-volt power supply is reduced to about 5.3 volts (see Figure 19-9). That's close enough to the +5 volts needed by most electronics. Using two diodes drops it down even further, to around 4.6 volts, and so on.

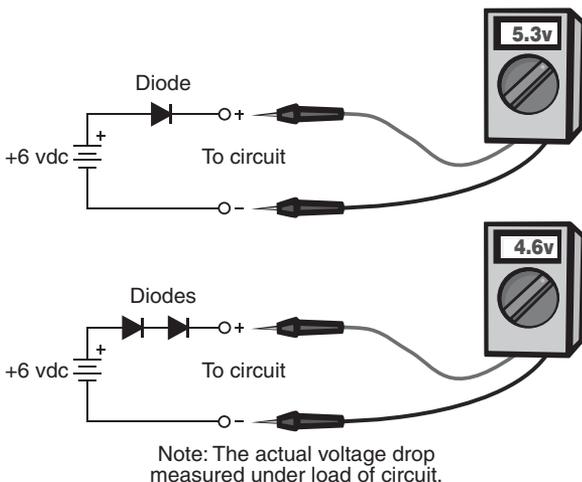


Figure 19-9 Diodes can be used to reduce the voltage to a circuit. Assuming silicon diodes, the voltage drop is about 0.7 volts. Combine diodes for higher voltage drops.

When selecting a diode for dropping voltage you need to also consider the current consumed by the circuit. Diodes are rated in amperes; pick a diode that can handle the current drawn by your circuit. The very common 1N4001 silicon diode can handle up to 1 amp, at 50 volts. The 1N5401 silicon diode handles up to 3 amps at 100 volts.



These types of diodes are often called *rectifiers*. Not all rectifier diodes are silicon. When in doubt, check the specs for the diode. Look at the *cut-in* or *forward voltage drop* specification. The voltage drop in most silicon diodes is about 0.7 volts; for Schottky and germanium diodes it's around 0.3 volts. LEDs, a type of diode, exhibit even higher forward voltage drops. But these generally cannot handle much current, so they're impractical as circuit voltage droppers.

ZENER DIODE VOLTAGE REGULATION

A quick and inexpensive method for providing a semiregulated voltage is to use zener diodes. A typical hookup diagram is shown in Figure 19-10. You can use zener regulation for circuits that don't consume a lot of power—say, under an amp or two.

With a zener diode, current does not begin to flow through the device until the voltage exceeds a certain level. This level is called the *breakdown voltage*. Any voltage over this level is then “shunted” through the zener diode, effectively limiting the voltage to the rest of the circuit. Zener diodes are available in a variety of voltages, such as 3.3 volts, 5.1 volts, 6.2 volts, and others. A 5.1 zener is well suited for use on circuits needing a +5 volt supply.

Zener diodes are rated by their tolerance—1 percent and 5 percent are common. If you need tighter regulation, get the 1 percent kind.

They're also specified by their power rating, expressed in watts. For low-current applications, a 0.25- or 0.5-watt zener should be sufficient; higher currents require larger 1-, 5-, and even 10-watt zeners. Note the resistor shown in Figure 19-10. It limits the current through the zener.

To calculate the value of this resistor, you need to know the maximum current draw of your circuit. You then do a bit of math:

1. Calculate the difference between the input voltage and the voltage rating of the zener diode. For example, suppose the input voltage is 7.2 volts, and you want to use a 5.1 volt zener:

$$7.2 - 5.1 = 2.1 \text{ volts}$$

2. Determine the current draw of your circuit. You want to add an overhead margin of about 200 percent. If, for example, the circuit draws 100 mA (milliamps), then

$$0.1 \times 2 = 0.2$$

In the preceding equation, 0.1 is 100 milliamps.

3. Determine the *value* of the resistor by dividing the current draw by the dropped-down voltage:

$$2.1 / 0.2 = 10.5 \text{ ohms}$$

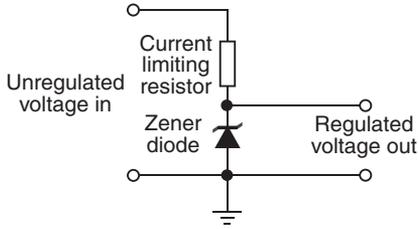


Figure 19-10 Zener diodes provide a simple form of voltage regulation. The value of the resistor depends on the incoming and regulated voltage levels and the amount of current the circuit is expected to draw. See the text on how to select the proper resistor value.

The nearest standard resistor value is 10 ohms, which is close enough.

4. Next, determine the *wattage* of the resistor. You do this by multiplying the difference in voltage from step 1, by the current draw of the circuit.

$$2.1 \times 0.2 = 0.42 \text{ watts}$$

Resistors are rated in watts and fractional watts: 1/8, 1/4, 1/2, 1, 2, and so on. Pick a resistor wattage at or above the calculated value. For this example, a 1/2-watt resistor will work.

- 5 Finally, determine the power dissipation for the zener diode. This is done by multiplying the current draw of the circuit by the voltage rating of the zener:

$$0.2 \times 5.1 = 1.02 \text{ watts}$$

You should use a zener rated at no less than 1 watt.



There is some simplification used in these calculation formulas—the reverse current of the zener is ignored, for example. But there are many approximations anyway, such as the overhead margin and use of standard component values. If you find your components get too hot, use a higher-wattage zener and resistor. If the voltage is too low, slightly decrease the value of the resistor.

LINEAR VOLTAGE REGULATION

Solid-state linear voltage regulators provide much more flexibility than zener regulators, and they're relatively inexpensive—a few dollars at the most. They are easy to get at any electronics parts outlet, and you can choose from among several styles and output capacities.

Two of the most popular voltage regulators, the 7805 and 7812, provide +5 volts and +12 volts, respectively (other voltages are available—just change the last two digits). You connect them to the + (positive) and – (negative or ground) rails of your robot, as shown in Figure 19-11. In normal practice, you also place some capacitors across the input and output of the regulator. These act to smooth out any instantaneous fluctuations in the voltage.

The smallest linear regulators are provided in the small TO-92 transistor package (see Chapter 31 for more on this). In fact, they look just like small transistors. For the 7805 and 7812 regulators, and depending on the manufacturer, these are often identified with an “L” within the part number, for example, 78L05. The TO-92 regulators are limited to use in circuits drawing 100 mA or less of current.

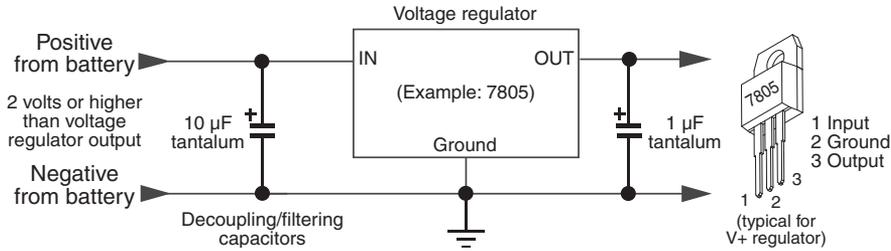


Figure 19-11 A linear voltage regulator requires no external parts, though it is customary (as good design) to include capacitors to help stabilize the voltage provided by the regulator.

More current is provided by regulators in the TO-220 style “power transistor” package. Use these when powering circuits that consume less than 500 milliamps. Other variations of the TO-220 style regulator can handle 1 amp or more. Check the datasheet that comes with the part you order.



Pinouts of voltage regulators can vary. Negative voltage regulators often have different pin assignments than positive voltage regulators. Reversing the connections to a voltage regulator will burn it out, so *always* double-check the datasheet for the device you are using. Always.

If you need more current, linear regulators are provided in larger TO-3 style transistor packages. They provide current outputs to several amps, depending on the exact device.

You can also get adjustable voltage regulators. These are some of the more common ones:

- The LM317T is an adjustable regulator in a TO-220 package. With some external parts (see its datasheet), you can adjust it to deliver from 1.2 volts to 37 volts, at a maximum of 1.5 amps.
- The LM317L offers the same voltage span, but is designed for use on circuits demanding 100 milliamps or less.



To enjoy the full current capacity of a voltage regulator you need to mount it on, or attach it to, a heat sink. Expect somewhat less current-carrying ability when a heat sink is not used.

Linear regulators require that the input be at least 1 volt, and usually 2 volts, over the expected output voltage. For example, for a 5-volt regulator, the unregulated supply should be about 2 volts higher, or 7 volts. By the same token, avoid applying too much voltage in relation to the output. The regulator throws off the extra voltage as heat, which is not only wasteful of energy but potentially harmful to the regulator. For that 5-volt regulator, the input voltage should be between 7 and 12 volts.

SWITCHING VOLTAGE REGULATION

Linear regulators take an incoming voltage and clamp it to some specific value. Linear regulation isn’t very efficient; as the voltage is stepped down to its desired level, excess energy (the difference between the input and output voltages) is wasted in the form of heat.

In AC-operated circuits this inefficiency is marginal, but it’s particularly notable in battery-powered systems, where battery life is limited.

Too, most linear regulators require a voltage source several volts higher than the expected output voltage. The difference is called *voltage drop*, and it's why you need at least 7 volts to power a 5-volt circuit.

An alternative to linear regulators is the *switching* (or switching-mode) voltage regulator, which is more expensive but more efficient. Most high-tech electronics equipment uses switching power supplies. They're common and not frightfully expensive.

A good example of a step-down switching voltage regulator is the MAX738, from Maxim. It comes in an 8-pin dual in line pin (DIP) IC package, among others; the DIP package is ideal for homebrew circuits. With just a few added parts you can build a simple, compact, and efficient voltage regulator. The output voltage is dependent on the external components that you use. Refer to the datasheet for the MAX738 for sample circuits.



Some kinds of switching voltage regulators can increase a lower voltage to a higher one. These are called *step-up* or *boost* regulators. One typical application is turning 3 volts from a pair of 1.5-volt alkaline cells into 5 volts, to run some microcontroller or other circuit. The Maxim MAX756 is a good example of a step-up switching regulator, able to turn any voltage from 1 to 5 volts into a regulated 5-volt output. Pretty nifty.

USING MULTIPLE VOLTAGE REGULATION

Not every subsystem in your robot will require the same voltage. Some parts might use 3.3 volts, while others use 5 volts. This requires that you use separate regulators for each voltage. Too, even if different subsystems require the same voltage, it's not generally a good idea to power them all from the same regulator. In many cases, it's better to use multiple regulators.

The concept of multiple regulation is simple, as shown in Figure 19-12. For each regulated subsystem you provide the regulator circuit. To prevent electrical noise from one subsystem from affecting another, be sure to add *decoupling capacitors* at each regulator. The capacitors effectively smooth out the voltage provided by the regulator, diminishing any instantaneous fluctuations in the power supply.

FYI

Read more about decoupling capacitors and how to use them, in Chapter 30, "Building Robot Electronics—the Basics." But, in a nutshell, a decoupling capacitor is the name given to a capacitor that is used to help filter out electrical noise in a power supply. The capacitor is placed between the positive and ground connections of the power supply.

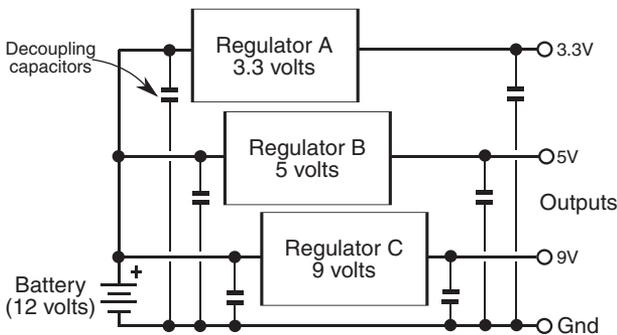


Figure 19-12 Use separate decoupling capacitors at the inputs and outputs of each separate voltage regulator used in your robot's power system. For extra measure you can add decoupling capacitors at the outputs. Typical capacitor values to use are shown in Figure 19-11.

Dealing with Power Brownouts

You see, robots aren't known to use power in an even, predictable manner. One moment the robot's sitting still, using very little power, waiting for the right time to pounce on your poor unsuspecting cat. The next moment it's screaming down the hall after the furry feline, burning amps like they're going out of style.

Every time the motors kick in, a high amount of current is drawn from the batteries. This increase in current consumption can make the voltage delivered by the batteries momentarily dip. If these same batteries also supply the electronics in your robot, a condition called *brownout*, or *sag*, could occur.

As the battery voltage drops below that needed for the regulator, the electronics go into a brownout mode. They get enough power to stay on, but not enough for reliable operation. To avoid possible brownouts:

- Use separate batteries for the electronics and the motors. This is the single best way to avoid brownout problems. Use a small-capacity battery or pack for the electronics, and a pack with larger AA, C, or D cells for the motors. Note: To make this work, *the ground leads of both battery supplies must be connected!*
- Use batteries with a higher per-cell voltage, to ensure enough overhead for proper voltage regulation. If you're using NiCd or NiMH cells, for instance, which put out 1.2 volts per cell, switch to rechargeable alkaline. These produce 1.5 volts per cell.
- Use one or more additional batteries to increase the voltage provided by the pack. Though nonstandard—and sometimes a bit hard to find—use a five-cell battery pack with your 1.2-volt NiCd or NiMH batteries. That increases the pack voltage from 4.8 volts to 6 volts.
- Power the electronics from a single 9-volt battery. Use appropriate voltage regulation, of course. The Arduino microcontroller boards have onboard regulation that will take the 9 volts input and provide the necessary 5 volts.
- Don't let your batteries get so discharged that they can't provide even the minimal operating current for your bot.
- Design for lower-voltage electronics. Some microcontrollers operate at 3.3 volts.

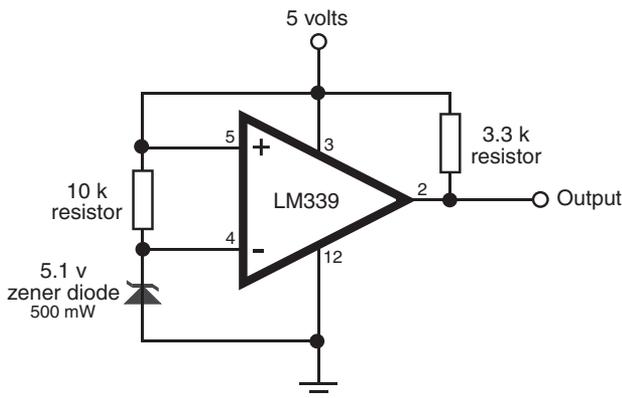


Figure 19-13 A voltage comparator and zener diode make for a convenient and easy-to-build battery-level monitor. You can adjust the trip point of the comparator—the input voltage at which its output changes—by experimenting with the value of the 10 kΩ resistor on the left.

Battery Voltage Monitors

Quick! What's the condition of the battery in your robot? With a battery monitor, you'd know in a flash. A battery monitor continually samples the output voltage of the battery during operation of the robot (the best time to test the battery) and provides a visual or logic output. A microcontroller-compatible battery monitor is shown in Figure 19-13. This monitor uses a 5.1-volt half-watt zener as a voltage reference along with an LM 339 quad comparator IC. Only one of the comparator circuits in the IC is used; you are free to use any of the remaining three for other applications.

The circuit is set to trip when the voltage sags below the (approximate) 5-volt threshold of the zener (in my test circuit the comparator tripped when the supply voltage dipped to under 4.5 volts). When this happens, the output of the comparator immediately drops to 0 volts.

Moving Your Robot

By definition, all mobile robots *move*. To propel themselves across the floor, a robot might use wheels, or perhaps tracks, maybe even legs. Moving your robot is called *locomotion*, and how these wheels, tracks, and legs are arranged is called the *drive geometry*. There are many variations of drive geometries, some relatively easy to achieve, and others not.

Selecting the right locomotion system and drive geometry involves figuring out what you want your robot to do. But it also takes assessing the mechanical requirements of constructing the drive mechanism. It's easy to "bite off more than you can chew," and design a robot propulsion system that is not practical for you to build. A good example is robots with legs. These are much harder to build than wheeled bots, and they often require regular maintenance to keep everything tight and aligned.

In this chapter you'll learn about locomotion systems and drive geometries, and how to select the best one for the robot you're building.



Chapters 26 and 27 in Part 4 contain additional specific details about using locomotion principles to power three common types of robots: wheeled, tracked, and legged. This chapter introduces you to the basic concepts, plus issues common to all three varieties of locomotion systems.

Choosing a Locomotion System

Let us take a quick look at the three main locomotion systems used with mobile robots. Don't worry if some of the terminology in the comparison table is new to you; I'll explain it to you as we go along.

Locomotion	Drive Considerations	Mechanical Considerations
Wheels	<ul style="list-style-type: none"> • Most common arrangement is 2 wheels on opposite sides of the base, with 1 or 2 casters or skids for balance. Typical variations include 4- and 6-wheeled bases. These do not require balancing casters/skids. • Size of wheel greatly influences traveling speed of robot. Larger wheels (for a given motor speed) make the robot go faster. • On 2-wheeled robots with support caster/skid, the wheels can be mounted centerline in the base or offset to the front or back. • Distance measuring (<i>odometry</i>) more reliable with wheeled bases. Accurate travel distance calculations are difficult with tracked and legged robots. 	<ul style="list-style-type: none"> • Mounting wheels to motors or wheels to shaft is the hardest part of building a wheeled base. R/C servo motors provide a consistent means for mounting small wheels to them, so these types of motors are quite common in small mobile robots. • Modest degree of accuracy needed in mounting the wheels to avoid <i>run-out</i>, side-to-side wobble as the wheel rotates.
Tracks	<ul style="list-style-type: none"> • The treads form a wide base that enhances stability of the vehicle. The mechanics of the treads creates a “virtual” wheel with a very large surface area that contacts the ground. • No need for a support caster or skid. • Though not as common, the treads may be augmented by wheels—similar to the half-track military vehicle. The treads are shorter and support only one end of the base. 	<ul style="list-style-type: none"> • Suitable tread material can be hard to find; most common approach is to hack a toy tank. • The large surface area of treads greatly increases friction; tracked vehicles can have trouble making turns, and the treads can pop off if they are made of flexible rubber. • Rubber treads (the most common on hacked toys) can stretch over time. A track tensioner mechanism is recommended.
Legs	<ul style="list-style-type: none"> • Variations include 2, 4, 6, and even 8 legs; 6 legs (<i>hexapod</i>) is the most common. • Most legged robots use <i>static</i> balance, meaning the arrangement of the legs on either side of the robot base prevents it from toppling over. More rare is <i>dynamic</i> balance, where weight on the base is shifted to compensate for stepping. • Joints of each leg are defined as <i>degrees of freedom (DOF)</i>: the more DOF, the more agile the platform, but the more difficult to build. 	<ul style="list-style-type: none"> • Of all locomotion types, legs require the greatest degree of machining and assembly. • Flexing of legs can cause stress in material; acrylic plastics can break over time. • Legs with independent articulation (each leg can move separately and independently) are the most difficult to construct. An easier alternative is the “linked gait” articulation, where the movements of legs are linked together. Fewer moving parts and motors required.

Locomotion Using Wheels

The drive geometry for robots that use wheels is defined by how each one is steered. There are a lot of choices.

DIFFERENTIAL STEERING

The most common way to move a robot is with *differential steering*. The most basic form consists of two wheels mounted on either side of the robot, as shown in Figure 20-1. It's called differential steering because the robot is steered by changing the speed and direction ("difference") between these two wheels.

A feature of most differentially steered robots is that they use one or two casters or skids, placed centerline over the robot in the front and/or back, to provide support for the base. See Chapter 26, "Build Robots with Wheels and Tracks," for more information on selecting and using casters and skids with your robot designs.

Variations of the two-wheeled base include four or six wheels (4WD, 6WD), but the idea is the same. On bases that use more than two wheels, support casters or skids aren't generally needed.

- One technique is the "dually" drive, where a single motor drives the two wheels on each side (see Figure 20-2). The wheels are linked together by a chain, belt, or gear system. The

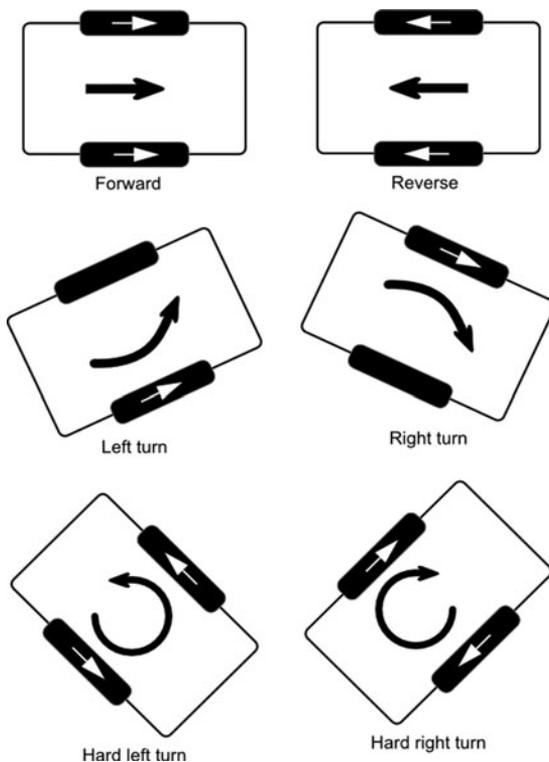


Figure 20-1 Differential steering involves using two motors on either side of the robot. The robot steers by changing the speed and direction of each motor.

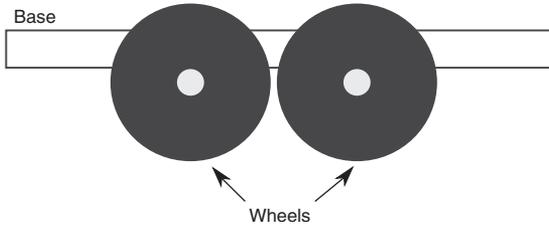


Figure 20-2 “Dually” wheels should be placed close to one another for maximum steering. If they are placed far apart, the robot cannot steer as easily.

wheels on each side are located close together. This aids in steering, where the robot will pivot at a virtual point midway between the two wheels on each side.

- Another technique in 4WD systems, easier to implement but more expensive, is to power each wheel with a separate motor. The motors on each side are powered as if they were one unit: both motors on the left turn on or off at the same time.
- And yet another technique for 4WD robots is to power only one motor per side and let the other one simply rotate freely.



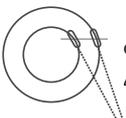
Please don't call this differential drive! Only robots that use a differential gearbox, like the one in an automobile, use “differential drive.” The correct term is differential steering—it's how the robot steers that defines how it putt-putts down the hall. In fact, you might have noticed that all of the drive geometries discussed in this section actually describe the way the robot steers.

One of the key benefits of differential steering is that the robot can spin in place by reversing one wheel relative to the other, as shown in Figure 20-3.

CAR-TYPE STEERING

Pivoting the wheels in the front is yet another method of steering a robot. Robots with *car-type* steering (see Figure 20-4) are not as maneuverable as differentially steered bots, but they are better suited for outdoor use, especially over rough terrain.

Why even bother with car-type steering? It seems so twentieth century. Well, there are a number of valid reasons to use it. One of the greatest drawbacks of the differentially steered robot is that it will veer off course if one motor is even a wee bit slow. You can compensate for this by monitoring the speed of both motors and ensuring that they operate at the same RPM. This, of course, adds to the complexity of the robot.



Somewhat better traction and steering accuracy are obtained if the wheel on the inside of the turn pivots to a greater extent than the wheel on the outside. This technique is called *Ackermann steering*, and it is found on most cars but not as many robots.

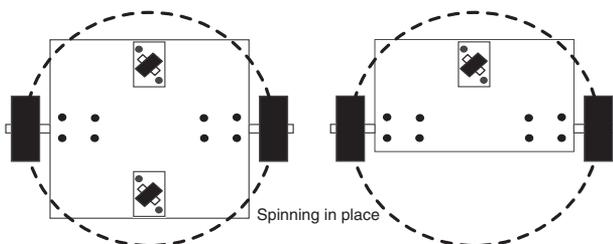


Figure 20-3 Differentially steered robots can turn in a circle within itself. This is called the steering circle, and the size of the circle depends on the dimensions of the robot and the placement of the wheels.

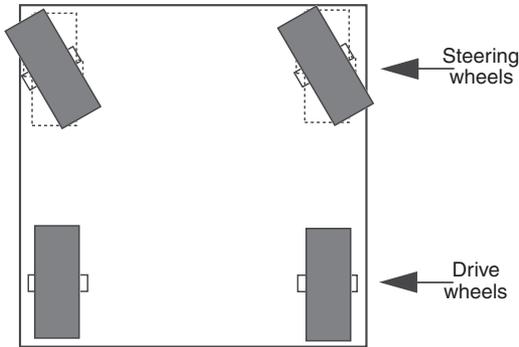


Figure 20-4 Car-type steering offers a workable solution for a robot used outdoors, but it's less useful for a robot used indoors or in places where there are many obstacles to steer around.

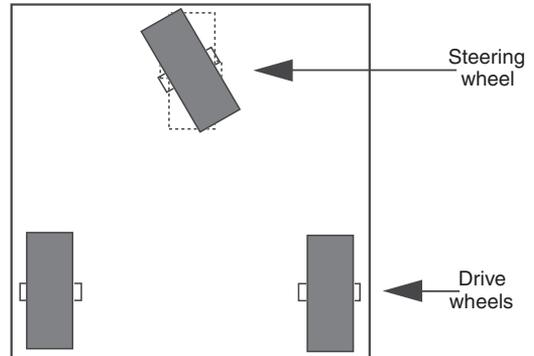


Figure 20-5 In tricycle steering, one drive motor powers the robot; a single wheel in front steers the robot. Beware of short wheelbases, as they can introduce tipping when the robot turns.

THREE-WHEELED TRICYCLE STEERING

Car-type steering makes for fairly cumbersome indoor mobile robots; a better approach is to use a single drive motor powering two rear wheels, and a single steering wheel in the front; the arrangement is just like a child's tricycle (Figure 20-5).

The robot can be steered in a circle just slightly larger than the width of the machine. Be careful of the wheel base of the robot (distance from the back wheels to the front steering wheel). A short base results in instability in turns, causing the robot to tip over in the direction of the turn.

Tricycle-steered robots require a very accurate steering motor in the front. The motor must be able to position the front wheel with subdegree accuracy. Otherwise, there is no guarantee the robot will be able to travel a straight line. Most often, the steering wheel is controlled by a servo motor; servo motors used a "closed-loop feedback" system that provides a high degree of positional accuracy.

There are two basic variations of tricycle drives:

- *Unpowered steered wheel.* The steering wheel pivots but is not powered. Drive for the robot is provided by one or two other wheels.
- *Powered steered wheel.* The steering wheel is also powered. The two other wheels freely rotate.

A subvariant of the tricycle base design reverses the functionality of the wheels: two wheels in the front of the robot steer, and a third wheel in the back provides support. The third wheel can even be a simple caster or omnidirectional ball (see the section on caster types, below).

OMNIDIRECTIONAL (HOLONOMIC) STEERING

All of the steering methods described so far are known as *nonholonomic*. This basically (and simplistically) means that in order for the robot to turn, it has to change the orientation of its body. A good example of nonholonomic steering is a car. It can turn, but only by following a circle described by the axis of its four wheels. The car cannot instantaneously move in any direction of the compass.

Holonomic drives are distinctive in that they allow motion in any direction, at any time. They can go straight ahead, then suddenly move 90° sideways—all without changing the orientation of the vehicle. A ball demonstrates holonomic movement: it can instantaneously travel straight, then move in any direction of the compass.

The common trait of holonomic steering systems is that the robot is omnidirectional, able to move in both the *x* and *y* directions with complete freedom. The most common form of holonomic robot base uses three motors and wheels, arranged in a triangle.

How Omnidirectional Steering Is Achieved

How exactly does this work? In the majority of robots that use holonomic steering, the secret is in the wheels. There are three wheels, each driven by a separate motor. Each wheel has rollers around its circumference—wheels within the wheels, as shown in Figure 20-6. The rollers are at some angle to the main wheel. The rollers provide traction to the wheel when the wheel is turning, but the rollers also let the wheel “slip” sideways to make turns.

The robot moves “forward” by activating any two motors; it turns by adjusting the speed or direction of any and all three of the motors (see Figure 20-7). These types of wheels were originally designed for materials handling, as a substitute for conveyor belts, but recently they’ve found new use in robot propulsion.

Other Forms of Omnidirectional Steering

Other forms of holonomic bases involve drive wheels that can each be independently rotated. These go by various names, such as *synchronized omnidirectional*. The rotation of each



Figure 20-6 An example wheel used in a holonomic robot. Instead of a solid rubber tire, the rim of the wheel is composed of several rollers that are set at opposite angles to the rotation of the wheel.

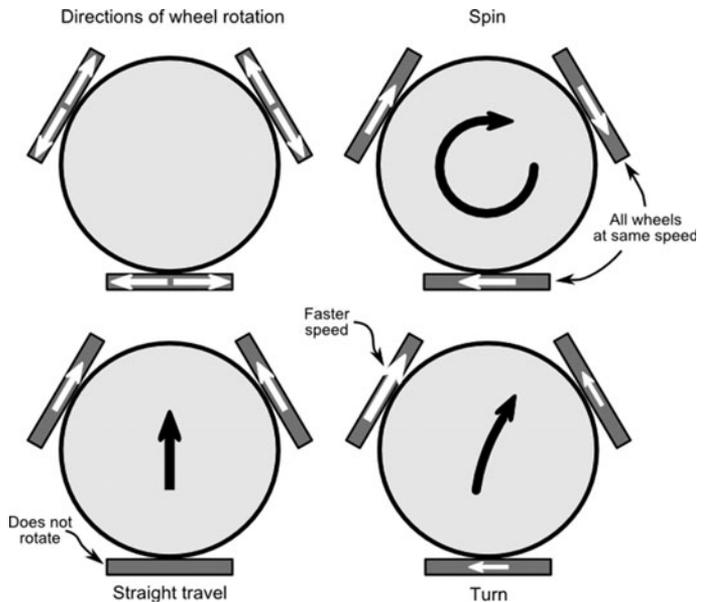


Figure 20-7 A holonomic vehicle steers by controlling the speed and direction of all of its wheels. For example, it goes forward by equally powering two wheels and letting the third “drag.” It spins by equally powering all three wheels. Turns are achieved by altering the speed and/or direction of each wheel.

drive wheel can be accomplished by separate steering motors or by one steering motor that is linked—via a chain or cogged belt—to all of the drive wheels.

Locomotion Using Tracks

Since World War I the tank has become the symbol of military battles. The hulking mass of iron plowing over the earth, the shrieking sound of metal crushing against the ground, the blasts of fire from its cannon, all combine a sense of awe and respect. Little wonder that the tank design is popular among robot builders. The same principles that make a military tank superior for uneven terrain apply to robots or, in fact, any other type of treaded vehicle.

Too, a number of robots from science fiction films run on tracks. There's Number Five from the movie *Short Circuit*, Robot B-9 from *Lost in Space*, and many others. Unlike their walking cousins, these robots actually look feasible. Their wide tracks provide a solid footing over the ground. Figure 20-8 shows plastic tracks on a small plastic robot.

Like the common two-wheeled bot, tracked robots are also differentially steered. Two long, chainlike *tracks*—also called *treads*—are mounted parallel to each side of the vehicle. A separate motor powers each track in either direction via a sprocket; the toothed design of the sprocket ensures that the drive mechanism doesn't just spin if the track gets jammed. The tracks are kept inline by the use of idler rollers, placed at intervals along the sides of the vehicle.



Tracks turn by skidding or slipping, and they are best used on surfaces such as carpet or dirt that readily allow low-friction steering. Very soft rubber treads will not steer well on smooth, hard surfaces. To reduce friction, one approach is to always steer by reversing the tread directions.

Because of the long length of the track, tank bots don't need a support caster or skid. The track acts as one giant wheel, one on each side.

- If both tracks move in the same direction, the robot is propelled in a straight line forward or backward.
- If one track is reversed, the robot turns (Figure 20-9).

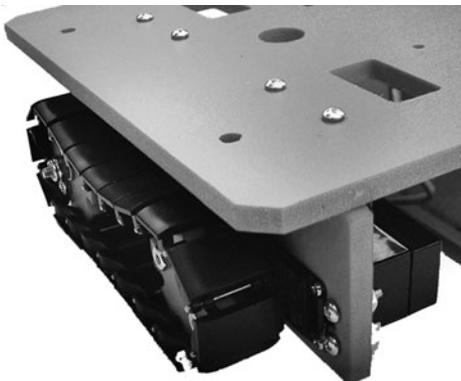


Figure 20-8 Individual links can be added or removed to lengthen or shorten these plastic tracks. Since the tracks don't stretch, they won't easily come off the drive sprocket of the robot. That can be a problem with rubber tracks.

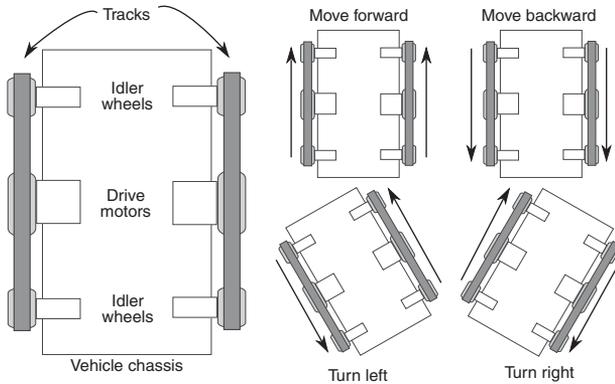


Figure 20-9 Differential steering using tracks is the same as with wheels, except that, to make a turn, one track needs to move in the opposite direction. If one track is stopped, the robot may “skitter” in the turn or the tracks may pop off.

This method is often referred to as *tank steering* or *skid steering*, but at the end of the day it’s the same as differential steering.

The main benefit of a tracked vehicle is its ability to navigate over rough terrain. The tracks enhance the “grip of the road,” allowing the robot to travel over loose dirt, sand, grass, and other surfaces that a wheeled robot can only dream about.

FYI

See Chapter 26, “Build Robots with Wheels and Tracks,” for additional information on track selection and use. You’ll also find several hands-on projects for building robots with tracks.

Locomotion Using Legs

Thanks to the ready availability of smart microcontrollers, along with the low cost of R/C (radio-controlled) servos, legged automatons like the one in Figure 20-10 are becoming a popular alternative for robot builders. Robots with legs require more precise construction than the average wheeled robot. They also tend to be more expensive.

Even a “basic” six-legged walking robot requires a minimum of 2 or 3 servos, with some six- and eight-leg designs requiring 12 or more motors. At about \$12 per servo (more for powerful higher-quality ones), the cost can add up quickly!



Figure 20-10 This six-legged hexapod robot uses 3 motors per leg, for a total of 18 motors. Because of the high number of motors, these kinds of robots are more difficult and more expensive to build. (Photo courtesy Lynxmotion.)

Obviously, the first design decision is the number of legs:

- Robots with one leg (“hoppers”) or two legs are the most difficult to build because of balance issues. In most two-legged bots the “feet” are oversized to offer the largest balancing area possible.
- Robots with four and six legs are more common. Six legs offer a static balance that ensures that the robot won’t easily fall over. At any one time, a minimum of three legs touch the ground, forming a stable tripod.
- Walking robots with eight (or more) legs are possible, but their construction cost and problems with higher weight make them largely impractical as a springboard for amateur robotics.

FYI

Check out Chapter 27, “Build Robots with Legs,” for more information on constructing multilegged robots.

Locomotion Using Other Methods

While robots with wheels, tracks, and legs are the most popular among robot builders, that doesn’t mean there aren’t other ways of moving a robot. Here are just a few alternatives that you might develop as you build up your robot construction skills:

- *Whegs* combine the action of wheels and legs into one unit. They’re a favorite at the Biologically Inspired Robotics Laboratory at Case Western Reserve University, where they’ve adapted the idea from several robots designed for space and military use. An attribute of most (but not all) whegs is that they are compliant, meaning there is built-in flexibility to conform to the terrain.
- *Flippers* are similar to whegs but are intended primarily for locomotion across very sandy terrain or water. Whegs and flippers share common traits, allowing for amphibious robots that can go from land to water.
- *Multisegment* robots mimic the locomotion of caterpillars, snakes, and other crawling creatures. The robot crawls by systematically moving each segment a little bit at a time. Figure 20-11 shows an example hobby robot that is equipped with five segments. On the bottom of the segments are “peds” that provide traction across the floor.

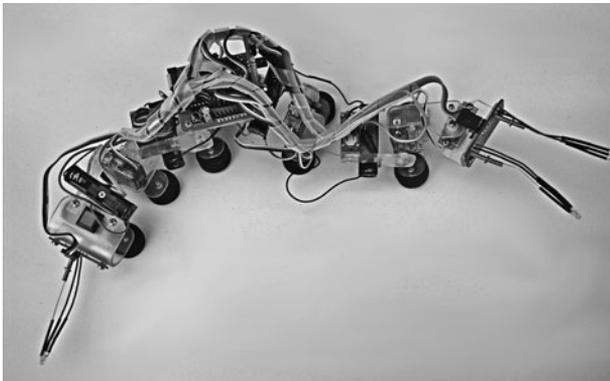


Figure 20-11 This segmented robot somewhat replicates the motion of a caterpillar. It uses radio control servo motors to move its segments side to side and up and down. Rubber pads on the bottom of the segments give it traction.

- *Hydra* robots combine two or more locomotion styles into one. For example, the robot may use articulated legs with wheels or small tracks as feet. Other types of hydra robots may combine the functionality of two front legs to serve as a gripper. The name hydra comes from Greek mythology; the Hydra was a beast with seven heads, who guarded the entrance to the underworld.
- *Unpiloted underwater* robots use various propulsion means, such as propellers and thrusters, to move around in fresh and saltwater.
- Similarly, *unpiloted flying* bots are scale versions of planes and helicopters.

On the Web: Managing the Weight of Your Robot

There's more than will fit here. Check out the RBB Online Support site (see Appendix A) for free articles on how to manage the weight of your robot and best distribute that weight to avoid tip-overs and other embarrassing design problems.

Choosing the Right Motor

Motors are the muscles of robots. Attach a motor to a set of wheels, and your robot can scoot around the floor. Attach a motor to a lever, and the shoulder joint for your robot can move up and down. Attach a motor to a roller, and the head of your robot can turn back and forth, scanning its environment. There are many kinds of motors; however, only a select few are truly suitable for homebrew robotics. In this chapter, we'll examine the various types of motors and how they are used.

AC or DC Motor?

Direct current—DC—dominates robotics; it's used as the main power source for operating the onboard electronics, for opening and closing relays, and, yes, for running motors that propel a robot across the floor. Figure 21-1 shows a gallery of over a half dozen small DC motors suitable for building small robots. They're inexpensive, and they adapt easily to most any robot design. You'll learn how in this chapter and the ones that follow.

The alternative motor type, alternating current (AC), is seldom used in robotics. AC motors are best suited for things like household fans and other applications where power comes from a wall socket.

FYI

Not sure what AC and DC mean? Refer to the lessons in *My First Robot*, found on the RBB Online Support site (see Appendix A). There you'll find a crash course in various electronics subjects.

When looking for DC-suitable motors, be sure the ones you buy are reversible. Few robotic applications call for motors that run in one direction only. DC motors are inherently bidirectional, but some design limitations may prevent reversibility, so this is something you have to be on the lookout for.

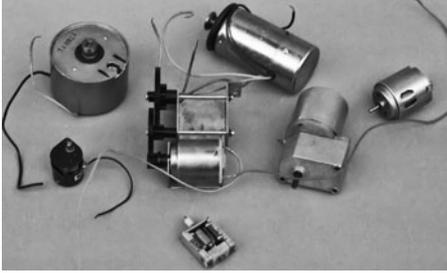


Figure 21-1 A menagerie of DC motors, with and without gear reduction boxes. These motors operate at low voltages, all of them under 12 volts and most under 6 volts—ideal for the typical battery-run robot.

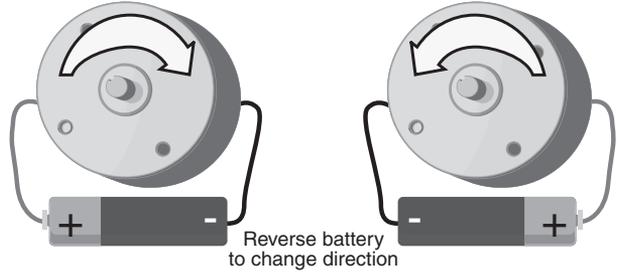


Figure 21-2 The direction of rotation of most DC motors may be altered simply by reversing the polarity of its power connections. With a battery or other DC power source connected one way, the motor spins clockwise. Reverse the polarity, and the motor spins counterclockwise.

The best and easiest test for reversibility is to try the motor with a suitable battery, as shown in Figure 21-2. Apply the power leads from the motor to the terminals of the battery or supply. Note the direction of rotation of the motor shaft. Now reverse the power leads from the motor. The motor shaft should rotate in reverse.

Continuous or Stepping Motor?

DC motors can be either continuous or stepping. Here is the difference: with a *continuous motor*, the application of power causes the shaft to rotate continuously (hence the name). The shaft stops only when the power is removed or if the motor is stalled because it can no longer drive the load attached to it.

With *stepping motors*, the application of power causes the shaft to rotate a few degrees, then stop. To keep rotating the shaft, power must be pulsed to the motor. Stepping motors are used when you want to control how far a motor turns, in either direction. They don't have the mechanical torque that a continuous DC motor has, but they're useful for certain tasks that don't need brute force. For example, one application is to spin a turret on top of a robot; on the turret might be a sensor of some kind or maybe a gun that shoots foam bullets. Chapter 22, "Using DC Motors," focuses entirely on continuous motors.



Stepping motors were covered in previous editions of *Robot Builder's Bonanza*. Because stepping motors are not as heavily used in robots as they used to be, the material has been removed from the printed book and is available on the RBB Online Support site (see Appendix A) as a free download.

Servo Motors

A special subset of continuous motors is the servo motor, which in typical cases combines a continuous DC motor with *electronic feedback* to ensure the accurate positioning of the motor. A common form of servo motor is the kind used in model and hobby radio control (R/C) cars and planes. We use these a lot in robotics, so this is a vital motor to get to know.

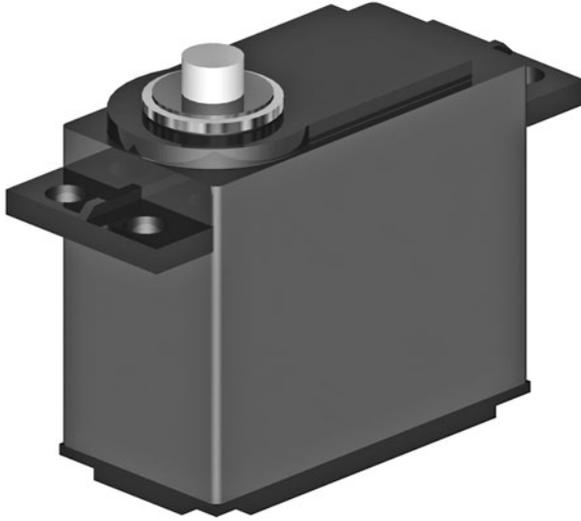


Figure 21-3 Servo motors for radio control (R/C) model airplanes and cars are DC motors, but with added control electronics and built-in gear reduction.

Compare the R/C servo motor in Figure 21-3 with the DC motors in Figure 21-1. The first thing you'll notice is that R/C servo motors come in a neat little rectangular box. This is one of their most alluring traits. Sizes of R/C servos are standardized (more or less), and they even have standardized mounting flanges, allowing you to easily add them to your robot creations.

R/C servos can be—and often are—used for the same jobs as stepper motors. Because they're cheaper, easier to find, and simpler to use than steppers, R/C motors have almost completely replaced the stepper motor in amateur robot designs.

For this reason, we devote a separate chapter just to them. See Chapter 23, “Using Servo Motors,” for more information on using R/C servo motors—not only to drive your robot creations across the floor but to operate robot legs, arms, hands, heads, and just about any other appendage.

Motor Specs

Motors come with extensive specifications. The meaning and purpose of some of the specifications are obvious, but for others, they aren't. Of all the specifications for motors, only a small handful are truly meaningful to the amateur robot builder, so I'll just concentrate on those. These specs are operating voltage, current draw, speed, and torque.

OPERATING VOLTAGE

All motors are rated by their *operating voltage*. With some small DC “hobby” motors, the rating is typically a range, something like 4.5 to 6 volts. For others, an exact voltage is specified. Either way, most DC motors will run at voltages higher or lower than those specified for it. A 6-volt motor is likely to run at 3 volts, but it won't be as powerful. It will also run slow.

Many motors will refuse to run, or will not run well, at voltages under about 40 or 50 percent of the specified rating. Similarly, a 6-volt motor is likely to run at 12 volts. As you may expect, as the speed of the motor increases, the motor will exhibit greater power.



I don't recommend that you run a motor continuously at more than 50 to 80 percent its rated voltage, at least not for long. The electrical windings inside the motor may overheat, which can cause permanent damage to the motor. Motors not designed for high-speed operation may turn faster than their construction allows, which literally could cause them to burn up.

If you don't know the voltage rating of a motor, you can take a wild guess at it by trying various voltages and seeing which one provides the greatest power with the least amount of heat and mechanical noise. Let the motor run for several minutes, then feel the heat on the outside of the motor case. Listen to the motor; it should not seem as if it is straining under the stress of high speeds.

CURRENT DRAW

Current draw is the amount of current, in milliamps or amps, that the motor requires from the power supply. Current draw is more important when the specification describes *motor loading*, that is, when the motor is turning something or doing work. The current draw of a free-running (no-load) motor can be quite low. But have that same motor spin a wheel propelling a robot across the floor, and the current draw might increase several hundred percent.

Most DC motors use a permanent magnet inside. In these motors, which are the most common, current draw increases with load. You can see this visually in Figure 21-4. The more the motor has to work to turn the shaft, the more current is required. The load used by the manufacturer when testing the motor doesn't follow any kind of standard, so in your application the current draw may be more or less than that specified.

A point is reached when the motor does all the work it can do and no more current will flow through it. The shaft stops rotating; the motor has *stalled*. This is considered the worst-case condition. The motor will never draw more than this current unless it is shorted out. If your robot is designed to handle the stall current, then it can handle anything.

When adding a motor to your robot you should always know the approximate current draw under load. All multimeters can be used to test current drawn by a motor. Learn how in the section "Testing Current Draw of a Motor," later in this chapter.

SPEED

The rotational *speed* of a motor is given in revolutions per minute, or *RPM*. Many continuous DC motors have a normal operating speed of 4000 to 7000 RPM. Certain special-purpose

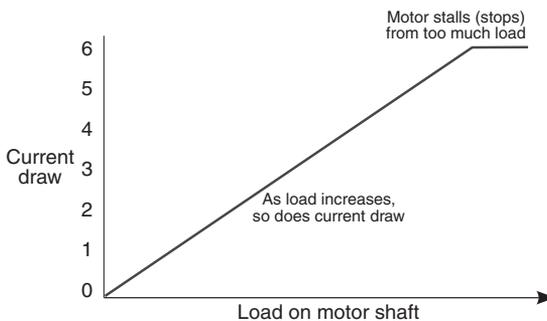


Figure 21-4 All motors draw current; the amount of current depends on the design of the motor and how much load is placed on the motor. As the load increases, so does the current. At some point, the load is too much for the motor and it stops turning, but it still consumes current. This is called the *stall current*.

motors, such as those used in tape recorders and computer disk drives, operate as slowly as 2000 to 3000 RPM, and there are motors that operate at 12,000 RPM and higher.

You don't need a lot of speed for most robotic applications. In fact, for DC motors used to move your robot, the speed of the motor needs to be reduced to no more than 200 or 250 RPM—often even slower.

The easiest way to slow down a motor is to attach some gears to it. This also increases the turning force (called *torque*; see the next section), allowing the motor to push bigger robots or lift heavier objects. Discover more about gears and how they're used in motors for robots in Chapter 24, "Mounting Motors and Wheels."

TORQUE

Torque is the force the motor exerts upon its *load*—the load is whatever it's moving. The higher the torque, the larger the load can be and the faster the motor will spin. Reduce the torque, and the motor slows down, straining under the workload. Reduce the torque even more, and the load may prove too demanding for the motor. The motor will stall to a grinding halt and, in doing so, eat up current—not to mention, put out a lot of heat.

Torque is perhaps the most confusing design aspect of motors. This is not because there is anything inherently difficult about it, but because motor manufacturers have yet to settle on a standard means of measurement. Motors made for industry are rated one way; motors for the military, another. And most motors for consumer or hobby applications come with no torque ratings at all.

At its most basic level, torque is measured by attaching a lever to the end of the motor shaft and a weight or gauge on the end of that lever, as depicted in Figure 21-5. The lever can be any number of lengths: 1 centimeter, 1 inch, or 1 foot. Remember this, because it plays an important role in torque measurement.

The weight can be either a hunk of lead or, more commonly, a spring-loaded scale, as shown in Figure 21-5. Turn the motor on, and it turns the lever. The amount of weight the motor lifts is its torque. There is more to motor testing than this, of course, but it'll do for the moment.

Now for the ratings game. Remember the length of the lever? That length is used in the torque specification.

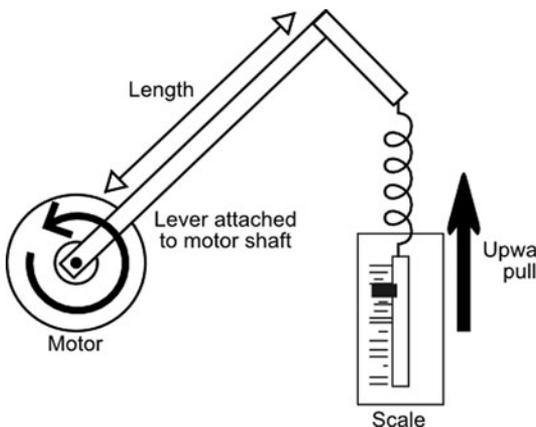


Figure 21-5 The power output or torque of a motor can be measured using a simple graduated spring scale (a fish-weighing scale will do). The motor is attached to the scale using a lever. The amount of pull plus the length of the lever are used to indicate the torque rating of the motor.

- If the lever is 1 inch long, and the weight successfully lifted is 2 ounces, then the motor is said to have a torque of 2 ounce-inches, or *oz-in*. (Some people reverse the “ounce” and “inches” and come up with “inch-ounces.” Whatever.)
- Or the torque may be stated in grams, not ounces. In this case, a lever calibrated in centimeters may be used. This gives you grams-centimeter, or *gm-cm*.
- Torque for very large motors may be rated in pound-feet, or *lb-ft*.
- Becoming more popular is the newton-meter unit of torque, slowly being adopted by motor manufacturers. You may see it as *N-m* or *Nm*. One N-m is equal to the torque that results from a force of 1 newton (no, not the fig kind) applied to a lever that is 1 meter long. (If you’re interested, the newton is equal to the amount of force required to accelerate a mass that weighs one kilogram at the rate of 1 meter per second per second.)

STALL OR RUNNING TORQUE

The typical motor is rated by its *running torque*—that is, the force it exerts as long as the shaft continues to rotate. For robotic applications, it’s the most important rating because it determines how large the load can be and still guarantee that the motor turns.

Manufacturers use a variety of techniques to measure running torque. The tests are impractical to duplicate in the home shop, unless you have an elaborate dynamometer and sundry other tools. Instead, you can empirically determine if the motors are sufficient for the job by constructing a simple test platform, as described in the next section.

Another torque specification, *stall torque*, is sometimes provided by the manufacturer instead of or in addition to running torque. Stall torque is the force exerted by the motor when the shaft is clamped tight. The motor does not turn.

JUDGING THE TORQUE OF MOTORS

If the motor(s) you are looking at don’t have running torque ratings, you must estimate their relative strength. There are formulas you can use to rate a motor for a specific task, and these have been detailed in numerous books on robotics; try *Building Robot Drive Trains* (Clark and Owings, McGraw-Hill, 2003). In truth, this is what most people do: they mount the candidate motors on a makeshift platform, attach wheels to them, and have the motors scoot the test bot across the floor. If the motors support the platform, start piling on weights. If the motors continue to operate with the additional load, then you know they’re suitable for the job. (This is called empirical evaluation. It’s how Thomas Edison did most of his inventing.)

You don’t need to build the whole robot just for this test. Employ a temporary construction using lightweight materials, such as heavy-duty cardboard or artists’ foam board. See Chapter 14, “Rapid Prototyping Methods,” for more information. With the techniques outlined in that chapter you can quickly test various motors and robot platform designs.

Such crude tests make more sense if you have a standard by which to judge others. If you’ve designed a robot before and are making another one, you’ll already know what kind of motors work for a robot of a general size and weight.

Testing Current Draw of a Motor

You can often just look at a motor and know it’ll have enough torque for your robot. Less ensured is knowing how much current the motor demands when it’s running. It’s not possible, or even advisable, to infer the current draw of a motor just by its size, shape, or type.

Knowing the current draw is very important: you need to make sure the batteries of your robot have the right capacity and that any electronics you use can handle the current. Should the motors draw too much current, your electronics could overheat and be permanently damaged.

Many motors are provided with a current draw specification. As noted earlier in this chapter, the spec may be the current of the motor when it's free-running. That's helpful information, but it's not enough to know how the motor will behave when it's pulling weight—or worse, if the motors are loaded down so much they stop turning.

You can use a multimeter to accurately test the current draw of your motor when it's free-running, under normal load, and even stalled—completely stopped. There are two methods, each described below.

FYI

If you are new to the concept of multimeters, you'll want to read the manual that comes with yours, and see Chapter 30, "Building Robot Electronics—the Basics," for more information. What follows assumes you already have at least a basic understanding of how to operate your multimeter.

DIRECT MOTOR CURRENT MEASUREMENT

The steps that follow assume your multimeter has a special input for testing high currents. Many do, and the input is labeled **10A** (or similar), like the one in Figure 21-6. It's a rare motor for a desktop robot that draws more than 10 amps, even when stalled, but even so, check the manual for your meter to be sure the 10A input has a replaceable fuse. (The fuse is usually located in the battery compartment.)

If any of the following are true, skip this section and move to "Indirect Motor Current Measurement."

- My meter does not have a 10A (or higher) input.
- My meter doesn't have an input fuse.
- I suspect the motor may draw current in excess of 10 amps.

Follow the connection diagram in Figure 21-7. Be sure to connect the red (+, positive) lead of your meter in the 10A socket. Then, connect the test leads to your motor using jumper

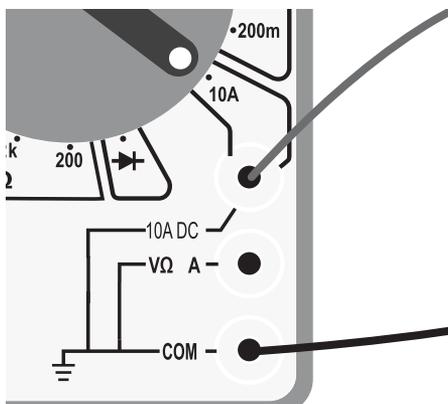


Figure 21-6 You may directly measure the current consumption of a motor by using a digital multimeter with a high amperage (10 amps or higher) input.

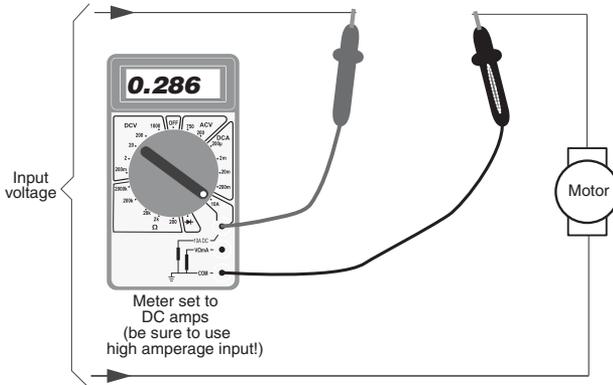


Figure 21-7 Test the current draw of a motor by interrupting the power line to the motor. Place the + and – leads of the multimeter between this power line, and take the reading. The meter must be dialed to read DC current.

wires, the kind with heavy-duty alligator clips on the ends. You need two jumpers, one each for the red and black test leads. Then,

1. Dial the meter to the 10A setting.
2. Apply power to the motor.
3. Observe the reading on the meter. It will be in amps. A reading of 0.30, for example, means 30 mA, or 0.3 of an amp. A reading of 1.75 means 1.75 amps.

This gives you the free-running torque of the motor. For other torque tests, you need to load down the motor.

Small hobby motors without a gearbox can be loaded down just by squeezing the shaft between your fingers. As you slow the motor down, watch the current increase. If you can stop the shaft from turning completely, the motor will be stalled and the current shown in the stall current. That's about as high as the current will ever get from the motor.

Small hobby motors with a gearbox can be loaded down by applying pressure to the output shaft of the motor or one of the gears in the gearbox nearest the shaft. This, of course, assumes the gears in the gearbox are accessible.



With some gearbox motors you can temporarily remove the motor from the gearbox. Test it alone, then put the motor back into the gearbox.

Larger motors are harder to test just by stopping the shaft with your fingers. It's better to attach a wheel to the motor (the larger the diameter, the better), then try to apply load to the wheel. Don't use a pair of pliers or another tool to clamp down on the motor shaft or gears, as this may cause damage. At the very least, you can take the free-running measurement.

INDIRECT MOTOR CURRENT MEASUREMENT

You can still determine current draw from your motors if your multimeter lacks a high current input or if the motor you're testing draws over 10 amps. The solution is to insert a low-value, high-wattage resistor inline between the + battery terminal and the motor.

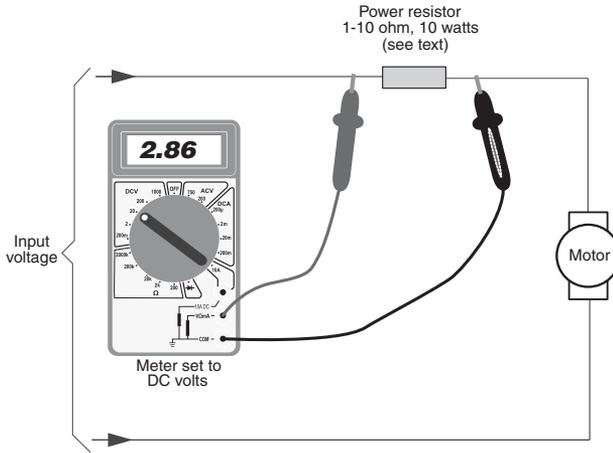


Figure 21-8 An alternative method for testing current draw is to place a low-ohm (1 Ω to 10 Ω) high-wattage resistor inline between the power source and the motor. Change the meter to read voltage. See the text for how to correlate the voltage reading to current.

The value of the resistor can be anything between 1 and 10 ohms, but it needs to be high wattage—10 or 20 watts, even more for motors that draw in excess of 10 amps.

1. Plug the red (+, positive) lead into the standard volts/ohm/current jack.
2. Use jumper clips to insert the resistor into the motor circuit, as shown in Figure 21-8.
3. Dial your multimeter to DC volts. If your meter is not autoranging, select a voltage at or just higher than the voltage you will apply to the motor.
4. Connect the red (+, positive) lead of your multimeter to the side of the resistor closest to the voltage source.
5. Connect the black (–, negative) lead of your multimeter to the other side of the resistor.
6. Apply power to the motor.

What you'll see is the voltage developed across the resistor. You then use one of the simple formulas of Ohm's law to determine the current flowing through the resistor. The formula is

$$I = E/R$$

where I is current, E is voltage, and R is resistance.

You are solving for I , or current, because you know the voltage developed across the resistor and you know the value of the resistor, in ohms. Let's suppose:

- The measured voltage is 2.86, and
- The resistor is rated at 10 ohms.

$$I = 2.86/10$$

or

$$I = 0.286, \text{ or } 286 \text{ milliamps}$$

Watch the voltage go up as you apply load to the motor shaft. See the previous section on how to test for currents under load and when stalled, using nongearred and geared motors.

Dealing with Voltage Drops

On most robots it's the motors that draw the most current. As the motors stop and start, the voltage provided to them can change. This happens because under heavy current draw, the voltage provided by the batteries can momentarily sag. The sag may be for only a fraction of a second, but it can be long enough to cause problems.

If your robot's control computer is connected to the same battery source as the motors, the voltage drop can cause what's known as a brownout; if the brownout causes the voltage to drop below a certain threshold (Figure 21-9), the control electronics may not operate correctly. Quite literally, your robot can go berserk!

Brownouts are particularly troublesome when using microcontrollers (see Part 7), which are small computers that run a program you devised. During a brownout, the microcontroller may spontaneously reset, causing it to rerun its programming from the beginning. This can actually occur several dozens or even hundreds of times a second.

During these brownouts your robot may become inactive (that's good), or it may lurch or spin or do something else unpredictable (that's bad).

There are several ways of avoiding voltage drops caused by motors:

- Add more volts to your robot's batteries. If the batteries normally supply 6 volts, but may "sag" to 4.5 volts during heavy motor draw, add another cell to bring the voltage to 7.2 or 7.5. The extra margin might prevent a voltage sag that causes a brownout.
- Add bigger batteries with a higher capacity (amp hours). The volts may be the same, but the added current capability can provide a reserve against voltage drops.
- Add a second battery pack to operate the electronics. This is my preferred method, because it effectively isolates the power supplies for the motor and electronics. As the motors draw current, they pull it from their own batteries and not from the pack powering the electronics.

For small robots you can often use a AAA- to D-size battery pack for the motors, and a 9-volt battery for the robot control electronics. If you have lots of electronics—microcontroller, multiple sensors, maybe a video camera—you might need to beef up the second battery. Use a separate multicell battery pack selected to provide the current needed for the electronics.



In order for the dual power supply technique to work, the ground (–, negative) lead from both battery packs must be connected together. This topic is discussed in more detail in Chapter 22.

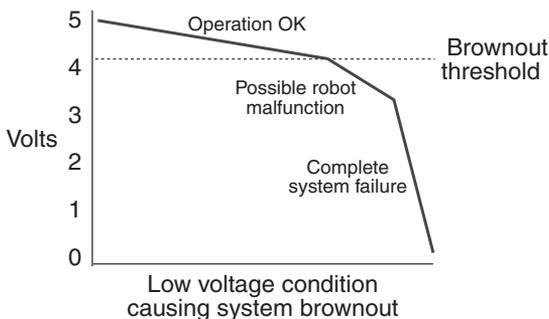


Figure 21-9 A brownout may occur when the system voltage of your robot falls below a certain minimum level. As the voltage falls under the brownout threshold, operation of the robot becomes unpredictable.

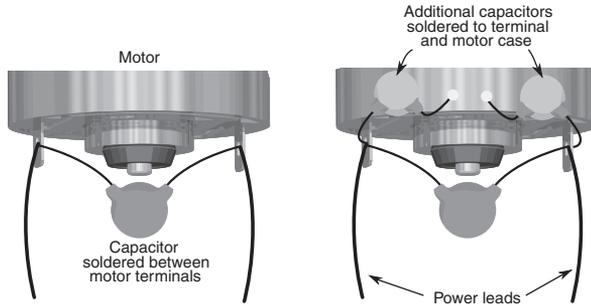


Figure 21-10 Electrical noise generated inside a DC motor can affect the operation of robot electronics. This noise can be suppressed by using small ceramic disc capacitors (0.01 to 0.1 μF) soldered to the motor terminals as shown.

Avoiding Electrical Noise

Electric motors generate lots of noise. In one form, the noise is like the static of a thunderstorm on an AM radio. It doesn't matter that the lightning bolt is miles away; the electrical charge travels through the atmosphere and into the radio's circuits, ending up as a loud snap, crackle, and pop in the speaker.

Fortunately, it's rather easy to cut down on the electrical noise produced by your robot's motors. The solution is to place one or more capacitors as near to the motor terminals as possible. The capacitor literally "soaks" up the electrical transients produced by the motors, which in turn reduces the noise they make.

- As part of good operating procedure, place a 0.1 μF ceramic disc capacitor across the motor terminals. Just solder the sucker right in there, along with the power leads to the motor.
- If the motor still generates too much electrical noise for the circuit, then as an extra precaution solder a 0.1 μF ceramic disc capacitor from each power terminal to the metal case of the motor. (This can be a bit tricky. To get the solder to stick to the motor case be sure it's clean. Rough up the metal with a small file. If needed, dab on some solder flux paste over the area and turn up the head of the soldering tool.)

Figure 21-10 shows the idea. The added capacitor(s) act to filter out the electrical noise from the motor, helping to reduce the amount of noise that travels back to the circuit.

Using DC Motors

DC motors are the mainstay of robotics. A surprisingly small motor, when connected to wheels through a gear reduction system, can power a hefty robot seemingly with ease. A flick of a switch, a click of a relay, or a tick of a transistor, and the motor stops in its tracks and turns the other way. A simple electronic circuit enables you to gain quick and easy control over speed—from a slow crawl to a fast sprint.

This chapter shows you how to apply continuous DC motors (as opposed to stepping or servo motors) to power your robots. The emphasis is on using motors to propel a robot across your living room floor, but you can use the same control techniques for any motor application, including gripper closure, elbow flexion, and sensor positioning. What follows applies to DC motors with and without gearboxes attached to them.

FYI This chapter discusses electronic components and circuits related to controlling DC motors. If you are brand-new to these subjects, be sure to see Chapter 30, “Building Robotic Electronics—the Basics,” and Chapter 31, “Common Electronic Components for Robotics.”

The Fundamentals of DC Motors

There are a many ways to build a DC motor. By their nature, all DC motors are powered by direct current—hence the name *DC*—rather than the alternating current (AC) used by most motorized household appliances.

THE PERMANENT MAGNET MOTOR: AFFORDABLE, EASY TO USE

Perhaps the most common DC motor is the permanent magnet type, so called because it uses two or more permanent magnet pole pieces that remain stationary. The turning shaft of the motor, the *rotor* (or *armature*), is composed of several sets of wires—called *windings*.

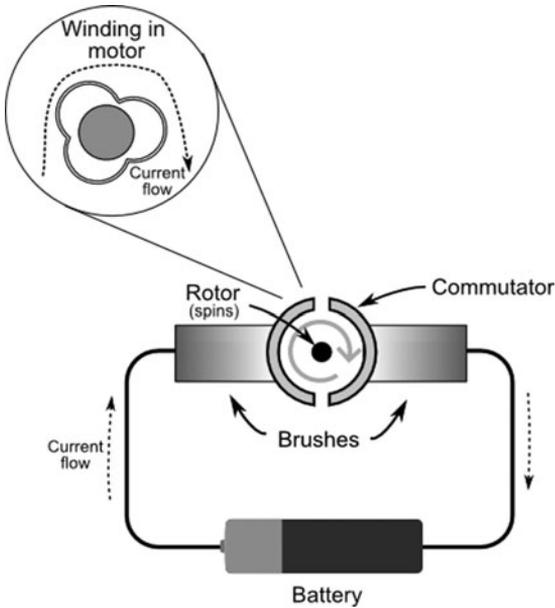


Figure 22-1 Simplified block diagram of a DC motor. Current flows from the battery to brushes, which are electrically connected to the commutator. Windings around the rotor are energized, causing the rotor to spin.

between the battery and the commutator. This is why, when used long enough, a DC motor will just go kaput.

Brushless motors use electronics, not brushes, to alternate the current between windings. Brushless motors are used extensively in computer disk drives, “noiseless” fans, CD and DVD players, and precision electronics. You should know about them, as motors pulled from these components and sold as surplus may require additional electronics in order to operate.

You may also encounter brushless motors used in R/C servo motors. These motors tend to be quite expensive and are reserved for applications where motor failure could be catastrophic, like suddenly losing control of a \$1000 R/C helicopter.

REVERSIBLE DIRECTION

One of the prime benefits of DC motors is that most (but not all) are inherently reversible. Apply current in one direction—the + and – on the battery terminals—and the motor spins clockwise. Apply current in the other direction, and the motor spins counterclockwise. This capability makes DC motors well suited for robotics, where it is often desirable to have the motors reverse direction. Use it to back a robot away from an obstacle or to raise or lower a mechanical arm.

If you’re buying your DC motors surplus, you may encounter some that are not reversible. This could be due to the way the motor windings are constructed inside the motor or it could be due to an intentional mechanical design. Read the description for the motor carefully. It will

These wires are wrapped around pieces of metal. At the end of the rotor is a *commutator*, which is used to alternately apply current to the windings.

Figure 22-1 shows a simplified diagram of current passing from a battery, through the commutator (shown here simplified), and energizing the rotor, which is in the middle. As the rotor spins, current is applied to the windings of the motor in such a way that the rotor is kept in motion. Only when current ceases to flow through the windings (or the motor shaft is physically blocked from turning) does the motor stop.

Note the two dark gray bars in the motor diagram shown in Figure 22-1. These are *brushes*, and they serve as terminals to apply the current from the battery to the commutator. On very inexpensive hobby motors, the brushes are often just pieces of copper wire, bent to a handy shape. On more expensive motors, brushes are made of conductive carbon. Both of these motors are known as *brushed motors*.

Both types of brushes can wear down over time, which can break the electrical connection

usually indicate whether it's bidirectional; or at least, if it's not, the description will specify that the shaft turns CW (clockwise) or CCW (counterclockwise) only.

See the sections later in this chapter under "Controlling a DC Motor" for various ways DC motors can be reversed.

Reviewing DC Motor Ratings

Motor ratings, such as voltage and current, were introduced in Chapter 21. Here's a quick recap of the main points of interest when selecting and using a DC motor for your robot:

- DC motors can often be operated at voltages above and below their specified rating. If the motor is rated for 12 volts and you run it at 6 volts, the odds are the motor will still turn but at reduced speed and torque. Conversely, if the motor is run at 18 volts, the motor will turn faster and will have increased torque.
- But this does not mean that you should intentionally underdrive or overdrive the motors you use. Significantly overdriving a motor may cause it to wear out much faster than normal. However, it's usually fairly safe to run a 10-volt motor at 12 volts or a 6-volt motor at 4 or 5 volts.
- DC motors draw the most current when they are *stalled*. Stalling occurs if the motor is supplied current but the shaft does not rotate. Any battery, control electronics, or drive circuitry you use with the motor must be able to deliver the current at stall, or major problems could result.
- The rotational speed of a DC motor is usually too fast to be directly applied in a robot. Gear reduction of some type is necessary to slow down the speed of the motor shaft. Gearing down the output speed has the positive side effect of increasing torque.

Controlling a DC Motor

As I've noted, it's pretty easy to change the rotational direction of a DC motor. Simply switch the power lead connections to the battery, and the motor turns in reverse. And when you want the motor to stop, merely remove the power leads to it.

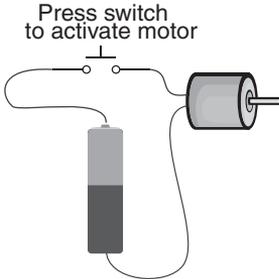
That's fine for when you're playing around on your workbench, but what are the options when the motor is part of a robot? You have several, actually, and each has its place. The ones you'll read about in this book are:

- Switch
- Relay
- Bipolar transistors
- MOSFET transistors
- Motor bridge modules

Motor Control by Switch

You can manually operate your robot using switches. This is a good way to learn about robot control and experiment with different types of robot base designs. The switches attach to your robot by wires. You can control both the operation and the direction of the motors.

SIMPLE ON/OFF SWITCH CONTROL



A very basic single-pole single-throw (SPST) *switch* can control whether a motor is on or off. Most robots use two motors, so having two switches lets you independently control each motor.

- To make the robot go forward, turn both switches on at the same time.
- To make the robot turn one direction or another, turn one switch on while leaving the other off.
- And, of course, to stop the robot, turn both switches off.

CONTROLLING DIRECTION USING SWITCHES

By using a double-pole, double-throw (DPDT) switch (read more about these in Chapter 31, “Common Electronic Components for Robotics”), you can control the direction of your motors—push the switch forward to have the motor spin one way; pull the switch back and the motor spins the other way.



Even if you don't plan on controlling your robot using switches, it's still handy to review the material that follows so that you understand how DPDT switches handle the reversal of the motor. The same basic technique of polarity reversal is used in all the other approaches.

Again, for the typical robot that uses two motors and two wheels, you use a pair of DPDT switches to control both motors. One switch operates the left motor, and the other switch operates the right motor. And if your DPDT switches have a *center-off* position, the same switches can be used to turn the motors off when you want to stop your robot.

See Figure 22-2 for an example of how to connect two DPDT switches from a battery pack to the pair of motors on your robot. Put the batteries and switches in a project box. Use two sets of wire pairs (four wires total) to connect your switch control panel to the motors on the bot.

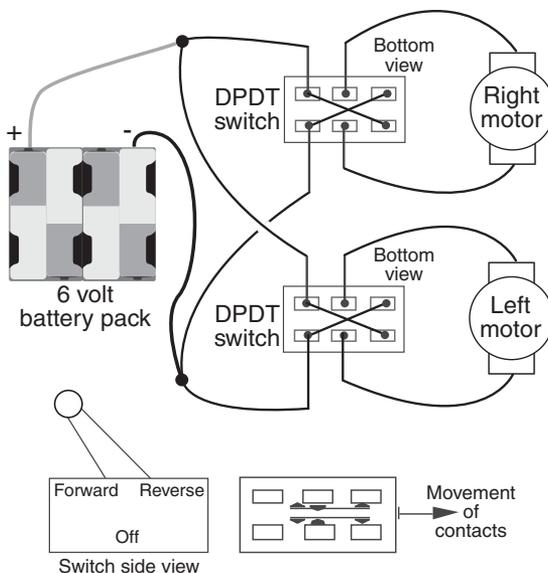


Figure 22-2 How to wire a double-pole, double-throw switch to control the direction of a DC motor. Use two switches to manually control the direction of two motors. A switch with a center-off position allows you to stop the motors.

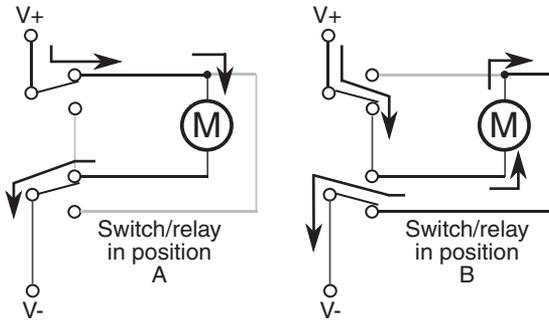


Figure 22-3 How current flows through the double-pole, double-throw switch. As shown here, the switch is wired in a way that it literally reverses the polarity of the battery connection to the motor.

Remember to use DPDT switches with a center-off position. When they are in the center position, the motors receive no power, so the robot does not move.



Try to get momentary-contact switches so that when you release them they spring back to the center. This makes it a lot easier to control your robot by flipping the switch. Momentary-contact DPDT switches cost a little more, but they're much more convenient.

Figure 22-3 shows how the switches (the technique also applies to relays, discussed next) are wired in order to achieve motor reversal. It may look a little a little weird at first, but it's actually quite logical in how it works. The switches are wired so that in one position—say, Position A—current from the battery flows through the motor in one direction. Flipping to Position B changes the direction of battery flow to the other way around. This naturally makes the motor go the other way.

Some things to try:

- Steer by turning on one motor only (leave the other one off). Note the speed of the turn.
- Now steer by making one motor go forward and the other motor go backward. Note the speed of the turn . . . it's faster. The robot actually spins, turning in place. This is how military tanks turn. It's referred to as tank steering, or, more commonly in robotics, *differential steering*.

Operating your robot with switches is great for learning how things work, but you'll soon want to graduate to hands-off methods, where your mechanical creations will steer themselves. The remainder of the DC motor control methods concentrate on techniques that allow for fully autonomous robots.

Motor Control by Relay

Before getting to the all-electronic methods of motor control, I want to take a moment to talk about another, somewhat more old-fashioned approach: using *relays*. Yes, I know this is the twenty-first century, and what's all this about using something like a relay that was invented almost 200 years ago. Daft indeed!

But there are many good reasons to look at relays, if only because they're a natural stepping-stone from switch control to fully electronic control. But also, small relays for small robots are very cheap and very easy to use.



While it's true that relays wear out in time, and they're slower than electronic motor control, neither is a particularly relevant excuse to avoid them. Small *reed* relays for the average-size desktop amateur robot can switch several hundred thousand times before even beginning to act funny. As for their slowness, the motors are even slower—the slowest part of your robot.

INSIDE A RELAY

A relay is an electrically operated switch. It's just like the manual switches detailed in the previous section, but with the added feature of being controlled via electric signals.

The operation of a relay is simple: an electric coil is placed around or near a piece of metal that acts as a switch plate. When current is applied to the coil, the coil acts like an electromagnet. The magnetism pulls the metal switch plate closer to it. This engages the switch.

Nearly all relays self-reset; that is, when current is removed from the coil, a spring on the switch plate pulls it back into its original position. This disengages the switch.

SIMPLE ON/OFF RELAY CONTROL

You can accomplish basic on/off motor control with a single-pole relay, just as you did with manual switches. The relay is wired so that when it's inactive (OFF) current from the battery is not switched to the motor. When the relay is activated (ON) the circuit between battery and motor is complete, and the motor turns.

How you activate the relay is something you'll want to consider carefully. You *could* control it with a push-button switch, but that doesn't get you anything more than using manual switches alone. Relays can easily be driven by digital signals, the kind from a simple board or microcontroller on your robot.

See Figure 22-4 for the basic way of connecting a relay to any kind of controlling electronics.

If you're new to electronics in general, a review of some terminology is in order.

- *Logical 0* (referred to as *LOW*) is digital terminology that means 0 volts is applied to the relay.
- *Logical 1* (referred to as *HIGH*) means that voltage (of some level) is applied to the relay. In most digital electronic circuits, this is 5 volts. In fact, we can just assume it's 5 volts, unless told otherwise.

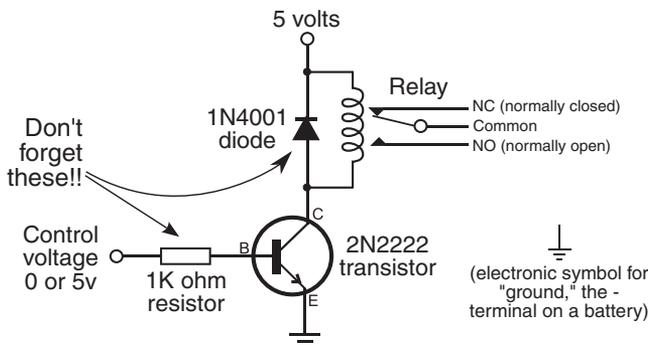


Figure 22-4 A relay can be used to electronically turn a motor on and off. A transistor and a resistor *drive* the relay; you control the operation of the relay by applying 0 or 5 volts as a *control voltage*.

- *Gates* are inputs or outputs on a digital circuit, such as from a computer or a microcontroller. As you might guess, you'd use an *output gate* when operating a relay via a digital circuit.

In this circuit, LOW turns the relay off and HIGH turns it on. The relay can be operated from most any digital gate. The chapters in Part 5 deal much more extensively with using electronic and computerized control. If you're wanting to try relays, familiarize yourself with the wiring diagram in Figure 22-4. It will prepare you for these later chapters.

CONTROLLING DIRECTION VIA RELAY

Changing the direction of the motor is only a little more difficult using relays than turning it on and off. As with the manual switches in the previous section, this requires a DPDT relay, wired in series after the on/off relay just described. Refer to Figure 22-5 for how these two relays are connected. With the switch contacts in the DPDT relay in one position, the motor turns clockwise. Activate the relay, and the contacts change positions, turning the motor counter-clockwise.

Again, you easily control the direction relay with digital signals. Logical 0 makes the motor turn in one direction (let's say forward), and logical 1 makes the motor turn in the other direction.



You need two relays, not just one, to duplicate the functionality you enjoyed with the DPDT center-off switch. An SPST relay is used to turn the motor on and off, and a DPDT relay is used to control the direction of the motor.

You can see how to control the operation and direction of a motor using just two signals (*data bits*) from a digital circuit like a computer or microcontroller. Since most robot designs incorporate two drive motors, you can control the movement and direction of your robot with just four data bits. In fact, this is true for all the electronic motor control schemes in this chapter.

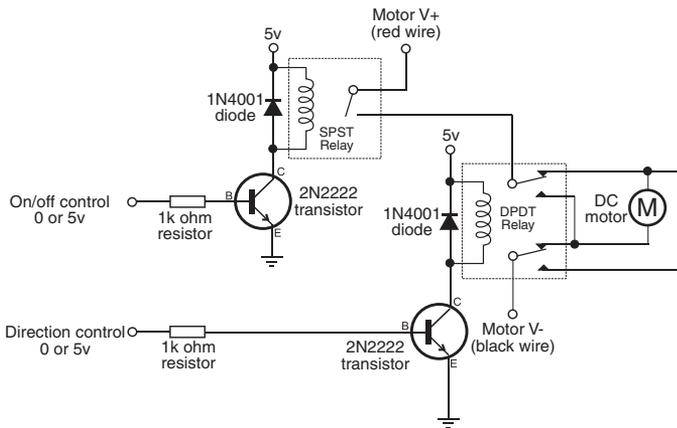


Figure 22-5 How to combine two relays to fully control a motor: turn it off and on, and reverse direction. The top relay is a single-pole type; the bottom relay is a double-pole type.

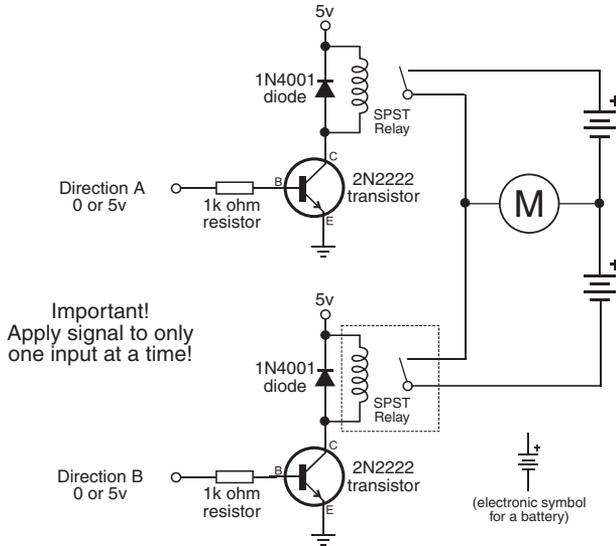


Figure 22-6 An alternative method for controlling a motor is to use two single-pole relays and a *split* power supply (it provides both positive and negative voltage; this can be done by simply combining two separate battery packs, as shown). With no input signal, the motor doesn't turn.

CREATING A RELAY HALF-BRIDGE

The SPST/DPDT method described here is the electrical equivalent of using switches, and it works with a wide variety of motors you might encounter in your robot travels. But there's yet another, and even simpler, method for controlling small, low-voltage hobby DC motors when using relays. It's ideally suited for those motors that are meant to be run from 1.5 to 3 volts, like those in the very popular Tamiya Twin Motor Gearbox kit.

Figure 22-6 illustrates using two SPST relays to make what's called a half-bridge. To make this work, you need a pair of battery packs, as shown. One pack provides the + positive voltage to the motor, and the other pack provides the - negative voltage.



You can use a pair of two-cell AA battery holders to power the motor, using either alkaline or rechargeable NiCd or NiMH batteries. When using alkalines, the voltage for each battery holder is 3 volts; it's 2.4 volts when using rechargeables. Either voltage is fine when using the Tamiya Twin Motor Gearbox or a DC motor of similar design.

Follow this table to control the motor, being careful to never turn both relays on at the same time.

Relay 1	Relay 2	Action
Open	Open	Motor OFF
Closed	Open	Motor Forward
Open	Closed	Motor Reverse
Closed	Closed	NO! Not allowed!!

If you prefer an all-electronic approach, see the next sections on using transistors, where the half-bridge concept comes up again.

CURRENT SPECS FOR RELAYS

When selecting relays for your robot, make sure the contacts are rated for the motors you are using. All relays carry contact ratings, and they vary from a low of about 0.5 amp to over 50 amps, at 125 volts. Higher-capacity relays are larger and require more current to operate. This means they need bigger transistors to trigger them and require much more care in selecting all the components in the system. Not very pretty.

As this book is primarily about creating amateur robots under about 20 to 30 pounds, you'll be using smaller motors, which means smaller relays. For the robots described in this book, you don't need a relay rated higher than 2 or 3 amps. If you plan on building bigger robots with much bigger motors, you should consider the motor bridge module, detailed later in this chapter.

SIMPLIFIED RELAY DRIVER ELECTRONICS

With two motors you need four relays, which means four transistors, four diodes, and four resistors. What are these extra components for?

- The transistors are used to boost the current from the digital gate (for example, from your microcontroller), as not all gates have sufficient current to directly drive the relay. The transistors act as current amplifiers.
- The diodes protect the transistors from current that flows backward from the relay coil when it is switched off. This happens because, when the relay coil is deenergized, some of the current that was flowing through it is regurgitated back out. This *back EMF* (EMF stands for electromotive force) can damage the transistor; the diode prevents this mess from happening.
- The resistors control the amount of current flowing from the digital gate to the transistor. Without the resistor, the transistor would suck up too much current and possibly damage the gate.

A typical value for these resistors is between 1 k Ω (1000 ohms) and 4.7 k Ω (4700 ohms), assuming 5-volt circuitry. The lower the value of the resistor, the more current will flow from the gate and to the base of the transistor. Use the 1 k Ω value for larger relays that need more current; otherwise, select a higher value.



Use the highest-value resistor connected to the base of the transistor that allows for reliable operation of the relay. You might start with 4.7 k Ω and work downward until you find a resistor value that works best for your specific circuit.

The transistor-resistor-diode combination is the typical way of connecting a relay to the rest of your electronics. But you can save some space and wiring time by using a single 16-pin IC that has everything built in. The ULN2003 integrated circuit contains seven *drivers*; each driver is the equivalent of the transistor-diode-resistor combo in the previous examples.

To use, connect the control gate to a driver input, and connect its output to the relay, as shown in Figure 22-7. The illustration shows using only one of the available seven drivers. You

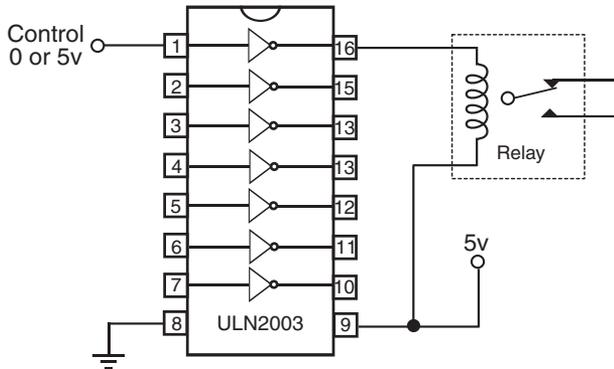


Figure 22-7 The ULN2003 Darlington array integrated circuit contains seven independent drive circuits. These drive circuits include the flyback diode shown in the previous relay diagrams. The ULN2003 is a simpler method when using many relays.

can use the remaining drivers for other relays or even something else completely—like lighting up some superbright light-emitting diodes. If a driver is not used, disable it by connecting its input to ground. Leave its output unconnected.



You may see similar circuit diagrams showing diodes used on the outputs of the ULN2003. That's okay. The chip has its own diodes built in, but some designers like to take the extra precaution and add their own. For the smallish relays typically used in hobby robots, the extra diodes are not usually needed, but you can add them if you wish.

Motor Control by Bipolar Transistor

No, bipolar transistors don't exhibit manic-depressive behavior. In this case, *bipolar* is merely the term used to describe their internal construction—which we won't be getting into here since it's not particularly relevant (besides, there are, like, 10,000 books and Web sites that already talk all about it). There are other types of transistors, but the ones we're interested in for the time being are the bipolar variety.



Bipolar transistors are more accurately known as *bipolar junction transistors*, or *BJTs*. Same thing, slightly more words.

For robotics motor control, you use a bipolar transistor much as you would a switch. In fact, the transistor acts just like a switch: apply or remove current, and the transistor (switch) turns on or off. In order to do its work as a motor control switch, the transistor needs a few extra common electronic components, specifically, a resistor and a diode. The purpose of these are described shortly.

BASIC TRANSISTOR MOTOR CONTROL

See Figure 22-8 for the most basic implementation of using a transistor to operate a motor. A digital LOW or HIGH signal is applied to the input of the transistor circuit. Depending on whether the input is LOW or HIGH, the motor turns on or it turns off.

- When you connect the Motor control input to 5 volts (HIGH), the motor turns.

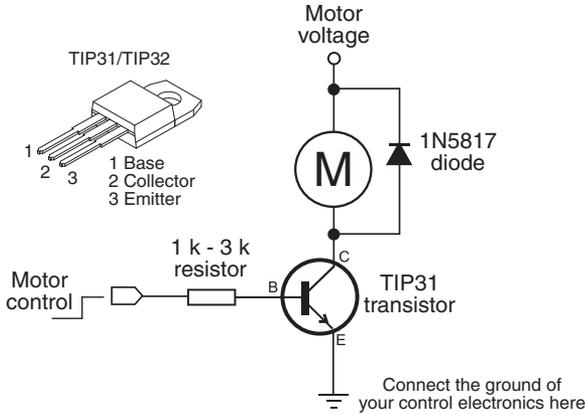


Figure 22-8 Fully electronic control of a motor is done using a transistor. This simple circuit starts and stops a motor, depending on the input signal. Select the value of the resistor so that the transistor switches fully on when the input signal is applied (the motor should turn at almost full speed).

- When you connect the Motor control input to the ground connection (LOW), the motor stops.



Transistors of the type shown in Figure 22-8 exhibit an inherent voltage drop between its collector (“C”) and emitter (“E”) connections. The drop is usually about 0.7 volt. It’s enough that you may notice your motor runs a bit slower than when it’s directly connected to the battery. This side effect can be largely avoided by using MOSFET transistors, described below.

Figure 22-9 goes a step further, creating a half-bridge using a two bipolar transistors. One transistor is the NPN type, and the other is a PNP type (see Chapter 31, “Common Electronic Components for Robotics,” for more information on this terminology). It’s best if you use so-called complementary pairs, NPN and PNP transistors that have similar specifications. Refer to the control table in Figure 22-9 for operating the motor. It’s a little unusual because of the use of both NPN and PNP transistors.

Note the use of diodes. When current to the motor is removed, the motor continues to spin until it comes to a natural stop. While spinning, the motor creates a backward-flowing current, which could damage transistor. When diodes are used in this fashion they are often called by various descriptive names—flyback diodes, snubber diodes, free-wheeling diodes, suppression

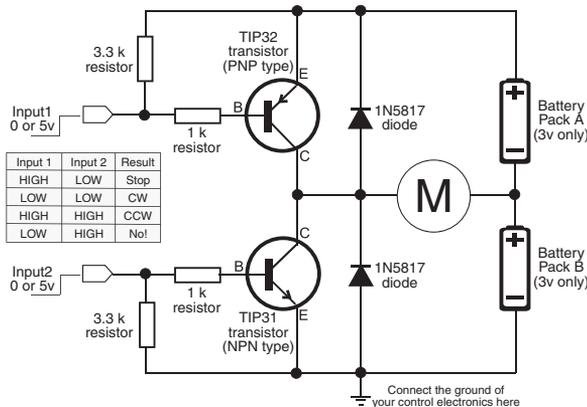


Figure 22-9 Simple split-supply motor control circuit, using two batteries (or battery packs) wired in series. Note that the transistors are different types: one is an NPN type, and one is a PNP type.

diodes, and a bunch of others. They all mean the same thing; the different terms describe their function, not their type.

You need two battery sources to provide the split voltage necessary for this circuit to run. Use a pair of two-cell battery holders, and wire them as shown.

FULL-BRIDGE TRANSISTOR CONTROL

A more elaborate form of transistor motor control uses the *full-bridge*, also called an *H-bridge*. The “H” comes from the way the motor is connected to its control circuitry, looking a wee bit like the letter H.

H-bridges require four transistors, along with associated components—resistors, flyback diodes, and perhaps more, depending on the design. They’re harder to build, and harder to get right. Given the availability of other motor control circuitry, it makes more sense to use these other techniques, which tend to be easier and cheaper.

Instead, you can use MOSFET transistors, as described in the following section, or rely on one of the many low-cost motor bridge ICs or modules. Those are covered later in this chapter. Both provide more flexibility, and they are often cheaper and easier to build.

Motor Control by Power MOSFET Transistor

MOSFET stands for “metal oxide semiconductor field effect transistor.” *Metal oxide* indicates the process used to manufacture them, and *field effect* refers to the way the transistor conducts current. *Power MOSFET* is a further classification that indicates the device is intended to drive some kind of load, like a motor.

MOSFETs look like bipolar transistors (they come in the same type of packages), but internally there are some important differences. First, due to their MOS construction, they are more susceptible to being damaged by static electricity. Always keep the protective foam around the terminals of the device until you’re ready to use it.

Second, the names of the terminals are different from what you find on bipolar transistors. These variations are discussed in more depth in Chapter 31, “Common Electronic Components for Robotics,” but for now just know that if you connect a MOSFET to your circuit incorrectly, odds are great that it’ll be destroyed the instant you turn on the power. Bipolar transistors aren’t usually so sensitive.

BASIC MOSFET MOTOR CONTROL SWITCH

A commonly available power MOSFET is the IRF5xx series, such as the IRF510, IRF520, and so on. These are available in the popular TO-220-style transistor case. These devices can control several amps of current, when on a suitable metal heat sink. The heat sink is a piece of metal that provides a large surface area to draw heat away from the MOSFET.

A basic circuit that uses a commonly available IRF510 power MOSFET is shown in Figure 22-10. It’s a simple on/off switch to control a motor. Apply a 5-volt signal to the gate connection of the transistor, and the motor is turned on. Resistors are used for additional protection of the circuit controlling the transistor.

Notice the *Motor voltage* label. The voltage to your motor can be different from the voltage that controls the MOSFET transistor. Often, the control electronics in your robot are powered by 5 volts; the voltage to your motors can be 5 volts or over, up to the specified limit

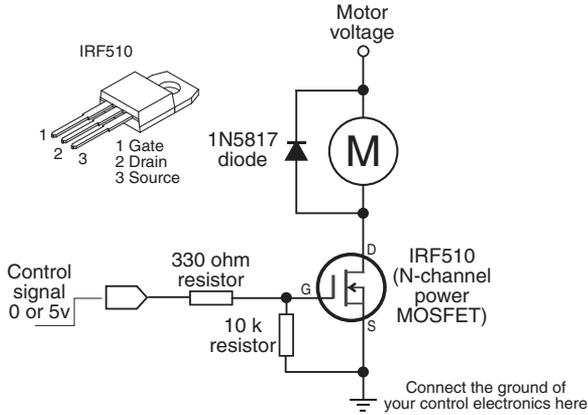


Figure 22-10 A MOSFET power transistor provides nearly the full voltage to the motor when it is turned on.

of the MOSFET, which is often at least 20 or 30 volts. In the case of the IRF510, the voltage limit is 100 volts.

The IRF510 is routinely available from many online sources, and it costs under \$1.50.

MOTOR H-BRIDGE USING MOSFET TRANSISTORS

Figure 22-11 shows the basic concept of the MOSFET transistor H-bridge. The gates of the transistors are connected to either ground or 5 volts. Turning on Q1/Q4 causes current to flow through the motor in one direction, making the motor spin clockwise. Turning on Q2/Q3 causes the current to flow through the motor in the opposite direction. The result: The motor spins counterclockwise.

The two types of MOSFET—N-channel and P-channel—refer to the microscopic conductive channel found inside the device. The two types differ in their chemical makeup, which in turn affects how the devices conduct electrons. The arrangement shown in the circuit takes advantage of N- and P-channel behavior to build an H-bridge that acts as close to a mechanical switch as possible.

Note that “turning on” an N-channel or P-channel MOSFET is relative and, for a P-channel transistor, may work opposite to what you think:

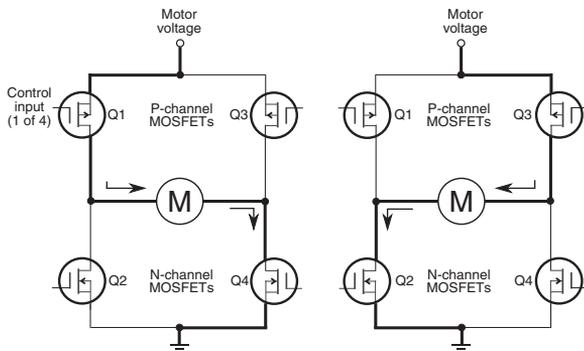


Figure 22-11 A (very) basic H-bridge motor control circuit, using four MOSFET transistors. Note that the two uppermost transistors are P-channel type; the two lower transistors are N-channel type. Don't get these crossed up.

Gate Signal	N-channel	P-channel
0 volts (LOW, logical 0)	Turns off	Turns on
5 volts (HIGH, logical 1)	Turns on	Turns off

Unlike bipolar transistors, which exhibit a drop in voltage when current is passed through them, MOSFET transistors pass through nearly all of the volts to the motor.



The N- and P-channel MOSFET transistors you use should be complementary pairs, that is, transistors that share similar specifications. This provides a balance in current-carrying capability. For example, you might use IRF530/IRF9530 or IRF540/IRF9540. Not all MOSFETs use such convenient numbering sequences to indicate pairing. You can consult a basic data book to find complementary pairs, and be sure to read the specifications provided by online retailers. Most retailers provide direct links to datasheets provided by the MOSFET manufacturers, and you're encouraged to review them and compare specifications.

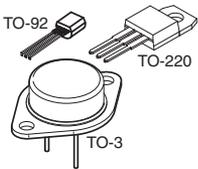


There are many ways to build MOSFET H-bridges, and each method has its distinct advantages. Rather than trying to cover them here—which wouldn't do them justice—see the bonus H-bridge projects on the RBB Online Support site (refer to Appendix A for details). Included are several tested variations, from fairly simple to somewhat complex.

COMMON DESIGN GOALS FOR TRANSISTOR H-BRIDGES

If you'd like to design your own MOSFET transistor H-bridge, keep these basic design goals in mind.

First, the transistors you choose must be capable of handing the current draw demanded by the motor. Refer to the motor specifications to determine their maximum current draw, or test it yourself using the steps in Chapter 21, “Choosing the Right Motor.”



Most motors draw at least 500 milliamps, and this exceeds the current-carrying capacity of MOSFETs that come in the smaller TO-92 package. For most motors you work with, stick with the devices in the TO-220 or TO-3 packages.

On larger-power MOSFETs, the case of the transistor doubles as the *drain* terminal. This is important if you mount the transistors on a common heat sink, and especially when you ground the heat sink to the metal frame of the robot.

Avoid the hassles of potential short circuits by getting a set of insulated transistor-mounting kits. These insulate the transistors electrically but still allow the heat sink to sink heat.

When using transistors in TO-220 or TO-3 packages, if you place the transistors close together on a circuit board be sure none of the metal cases touch one another.

Most H-bridge designs use flyback diodes placed in parallel with each transistor. Without these, the back EMF from the motor could damage the transistor. You'll know this has happened if the transistor constantly runs the motor, but at a reduced voltage. While most modern power MOSFET transistors include a diode as part of its internal construction, many robo-builders suggest adding your own external diodes. A fast-acting Schottky diode, such as the 1N5817, is usually a good choice when using small to medium-size motors.

And finally, remember that motors produce lots of noise. You'll want to place an aluminum

or tantalum electrolytic capacitor (47 μF to 330 μF) across the motor voltage and ground connections, as close to the transistors as possible.

Motor Control by Bridge Module

Circuits for controlling motors are big business, and it shouldn't come as a surprise that dozens of companies offer all-in-one solutions for running motors through fully electronic means. These products range from inexpensive \$2 integrated circuits to sophisticated modules costing tens of thousands of dollars. Of course, we'll confine our discussion to the low end of this scale!

Motor control modules incorporate the H-bridge design you learned about in the previous section. The module may consist of just an IC, or it may be a premade circuit board with the H-bridge electronics on it. Either way, motor control bridges have two or more pins on them for connection to control electronics and, of course, connections to power and to the motors.

Typical functions for the pins are:

- *Motor power.* Connect these to the battery or other source powering the motors. I like to use a completely separate battery pack for the motors and the rest of the robot's electronics. Using the motor power pins on the motor bridge, this is very easy to do.
- *Motor enable.* When enabled, the motor turns on. When disabled, the motor turns off. Some bridges let the motor "float" when disabled; that is, the motor coasts to a stop. On other bridges, disabling the motor causes a full or partial short across the motor terminals, which acts as a brake to stop the motor very quickly.
- *Direction.* Setting the direction pin changes the direction of the motor.
- *PWM.* Most H-bridge motor control ICs are used to control not only the direction and power of the motor but its speed as well. The typical means used to vary the speed of a motor is with pulse width modulation, or PWM. This topic is described more fully under "Controlling the Speed of a DC Motor." On many H-bridge ICs, the motor enable and PWM input are the same.
- *Brake.* On bridges that allow the motor to float when the enable pin is disengaged, a separate brake input is often used to specifically control the braking action of the motor.
- *Motor out.* These are outputs for connecting to the motor.

The better motor control bridges incorporate overcurrent protection circuitry, which prevents them from being damaged if the motor pulls too much current and overheats the chip. Some even provide for *current sense*, useful when you want to determine if the robot has become stuck.

Recall from earlier in this chapter that DC motors will draw the most current when they are stalled. If the robot gets caught on something and can't budge, the motors will stop, and the current draw will spike.

USING THE L293D AND 754410 MOTOR DRIVER ICs

Among the most common—and least expensive—motor bridge ICs are the L293D and its close cousin, the 754410. Both come in small 16-pin IC packages, and their hookup is identical. The big difference between the two is the maximum amount of current that the chip can handle.

- The L293D can supply up to 600 mA of current (continuous, per channel).
- The 754410 can supply up to 1.1 A of current (continuous, per channel).

On both chips, the supply voltage ranges from 4.5 to 36 volts, and they have connections for not just one motor but two.

From here on out I'll refer just to the L293D, but know the discussion applies to the 754410 as well.

When buying the L293D, be sure it has a *D* at the end of it. "D" denotes diodes. Recall the use of diodes used to protect components in the transistorized H-bridges described earlier in this chapter. The L293D (and the 754410) have these diodes built in. If you don't get the D version of the L293, you'll need to add these diodes yourself.

While we're on the subject of diodes: There is some disagreement among robo-builders about whether the diodes built into the 754410 are intended for flyback protection. The datasheet for the 754410 is not clear on the subject. Adding to the confusion is that several versions of the datasheet contain other errors. If you want to be sure that the 754410 is fully protected, you can add external diodes, such as 1N5817. But if playing safe is the key, this suggestion applies equally to the L293D and any other motor bridge IC.

Figure 22-12 shows the basic connection for the L293D to a pair of motors. Notice the two power pins for the L293D: one is to power the IC, and it should be 5 volts. The other is the power for the motor, and that can be up to 36 volts. The minimum allowed motor voltage is 4.5 volts.

Each motor is controlled by a set of three pins, also called input lines. The motors are referenced as *Motor1* and *Motor2*.

For Motor1, the control lines are labeled Input1, Input2, and Enable1. For Motor2 the lines are marked Input3, Input4, and Enable2.

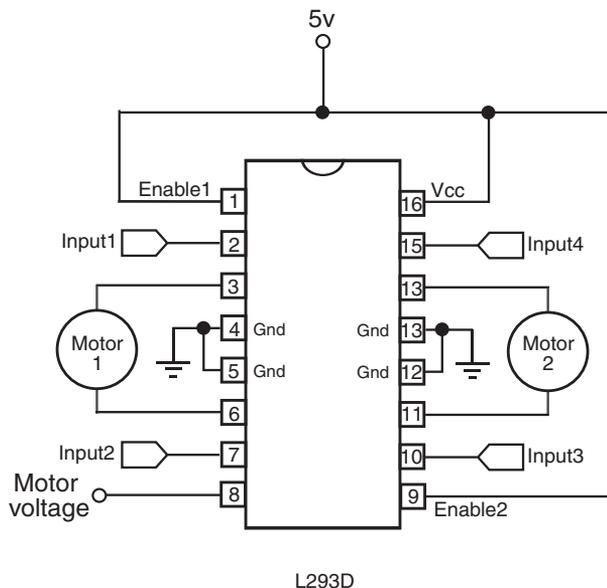


Figure 22-12 H-bridge integrated circuits incorporate all the necessary electronics to drive a motor.

In order to activate Motor1, the Enable1 line must be HIGH. You then control the motor and its direction by applying a LOW or HIGH signal to the Input1 and Input2 lines, as shown in this table.

Motor 1		
Input1	Input2	Action
LOW	LOW	Motor stops
LOW	HIGH	Motor turns forward
HIGH	LOW	Motor turns backward
HIGH	HIGH	Motor stops

Controlling Motor2 is the same, except it uses Input3 and Input4.

The enable lines are also used to control the speed of the motor (Enable1 for Motor1, and Enable2 for Motor2). Instead of a constant HIGH signal, you can apply a rapid succession of LOW/HIGH pulses; the length of the pulses determines how fast the motor goes. For more, see “Controlling the Speed of a DC Motor,” later in this chapter.

The L293 series and 754410 chips have a “fast motor stop” feature when the Enable line is HIGH and both Inputs are LOW or HIGH (either, doesn’t matter). Depending on the motor, this may not produce much of a braking effect. If you need to quickly stop the motor, momentarily reverse its direction, then stop it. If you don’t want the “fast motor stop” feature, disable the motor by bringing the Enable line LOW.



The four center pins of the L293D serve as the IC’s ground connectors and can also be used as part of a heat sink. In order for the chip to drive its maximum current, you should add a larger metal heat sink over the IC. You can buy these premade or solder on strips of bare (uncoated) copper metal pieces to the center pins in a kind of “wing” arrangement.

BONUS PROJECT: USING THE L298 MOTOR DRIVER IC

The L298 is another popular motor driver IC, capable of handling up to 2 amps per motor. Using it is a bit more involved than using the L293D, though the principle of operation is the same. See the RBB Online Support site for a bonus project using the L298, including programming examples for the Arduino and PICAXE microcontrollers.

“INTELLIGENT” MOTOR BRIDGE MODULES

A recent trend is *serial motor control*, which essentially consists of “set and forget” modules that do most of the hard work for you. Using a microcontroller attached to the module via a simple serial communications connection, you command the motor to start, stop, and reverse. All modern microcontrollers support serial communications in one form or another, so this technically is available on any robot that uses a microcontroller as a central brain.

What’s more, you can order the motor to change speed, without having to worry about the complexities of motor speed control (described in the next section). The number of speeds

available to you depends on the module, but it's not uncommon to have at least 64 speed steps—all the way from a slow crawl to full pedal-to-the-metal bore.

Many of these intelligent bridge modules are available in single- or dual-motor versions, with current capacities of 50, 75, even 100 amps—ideal if you're building a very large or combat robot.



Also available are *ESC motor speed controllers*, originally intended for use with high-speed R/C racing vehicles. Find these at any R/C hobby store. Though ESC motor speed controllers are designed for use with R/C receivers, you can use an ordinary microcontroller to simulate the signals that it expects to see. Just treat it like a servo motor.

Controlling the Speed of a DC Motor

There will be plenty of times when you'll want the motors in your robot to go a little slower or perhaps track at a predefined speed. Speed control with continuous DC motors is a science in its own right, but the fundamentals are quite straightforward.

NOT THE WAY TO DO IT

Before exploring the right way to control the speed of motors, let's examine how *not* to do it. Many robot experimenters first attempt to vary the speed of a motor by using a potentiometer. While this scheme can work, it wastes a lot of energy. Turning up the resistance of the potentiometer (which is a variable resistor) decreases the speed of the motor, but it also causes excess current to flow through the pot. That current creates heat and draws off precious battery power.

BASIC SPEED CONTROL

A better way is to feed the motor short on/off pulses of its usual voltage. The pulses are very fast, so fast that the motor doesn't have time to respond to each on/off change. What happens is that the motor ends up averaging the ons with the offs, so, effectively, less voltage gets to the motor.

This system of motor speed control is called *pulse width modulation*, or *PWM*. It is the basis of just about all motor speed control circuits. The longer the duration of the pulses, the faster the motor because it is getting full power for a longer period of time. The shorter the duration of the pulses, the slower the motor.

Check out Figure 22-13, which shows the on and off nature of PWM. The time between each pulse is called the *period*, and it's usually just a brief moment in time—microseconds. In the typical PWM system, there may be from 500 to over 20,000 of these periods each second.

Notice that the power or voltage delivered to the motor does not change—it's always 5 volts (or 6 volts, or 12 volts, or whatever). The only thing that changes is the amount of time the motor is provided with this voltage. The longer the on time in relation to the off time, the more power the motors gets. Most motors function adequately at PWM ratios of 25 percent or higher. Depending on the motor and other factors, at lower PWM rates the motor may not receive enough power to turn its load.

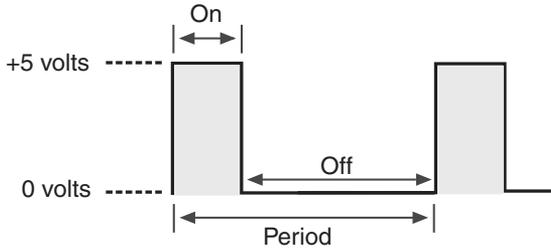


Figure 22-13 The speed of nearly all DC motors may be varied by changing the duty cycle (on versus off times) of its supply voltage. The longer the on time, the faster the motor will turn.

Remember: The frequency of the pulses—how many occur in a second—does not change, just their relative on and off times. PWM frequencies of 1 kHz (1000 cycles per second) to over 20 kHz are commonly used, depending on the motor.

Unless you have a specification sheet from the manufacturer of the motor, you may have to do some experimentation to arrive at the “ideal” pulse frequency to use. You want to select the frequency that offers maximum power with minimum current draw.

Bonus Projects: Interfacing to Motor Bridge Modules

I’ve added a number of bonus motor bridge projects to the RBB Online Support site, including using several brands of serial motor speed controllers with the Arduino microcontroller. You’ll find connection schematics, programming code, and parts lists.

Also included are hands-on sample code projects for using the venerable L293D, and the L298, with the three controllers highlighted in this book: Arduino, PICAXE, and BASIC Stamp. See Appendix A for more details about the RBB Online Support site.

The sample code demonstrates motor direction control, plus PWM speed control.

Using Servo Motors

DC motors are inherently an *open feedback* system—you give them juice, and they spin. How much they spin is not always known, at least not without additional mechanical and electronic parts.

Servo motors, on the other hand, are a *closed feedback* system. This means the output of the motor is coupled to a control circuit. As the motor turns, the control circuit monitors the position. The circuit won't stop the motor until the motor reaches its proper point. All without your having to do anything extra.

Servo motors have earned an important place in robotics. And fortunately for robot builders, another hobby—model radio control—has made these motors plentiful, easy to use, and quite inexpensive.

In this chapter you will learn what you need to know to use radio control (R/C) servos in your robot projects. While there are other types of servo motors, it is the R/C type that is commonly available and affordable, so I'll be sticking with those only.

FYI

Be sure to also check out Chapter 24, "Mounting Motors and Wheels," to learn how to attach servos to your bot, and the chapters in Part 7, "Microcontroller Brains," on how to program servos to do various wonderful things.

How R/C Servos Work

Servo motors designed to be operated via a radio-controlled link are commonly referred to as *radio-controlled* (or *R/C*) servos, though, in fact, the servo motor itself is not what's radio controlled. The motor is merely connected to a radio receiver on the model plane or car. The servo takes its signals from the receiver.

This means you don't need to control your robot via radio signals just to use an R/C servo



Figure 23-1 A standard-size servo motor for radio-controlled model airplanes and cars. The flanges allow easy mounting, and the output of the servo can be readily attached to wheels, linkages, and other mechanisms. (Photo courtesy Hitec RCD.)

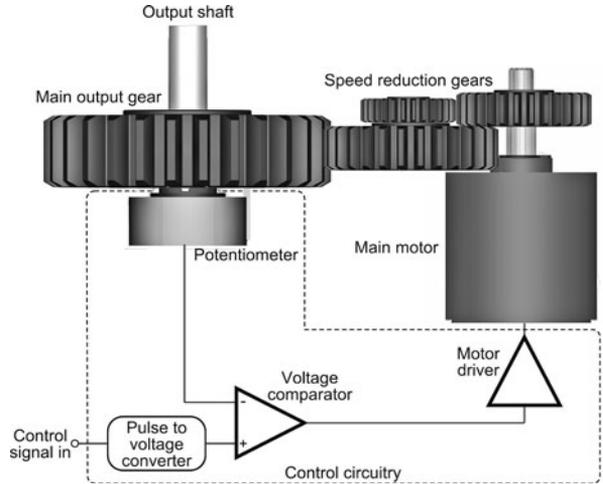


Figure 23-2 How an RC servo works. A control signal causes the motor to turn in one direction or the other, depending on the current position of the output shaft.

(unless you want to, of course). You can control a servo with your PC, a microcontroller such as the Arduino.

Figure 23-1 shows a typical standard-size R/C servo motor, which is used with model flyable airplanes and model racing cars. It measures about $1\text{-}1/2'' \times 3/4'' \times 1\text{-}3/8''$. For this style of servo, its size and the way it's mounted are the same, regardless of the manufacturer. That means that you have your pick of a variety of makers and can compare prices. There are other common sizes of servo motors besides that shown in the figure, however. I'll get to those in a bit.

A PEEK INSIDE

Inside the servo is a motor and various other components, neatly packaged (see Figure 23-2). While not all servos are exactly alike, all have these three major parts: motor, reduction gear, and control circuitry.

- *Motor.* A DC motor capable of reversing direction is at the heart of the servo.
- *Reduction gears.* The high-speed output of the motor is reduced by a gearing system. Many revolutions of the motor equal one revolution of the output gear and shaft of the servo.
- *Control circuitry.* The output gear is connected to a potentiometer, a common electronic device similar to the volume control on a radio. The position of the potentiometer indicates the position of the output gear.

The motor and potentiometer are connected to a control board, all three of which form a *closed feedback loop*. The servo is powered by 4.8 to 7.2 volts.



You can't run an R/C servo simply by connecting it to a battery. It needs special control signals to operate. As it turns out, it's not terribly hard to control a servo using simple programming. Numerous programming examples are provided in Part 7.

LIMITING ROTATION

As you can surmise, servo motors are designed for limited rotation rather than for continuous rotation like a DC motor. While there are servos that rotate continuously, and you can modify one to freely rotate (see later in this chapter), the primary use of the R/C servo is to achieve accurate rotational positioning over a range of up to 180° .

While 180° —half a circle—may not sound like much, in actuality such control can be used to steer a robot, move legs up and down, rotate a sensor to scan the room, and more. The precise angular rotation of a servo in response to a specific digital signal has enormous uses in all fields of robotics.

Control Signals for R/C Servos

The control signal that commands a servo to move to a specific point is in the form of a steady stream of electrical pulses. The exact duration of the pulses, in fractions of a millisecond (one-thousandths of a second), determines the position of the servo, as shown in Figure 23-3.

Note that it is not the number of pulses per second that controls the servo, but the *duration* of the pulses that matters. This is very important to fully understand how servos work and how to control them with a microcontroller or other circuit.

Specifically, the servo is set at its center point if the duration of the control pulse is 1.5 milliseconds. Durations longer or shorter command the servo to turn in one direction or the other.

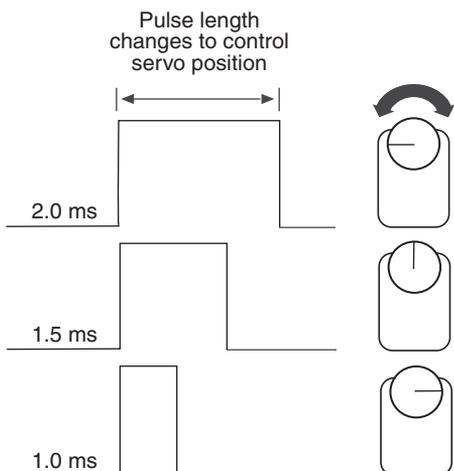


Figure 23-3 The length of the control pulses determines the angular position of the servo shaft. The pulse range is from 1.0 millisecond (or 1000 microseconds) to 2.0 ms (2000 μs). A pulse of 1.5 ms (1500 μs) positions the servo shaft in the middle.

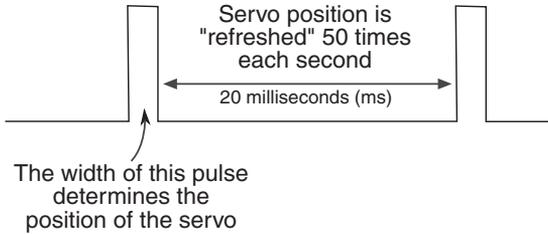


Figure 23-4 Control pulses are repeated ("refreshed") at roughly 50 Hz (50 times each second).

- A duration of 1.0 milliseconds (ms) causes the servo to turn all the way in one direction.
- A duration of 2.0 ms causes the servo to turn all the way in the other direction.
- And to recap, a duration of 1.5 ms causes the servo to return to its midpoint.

The servo needs about 30 to 50 of these pulses per second, as shown in Figure 23-4. This is referred to as the *refresh* (or *frame*) rate; if the refresh rate is too low, the accuracy and holding power of the servo are reduced. If there are way too many pulses per second, the servo may jitter and fail to work properly.

PULSES ALSO CONTROL SPEED

As mentioned, the angular position of the servo is determined by the duration of the pulse. This technique has gone by many names over the years. One you may have heard is *digital proportional*—the movement of the servo is proportional to the digital signal being fed into it.

The power delivered to the motor inside the servo is also proportional to the difference between where the output shaft is and where it's supposed to be. If the servo has only a little way to move to its new location, then the motor is driven at a fairly low speed. This ensures that the motor doesn't "overshoot" its intended position.

But if the servo has a long way to move to its new location, then it's driven at full speed in order to get it there as fast as possible. As the output of the servo approaches its desired new position, the motor slows down.

People often refer to the pulses used to control an R/C servo as *pulse width modulation*, or PWM. That's okay, but it can lead to some confusion.

Technically speaking, R/C servos employ what might be better termed *pulse duration modulation*. With PWM (detailed in Chapter 22, "Using DC Motors"), it's the duty cycle of the pulses—the ratio that each pulse is on versus off—that matters. R/C servos don't care about duty cycles or ratios. All they care about is how long the pulse is. As long as the servo receives at least 20 of these pulses per second (50 is better), it's happy.

You can call whatever goes on inside a servo anything you like, just as long as you remember that a PWM signal for a DC motor bears no relation to the "PWM" signal used to control an R/C servo. In fact, trying to use a PWM signal intended for a DC motor will likely overheat and damage an R/C servo.

VARIATION IN PULSE WIDTH RANGES

Most standard servos are designed to rotate back and forth by 90° to 180°, given the full range of timing pulses. You'll find the majority of servos will be able to turn a full 180°, or very nearly so.

The actual length of the pulses used to position a servo to its full left or right positions varies among servo brands, and sometimes even among different models by the same manufacturer. You need to do some experimenting to find the optimum pulse width ranges for the servos you use. This is just part of what makes robot experimenting so much fun!

The 1-to-2-ms range has built-in safety margins to prevent possible damage to the servo. Using this range provides only about 100° of turning, which is fine for many tasks.

But if you want a full stop-to-stop rotation, you need to apply pulses shorter and longer than 1 to 2 ms. Exactly how long depends entirely on your specific servo. Full rotation (to the stop) for one given make and model of servo might be 0.730 ms in one direction and 2.45 ms in the other direction.

You *must* be very careful when using shorter or longer pulses than the recommended 1-to-2-ms range. Should you attempt to command a servo beyond its mechanical limits, the output shaft of the motor will hit an internal stop, which could cause gears to grind or chatter. If left this way for more than a few seconds, the gears may be permanently damaged.

The 1.5 ms “in-between” pulse may also not precisely center all makes and models of servos. Slight electrical differences even in servos of the same model may produce minute differences in the centering location.



Timing signals for R/C servos are often stated in milliseconds, but a more accurate unit of measure is the *microsecond*—or millionth of a second. In the programming chapters that follow you’ll more often see timing pulses for servos stated in microseconds.

To convert milliseconds to microseconds, just move the decimal point to the right three digits. For example, if a pulse is 0.840 milliseconds, move the decimal point over three digits and you have 0840, or 840 microseconds (lop off the leading zero; it’s not needed).

The Role of the Potentiometer

The potentiometer of the servo plays a key role in allowing the motor to set the position of its output shaft, so it deserves a short explanation of its own.

The potentiometer is mechanically attached to the output shaft of the servo (in some servo models, the potentiometer *is* the output shaft). In this way, the position of the potentiometer very accurately reflects the position of the output shaft of the servo.

The control circuit in the servo compares the position of the potentiometer with the pulses you feed into the servo. The result of this comparison is an *error signal*. The control circuitry compensates by moving the motor inside the servo one way or the other. When the potentiometer reaches its final proper position, the error signal disappears and the motor stops.

Special-Purpose Servo Types and Sizes

While the standard-size servo is the one most commonly used in both robotics and radio-controlled models, other R/C servo types, styles, and sizes also exist.

- *Quarter-scale* (or *large-scale*) servos are about twice the size of standard servos and are significantly more powerful. Quarter-scale servos make perfect power motors for a robot arm.
- *Mini-* and *micro* servos are diminutive versions of standard servos and are designed to be used in tight spaces in a model airplane or car—or robot. They aren’t as strong as standard servos.

- *Sail winch* servos are designed with maximum strength in mind and are primarily intended to move the jib and mainsail sheets on a model sailboat.

See the “Typical Servo Specifications” table, later in this chapter, for a size comparison of these various types.

Gear Trains and Power Drives

The motor inside an R/C servo turns at several thousand RPMs. This is way too fast to be used directly; all servos employ a gear train that reduces the output of the motor to the equivalent of about 50 to 100 RPM. Servo gears can be made of nylon, metal, or a proprietary material.

- *Nylon* gears are the lightest and least expensive to make. They’re fine for general-purpose servos.
- *Metal* gears are much stronger than nylon and are used where brute-force power is needed, but they significantly raise the cost of the servo. They’re recommended for heavier walking robots or large robotic arms. On many servos only some of the gears are metal; the rest are a heavy-duty nylon or other plastic material.
- Gears made of *proprietary* materials include Karbonite, found on Hitec servos. These materials are offered as stronger alternatives to plastic.



Replacement gear sets are available for many servos, particularly the medium- to high-priced ones (\$20+). Should one or more gears fail, the servo can be disassembled and the gears replaced.

Output Shaft Bushings and Bearings

Besides the drive gears, the output shaft of the servo receives the most wear and tear. On the least expensive servos this shaft is supported by a plastic or resin bushing, which obviously can

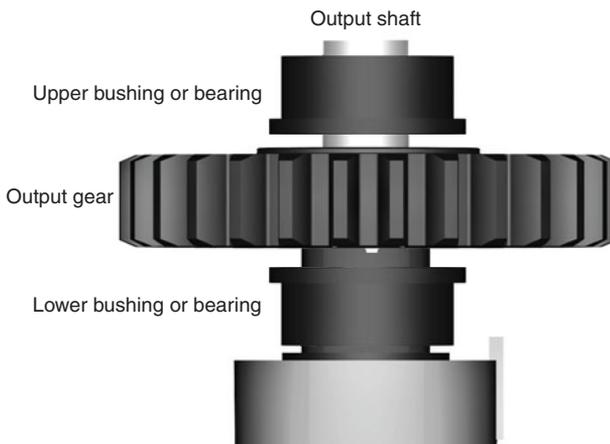


Figure 23-5 Ball bearings or bushings may be placed at the bottom and/or top of the servo output gear, in order to prolong the life of the servo motor.

wear out very quickly if the servo is used heavily. A *bushing* is a one-piece sleeve or collar that supports the shaft against the casing of the servo.

Metal bushings, typically made from lubricant-impregnated brass (often referred to as *Oilite*, a trade name), last longer but add to the cost of the servo. The better servos come equipped with *ball bearings*, which provide longest life.

When looking at servos, you'll often see a notation regarding the bearing type, either bushing or bearing, and whether it's metal or plastic. You also may see a notation for "Top" or "Bottom"; this refers to having a bushing or bearing on the top and/or bottom of the output gear (top *and* bottom is best), like that shown in Figure 23-5.

Typical Servo Specs

R/C servo motors enjoy some standardization. This sameness applies primarily to standardized servos, which measure approximately 1.6" × 0.8" × 1.4". For other servo types the size varies somewhat among makers, as these are designed for specialized tasks.

Table 23-1 outlines typical specifications for several types of servos, including dimensions, weight, torque, and transit time. Of course, except for the size of standard servos, these specifications can vary according to brand and model. Keep in mind that there are variations on the standard themes for all R/C servo classes.

A couple of the terms used in the specs require extra discussion.

- As explained in Chapter 21, "Choosing the Right Motor," the *torque* of the motor is the amount of force it exerts. Servos exhibit very high torque thanks to their internal gearing.
- The *transit time* (also called *slew rate*) is the approximate time it takes for the servo to rotate the shaft a certain number of degrees, usually 60°. The faster the transit time, the faster-acting the servo will be.



You can calculate equivalent RPM by multiplying the 60° transit time by 6 (to get full 360° rotation), then dividing the result into 60. For example, if a servo motor has a 60° transit time of 0.20 seconds, that's one revolution in 1.2 seconds ($0.2 \times 6 = 1.2$), or 50 RPM ($60/1.2 = 50$).

Table 23-1 Typical Servo Specifications

Servo Type	Length	Width	Height	Weight	Torque	Transit Time
Standard	1.6"	0.8"	1.4"	1.3 oz	42 oz-in	0.23 sec/60°
1/4-scale	2.3"	1.1"	2.0"	3.4 oz	130 oz-in	0.21 sec/60°
Mini-micro	0.85"	0.4"	0.8"	0.3 oz	15 oz-in	0.11 sec/60°
Low profile	1.6"	0.8"	1.0"	1.6 oz	60 oz-in	0.16 sec/60°
Sail winch small	1.8"	1.0"	1.7"	2.9 oz	135 oz-in	0.16 sec/60° 1 sec/360°
Sail winch large	2.3"	1.1"	2.0"	3.8 oz	195 oz-in	0.22 sec/60° 1.3 sec/360°

Connector Styles and Wiring

While some servo specifications may vary from one model to another, one thing that's much more standardized is the connectors used to plug servos into their receivers.

CONNECTOR TYPE

There are three primary connector types found on R/C servos:

- “J” or Futaba style
- “S” or Hitec/JR style
- “A” or Airtronics style

Servos made by the principal servo manufacturers—Futaba, Airtronics, Hitec, and JR—employ the connector style popularized by that manufacturer. In addition, servos made by competing manufacturers are usually available in a variety of connector styles, and connector adapters are available.

PINOUT

The physical shape of the connector is just one consideration. The wiring of the connectors (called the *pinout*) is also critical. Fortunately, all but the “old-style” Airtronics servos (and the occasional oddball four-wire servo) use the same pinout, as shown in Figure 23-6.

With very few exceptions, R/C servo connectors use three wires, providing ground, DC power (+V), and signal. The +V DC power pin is virtually always in the center; that way, if you manage to plug the servo in backward, there's less chance of damage. (An exception to this is what's referred to as “old-style Airtronics,” a wiring scheme no longer in use. You may encounter it if you have some older-model Airtronics servos.)

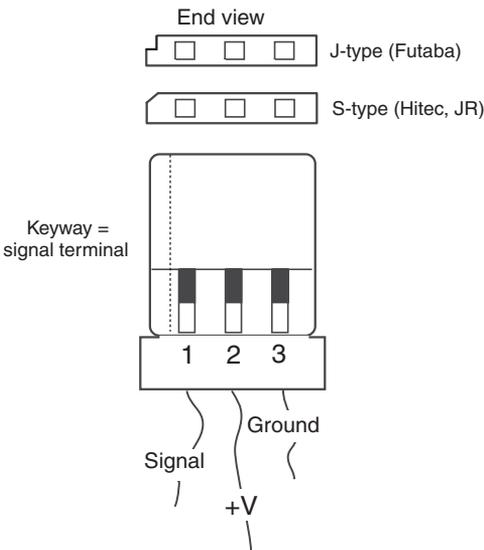


Figure 23-6 Standard three-pin connector used on the vast majority of RC servo motors. The connector may or may not be “keyed” using a groove or notch.

COLOR CODING

Most servos use color coding to indicate the function of each connection wire, but the actual colors used for the wires vary among servo makers. Some of the more common wire color coding is::

- White, orange, or yellow: Signal
- Red: +V (DC power)
- Black or brown: Ground

USING SNAP-OFF HEADERS FOR MATED CONNECTORS

R/C servos and their mating receivers use polarized connectors to help prevent plugging things in back-

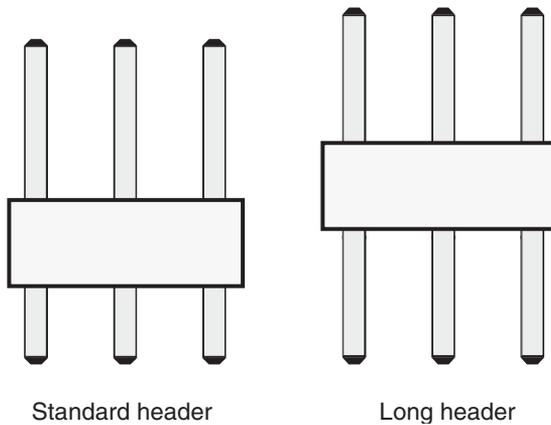


Figure 23-7 A standard male header, versus one that uses long pins on both ends. You want the long-pin version to connect your RC servos to a solderless breadboard.

ward. These polarized connectors are fairly expensive, and most folks instead use 0.100" “snap-off” *pin headers*, common in electronics. You can buy these things at any local or online electronic parts outlet, and they’re pennies a piece.

For a servo, snap off three pins, then solder them to your circuit board. Since these header pins lack any kind of polarization, it’s possible to plug servos in backward. You’ll want to mark how the servo connector should attach to the header, to help prevent this.

Fortunately, reversing the connector *probably* won’t cause any damage to either the servo or the electronics, since reversing the connector merely exchanges the signal and ground wires. This is *not* true of the “old-style” Airtronics connector: if you reverse this connector, the signal and DC power (+V) lines are swapped. In this case, both servo and control electronics can be irreparably damaged.



Damage *can* and *will* occur if you wire up the pin headers wrong. Mix up the Ground and +V pins, and within seconds your servo will be *permanently ruined*—perhaps even die a violent death. I’ve seen the bottoms of servos blown out when they have been connected backward in a circuit.

You can also use headers with solderless breadboards to easily and quickly connect servo motors to the rest of your robot electronics, but it works best if you use the kind of header where the pins are long on both ends (Figure 23-7).

Analog versus Digital Servos

The most common, and most affordable, R/C servos are analog, meaning their control electronics uses traditional circuitry for controlling the motor. Digital servos use onboard microcontrollers to enhance their operation.

Among the added features of digital servos include higher-power and programmable behavior. With the proper external programmer (available separately, and at extra cost) it’s possible to control the maximum speed of the servo, for example, or make the servo always start from power-up in a specific position.

Except for higher torques, from an applications standpoint there is little difference between analog and digital servos. You control both the same way. However the higher torque of a

digital servo means the motor is drawing more current from its power source, which means batteries tend not to last as long between charges.

For most robotics applications, digital servos are not required. You can get by with the less expensive standard analog servos. An exception is when building a walking robot, where the extra torque of digital servos comes in handy. Six-legged walking bots may use 12 and even 18 servos just for the legs. The higher torque helps to offset the added weight of all those servos.

Electronics for Controlling a Servo

Unlike a DC motor, which runs if you simply attach a battery to it, a servo motor requires proper interface electronics in order to rotate its output shaft. While the need for interface electronics may complicate to some degree your use of servos, the electronics are actually rather simple. And if you plan on operating your servos with a PC or microcontroller (such as the Arduino, PICAXE, or BASIC Stamp), all you need for the job is a few lines of software.

A DC motor typically needs power transistors, MOSFETs, or relays if it is interfaced to a computer. A servo, on the other hand, can be directly coupled to a circuit or microcontroller with *no additional electronics*. All of the power-handling needs are taken care of by the control circuitry in the servo, saving you the hassle. This is one of the key benefits of using servos with computer-controlled robots.

CONTROLLING A SERVO VIA A MICROCONTROLLER

All microcontrollers can be used to control an R/C servo. The basic connection scheme is shown in Figure 23-8.

- The microcontroller and servo can share the same power source, assuming the controller has an onboard regulator, but it's much better to use a separate source for the servo. Why? Servos draw a lot of current when they're first turned on or are in motion. By using sepa-

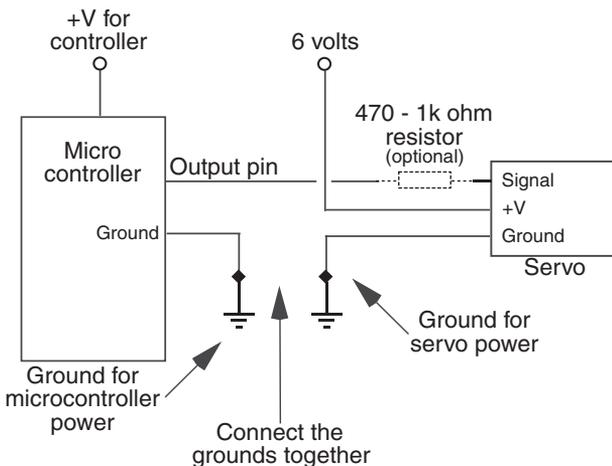


Figure 23-8 General connection diagram for attaching a microcontroller to a servo motor. The 470 to 1 k Ω resistor is optional, and is included to prevent excessive current draw by the servo.

rate sources—a 9-volt battery for the voltage-regulated controller, for example, and a set of 4 AA cells for the servo—you avoid messy power line problems.

- When using separate power sources, be sure to connect the grounds from these sources together. Otherwise, your server will not work properly.
- Only one input/output (I/O) line from the microcontroller is needed to operate the servo. You may insert an optional 470 Ω (ohm) 1/8-watt resistor inline between the controller and the Signal input of the servo. This helps protect the microcontroller in case the servo has an (unlikely) internal electrical problem.



Be sure to power the servos using a separate battery pack. Connect as shown. You might also wish to add a 1 μF to 22 μF tantalum capacitor between the +V and ground of the servo power source to help kill any noise that may be induced into the electronics when the servo turns on and off.

Tantalum capacitors are polarized. Be sure to connect the + lead of the capacitor to the +V pin of the servo. For more information about tantalum and other types of capacitors, see Chapter 31, “Common Electronic Components for Robotics.”

FYI

See the chapters in Part 7, “Microcontroller Brains,” for specific hookup diagrams and programming code for using servos with the Arduino and other microcontrollers. Be sure to also visit the RBB Online Support site (see Appendix A) for additional programming examples.

USING A SERIAL SERVO CONTROLLER

Even the fastest microcontrollers have trouble generating signals for more than 8 or 10 servos at a time—and you may have that many, or more!, if you’ve designed a six-legged walking robot.

Even if your microcontroller is a speed demon and has no trouble creating pulses for 12, 18, or even 24 servos, you may not want to use it for that task. Instead, you may wish to use a dedicated serial servo controller. It acts as a pulse-making coprocessor.

Servo controllers connect to your microcontroller via a serial communications line. Using programming code (examples are provided with the unit you buy), you then send commands to each servo, telling it to move to a certain position. The work of producing the properly timed pulses is the job of the servo controller, thus freeing up your microcontroller to do other, more important jobs.

BONUS PROJECT: CONTROLLING A SERVO VIA AN LM555 TIMER IC, AND MORE

See the RBB Online Support site for additional methods of controlling an R/C servo, including using an LM555 timer IC, practical examples of using several popular serial servo controllers (like the Lynxmotion SSC-32), and more. See Appendix A for details on how to access the RBB Online Support site.

USING GREATER THAN 7.2 VOLTS

Servos are designed to be used with rechargeable model R/C battery packs, which put out from 4.8 to 7.2 volts, depending on the number of cells they have. Servos allow a fairly wide

latitude in input voltage, and 6 volts from a four-pack of AAs provides more than enough juice. As the batteries drain, however, the voltage will drop, and you will notice your servos won't be as fast or strong as they used to be.

But what about going beyond the voltage of typical rechargeable batteries used for R/C models? Indeed, *some* servos can be operated at 7.2 volts, but always check the datasheet that comes with the servos you are using. Unless you need the extra torque or speed, it's best to keep the supply voltage to your servos at no more than 7.2 volts, and preferably between the rated 4.8- to 6-volt range specified in the manufacturer's literature.

WORKING WITH AND AVOIDING THE "DEAD BAND"

References to the Grateful Dead notwithstanding, all servos exhibit what's known as a *dead band*. The dead band of a servo is the maximum time difference between the incoming control signal and the internal reference signal produced by the position of the potentiometer. If the difference equates to less than the dead band—say, 5 or 6 microseconds—the servo will not bother trying to nudge the motor to correct for the error.

Without the dead band, the servo would constantly "hunt" back and forth to find the exact match between the incoming signal and its own internal reference signal. The dead band allows the servo to minimize this hunting so it will settle down to a position close to, though maybe not exactly, where it's supposed to be.

The dead band amounts vary and are often listed as part of the servo's specifications. A typical dead band is 5 μ s. If the servo has a full travel of 180° over a 1000- μ s range, then the 5- μ s dead band equates to 1 part in 200. If your control circuitry has a resolution higher than the dead band, then small changes in the pulse width values may not produce any effect. For instance, if the controller has a resolution of 2 μ s, and if the servo has a dead band of 5 μ s, you must make changes in the pulse width in no less than 5- μ s increments.

Using Continuously Rotating Servos

So far I've only talked about servos that are meant to turn a portion of a circle. These are used when precise angular positioning is required, such as scanning a sensor from side to side.

But R/C servos can also rotate continuously, either by design or via a modification that you perform yourself. R/C servos make terrific drive motors for your robot. They tend to be less expensive than comparable DC gear motors of the same specification, and they come with their own driver electronics. They're definitely worth considering for your next robot.

Servos that rotate continuously act like an ordinary geared DC motor, except they're still controlled by sending the motor pulses.

- To make the motor go in one direction, send it 1 ms pulses.
- To make the motor go in the other direction, send it 2 ms pulses.
- To make the motor slow to a stop, send it 1.5 ms pulses.
- To make the motor stop altogether, stop sending it pulses.



Stopping the motor by ceasing the pulses works for all except digital servos. With most digital servos, when pulses are stopped the servo will merely continue with the last good position information it received.

You're not likely to use digital servos for continuous rotation, so this problem seldom comes up in real life. But keep it in mind just in case.

As of this writing, there's just a small handful of servos made for continuous rotation. These include the GWS S-35, the Parallax Continuous Rotation Servo, and the SpringRC SM-S4303R. These are available from a variety of online resellers; see Appendix B, "Internet Parts Sources," for more information.

Modifying a Standard Servo for Continuous Rotation

You can convert most any servo to continuous rotation. Once modified, they no longer are capable of precise angular rotation, but they're perfectly suited as drive motors for wheeled and tracked bots.

There are a number of methods you can use to modify an R/C servo for continuous rotation. The process involves removing the mechanical stops, and—for some types of servo surgery—making a change in the electrical connections inside.

WAYS TO MODIFY A SERVO

There are many ways to modify R/C servos. In reverse order of difficulty:

- *Power-train mod.* For this modification, the control electronics are removed completely, and the motor is driven by an external H-bridge. Any stops are removed from the power train, and the potentiometer is either removed or its shaft is otherwise disengaged from the servo drive. This is a lot of work, and most people don't bother.
- *Signal tap-in mod.* In this conversion the electronics of the servo remain, but, as usual, the mechanical stops are removed and the potentiometer is disengaged. In this variation, the robot's computer or microcontroller is connected to the H-bridge chip within the servo's circuitry. Not all servos are so adaptable, however. Several Internet sites detail this tapping-in process for the BAL6686 H-bridge IC, used in certain Futaba models and other servos. Do a Web search for "BAL6686" for a number of useful sites that describe this.
- *Potentiometer mod.* Here all the control circuitry is left in place, but the potentiometer is either removed completely (and replaced with a couple of resistors) or disengaged from the power train. This is the easiest modification to make, and the most popular.

Of these methods I'll talk only about the last one, as the others have far more restricted uses and are much more difficult. The potentiometer mod that follows is relatively quick and easy, requires no soldering, and needs basic basic tools.

BASIC MODIFICATION INSTRUCTIONS

Servo modification varies somewhat among makes and models, but the basic steps are the same:

1. Remove the case of the servo to expose the gear train, motor, and potentiometer. This is accomplished by removing the four screws on the servo case and separating the top and bottom.
2. File or cut off the nub on the underside of the output gear that prevents full rotation. This typically means removing one or more gears, so you should be careful not to

misplace any parts. If necessary, make a drawing of the gear layout so you can replace things in their proper location.

3. Remove the retainer clip at the bottom of the output gear. Doing this disengages the potentiometer from the gear, so that the pot no longer turns when the gear does. On some servos there is no retainer clip; the servo engages into a spline molded into the bottom of the gear. On these you need to carefully drill out the bottom of the gear.
4. Reassemble the case.

In the following section you'll find detailed modification instructions for the Hitec HS-422—the same instructions apply to many other servos, such as the venerable Futaba S-148 and the GWS S03. With the same or minor variations, the steps that follow can be applied to similarly designed servos.

TOOLS YOU NEED

You'll need the following tools to complete the conversion process:

- #0 Phillips screwdriver
- 1/8" or smaller flat-bladed screwdriver
- "Nippy": cutters, X-Acto blade, or razor saw
- Small, flat jeweler's file

CHOOSING A SERVO TO MODIFY

The best servos, from a standpoint of easy modification:

Use a lower ball bearing or bushing that supports the output gear. At the very least, the output gear should be supported by a molded-in ledge, rather than directly on the potentiometer shaft.

Use a removable potentiometer shaft clip. The clip can be readily removed in order to disengage the output gear from the potentiometer shaft. Servos that lack a removable clip will use instead a molded-in channel that the potentiometer fits into. This is the case of the Hitec HS-311 and the Futaba S-3003. If your servo is so constructed, you'll need to carefully drill out the bottom of the output gear in order to remove the spline, as detailed later in this chapter.

Use a plastic output gear, rather than a metal gear. The better, heavy-duty servos use a metal output gear. While the metal is able to handle increased stress, it's significantly harder to modify. If your servo of choice uses a metal output gear, a motorized hobby tool such as the Dremel will make the job of grinding down the nub much easier.

Are normal sized or larger. Mini and micro servos are more difficult to modify. You may ruin one or two before you get the hang of it.

STEPS FOR MODIFYING A HITEC HS-422

The Hitec HS-422 is a good middle-of-the-road servo: low price, available anywhere, and well made. It employs both a top and bottom oil-impregnated brass bushing and a removable clip on the underside of the output gear. Perfect!

Throughout the following, use care to avoid wiping off or absorbing through your skin too

much of the lubricant used for the internal gears of the servo. If you think lubrication has been lost, you can always add more just prior to reassembly. Clear (or white) gear grease is available at any hobby store that sells R/C parts. Apply the grease sparingly. Do not use a spray-on lubricant such as WD-40.



Before performing the modification, test the servo for proper operation, just to be sure it's not bad to begin with. Though rare, some servos fail to work right out of the box. Once it is modified, you cannot return the servo for in-warranty replacement.



1. Using a Phillips screwdriver, remove the horn wheel, if one is attached to the servo.
2. Untighten the four casing screws from the bottom of the servo. If they don't get in the way in the following steps, you can keep the screws in place in the bottom of the case. Unscrew just enough to remove the top of the servo case. Note that on a few servos, notably the S03 series from GWS, the case screws are removed from the top, and these should be removed all the way and set aside while you work.
3. Observe the orientation of all the gears. Remove the center gear, being careful not to unseat its metal shaft. On most servos the center gear cannot be easily removed without also lifting up the output gear at the same time. Place the center gear aside.
4. Remove the output gear.
5. Using nippy cutters, X-Acto blade, or razor saw, remove the nub on the top side of the output gear. I prefer the cutters, *but exercise caution!* The harder the plastic, the more likely the nub will *break off at high speed*. Wear eye protection. Always nip first on the long side, to prevent possible breakage of the gear, and cut off *only a small portion* at a time. When using an X-Acto blade or razor saw, the obvious precautions against cutting your fingers off should be observed. Work slowly.
6. Odds are, no matter what cutting technique you use, a small portion of the nub will remain. This can be filed down with a small flat file.
7. Use the small-bladed screwdriver to remove the metal retaining ring from the underside of the output gear. This ring retains the potentiometer shaft clip and also serves as a bearing surface.
8. Use the small-bladed screwdriver to remove the retaining clip.
9. Replace the metal retaining ring back into the output gear.
10. Align the potentiometer shaft so that it's centered. If needed, rotate it back and forth to find the center.
11. As an optional step, you may want to connect the servo to your control circuit. Apply 1.5 ms (1500 μ s) pulses. If the motor turns (even slowly), rotate the potentiometer shaft until all rotation is "nulled out."
12. Once set to its center, you can leave the shaft as is or apply a *very small* dab of cyanoacrylate glue (Super Glue) to keep the shaft in place. Do not apply too much glue, or the potentiometer may be damaged.
13. Reassemble by placing the output gear on its seat over the potentiometer. Replace the middle gear, and observe that all gears properly mesh. Add more grease at this point, if needed. Finally, replace the top case and the four case screws. Don't overtighten the screws.

Test the servo for proper operation by connecting it to your control circuit. A series of 1.5 ms pulses should stop the servo. A signal of 1.0 ms pulses should rotate the servo in one direction; 2.0 ms pulses should rotate the servo in the other direction.



These steps are virtually the same for several other popular low-cost servos, including the Futaba S-148, and the S03 series of servos from GWS.

STEPS FOR MODIFYING A FUTABA S3003 SERVO

The Futaba S3003 is a low-cost alternative to servos with metal bushing or ball bearings. As with many servos of its type, it doesn't use a retained clip for the potentiometer. You need to drill out the bottom of the gear so that the gear will no longer engage with the potentiometer.

1. Follow steps 1 through 6, above.
2. Using a spare servo horn (the larger, the better), attach the horn to the output gear using the mounting screw provided with the servo.
3. For steps 7 through 9, instead, carefully drill out the bottom of the output gear with a 3/16" (or thereabouts) drill bit. Remove as much of the plastic as necessary so that the potentiometer shaft will not engage the bottom of the gear. Use a drill press if you have one. See Figure 23-9. If you don't have a drill press, get someone to hold the output gear (see the note below) while you drill out the spline.
4. Complete the remainder of the steps above.

When you drill out the spline in the bottom of the gear, hold the gear steady by clamping the servo horn—not the gear—using a pair of heavy-duty pliers. Do not clamp by the shaft or gear face, as this could wreck the plastic and you'll end up with a badly functioning servo. Be sure not to ream or drill out too much, or you'll ruin the gear.

TESTING THE MODIFIED SERVO

After reassembly but before connecting the servo to a control circuit, you'll want to test your handiwork to make sure the output shaft of the servo rotates smoothly. Do this by attaching a control disc or control horn to the output shaft of the servo. *Slowly* and carefully rotate the disc or horn and note any snags. Don't spin too quickly, as this will put undue stress on the gears.

If you notice any binding while you're turning the disc or horn, it could mean you didn't remove enough of the mechanical stop on the output gear. Disassemble the servo just enough to gain access to the output gear and clip or file off some more.

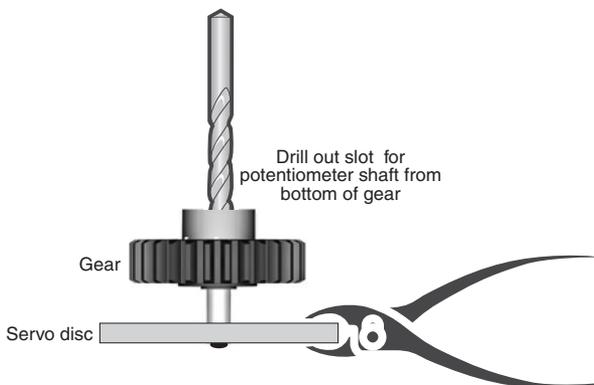


Figure 23-9 If modifying a servo where the slot for the potentiometer shaft is molded into the plastic of the output gear, attach the gear to a servo horn and drill out the bottom with a 3/16" drill bit. Hold the servo horn with pliers.

LIMITATIONS OF MODIFIED SERVOS

Modifying a servo for continual rotation carries with it a few limitations, exceptions, and “gotchas” that you’ll want to keep in mind:

- The average servo is not engineered for lots and lots of continual use. The mechanics of the servo are likely to wear out after perhaps as little as 25 hours (that’s total elapsed time, which for a robot is quite a long time), depending on the amount of load on the servos. Models with metal gears and/or brass bushing or ball bearings will last longer.
- Standard-size servos are not particularly strong in comparison to many other DC motors with gearboxes. Don’t expect a standard servo to move a 5- or 10-pound robot. If your robot is heavy, consider using either larger, higher-output servos (such as 1/4-scale or sail winch) or, better yet, DC motors with bolted-on gearboxes.
- Last and certainly not least, remember that modifying a servo voids its warranty.

Using Servo Motors for Sensor Turrets

After all this talk of using R/C servos as robot drive motors it’s easy to forget what they were created for in the first place: for precise position and control. A common application in robotics is the sensor turret, so called because it acts as a rotating turret (like a cannon gun turret) for one or more robot sensors. Typical sensors for turrets include ultrasonic and infrared proximity detection—these are detailed in Chapter 43, “Proximity and Distance Sensing.”

The concept of the rotating sensor turret is simple: put a sensor on top of the servo, and then “scan” it back and forth by alternating the position of the servo left and right. A simple way to mount a sensor is atop a servo horn. You can use double-sided foam tape, hot-melt glue, even a rubber band (wound around a couple of times) to secure the sensor to the horn.

Or use a specialty sensor bracket, like the one in Figure 23-10. Sensor brackets come in all shapes and sizes; the one in the picture is designed for the typical two-transducer ultrasonic sensor, such as the Devantech SRF05 or the Parallax Ping. Holes in the back of the bracket allow the connection wires to come through.

You can mount the servo to the robot using a variety of methods. Foam tape and glue are always options, but I prefer mechanical fasteners that can be easily removed and reattached; the fasteners make it easier to build and change the robot.



Figure 23-10 A servo used to rotate a sensor, mounted on a bracket.

Mounting Motors and Wheels

You've got two motors. You've got a robot body. What comes next isn't always simple: you have to somehow mount the two motors onto the robot body, hopefully without making the thing look like a junkyard reject! Then there's the problem of attaching the wheels.

DC motors and R/C servos each have their own means of mounting to a robot platform or frame. Some are easier than others. In this chapter you'll learn ways to mount both common and not-so-common motors to robot frames and platforms. And you'll learn about attaching wheels to those motors.

To round out the chapter, you'll find helpful information about using standard drivetrain components—things like gears, chain, belts, and couplers.

Mounting DC Motors

There are no hard standards in the design of a DC motor. Depending on how the motor was meant to be used, it may be a snap to mount, or it could be cumbersome, requiring a hodgepodge of hardware.

- Generally speaking, motors meant for use in a variety of applications tend to have holes, brackets, or flanges that make mounting easier.
- Those motors engineered to work with just a specific product rely on the design of that product for secure mounting. There are no holes, no brackets, no flanges.

Motors meant for robotics (or at least home-shop tinkering) are made with mounting in mind. These include the various Tamiya motor kits, all of which have flanges or other means for secure mounts. Solarbotics, Pololu, and several other robotics-centric online retailers import motors that have been especially selected because they offer mounting ease. Or else they provide their own mounting solutions, especially crafted for the motors they offer.

These and other motors like them are the easiest to work with, and if you're just starting out in robotics, these should be the ones you choose. It'll make your life much easier!

USING MOUNTING HOLES

Some motors have already-threaded holes on their faceplate (the side where the shaft comes out), like that in Figure 24-1. The threads may be imperial (inch) or metric. For imperial, 2-56 and 4-40 threads are the most common. For metric, you'll often find holes tapped for 3mm or 4mm machine screws.

If you have a die-and-tap set, you can drill your own holes or retap existing holes. To do this you must disassemble the motor to remove the faceplate so you can drill new holes. Disassembly is required to prevent metal shavings (from drilling and tapping) from getting into the motor, which will render it inoperable.



It's important that the machine screws you use don't go too far into the motor, or else they may obstruct the rotating parts inside. If you don't have the correct length of screw, you can either cut it to size or add washers on the outside. Washers don't look as good, but they're the easier solution.

If you're mounting the motor perpendicular to the body of the robot, you may need to come up with a bracket to secure the motor to. More about brackets later in this chapter.

USING BUILT-IN FLANGES

Oh, happy joy—a motor with its own mounting flanges! It doesn't get much better than this, so enjoy while you can. You know what you need to do here: just find some nuts and screws that fit, mark where the holes should go on your robot, and attach.

When mounting with flanges, I like to insert the head of the screw through the flange on the motor side (adding a flat washer if the head might come through the flange). A nut secures everything on the other side (Figure 24-2).



Figure 24-1 Motors with faceplate holes may use a flange for mounting. This method requires the proper size of machine screws; if the screws are too long, they'll interfere with the moving parts inside the motor.

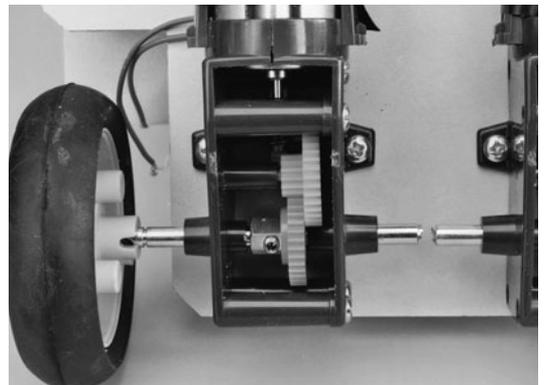


Figure 24-2 On motors with a mounting flange, insert the fastener from the flange side, when possible. In this way the exact length of the screw is not as critical, and you can use self-tapping screws that secure into the base material (wood, plastic, etc.).

USING BRACKETS

You need brackets if you must attach a vertically mounted motor to a horizontal surface. Metal brackets are common finds at any home improvement store. They tend to be a bit bulky, being made for general household applications. For smaller bots I prefer lightweight plastic brackets available at specialty online stores and miniature metal hardware available from the larger electronic parts outlets.

You can also fashion your own mounting brackets using metal or plastic. Cut the bracket to the size you need, and drill mounting holes. This technique works well when you are using servo motors for radio-controlled model cars and airplanes.

Brackets from Metal

You can make your own L-shaped metal brackets by bending a strip of aluminum, brass, or stainless steel metal. You can find these metal strips at many hobby stores. Aluminum and brass are easier to work with. They're best for smaller motors. For larger and heavier motors, opt for stainless steel.

1. Cut the strip to the desired length—for example, make the strip slightly more than 2" long for an L-shaped bracket that is 1" × 1". Each side of the L is a "leg" of the bracket, and it's 1" long.
2. Mark the midpoint of the bracket with a pencil. For that 2" bracket, for instance, place a mark 1" in.
3. With the metal strip held securely in a vise, drill at least one hole in each leg (see Figure 24-3). For larger brackets, drilling two holes is even better. Stay away from the midpoint mark you made in step 2.
4. For this next part, use a vise secured to your workbench. Put the strip into the vise so that the midpoint mark is just visible, and tighten the vise.

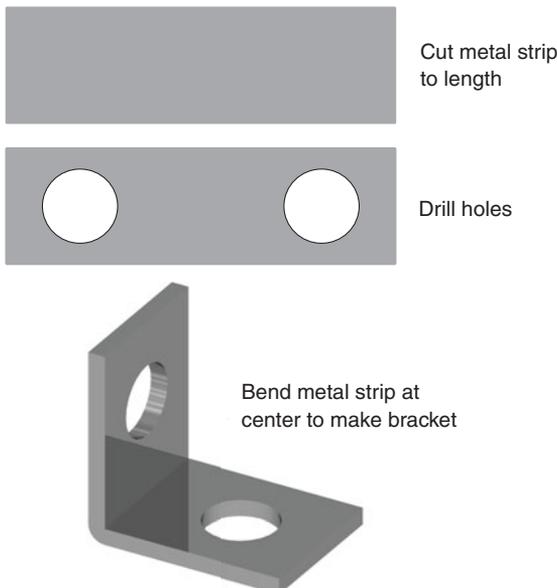


Figure 24-3 Make your own metal angle brackets by cutting strips of metal (copper, brass, aluminum) to length, then drilling holes. Bend at the middle to finish the bracket.

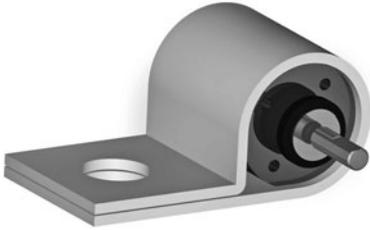


Figure 24-4 Pipe holders or straps (in metal or plastic) may be used to mount motors with round casings. Plastic straps provide more leeway in matching the size of the motor.

5. Use a hammer and a block of wood to fold over the exposed leg of the bracket. You want a neat 90° angle.



When making small brackets, drill the holes first, then cut the strip to length. You can make several bracket pieces at a time this way. If you need to make lots of brackets, draw the layout on a piece of paper and use it as a template for placing the holes and cuts.

Bracket Blocks from Wood and Plastic

You can make convenient block-style mounting brackets out of small pieces of 1/2" or thicker wood or plastic. Start with a minimum 1.5"-long block. On one face of the block drill two holes close to the outside edges. On the other face drill at least one hole toward the center part of the block. You can then use the block as a type of L bracket.

For larger motors, use bigger and longer blocks. You may need two and even three mounting holes per face. Drill larger holes for bigger machine screws.

For plastic materials, visit your local neighborhood plastics retailer (check the Yellow Pages for one near you). Most have a discard bin in the front showroom with odds and ends. Look for small scraps of thick nylon, acetal resin (Delrin), and ABS plastic. The scrap is usually sold by the pound; enough material for a half dozen blocks should cost only a few dollars.

USING CLAMPS

If the motor lacks mounting holes, you can use clamps to hold it in place. U-bolts, available at the hardware store, are excellent solutions. Choose a U-bolt that is large enough to fit around the motor.

A technique that works with smaller motors is to use hold-down straps designed for EMT (electrical) conduit pipes. The straps, like the one in Figure 24-4, are available in various sizes, to hold down pipes of different diameters. These pipe straps are available at your local home improvement and hardware stores, in both metal and plastic. The plastic ones are easier to work with and lighter.

Mounting and Aligning Motors with Aluminum Channel

The same (but somewhat larger) aluminum channel used to construct robot frames can be used to mount and align DC motors. Find a channel that's large enough inside for your motors to drop in place. A snug fit is best. This technique not only allows you to use a motor that otherwise lacks a convenient means of mounting, but guarantees that the two motors are precisely aligned with one another.



Figure 24-5 Motors in a differentially steered robot may be secured and aligned by placing them inside a strip of aluminum U-channel. The inside dimensions of the channel must be large enough to accommodate the girth of the motor. Use plastic cable-ties to hold the motors in place.

Cut the channel to length and, as shown in Figure 24-5, place the motors within the channel, end to end. If the motors protrude from the channel, you might be able to secure them in place using nothing more than a cable tie—for demonstration purposes, the illustration shows one tie, but you’ll probably need several to hold the motor in place. Cinch up the tie so that it firmly holds the motor.

Mounting R/C Servos

The world is quite a different place when working with R/C servos. By their nature, servos have mounting flanges; what’s more, servo sizes and mounting configurations are fairly standardized, giving you the option of premade mounting solutions if you don’t want to “roll your own.”

Whatever the method, servos should be securely mounted to the robot so the motors don’t fall off while the thing is in motion. Over the years, I’ve found “hard mounting”—gluing, screwing, or bolting—the servos onto the robot body to be the best overall solution. These techniques greatly reduce the frustration level of hobby robotics.



An exception to hard mounting is when creating so-called rapid prototypes, robots that test design principles and aren’t necessarily meant to last long. For these, things like Velcro, sticky tape, and tie-wraps are adequate for the job. Read more about this concept in Chapter 14, “Rapid Prototyping Methods.”

ATTACHING SERVOS WITH SCREWS

Unless you have a good reason not to, the best way to mount R/C servo motors to your robots is with screw fasteners. You have two kinds of screws to choose from:

- Self-tapping metal or wood screws don’t need a nut on the other end to hold things in place. Drill a small pilot hole to start, then insert the screw. The threads of the screw dig into the material and hold it into place.
- Machine screws and nuts are ideal if you need to disassemble your creation and rebuild it, or use parts for something else.

Specialty Premade Servo Mounts

With the popularity of servos for robotics applications there’s no shortage of mounts for all types and sizes of servos. These are available from online sources and are not likely to be something you’ll find at a local store; see Appendix B, “Internet Parts Sources,” for a selected list of robotics specialty outlets, many of which offer servo mounts.

Making Your Own Servo Mounts

You can also construct your own servo mounting brackets using 1/8"-thick aluminum or plastic. A template is shown in Figure 24-6. (Note: The template is not to scale, so don't trace it to make your mount. Use the dimensions to fashion your mount to the proper size. If you'd like a scale version of this mount, visit the RBB Online Support site; see Appendix A for the details.)

The first step in constructing your own servo mounting brackets is to cut and drill the aluminum or plastic following the template. Use a small hobby file to smooth off the edges and corners. Use 4-40 screws and nuts, or #4 self-tapping screws, to attach the servo mount to wood or plastic.

Servo "Tab" Mounting Brackets

For R/C servo motors of any size you can create simple mounts using a pair of "tab brackets. Start with a strip of 1/8"-thick aircraft plywood or plastic, and drill a pair of holes to match those on the flange of the servo motor. No closer than 3/8" from these holes, drill one or two holes for a metal or plastic angle bracket.

After drilling, cut the strip near the pair of servo flange holes. Repeat this process again for the second bracket. Once you get the hang of it you can drill and cut a pair of servo brackets in just minutes. Figure 24-7 shows a pair of tab mounting brackets on a servo.

ATTACHING SERVOS WITH GLUE

Gluing is a quick and easy way to mount servos on most any robot body material, including heavy cardboard and plastic. For a permanent bond, use only a strong glue, such as two-part epoxy. For a somewhat less permanent construction, I prefer hot-melt glue because it doesn't emit the fumes that epoxy does and it sets much faster.

When gluing, it is important that all surfaces are clean. Rough up the surfaces with a file or heavy-duty sandpaper for better adhesion. If you're gluing servos to LEGO parts, apply a generous amount so the extra adequately fills between the "nubs." LEGO plastic is hard and smooth, so be sure to rough it up first.

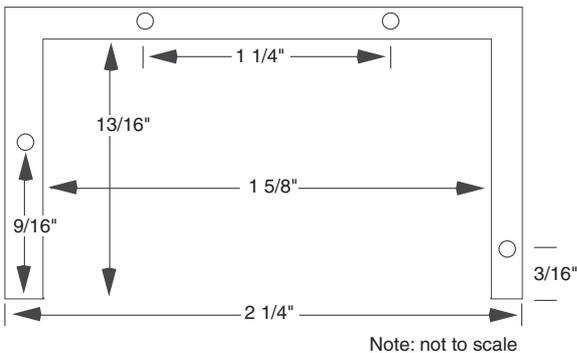


Figure 24-6 Make your own radio control servo mounts following this template (it's not shown here to scale; be sure to draw it out on paper rather than trace it from the book). Cut the mount in wood, plastic, or metal, but make sure the material is thick enough to support the motor.



Figure 24-7 A pair of servo mounting tabs, shown with servo and 3/8" metal angle brackets for securing to a robot base.



If you don't need or want to permanently attach the servos, use less hot-melt glue or use a dab of ordinary white household glue. The bond is weaker, allowing you to more easily pry the servo off and reuse it for something else.

Mounting Drivetrain Components to Shafts

Drivetrain components are things like wheels, gears, sprockets, and other stuff that attach to your robot's motors. Unless the parts are specifically designed for one another, connecting the shaft of the motor to drivetrain components is one of the more difficult tasks in building a robot.

Still, it's not impossible, and robotics wouldn't be as much fun without the occasional challenge. What follows are the most common methods used to connect a motor shaft to a wheel, gear, or other drivetrain component.

PRESSFITTING

For smaller drivechains, it's not uncommon to use pressfit parts, where the shaft fits very tightly into the wheel or other component. Usually these are manufactured to fit this way, and the parts are assembled with heavy-duty hydraulic presses. You probably don't have such a press in your garage; you'll need to instead use a small hammer, a bit of spit for lubrication, and a few well-chosen curse words to knock the shaft into position.

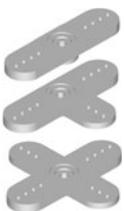
SETSCREWS

A setscrew physically clamps the wheel or other part directly to the shaft. Setscrews are convenient and elegant, but they're not common in amateur robotics because drivetrain parts that use them tend to be more expensive. You're more likely to encounter setscrews when using small metal R/C parts like gears or locking collars (see the section that follows, on making your own shaft couplers).

Setscrews usually have hex socket heads, so you need a hex wrench in order to remove or tighten the screw. When purchasing R/C parts that use setscrews, the wrench is usually included.

PROPRIETARY INTERLOCK

Some motors and wheels (and other drivechain components) are made to go together. Good examples are the various DC gearbox motors and wheels from Tamiya. Many of the motors and wheels are designed for interchangeability. The wheels fit onto the motor shaft using nuts, pins, or other means.



SERVO HORNS

The output shaft of R/C servo motors has about two dozen tiny splined teeth carved around its circumference. These splines, and a metal screw that's inserted into the bore of the shaft, are made to secure various styles and sizes of *servo horns*. Most servos come with at least one of these horns—typically a round or small, X-shaped jobbie.

Servo horns are made of plastic or metal (when of metal, it's usually aluminum). You

can drill holes for attaching things to the horns. This is one of the primary ways of securing wheels to a servo. You can also glue parts to the horns, though this works best when bonding plastic pieces to plastic horns. Hot-melt glue and epoxy are good choices.

ADHESIVES

Some motor shafts, wheels, and other drivetrain components can be bonded with an adhesive. This technique works best for plastic parts—don't try to glue a metal shaft to a plastic wheel.

An example of using adhesives is to cement a LEGO axle into a non-LEGO plastic wheel. You can then use the axle/wheel combo in a LEGO creation. Drill out the hub of the wheel so it's just smaller than the axle. Then gently tap the axle into the hole using a small hammer. You may set the axle in place using epoxy or household adhesive.

Mounting Wheels to DC Gear Motors

In the world of amateur robotics, you can now find motors that match the wheels made for them. Those are the easiest to use when building a bot, but there are many other options, too. With some effort, you can adapt a wheel to most any kind of motor, using either a direct connection—as described in this section—or a coupler, as detailed later in the chapter.

USING MATCHING MOTORS AND WHEELS

When I first started in robotics I used to spend an inordinate amount of time combing through various surplus catalogs, looking for motors and wheels that could go together. It wasn't always easy to find matches. Today there are numerous specialty online robotics retailers that offer low-cost DC motors and wheels that are designed to complement one another.

And when I say low-cost, I do mean low-cost—it's easy to find a motor-and-wheel set that's priced under \$10 each. You need two for a basic bot. Add some batteries, wire, some simple electronics (or even a microcontroller like the PICAXE), and you have a fully functional autonomous robot for under \$30. Not bad.

BUILDING CUSTOM WHEELS

While matching motors and wheels are handy, selection isn't always great. You may want a smaller or larger wheel than what's offered, or you may not like the width of the rubber treads on the wheel. That's when you need to come up with your own motor-to-wheel solution.

FYI

Be sure to also see the section "Using Rigid and Flexible Couplers" for more ideas on how to match wheels to motor shafts. Using couplers requires a bit more work than the methods outlined here, but sometimes you have no other choice.

Wheels with Setscrews

If your wheel already has a hub with a setscrew, you're in business . . . assuming the wheel hub is the right size for the motor shaft. If it is, you're ready to go. But if it's not, there are a couple of things you can do to solve the problem:

- If the wheel hub is too small, drill it out to fit the shaft. Obviously, this works only if the hole you drill isn't so large that it destroys the wheel hub.

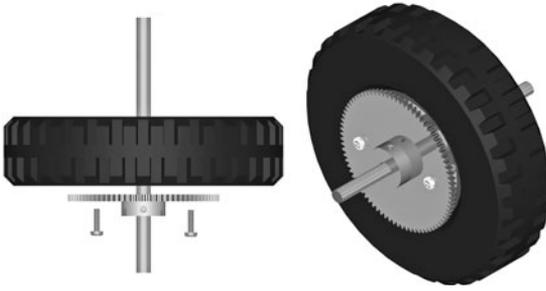


Figure 24-8 The concept behind using a metal or plastic gear as a flange for mounting a wheel to a motor shaft. Attach the face of the gear against the wheel hub; the gear provides a hub that can then be secured to the motor shaft.

- Use a reducing bushing, which is basically a short length of hollow tube. The inside of the bushing is sized to match the motor shaft; the outside, the wheel hub. You then carefully drill a hole through the bushing. Use a longer setscrew, if needed, to make it extend through the bushing and make contact against the shaft.

Using Flanges

But what if the wheel lacks a setscrew? One solution is to use a *flange* that does have a setscrew, and a hub with a properly sized bore that matches the motor shaft. You attach the flange to the wheel using fasteners, then mount the flange to the shaft.

A flange is anything smaller than the diameter of the wheel and that has a setscrew and hub compatible with the motor shaft you want to connect it to. Specially made flanges are available through Lynxmotion, Jameco, Servo City, and other online retailers. The flanges come in different bore sizes. Cost is modest.

A convenient ready-made flange is a surplus gear. Drill two or three holes through the face of the gear and matching holes into the wheel, being sure to keep the gear and wheel concentric. Mount the flat part of the gear against the wheel (see Figure 24-8), then attach the assembly to the motor shaft.

Mounting Wheels to R/C Servos

Servos reengineered for full rotation are most often used for robot locomotion and are outfitted with wheels. Since servos are best suited for small- to medium-size robots (under about 3 pounds), the wheels for the robot should ideally be between 2" and 5" in diameter, and lightweight.

WHEELS ENGINEERED FOR R/C SERVOS

By far the easiest way to attach wheels to servos is to use a wheel that's specially engineered for the job. Many specialty robotics retailers sell wheels meant for use with the standard-size Hitec and Futaba servo.

Now for the bad news: Your choice of wheel diameters is pretty limited. You'll find just a few sizes, with 2-1/2" (give or take a few fractions of an inch) the most common. If you need a smaller or larger size, you can always make your own wheels, as detailed next.



Figure 24-9 Most any kind of wheel can be converted for use with an R/C servo by attaching a servo horn to the side of the wheel. If your servo didn't come with an assortment of horns, you can purchase them separately.



Servos differ in the type of spline used on their output gear. The three most common spline types are noted simply by the servo manufacturer that popularized them: Hitec, Futaba, and Airtronics. If you are purchasing wheels for your servos, make sure the wheels use a matching hub spline. Wheels made for standard-size Futaba servos will also work on any other brand that uses the same Futaba-style spline, such as GWS.

MAKING YOUR OWN WHEELS FOR SERVOS

The general approach for attaching wheels to servos is to use the round servo horn that comes with the servo and secure it to the wheel using screws or glue (see Figure 24-9). The underside of the horn fits snugly over the output shaft of the servo. Here are some ideas:

Lightweight foam tires, popular for model airplanes, can be glued or screwed to the servo horn. Popular brands are Dave Brown and Du-Bro, and these can be found at most any well-stocked R/C hobby store. The tires are available in a variety of diameters, with the 2", 2-1/2", and 3" diameters the best for small bots.

Large LEGO "balloon" tires have a recessed hub that exactly fits the small, round servo horn included with Hitec and many other servos. You can simply glue the horn into the rim of the tire.

A gear glued or screwed into the servo horn can be used as an ersatz wheel or as a gear that drives a wheel mounted on another shaft.

Homemade O-ring wheels can be constructed out of two plastic discs, cut to any diameter you like—though about 3-1/2" is a practical maximum. The O-ring is the rubber tire of the wheel. At the center of the discs, mount a large, round servo horn, then fasten the pieces together using miniature machine screws and nuts.

Pulley horns look like three discs cemented together with a space in between. You can turn the pulley horn into a unique wheel by adding a pair of rubber O-rings as tire treads.

Urethane skateboard/inline roller-skate wheels make for incredibly "grippy" wheels for robots. The trick, if it can be called that, is to find metal or plastic discs that just fit into the hub of the wheel. Most skate and inline blade wheels use metric sizes, with a hub diameter of 22 mm. A 0.625"-diameter fender washer fits into the bore of the wheel but is (usually) large enough to stop against the ridge that's molded into the center of the wheel. Hold all the pieces together with a 4-40 machine screw that goes from the outside of the wheel and directly into the servo motor output shaft.

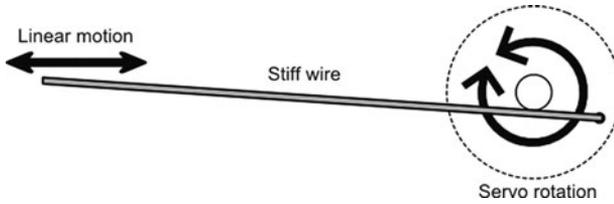


Figure 24-10 Rotary motion can be converted to linear motion by using a pushrod linkage. As the servo rotates, the end of the linkage moves back and forth.

Attaching Mechanical Linkages to Servos

A key benefit of using R/C servos is the variety of ways you can connect stuff to them. In model airplane and car applications, servos are most often connected to a push/pull linkage (called a *pushrod*). As the servo rotates, the pushrod draws back and forth, like that in Figure 24-10.

You can use the exact same hardware designed for model cars and airplanes with your servo-equipped robots. Visit the neighborhood hobby store and scout for possible parts you can use. Look for pushrods and clevis ends.

CONTROLLING LINEAR MOVEMENT

So you can see how pushrods allow you to convert the rotation of a servo to linear movement. There are two key ways of controlling the amount of linear movement you get:

- Use a larger or smaller horn on the servo. The larger the horn, the larger the movement of the pushrod.
- Use a specific pivot point for the pushrod, a mechanical constriction like an eyelet, channel, or hole that limits the movement of the pushrod to just back-and-forth motion. As shown in Figure 24-11, the closer the pivot is to the servo, the wider the movement pattern; conversely, the farther away from the servo, the narrower the pattern.

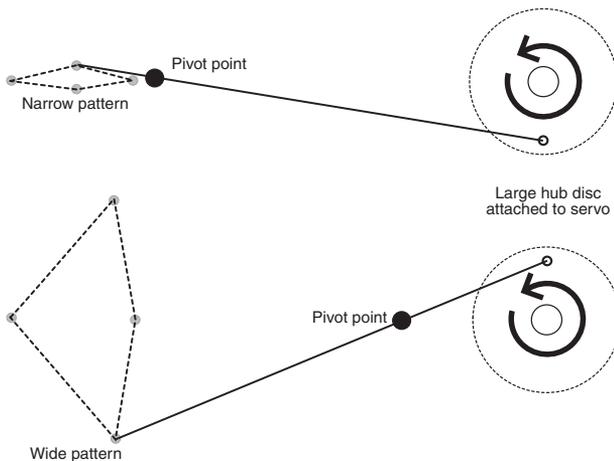


Figure 24-11 Control the angular displacement of the pushrod by changing the location of the pivot point, which is simply a mechanical conduit (small tube, hole, plastic grommet) that restricts side-to-side motion. Avoid placing the pivot point too close to either end of the pushrod.



There will always be some side-to-side motion in the linear movement. For this reason, it's a good idea to design a bit of "slop" in the pushrod system, such as a loose clevis end on either end of the pushrod.

ADDING LUBRICATION

Parts that slide against each other should be kept lubricated, to prevent them from binding up. Use a thick grease, not oil. Grease for hobby R/C is available at any store that carries these parts. It comes in a small tube; you want the white or clear stuff. My favorite is white lithium grease.

Drivetrain Components for Robotics

Pushrod mechanical links are one form of drivetrain—get the power of a motor from one place to another. There are hundreds of drivetrain components you might use in your robot creations, but the following table summarizes the most commonly used. These drivetrain parts go between the motor and the wheel or other driven element, such as legs, tracks, or arm segments.



Gears

Gears are a principal component of drivechains and are primarily used in robotics to reduce the speed, and increase the torque, of the wheel drive motors. They are also used to share the power between several wheels or other components.

Because of the mechanical precision required to properly mesh gears, most amateur robot builders do not construct their own gear assemblies. More about gears later in the chapter.



Timing Belts

Timing belts are also called *synchronization* or *toothed* belts, and they can be used to make tracked robots, as well as substitute for more complex gear systems.

Widths range from 1/8" to 5/8", and lengths from just a few inches to several feet in diameter. Material is usually rubber. Belts are rated by the pitch between "nubs" or "cogs," which are located on the inside of the belt. The cogs interface with matching rollers.



Endless Round Belts

Endless round belts are used to transfer low-torque motion. The belt looks like an overgrown O-ring and, in fact, is often manufactured in the same manner.

Grooved pulleys are used with round belts. The diameter of the pulleys can be altered to change torque and speed. Like timing belts, you can use round belts as the tire material on wheels and for unusual treads in a tracked robot.



Roller Chain

Roller chain is exactly the same kind used on bicycles, only in most robotics applications it's smaller.

Roller chain is available in miniature sizes, down to 0.1227" pitch (distance from link to link). More common is the #25 roller chain, which has a 0.250" pitch. For reference, most bicycle chain is #50, or 0.50" pitch. The chain engages a sprocket that has teeth of the same pitch.



Idler Wheels

Idler wheels (also called idler pulleys or idlers) take up slack in belt- and chain-driven mechanisms. The idler is placed along the length of the belt or chain and is positioned so that any slack is pulled away from the belt or chain loop. Not only does this allow more latitude in design, it also quiets the mechanism.



Couplers

Couplers come in two basic styles: rigid and flexible. They are used to directly connect two shafts together, so you don't need a gear or belt to transfer the power. Couplers are common when connecting wheels and motors that aren't otherwise designed for one another.



Bearings

Bearings are used to reduce the friction of a spinning component, such as a wheel or idler, around a shaft. There are lots of types of bearings, but ball bearings are the most common. The ball bearing is composed of two concentric rings; between each ring is a row of metal balls. The rings—and the ball bearings—are held in place by a flange.



Bushings

Bushings and bearings serve the same general purpose, except a bushing has no moving parts. (Note: Some people call these *dry bearings*.) The bushing is made of metal or plastic and is engineered to be self-lubricating. Bushings are used instead of bearings to reduce cost, size, and weight, and are adequate when friction between the moving parts can be kept relatively low.

Using Rigid and Flexible Couplers

Couplers are used to connect two drive shafts together end to end. A common application is to use a coupler to connect the drive shaft of a motor with the axle of a wheel. Couplers can be rigid or flexible.

Rigid couplers are best used when the torque of the motor is low, as it would be in a small tabletop robot. Conversely, flexible couplers are advised for higher-torque applications, as they are more forgiving of errors in alignment. Why is this? A rigid coupler may shear off or damage the motors or shafts when misaligned.

PURCHASING READY-MADE COUPLERS

There are many types of commercially available rigid and flexible couplers, and cost varies from a few dollars to well over \$50, depending on materials and sizes. Common flexible couplers include helical, universal joint (similar to the U-joint in the driveshafts in older cars), and three-piece jaw.

The couplers attach to the shafts either with a press fit, by a clamping action, by setscrews, or by keyway. Press fit and clamp are common on smaller couplers for low-torque applications; setscrews and keyways are used on larger couplers.

Three-piece jaw couplers (Figure 24-12; see-through and partially disassembled), like those made by Lovejoy, consist of two metal or plastic pieces that fit over the shafts. These

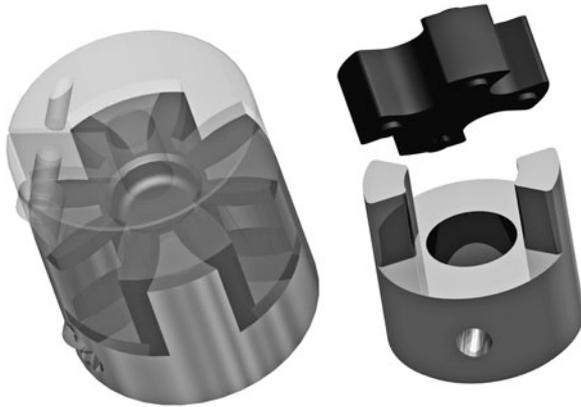


Figure 24-12 A three-piece *jaw coupler* uses two metal or plastic “jaws” that connect to the driveshafts and a softer (often rubber) inner piece that provides the flexibility.

are the “jaws.” A third, rubberized piece, the spider, fits between the jaws and acts as a flexible cushion.

One advantage of three-piece couplers is that because each piece of the jaw is sold separately, you can readily “mix and match” shaft sizes. For example, you can purchase one jaw for a 1/4” shaft and another for a 3/8” shaft. Both jaws must have the same outside diameter.

MAKING YOUR OWN RIGID COUPLERS

To save money, you can make your own rigid couplers using metal tubing, metal or plastic standoffs, or threaded couplers.

Couplers from Tubing

You can get brass, steel, and aluminum tubing of various diameters at hobby stores; larger aluminum tubing can be found at home improvement outlets. Most tubing is sold by its inner diameter—1/8” tubing measures an eighth of an inch inside, for example. The thickness of the tubing determines its outside diameter. Get tubing that fits over the motor or wheel shaft you’re attaching to.



When matching up tubing, note its I.D. (inside diameter). Tubing at the hobby store is meant to fit into the next size larger, as in a folding telescope. You can use this feature to match one shaft diameter to another. Typical thickness of the tubing is between 0.014” and 0.049”, though this varies somewhat by brand.

For lightweight robots with small motors, look at 1/8” or 3/16” I.D. If possible, bring your motor or wheel (or just the shaft) to the store with you so you can test the pieces for proper fit.

To use, cut the tubing to length. Use a tubing cutter instead of a saw. You can then secure the tubing to the shaft in several different ways, including the following:

- Crimp it on with an appropriately sized metal collar. The collar comes with a setscrew. Carefully tighten the setscrew over the tubing (see Figure 24-13). This type of metal collar is available at any R/C hobby store and often goes by the name Dura Collar. If you can’t find a collar just the right size for the tubing, you may need to select the next size smaller, then drill it out. The collar is typically plated brass, so it’s not as hard to drill as it may look.

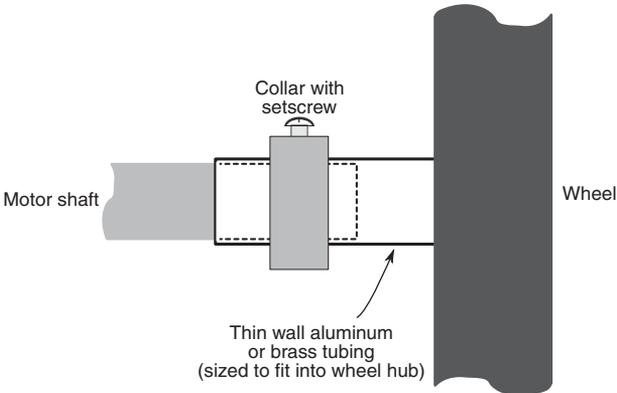


Figure 24-13 Thin tubing doesn't have enough "bite" for a setscrew. But you can use a metal collar around the tubing to provide compression against a solid motor shaft. Tighten the setscrew in the collar for a solid fit.



Figure 24-14 Metal collars may be used on wheels with an outsized hub (the hub protrudes from the wheel, giving more surface area to the shaft). Place the collar around the hub of the wheel. The compression tightens the hub around the motor shaft.

- Use large wire crimper pliers to squeeze the tubing around the shaft. This works only for thin-walled tubing and when working with smaller motors that don't develop a lot of torque.

When using the crimp-on metal collar trick, you can help cinch up the fit by drilling a small ($1/16$ " hole in the tubing where the setscrew for the collar will go. This works well when using thicker-walled tubing. When assembling the collar over the tubing, carefully line up the hole with the setscrew, then tighten the screw.



Don't forget that some wheels have outsized hubs, like the one in Figure 24-14. You may not even need to use tubing to make a coupler. Just attach the collar around the existing hub of the wheel, and tighten the set screw. With plastic wheels the hub will deform, cinching around the shaft for a fairly tight fit.

Couplings from Standoffs and Threaded Couplers

Commercially available rigid couplers often use setscrews to hold the coupler to the shaft. Thin-wall tubing is too thin for a setscrew. You need a thicker-walled coupler.

There are two ready-made sources of short metal and plastic begging to be turned into shaft couplers: standoffs and something called threaded couplers.

Standoffs are commonly used in electronics projects to keep two circuit boards or other pieces separate from one another. I regularly use metal and plastic standoffs to add additional "decks" to my robots. The decks are like tiers in a wedding cake. The standoffs are typically 1" or longer and are made of nylon, steel, or aluminum. I like working with aluminum over steel, as aluminum is easier to drill and cut. You can get either the threaded or the nonthreaded kind, but the unthreaded kind is easier to work with.

Threaded couplers are just like standoffs but are almost always made of heavier steel. You get them at the local hardware or home improvement store. They come in thread sizes from 6-32 to 5/16" and larger. The 6-32 thread size is about right for 1/8" shafts.

To make shaft couplers, you should have a vise and a drill press in order to make accurate holes. You also need a tap set with drill bits. The bits are already correctly sized for the taps.



Nylon can't hold a thread as well as aluminum or steel. If you go with a nylon standoff, use it only with shafts that have a flattened side to them. Round shafts require more pressure to keep them from turning, but this added pressure of the setscrew against the nylon just strips out the threads.

1. Secure the standoff or coupler into the vise. Make it nice and snug.
2. Use a center punch to mark where you will be drilling. You want two holes, no closer than about 1/4" from the ends. The punch makes a mark in the plastic or metal that serves as a pilot for keeping the bit on track.
3. For metal standoffs and couplers, apply some light machine or cutting oil on the mark. Slowly drill the holes for the setscrews all the way through one side of the standoff or coupler. Don't drill through to the other side unless you want to put setscrews on that side as well.
4. Turn the standoff or coupler on end and tighten in the vise. Change the drill bit to the size of the shaft you're using. For example, if the shaft is 1/8" in diameter, use a 1/8" drill bit.
5. Drill halfway down the length of the standoff or coupler, then turn it upside down. Drill through the second half.
6. Now to tap the holes for the setscrews. Change the bit (if needed) to the correct size for the setscrew you wish to use. See Appendix C, "Mechanical Reference," for a handy list of drill bits and tap sizes for making different kinds of holes.
7. Put a drop or two of machine or cutting oil into the holes you've drilled (metal standoff or coupler only), then carefully tap them to make threads. You can skip the cutting oil when tapping into plastic.



I'm not going to give a lot of detailed explanation about how to use taps to thread a hole, as this information is everywhere on the Internet. Do a quick Web search for phrases like *how to drill and tap*.

MAKING YOUR OWN FLEXIBLE COUPLERS

For many desktop robotics jobs, flexible plastic rubber tubing will work just dandy as a coupler. Select the rubber tubing so that it is just slightly smaller than the motor shaft and wheel axle you are using, and press it on for a good fit. If the motors develop too much torque for the press-fit to hold, use small worm-gear clamps to hold the tubing in place. You can get these worm-gear clamps at any hardware store.

See Figure 24-15 for one example. The wheel is held in place with a *pillow block*, which is merely a piece of wood, plastic, or metal with a hole in it that's fastened to the base of the bot. Use a long machine screw as the wheel axle; hold the wheel-axle combo in the pillow block with a pair of nylon insert locking nuts, as shown. Plastic tubing is available at hardware and home improvement outlets, as well as many hobby stores and pet stores that sell fish aquarium supplies.

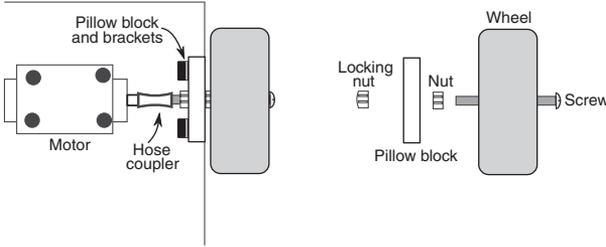


Figure 24-15 Flexible rubber tubing or hose may be used to connect a motor shaft to a wheel with its own axle. Shown here is an axle made with a machine screw, but any type of axle compatible with the inner diameter of the tubing will work. Use hose clamps around the tubing if the fit is too large.



Try to get tubing the same size or slightly smaller than the shaft diameter. Prior to fitting, you can put the tubing into hot water to soften and expand it. With the tubing still warm, slip it over the shaft. Wait for the tubing to cool, then do the twist test to see if it'll work as a shaft coupling.

Tubing is sized in different ways—it is sometimes sold by its inside diameter (I.D.) and sometimes by the outside diameter (O.D.). Bring your parts into the store for a dry fit. Tubing sold by the foot is the most economical, as you can buy just short lengths at a time. You don't need much.

Working with Different Shaft Types

Motor shafts come in several shapes and forms. A few of the more common ones are shown in Figure 24-16. Most motors use a simple round shaft; most secure to a gear or wheel hub using a tight friction fit. A flatted or “D” shaft is best when using a setscrew, as the tip of the screw can settle into the flat depression. Flatted shafts may also be used for friction fit. The “D” helps prevent the shaft from spinning inside the wheel or gear hub.

Some motors have threaded shafts. For example, several motors in the Tamiya educational motor lineup have a short male-threaded shaft end. Using locking nuts you can secure wheels and other components onto the end. R/C servo motors use a female-threaded shaft to secure a servo horn or other accessory to the motor. On servo motors, the shaft is also splined to help prevent slippage.

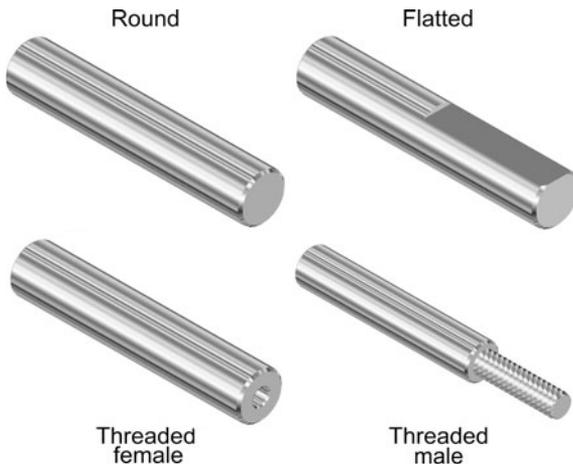


Figure 24-16 Common shaft types you will encounter: round, flatted, and threaded (both male and female). The flatted type is also called a D-shaft, because it resembles the letter D.

Other shaft types you may encounter include the hex and square, so called because of their hexagonal or square shape. They're used with wheels, gears, and other parts that have matching-shaped hubs.

Everything You Always Wanted to Know about Gears

We've already discussed the fact that the normal running speed of motors is far too fast for most robotics applications. Locomotion systems need motors with running speeds of 75 to 200 RPM. Any faster than this, and the robot will skim too quickly across the floor. Arms, gripper mechanisms, and most other mechanical subsystems need even slower motors. The motor for positioning the shoulder joint of an arm needs to have a speed of less than 20 RPM; 5 to 10 RPM is even better.

There are two general ways to decrease motor speed significantly: build a bigger motor (impractical) or add gear reduction. Gear reduction is used in your car, on your bicycle, in the washing machine and dryer, and in countless other motor-operated mechanisms.

GEARS 101

Gears have two main applications:

- To transfer power or motion from one mechanism to another.
- To reduce or increase the speed of the motion between two linked mechanisms.

The simplest gear systems use just two gears: a drive gear, and a driven (or output) gear. More sophisticated gear systems, referred to as *gear trains*, *gearboxes*, or *transmissions*, may contain several or even dozens of gears. Motors with attached gearboxes are said to be *gearbox motors*.



We use these gearbox motors a lot in robotics. R/C servos have their own gearbox built in, and most of the DC motors we use to power wheels and tracks of bots have a gearbox of some type. Though these already have a gearbox for speed reduction, there are many applications for adding external gears, such as making one motor drive two wheels at the same time.

GEARS ARE LEVERS IN THE ROUND

In a way, gears are round levers, and it may help to explain how gears function by first examining the basic mechanical lever.

Here goes: Place a lever on a fulcrum so that the majority of the lever is to one side. Push up on the long end, and the short side moves in proportion. Although you may move the lever several feet, the short end is moved only a few inches. But note that the force available on the short end is proportionately larger than the force applied on the long end.

Now back to gears. Attach a small gear to a large gear, as shown in Figure 24-17. The small gear is directly driven by a motor. For each revolution of the small gear, the large gear turns one half a revolution. Expressed another way, if the motor and small gear turn at 1000 RPM, the large gear turns at 500 RPM. The gear ratio is said to be 2:1.

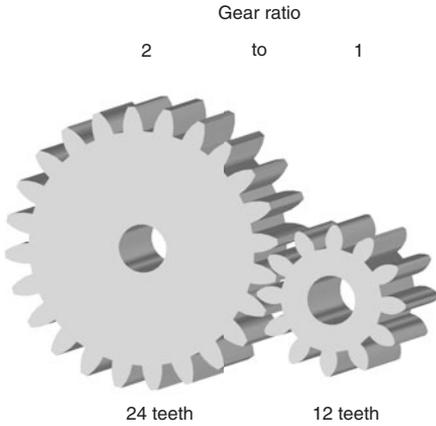


Figure 24-17 Two meshing gears, showing a 2:1 gear ratio. The gear on the left has twice the number of teeth as the gear on the right.

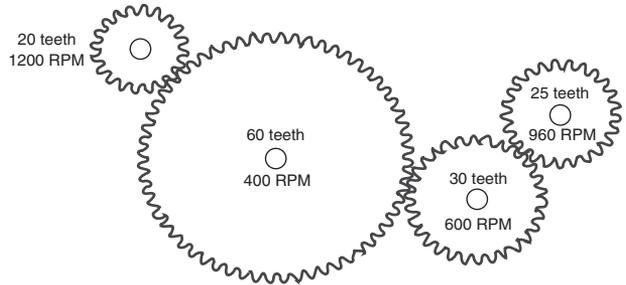


Figure 24-18 The number of gear teeth directly relates to speed.

As with a lever, another important thing happens: *decreasing the speed of the motor also increases its torque*. The power output is approximately twice the input. Some power is lost in the reduction process due to the friction of the gears. If the drive and driven gears are the same size, the rotation speed is neither increased nor decreased, and the torque is not affected (apart from small frictional losses). You can use same-size gears in robotics design to transfer motive power from one shaft to another.

ESTABLISHING GEAR REDUCTION

Gears are an old invention, going back to ancient Greece in about the third century BC. Today’s gears are much refined, though they’re still based on the old designs in which teeth from the two mating gears mesh with each other. Force is transferred from one gear to another.

Gears with the same size teeth are usually characterized not just by their physical size but also by the number of teeth around their circumference. In the example in Figure 24-17, the small gear contains 12 teeth, the large gear 24 teeth. And you can string together a number of gears one after the other, all with varying numbers of teeth (see Figure 24-18). Attach a tachometer to the hub of each gear, and you can measure its speed. You’ll discover the following two facts:

- The speed always decreases when going from a small to a large gear.
- The speed always increases when going from a large to a small gear.

There are plenty of times when you need to reduce the speed of a motor from 5000 RPM to 50 RPM. That kind of speed reduction requires a reduction ratio of 100:1. To accomplish that with just two gears you would need, as an example, a drive gear that has 10 teeth and a driven gear that has 1000 teeth—quite impractical.

Instead, you reduce the speed of a motor using multiple gears, as in Figure 24-19. Here, the drive gear turns a larger “hub” gear, which in turn has a smaller gear permanently attached

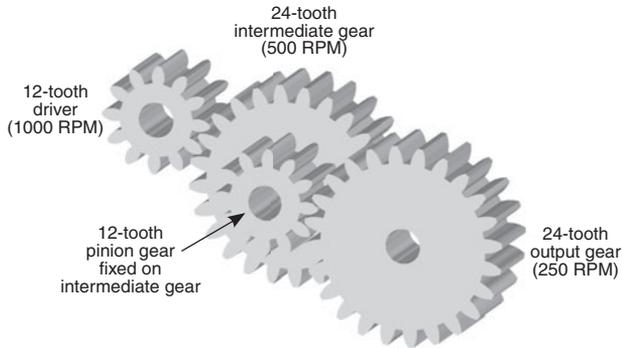


Figure 24-19 True speed reduction is accomplished by ganging gears together. The idea is to always go from fewer teeth to more teeth; with each step, the speed is reduced (and torque is increased).

to its shaft. The small hub gear turns the driven gear to produce the final output speed, in this case 250 RPM. You can repeat this process over and over again until the output speed is but a tiny fraction of the input speed.



There are many other ways to achieve large-ratio gear reductions, and this is just one of them. Other techniques include worm, planetary, and hypoid gears; you might find gearbox motors that use these techniques, but they tend to be more expensive.

VARIATIONS IN GEAR TEETH

All of the examples shown so far have been of *spur gears*, which are the most common. They're used when the drive and driven shafts are parallel. *Bevel* gears have teeth on the surface of the circle rather than the edge. They are used to transmit power to perpendicular shafts. *Miter* gears serve a similar function but are designed so that no reduction takes place.

Spur, bevel, and miter gears are reversible—the gear train can be turned from either the drive or the driven end. Conversely, *worm* and *leadscrew* gears transmit power perpendicularly and are not usually reversible. The lead screw resembles a threaded rod.

Rack gears are like spur gears unrolled into a flat rod. They are primarily intended to transmit rotational motion to linear motion.

COMMON GEAR SPECIFICATIONS

Here are some common gear specifications to keep you warm at night.

Pitch: The size of gear teeth is expressed as pitch, which is roughly calculated by counting the number of teeth on the gear and dividing it by the diameter of the gear. Common pitches are 12 (large), 24, 32, 48, and 64. Odd-size pitches exist, of course, as do metric sizes.

Pressure angle: The degree of slope of the face of each tooth is called the pressure angle. A common pressure angle is 20° , although some gears, particularly high-quality worms and racks, have a $14\text{-}1/2^\circ$ pressure angle.

Tooth geometry: The orientation of the teeth on the gear can differ. The teeth on most spur gears is perpendicular to the edges of the gear. But the teeth can also be angled, in which case it is called a helical gear. There are a number of other unusual tooth geometries in use, including double-teeth and herringbone.

USING MOTORS WITH GEAR REDUCTION

It's always easiest to use motors that already have a gear reduction box built onto them. When selecting gear motors, you'll be most interested in the output speed of the gearbox, not the actual running speed of the DC motor itself. Note also that the running and stall torque of the motor will be greatly increased at the output of the gearbox.

When using motors without built-in gear reduction, you'll need to add reduction boxes or make your own. Although it is possible to do this yourself, there are many pitfalls:

- Shaft diameters of motors and ready-made gearboxes may differ, so you must be sure that the motor and gearbox mate.
- Separate gear reduction boxes are hard to find. Most must be cannibalized from salvage motors. Old AC motors are one source of surplus boxes.
- Machining the gearbox requires precision, since even a small error can cause the gears to mesh improperly.

WHERE TO FIND GEARS

Gears can be expensive, especially the metal ones that are machined from a piece of solid metal. Online sources like Boston Gear, Small Parts, W.M. Berg, and Stock Drive offer these and most any other gear imaginable, but at costs that make the average robot builder blanch.

As long as your requirements aren't too unusual, you may be able to locate the gears you want from other products and sources.

- *Hobby and specialty retailers.* Next time you're at a hobby store, look for replacement gear sets for servos and drive motors for R/C cars and airplanes. Some are plastic; others are metal (usually either aluminum or brass).
- *Toy construction sets.* Don't laugh! Toys like LEGO, Erector, and Inventa come with gears you can use in your robotics projects. Most are on the large side and are made of plastic.
- *Surplus catalogs.* New gears can be expensive; surplus gears can be quite affordable. You can often find new gears, plastic or metal, for about 10 cents on the dollar, compared to the cost of the same gear new. The only problem: Selection can be limited, and it can be hard to match gear sizes and pitches even when buying gears from the same outlet.
- *Rechargeable electric screwdrivers.* Inside are numerous gears, typically in a "planetary" configuration, used to produce their very high speed reductions. Before raiding the screwdriver for just the gears, consider using the motor, too.
- *Hacked toys* Discarded and discounted toys make for good gear sources. These include friction- and battery-powered toy cars, "dozer" toys, even some action figures. Tear the toy apart for the treasure inside. These gears tend to be small and made of plastic.
- *Old kitchen appliances.* Go to resale stores and garage sales and look for old food mixers, electric knives, even electric can openers. Unlike toys, kitchen appliances commonly use metal gears—or at the least, very strong plastic gears.

Robot Movement with Shape Memory Alloy

A metal with a memory? You bet. As early as 1938, scientists observed that certain metal alloys, once bent into odd shapes, returned to their original form when heated. This property was considered little more than a laboratory curiosity, but research into metals with memory took off in 1961, when William Beuhler and his team of researchers at the U.S. Naval Ordnance Laboratory developed a titanium-nickel alloy that repeatedly displayed the memory effect. Beuhler and his cohorts developed the first commercially viable shape memory alloy, or SMA. They called the stuff Nitinol, a fancy-sounding name derived from *Nickel Titanium Naval Ordnance Laboratory*.

Shape Memory Alloy Comes to Robotics

In 1985, a Japanese company, Toki Corp., unveiled a type of shape memory alloy specially designed to be activated by electrical current and made this material available in small quantities to business and hobbyists. The availability of short and inexpensive lengths of shape memory alloy for experimental use greatly enhanced the spread of interest in the material.

Toki's shape memory alloy, trade-named BioMetal, offered all of the versatility of the original Nitinol, with the added benefit of near instant electrical actuation.

BioMetal and materials similar to it—Muscle Wire from Mondo-Tronics or Flexinol from Dynalloy—have many uses in robotics, including novel locomotive actuation. From here on out we'll refer to this family of materials generically as shape memory alloy, or simply SMA.

Basics of Shape Memory Alloy

SMA is basically a strand of metal wire made with nickel and titanium. You know that titanium is a kind of “super-space-age” metal, about as strong as steel, yet some 50 percent lighter in

weight. Though the material may be very thin (a typical thickness is 0.006" (six mil)—slightly wider than a strand of human hair), it's exceptionally strong.

In fact, the tensile strength of SMA rivals that of stainless steel: the breaking point of just one slender wire is a whopping 6 pounds. Even under this much weight, SMA stretches little. In addition to its strength, SMA also shares the corrosion resistance of stainless steel.

Shape memory alloys change their internal crystal structure when exposed to certain higher-than-normal temperatures, and this includes the induced temperatures caused by passing an electrical current through the wire. The structure changes again when the alloy is allowed to cool.

More specifically, during manufacture the SMA wire is heated to a very high temperature, which embosses or "memorizes" a certain crystal structure. The wire is then cooled and stretched to its practical limits. When the wire is reheated, it contracts because it is returning to the memorized state. That's why SMA is often referred to as "memory wire."

Shape memory alloys have an electrical resistance of about 1 ohm per inch. That's far more than ordinary hookup wire, so SMAs will heat up more rapidly when an electrical current is passed through them. The more current that passes through, the hotter the wire becomes and the more contracted the strand.

Under normal conditions, a 2" to 3" length of SMA is actuated with a current of about 450 milliamps. That creates an internally generated temperature of about 100 to 130°C; 90°C is required to achieve the shape memory change. Most SMAs can be manufactured to change shape at most any temperature, but 90°C (194°F) is a standard value for off-the-shelf material.

Excessive current should be avoided. Why? Extra current causes the wire to overheat, which can greatly degrade its shape memory characteristics. For best results, current should be as low as necessary to achieve the contraction desired. SMAs will contract by 2 to 4 percent of their length, depending on the amount of current applied. The maximum contraction of typical SMA material is 8 percent, but that requires heavy current that can, over a period of just a few seconds, damage the wire.

Using Shape Memory Alloy

Shape memory alloys need little support paraphernalia. Besides the wire itself, you need some type of terminating system (the hardest part!), a bias force, and an actuating circuit. We'll discuss each of these in the following sections.

TERMINATING SYSTEM

Because SMA can't be readily soldered to anything, the ends have to be mechanically terminated—not the "Are you Sarah Conner?" type of terminating, but the kind involving crimp-on lugs and other kinds of solderless connectors. Because SMAs physically contract, using glue or another adhesive will not secure the wire to the mechanism. These and other crimp terminators are available from companies that sell shape memory alloy wire.

You can make your own crimp-on connectors using 18-gauge or smaller solderless crimp connectors—the smaller, the better. Although these connectors are rather large for the typical thin 0.006" SMA, you can achieve a fairly secure termination by folding the wire in the connector and pressing firmly with a suitable crimp tool, as shown in Figure 25-1. Be sure to completely flatten the connector. If necessary, place the connector in a vise to squish it all the way shut.

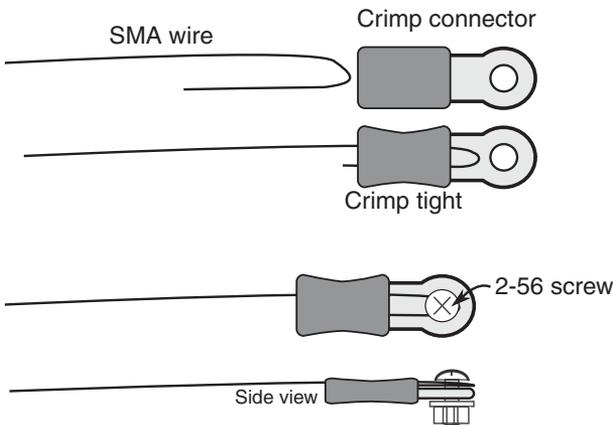


Figure 25-1 Use small crimp-on connectors to terminate the ends of the shape memory alloy wire. Small tools like those for making jewelry are a help.

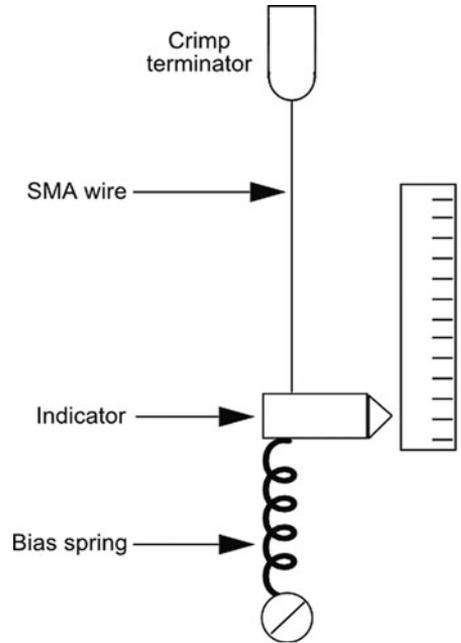


Figure 25-2 A bias spring or weight is required as a counterbalance on the SMA wire.

Figure 25-1 also shows a variation using miniature 2-56 size machine screws (available at hobby stores). You still use the crimp-on connector, but you carefully loop the wire around the machine screw.



Experience in making personal jewelry comes in handy when working with shape memory alloy. Many of the tools, techniques, and findings used to make things like necklaces and earrings can be used with SMA, too. *Findings* are the little doodads used along with the jewel in jewelry—metal chain, clasps, pins, and, of course, crimp-on connectors. Locate these tools and parts in well-stocked hobby stores and online at places such as Fire Mountain Gems.

BIAS FORCE

Apply current to the ends of an SMA wire, and it just contracts in air. To be useful,

- The wire must be attached to one end of the moving mechanism.
- It must be biased at the other end, like that in Figure 25-2. The bias can be a spring or even an unmovable part.

Besides providing physical support, the bias offers the counteracting force that returns the SMA wire to its limber condition once current is removed from the strand. You can also use a second SMA wire that contracts in the other direction and restores the state of the first wire, but this involves some very detailed mechanical construction.

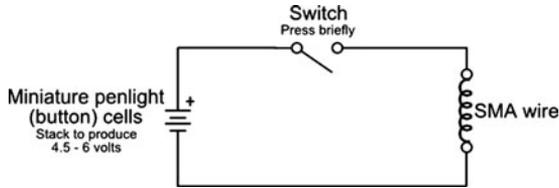


Figure 25-3 A simple switch in series with a battery forms a basic SMA driving circuit. The low current provided by the penlight cells delivers reduced current to the SMA wire, but don't leave the switch closed for more than a second.

ACTUATING CIRCUIT

Basic SMA wire can be actuated with three or four 1.5-volt button cell (SR44) batteries (the cheap ones). Because the circuit through the SMA wire is almost a dead short, the batteries deliver close to their maximum current capacity. The average 1.5-volt button cell (SR44) battery has a maximum current output of only a few hundred milliamps, so the current is limited through the wire. You can connect a simple on/off switch in line with the battery, as detailed in Figure 25-3, to contract or relax the SMA wire. Press the button only briefly . . . just long enough for the wire to instantaneously contract.



The problem with this setup is that it wastes battery power, and if the power switch is left on for too long it can damage the SMA strand. A more sophisticated approach uses an LM555 timer IC that automatically shuts off the current after a short time. To save space, this bonus project is found on the RBB Online Support site. See Appendix A for details.

Operating SMA Using a Microcontroller

A microcontroller is the perfect tool for controlling shape memory alloy. With a simple program you can accurately control the amount of time an SMA wire is actuated. In contrast to using something like the LM555 timer, where you need to change component values to alter the amount of time the SMA is charged, all you need to do with a microcontroller is change a line or two of code.

A microcontroller also allows you to fully automate a sequence of SMA actuations, which you could use to create such things as a miniature caterpillar or some other clever arrangement of SMA wire.

You cannot connect the SMA wire directly to the microcontroller, however. You need to boost the current fed to the wire using a transistor or other driver. Figure 25-4 shows how to use a TIP120 Darlington power transistor, as well as a driver from an inexpensive and easy-to-use ULN2003 IC. This chip is composed of seven (count 'em, *seven!*) Darlington transistors, each capable of handling about half an amp.

With either approach, you can provide a higher voltage to the wire than the 5 volts shown; the increased voltage provides stronger and faster pulls. Experiment with different voltages (up a practical maximum of 9 to 12 volts), being careful to prevent excessive current through the wire. The higher the voltage, the easier it is to burn out your SMA strand. With higher voltages you might consider adding a 10 to 15 Ω (2-watt or higher) resistor between the V+ power supply and the SMA wire connection.

The sketch *sma.pde* is an example Arduino program that pulses the SMA wire (via a resistor and transistor, a ULN2003 IC interface) at 5-second intervals. Each pulse is limited to 250 milliseconds (a quarter of a second). You can also use the PWM feature of the Arduino to

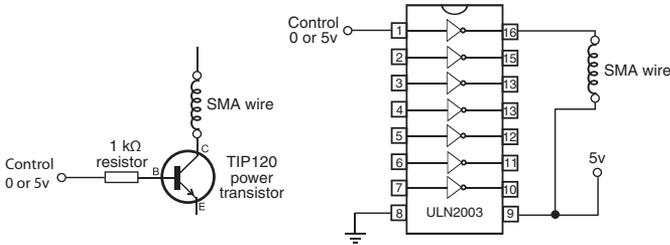


Figure 25-4 Two ways to activate SMA wire via a microcontroller: using a power transistor (in this case, a TIP120 Darlington) and the trusty ULN2003 integrated circuit. Shown here is an SMA wire connected to one of the seven drivers of the ULN2003.

vary the duty cycle (on-versus-off) of the short pulse applied to the wire. Experiment with the time delays and PWM duty cycles. The example program uses a 3/4-second ON time.

```
101010
010101
101010
010101
```

sma.pde

You can also find this code at the RBB Online Support site. See Appendix A, “RBB Online Support.”

```
int smaPin = 9; // Connect SMA driver circuit to
                // pin D9
int ledPin = 13; // Use integrated LED
void setup()
{
  pinMode(smaPin, OUTPUT); // Set pins as outputs
  pinMode(ledPin, OUTPUT);
}
void loop()
{
  digitalWrite(smaPin, HIGH); // Activate wire
  digitalWrite(ledPin, HIGH); // Turn LED on
  delay(250); // Keep on 1/4 second
  digitalWrite(smaPin, LOW); // Deactivate wire
  digitalWrite(ledPin, LOW); // Turn LED off
  delay(5000); // Relax/cool for 5 seconds
  analogWrite(smaPin, 128); // Activate wire at 1/2 duty
  digitalWrite(ledPin, HIGH); // Turn LED on
  delay(750); // Keep on doer 1/2 second
  digitalWrite(smaPin, LOW); // Deactivate wire
  digitalWrite(ledPin, LOW); // Turn LED off
  delay(5000); // Relax/cool for 5 seconds
}
```



Be sure to power the driver circuit for the SMA wire from its own battery or power supply. Do not use the 5V pin on the Arduino to drive the wire, as the instantaneous current draw from the wire could overwhelm the Arduino’s voltage regulator. As usual, be certain to connect the grounds of the Arduino and your separate battery/power supply.

Experimenting with SMA Mechanisms

With the SMA properly terminated and actuated, it’s up to you and your own imagination to think of ways to use it in your robots. Figure 25-5 shows a typical application using an SMA wire in a pulley configuration. Apply current to the wire and the pulley turns, giving you rota-

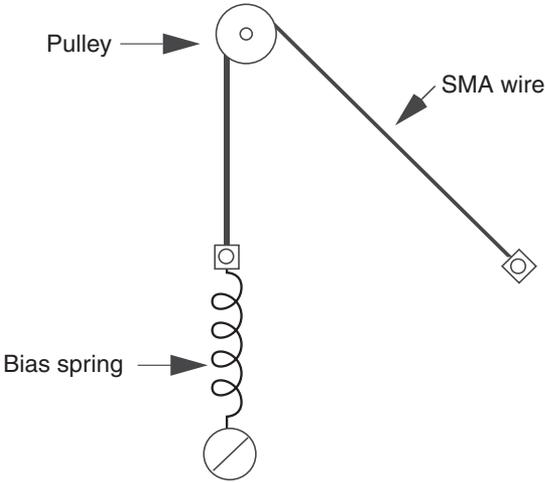


Figure 25-5 Concept of using an SMA wire with a mechanical pulley.

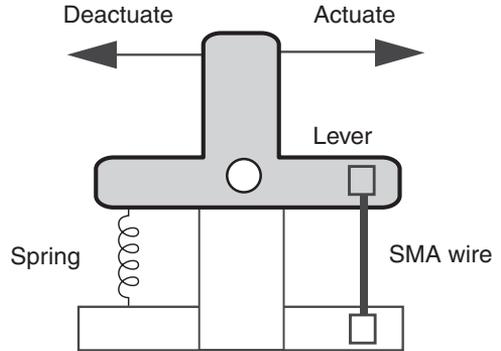


Figure 25-6 The bell crank changes the contraction of the SMA wire for sideways movement. The spring allows the crank to return to its original position after current is removed from the SMA wire.

tional motion. A large-diameter pulley will turn very little when the SMA tenses up, but a small-diameter one will turn an appreciable distance.

Then in Figure 25-6 there's a length of SMA wire used in a lever arrangement. Here, the metal strand is attached to one end of a bell crank. On the opposite end is an ordinary expansion spring, something you can get at most any hardware store. Applying juice to the wire causes the bell crank to move. The spot where you attach the drive arm dictates the amount of movement you will obtain when the SMA contracts.

SMA wire is tiny stuff, and you will find that the miniature hardware designed for model R/C airplanes is most useful for constructing mechanisms. Most any well-stocked hobby store will provide a full variety of bell cranks, levers, pulleys, wheels, gears, springs, and other odds and ends to make your work with SMA more enjoyable.

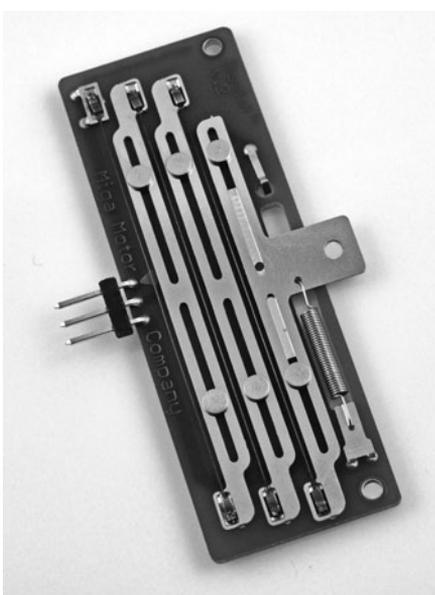


Figure 25-7 Commercially made SMA linear motor. (Photo courtesy Miga Motor Company.)

Using Ready-Made SMA Mechanisms

Several companies have taken up the shape memory alloy mantra and offer unusual and handy ready-made SMA mechanisms. Price is reasonable, especially considering that the products are ready for instant use; you don't need to crimp on any connectors or attach any bias springs. One such company is Miga Motors (www.migamotors.com)

.com), which sells their wares through their own site plus a network of worldwide distributors, such as RobotShop and SparkFun.

Figure 25-7 shows the Miga Motors Dash, an SMA linear actuation that provides an astonishing 1.75 pounds of output force and a 1/4" stroke. It's intended for such applications as electronic door locks, but it has obvious uses in robotics as well, such as moving legs in a walking bot. Likewise, rotary SMA actuators can be used in place of servo motors in many applications. They're spring-loaded, so once current is removed, the actuator returns to its starting position.

This page intentionally left blank

Hands-On Robotic Projects

This page intentionally left blank

Build Robots with Wheels and Tracks

As you discovered in Part 2 of this book, you can construct practical and functional mobile robots using only common tools and readily available materials. You can use wood, plastic, or metal—or, for quick prototypes, heavy-duty cardboard or foamboard intended for art projects. (What’s a robot but a fancy art project?!)

This chapter extends what you’ve discovered in previous pages, offering numerous plans and design concepts for building robots that run about on wheels and tracks. The choice of construction material is up to you, and you’re free to experiment with different sizes and assembly techniques.

See Chapter 27 if you’re wanting to build a robot that uses legs to move around.

FYI

See also Chapters 20 through 23 for information on general robot design, as well as information on powering your robot with motors. And be sure to check out the free bonus projects on the RBB Online Support site, detailed in Appendix A.

Basic Design Principles of Rolling Robots

With few exceptions, bots that roll use wheels or tank treads to get from one place to another. As you read in Chapter 20, “Moving Your Robot,” wheeled robots use a number of steering techniques. The most common—for wheels or treads—is two motors on each side of the vehicle.

DRIVE MOTOR ARRANGEMENTS

The most popular mobile robot design uses two identical motors to spin two wheels on opposite sides of the base. These wheels provide forward and backward locomotion, as well as left

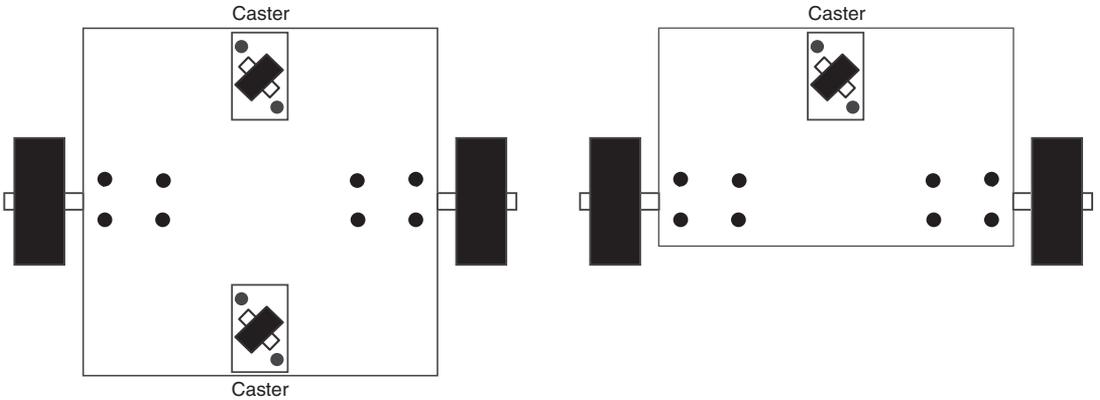


Figure 26-1 Center- versus front-drive motor mounting. With the center-drive arrangement, you typically need a caster on each end, though you can use just one if the robot is properly balanced.

and right steering. If you stop the left motor, the robot turns to the left. By reversing the motors relative to one another, the robot turns by spinning on its wheel axis (“turns in place”). You use this forward-reverse movement to make “hard” or sharp right and left turns.

Centerline Drive Motor Mount

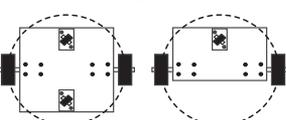
You can place the wheels—and, hence, the motors—just about anywhere along the length of the platform. If they are placed in the middle, as shown in Figure 26-1, you should add casters to either end of the platform to provide stability. Since the motors are in the center of the platform, the weight is more evenly distributed across it.

A benefit of centerline mounting is that the robot has no “front” or “back,” at least as far as the drive system is concerned. Therefore, you can create a kind of *multidirectional* robot that can move forward and backward with the same ease. Of course, this approach also complicates the sensor arrangement of your robot. Instead of having bump switches only in the front of your robot, you’ll need to add some in the back in case the robot is reversing direction when it strikes an object.

 Depending on the size of the robot and its weight distribution, you may be able to get by with just one caster. By placing slightly more weight over the caster, the bot will favor tipping to that side.

Front-Drive Motor Mount

You can also position the wheels on one end of the platform. In this case, you add one caster on the other end to provide stability and a pivot for turning, also shown in Figure 26-1. Obviously, the weight is now concentrated more on the motor side of the platform. Even out the weight distribution by putting the batteries in the center of the platform.

 One advantage of front-drive mounting is that it simplifies the construction of the robot. Its “steering circle,” the diameter of the circle in which the robot can be steered, is still the same diameter as the centerline drive robot. However, it extends beyond the front/back dimension of the robot. This may or may not be a problem, depending on the overall size of your robot and how you plan to use it.

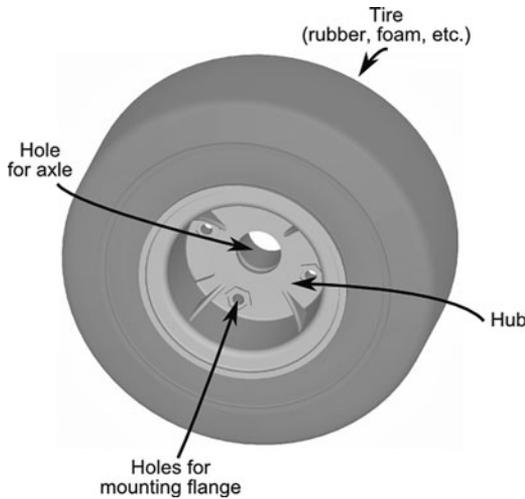


Figure 26-2 The construction of the typical wheel is a rubber (or other material) tire mounted on a hub.

PICKING THE RIGHT WHEELS

Wheels are made up of tires (or tyres, in the United Kingdom), mounted on hubs (Figure 26-2). A tire is rubber, plastic, metal, or some other material, and the hub is the portion that attaches to the shaft of the axle or motor. The hub usually has a hole for an axle, which is most often the driveshaft of a motor.

Some wheels for robots are molded into one piece. Others, such as the Dave Brown Lite-Flight wheels, are composed of separate pieces assembled at the plant. The Lite-Flight wheels use a plastic hub that attaches to the motor shaft or axle, and onto the hub is mounted a foam tire.

Wheel Materials

The first order of consideration is the materials used for the wheel. The least expensive wheels, like those used on low-cost toys, are molded in one piece, usually a hard plastic. The wheel doesn't have a separate tire and hub. While these wheels are acceptable for some robots, you probably want a softer tire surface. This requires a softer rubber or foam, over a rigid hub.

Rubber over plastic: The hardness of the rubber greatly influences traction. One common measure of hardness is the *Durometer*, tested by a device called (get this!) the durometer. There are several durometer scales, each labeled with a letter, such as A or D. A Durometer of 55A is relatively soft and pliable; 75A is medium, and 95A is quite hard.

Rubber over metal: Typical of wheels made for R/C racing, these are heavier and sturdier, and are well suited to bigger robots. You can also get small rubber tires mounted on aluminum hubs. These are typically sold at hobby stores as tail wheels for model airplanes.

Foam over plastic: Foam wheels are also a mainstay in the R/C racing field. Like their rubber counterpart, hardness varies.

Rubber/foam over spoked wheels: As the size of the wheel increases, so does its weight. Spokes are used to reduce the weight of very large wheels. Smaller bicycle or wheelchair wheels are suitable for larger robots.

Pneumatic wheels: Traditional foam and rubber tires are merely fitted over their hubs. In a pneumatic wheel, the tire is filled with air, which gives the wheel more bounce, but with added rigidity. Wheels for wheelbarrows and some wheelchairs are pneumatic.

Airless tires: Similar in concept to the pneumatic wheel, airless tires are hollow and filled not with air but with a rubber or foam compound. They are common on wheelchairs and heavy-duty materials-handling carts. They're great for larger bots that have to carry a lot of weight.

Wheel Diameter and Width

There are no standards among wheel sizes. They vary by their diameter, as well as their tread width (the tread is the plastic, foam, or rubber material that contacts the ground).

- The larger the diameter of the wheel, the faster the robot will travel for each revolution of the motor shaft. You can quickly calculate linear speed if you know the speed, in revolutions per minute or second, of the motor. Simply multiply the diameter of the wheel by π , or 3.14, then multiply that result by the speed of the motor. See the section "Using Wheel Diameter to Calculate the Speed of Robot Travel" for more details.
- The larger the diameter of the wheel, the lower the torque from the motor. Wheels follow the laws of levers, fulcrums, and gears. As the diameter of the wheel increases, the amount of torque delivered by the wheel decreases.
- Wider wheels provide a greater contact area for the wheel, and therefore traction (from friction) is increased.
- The wider the wheels, the more the robot will tend to stay on course (called *tracking*). With narrow wheels, the robot may have a tendency to favor one side or the other when there is even the slightest misalignment of the wheels. Conversely, if the wheels are too wide, the friction created by the excess wheel area contacting the ground may hinder the robot's ability to make smooth turns.

When selecting the wheel diameter and width, match the wheel to the job. A robot with modest-size wheels of fairly narrow proportions (say, 1/4" wide for a wheel of 2.5" to 3" in diameter) will be more agile than if it were equipped with much wider wheels.

Wheel Placement and Turning Circle

Where the wheels are located on the robot base affects the turning circle of the robot. Whenever possible, locate the wheels within the body of the base, rather than outside it. This decreases the effective size of the robot and allows it to turn in a tighter circle. Figure 26-3 shows wheels mounted both within the area of the base and outside it.

UNDERSTANDING WHEEL TRACTION

As in a car, wheels on your robot are meant to grip the driving surface. This provides traction and allows it to move forward. Yet, oddly enough, with robots both too little and too much grip can be a bad thing.

Picking up from Chapter 20, "Moving Your Robot," let's look at how a differentially steered robot is designed. It has two motors and wheels mounted on opposite sides. Traction going straight ahead is simple: when both motors are activated in the same direction, the robot moves forward or backward in a straight line.

Wheel traction becomes an issue in turns. There are two ways to turn a differentially

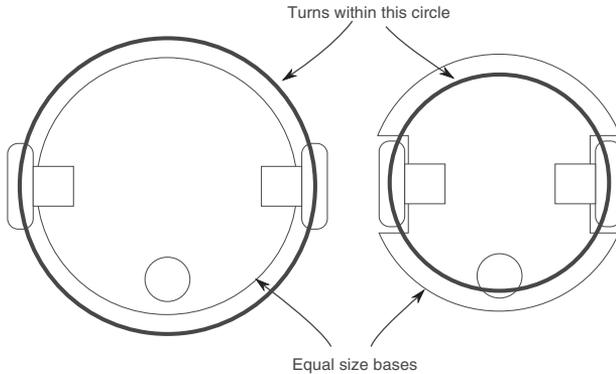


Figure 26-3 The *turning circle* describes the area occupied by a robot when it makes a spinning turn. Given the same-diameter base, wheels on the inside of the base circumference have a smaller turning circle.

steered robot: by reversing the motors relative to one another (hard turn) or by stopping one motor and activating the other (soft turn). In both kinds of turns,

- Inadequate traction causes the wheels to slip, so it's anyone's guess where the robot will be heading afterward.
- Excessive traction can cause “chatter”—the wheels grip the road surface so well, they have to bounce in order to negotiate the turn. The effect is most pronounced in soft turns and is compounded in 4- and 6-wheel designs.
- Four or more driven wheels, mounted in sets on each side, will function much like tank tracks. In tight turns, the wheels will experience significant friction and skidding. If you choose this design, position the wheel sets closer together.

Most wheels for robotics use a rubber tire material. The softness of the rubber and its surface help determine its *compliance*. A very soft and mushy tire material—like that found on some model racing cars—may cause too much traction, hindering proper steering. A very hard tire material, such as a hard plastic, may not provide enough grip.

The effectiveness of any tire material is determined by the surface it rolls over. A hard tire on a hardwood floor can be a bad combination; a moderately soft tire on Berber carpet is a much better combination.

USING WHEEL DIAMETER TO CALCULATE THE SPEED OF ROBOT TRAVEL

The speed of the drive motors is one of two elements that determine the travel speed of your robot. The other is the diameter of the wheels. For most applications, the speed of the drive motors should be under 130 RPM (under load). With wheels of average size, the resultant travel speed will be approximately 4 feet per second. That's actually pretty fast. A better travel speed is 1 to 2 feet per second (approximately 65 RPM), which requires smaller-diameter wheels, a slower motor, or both.

How do you calculate the travel speed of your robot? Follow these steps:

1. Divide the RPM speed of the motor by 60. The result is the revolutions of the motor per second (RPS). A 100 RPM motor runs at 1.66 RPS.
2. Multiply the diameter of the drive wheel by π , or approximately 3.14. This yields the circumference of the wheel. A 7" wheel has a circumference of about 21.98 inches.

3. Multiply the speed of the motor (in RPS) by the circumference of the wheel. The result is the number of linear inches covered by the wheel in 1 second.

With a 100 RPM motor and 7" wheel, the robot will travel at a top speed of 36.49" per second, or about 3 feet. That's about 2 miles per hour! You can slow down a robot by decreasing the diameter of the wheel. By reducing the wheel to 5" instead of 7", the same 100 RPM motor will propel the robot at about 25 inches per second.

Bear in mind that the actual travel speed of your robot when it's fully accessorized may be less than this. The heavier the robot, the larger the load on the motors, so the slower they will turn.

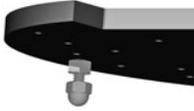
For your reference, here is handy table comparing travel speed, in inches per second, against a variety of motor RPM and several common small wheel sizes.



See the Robot Speed Calculator on the RBB Online Support site (refer to Appendix A). Enter the motor RPM and the diameter of the wheels you're using, and the calculator tells you the travel speed in inches per second.

SUPPORT CASTERS AND SKIDS FOR WHEELED ROBOTS

Differentially steered robots need something on the front and/or the back to prevent them from tipping over. There are several common approaches, listed below.



Nonrotating Skid

The purpose of a skid is to glide over the ground without using any moving parts. The skid is rounded (a cap or acorn nut works well) to facilitate a smooth ride. Polished metal, hard plastic, or Teflon are common choices. For obvious reasons, skids are not suitable for robots that may travel over uneven surfaces or when there are many obstructions, like cables and old socks.



Swivel Caster

Swivel casters are available with wheel diameters from 1" to over 4". Match the size of the caster with the size of the robot. You'll find the common 1-1/4" to 2"-diameter caster wheel is suitable for most medium robots. For larger bases you can opt for the 3" and even 4" casters. Swivel casters are commonly available with plate or stem mounting and in the following wheel styles:

- Single wheel
- Dual wheel ("twin wheel")
- Ball style

The ball style is used with furniture and tends to be heavy. If you use it at all, reserve it for heavier robots. Single-wheel casters are the most common and easiest to find. Look for a caster that swivels easily.



Ball Caster

Ball casters act as omnidirectional rollers. Unlike swivel casters, which must rotate to point in the direction of travel, ball casters are ready to move in any direction at any time. This makes them ideal for use as support casters in robots.

The size of the ball varies from pea-sized to over 3" in diameter, and they are available in steel, stainless steel, or plastic. Pololu sells a variety of small ball casters for desktop robots; industrial supply outlets such as Grainger, McMaster-Carr, and Reid Tool and Supply offer the bigger ones.



Omnidirectional Wheel

Omnidirectional wheels are basically rollers mounted on the tread of a tire. The tire turns on an axis like any other, but the rollers allow for movement in any direction. For what they do as casters, omnidirectional wheels mean extra cost, size, and weight. I've not found that they work any better than a ball transfer or even a well-made swivel caster.



Tail Wheel

One alternative to the swivel caster is the tail wheel, used on R/C model airplanes (and, therefore, available at most hobby stores). The wheels come in sizes ranging from about 3/4" to over 2" and are used with specific mounting hardware.

In summary: For a small robot, under a couple of pounds and measuring 7" in diameter or less, a nonrotating skid is usually acceptable. For centerline motor mounting, use two skids: one each in the front and rear of the bot. For larger or heavier robots, a skid may dig into soft surfaces, or it may snag on bumps, cables, and other obstructions. For these, use a swivel caster or a ball caster.

Successful Use of Casters

The casters on your robot must not impede the direction or speed of the machine's travel. Cheap swivel casters can catch and not swivel properly when the robot changes direction.

Keep these points in mind when selecting and using casters with your robots:

- Test them for smooth swivel action. Casters with ball bearings tend to give better results.
- In most cases, since the caster is provided only for support and not traction, the caster wheel should be a hard material to reduce friction.
- When using two casters on either end of the base, there's a possibility of the robot becoming trapped if the casters touch ground but the drive wheels do not. You can fix this by

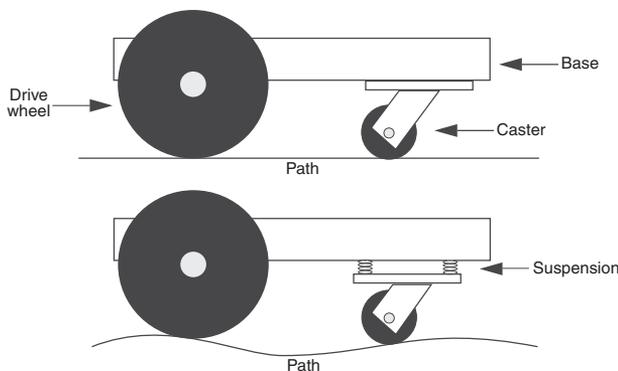


Figure 26-4 Spring-loading can overcome some of the common problems of using a caster over uneven terrain. The height of the caster adjusts to compensate for bumps.

using only one caster instead of two, or place slightly more weight over the end with the caster. You may also reduce the height of the casters, or mount the casters on a spring suspension, as shown in Figure 26-4. Select a spring that, under normal load, just begins to compress from the weight of the robot.

Two-Motor BasicBot

The BasicBot is a simple differentially steered base that's easy to construct of wood, plastic, cardboard, picture mat, or foamboard. It's an ideal first robot, and its round shape makes it well suited for use as a wall follower, maze solver, or other robot that works in confined spaces. The base measures 5" in diameter; many craft and hobby stores sell 1/8"- or 1/4"-thick wood and plastic already cut into this size (or close to it), saving you from cutting out a circle using a saw or mat-cutting blade. A finished BasicBot is shown in Figure 26-5.

The BasicBot uses the following motors and mechanical parts, all of which are available at Tower Hobby and many other online hobby stores (see Appendix B for Web sites).

- Tamiya Twin-Motor Gearbox, #70097. The motor comes as a kit and is assembled in about 20 minutes using a screwdriver and small needle-nose pliers.
- Tamiya Ball Caster, #70144. You get two ball caster units; you need only one for the BasicBot, so save the second caster for another project. Construction takes about five minutes.
- Tamiya Truck Tire Set, #70101. You get four tires; you need only two.



Tamiya also offers the model #70168 Double Gearbox kit. It is functionally identical to the Twin-Motor kit, except its dimensions are slightly different. This means that if you use the double gearbox you'll need to adjust the drilling pattern in order to properly mount the motor to your robot base.

CONSTRUCTING THE BASICBOT

Refer to Figure 26-6 for the cutting and drilling layout. Use a 1/8" bit to drill the holes. The location of the four holes on the base isn't supercritical, but the spacing is. You may wish to use the constructed motor and ball caster to mark off the holes.

1. Begin by constructing the twin-motor gearbox according to the instructions that come with it. You have the choice of building the motors with a 58:1 or 203:1 gear ratio. Opt for the 58:1 ratio if you'd like a faster robot. For maze following and other tasks you're better off with the slower, 203:1 ratio.
2. Before inserting the motors into the gearbox, solder wires to them and connect the wires to a set of switches or control electronics. See Chapter 22, "Using DC Motors," for ways to control small motors.
3. Construct the ball caster according to the instructions that come with it. The caster comes with various pieces to alter its height. Your finished caster should measure about 1" from the base to the ball socket.
4. Mount the motor box and ball caster. Assuming a 1/4"-thick base, use 4-40 × 1/2" machine screws and nuts. You can use 3/8"-long screws if the base is 1/8" thick.
5. Mount rubber tires onto two of the truck tires, then insert the wheels over the motor shaft. Figure 26-7 shows the underside of the BasicBot, with motor and ball caster attached.



Figure 26-5 The BasicBot uses one or two decks, a Tamiya Twin Motor gearbox kit, a set of small tires, and a ball caster for balance. Construction takes less than an hour.

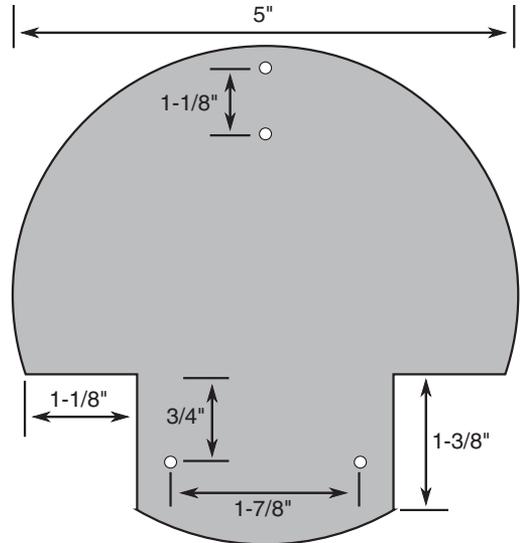


Figure 26-6 Cutting and drilling layout for the BasicBot. All holes are $1/8$ ".

ADDING A SECOND DECK

The one-deck BasicBot has room for a small microcontroller board, mini solderless breadboard, and a four-cell AAA battery holder. You can increase the area of the BasicBot if you need more space. Make a second deck with another 5"-diameter circle. Drill matching holes in both the bottom and the second deck. Use 1"- or 1-1/2"-long metal or plastic standoffs to act as "risers" to separate the two decks.

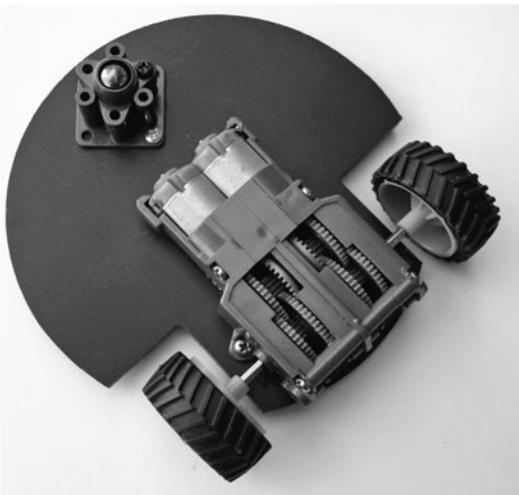


Figure 26-7 Underside of the BasicBot, showing the twin motor gearbox and ball caster.

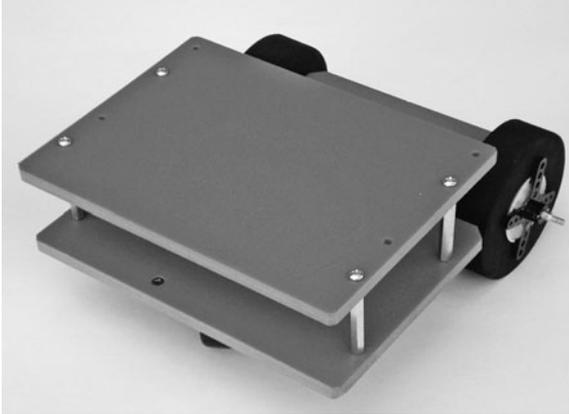


Figure 26-8 Completed RoverBot, constructed using only straight cuts. You can use wood, plastic, or metal for the body pieces. See the RBB Online Support site for full plans on constructing the RoverBot.

USING MORE EFFICIENT MOTORS

The Tamiya Twin-Motor kit makes for an affordable and easy-to-use motor set for small desktop robots, but these motors are not terribly efficient, and they consume a lot of current.

You can replace the DC motors used in the twin-motor kit with more efficient ones. The gearbox kit uses a pair of Mabuchi FA-130-size motors. You may substitute for a more efficient motor, as long as it conforms to the “130” motor size. Pololu and several other online sources offer replacement 130-size motors that consume much less current than the stock units that come with the Tamiya kit.

Bonus Project: Double-Decker RoverBot

Figure 26-8 shows a double-deck tabletop robot that I’ll call RoverBot. This base requires only rudimentary construction skills (straight cuts only) and can be made using a number of materials, including 1/4”-thick aircraft-grade plywood, 1/4”-thick expanded PVC sheet, or 1/8” acrylic plastic. It’s designed to be easy to build and easy to expand.

This is a free bonus project available on the RBB Online Support site (see Appendix A). On the support site you’ll find complete construction plans, including cutting and drilling layouts, parts list and sources, and assembly instructions.

Building 4WD Robots

What’s better than a two-wheeled robot? Why, four wheels, of course! Four-wheel-drive (4WD) bots are able to traverse more kinds of terrain than their two-wheeled cousins, moving from indoors to out with ease. They’re the preferred method of exploring grassy or dirt areas. And because they have four wheels, 4WD robots are statically balanced and have no need (or use) for a caster. Like a 2WD robot, 4WD bases use differential steering to explore their world.

Alas, 4WD robots are a bit harder to construct, and, depending on how they're designed, they cost more. But the advantages of a 4WD base often outweigh the disadvantages of a higher price tag and extra time in the shop.



What applies to the typical 4WD robot also applies to those using six (or more) wheels. As you add drive wheels, the complexity, weight, and cost of the robot can skyrocket. You might consider instead a tracked base, which functions like a multi-wheel-drive system, with an infinite number of wheels. See the following section for more on robots that use tracks.

SEPARATE MOTORS OR LINKED DRIVE?

While there are four-wheel-drive systems where two of the wheels are unpowered, in the typical 4WD all four wheels provide oomph to the robot. That means you either must use separate motors for each wheel or somehow link the wheels together so they're driven by the same motors. Figure 26-9 shows the basic concept.

- Separate motors cost more but require less mechanical complexity. You merely add two more motors and wheels to the base. With separate motors you can also control them individually, which offers some benefits over loose terrain like unpacked dirt.
- Linked drive saves the added cost of two extra motors, but requires you to develop a system where each motor powers two wheels at the same time.

With either method, the wheels of the 4WD robot are the same diameter and are placed toward the center of the base. The farther apart the wheels on each side, the more difficult the steering. On many four-wheel-drive bots, the wheels are placed with only minimal separation.



Of the two, 4WD systems with separate motors provide the most power, simply because each motor is dedicated to a single wheel. On linked-drive systems, the one motor is shared between wheels, so the overall power of the base is less.

Constructing a Separate-Motor 4WD Robot

A simple yet fully functional 4WD robot can be built using three pieces of wood, plastic, or metal. The motors (let's say, R/C servos) are mounted in what I call *side rails*. They're just overgrown motor mounts, with cutouts for the body of the motor and holes for the screws. The side rails attach to a deck piece by way of any kind of corner angle bracket.

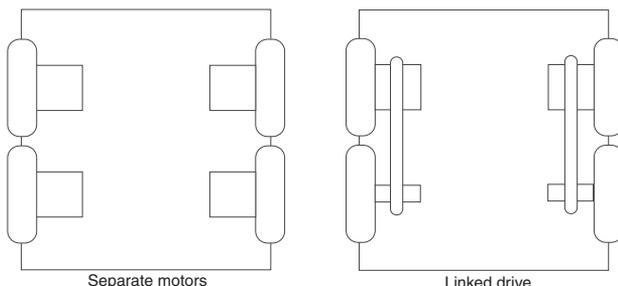


Figure 26-9 4WD robots take two general forms: those with four independent motors and those with two motors. The wheels on each side of the bot are linked to a single motor.



Figure 26-10 A 4WD robot, using servo motors and some oversized plastic wheels ripped off from a toy. The base is constructed by attaching the servos in *side rails*, and the side rails are attached to a top plate.

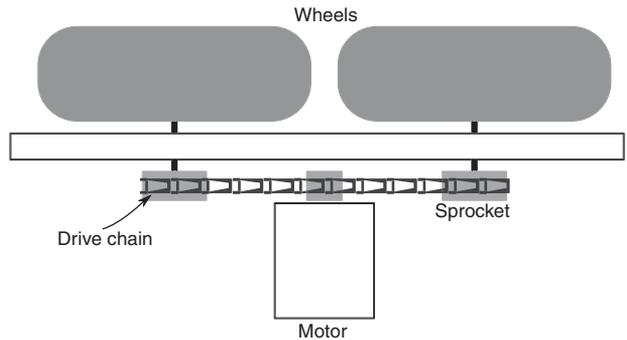


Figure 26-11 Chains (shown here) and belts can be used to link the wheels of a 4WD robot that uses only two motors. The chain allows for less precise construction requirements.

Figure 26-10 shows the basic concept of a 4WD base. The spacing between each pair of servo motors depends on the diameter of your wheels and the way the servos are mounted in the side rails (i.e., output shafts facing one another provide less space than if they face outward). In all, it's a pretty straightforward design: mount the servos in side rails; stick the side rails to the base.

The choice of corner angle brackets is up to you. I used some plastic brackets I had lying around the shop, but the common $3/4" \times 1/2"$ -wide corner angle bracket available at any hardware store works as well.

On my prototype I used a set of six $4-40 \times 1-1/2"$ machine screws, along with some $5/8"$ nylon standoffs I bought surplus, as risers between the bottom deck of the bot and an optional second deck. Holes are cut in the bottom deck to feed wires through for connecting to batteries, microcontroller, and other electronics.

Constructing a Linked-Drive 4WD Robot

A 4WD linked-drive system uses a single motor on each side of the robot and a power train coupling to connect each motor to its wheels. The most common techniques for coupling the motor and wheels are gear, chain, or belt drive.

- Belt and chain drive are probably the easiest methods, because both offer a bit of “slop” in aligning all the parts. A central motor (on each side of the bot) powers both wheels using either a flexible belt or a segmented chain; see Figure 26-11 for details. For best results, the belt should be *cogged*; that is, it should have nubs molded in that assertively mesh with teeth in the sprockets used on the motor and wheel shafts.
- Gear drive uses a main drive gear from the motor to mesh with subgears attached to the wheels (Figure 26-12). This is the method most commercially made 4WD toys use, but it requires extra precision when constructing a homebrew solution.

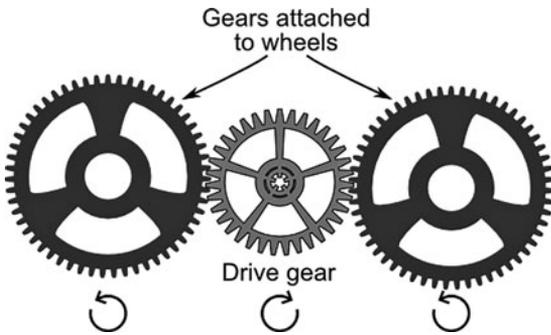


Figure 26-12 Gears used to transfer power from a central motor to a pair of wheels. The gears must be carefully positioned or else the teeth may not mesh properly.



Figure 26-13 Tank-style robots use rubber, plastic, or metal treads, along with a drive motor and a series of unpowered *idler* wheels that keep the treads in place. Low-cost toys are a common source of useful rubber treads.



You must be very careful to align the gears with the proper spacing, or else they won't mesh properly. The mechanism will bind if the gears are too close together, or slip and chatter if they're too far apart.

For small robots (let's say, under about 8" to 10"), Tamiya makes several products that can be used to create belt- or chain-drive 4WD bases. First is the Tamiya Ladder-Chain Sprocket set (#70142), which consists of plastic molded sprockets and individual links of chain that you connect together. Add or remove links to make the chain the exact length you wish. Because of the parts assortment you get, you will need two complete sets for one robot base.

Another method is the Tamiya Track and Wheel Set (#70100), which comes with molded plastic sprockets and rubber 1"-wide tracks. This set is normally used to make a tracked vehicle (see later in this chapter), but it also works well as a belt-drive system for a 4WD vehicle.

Building Tank-Style Robots

Another popular form of the rolling robot is the tank tread design, so called because it uses treads (or *tracks*) similar to those on military tanks. Like 4WD robots, tank tread bots don't need a balancing caster or skid. They use differential steering like 2WD and 4WD bases and are expressly designed for use over uneven terrain. Figure 26-13 shows a representative homebrew tracked-drive robot, made with a rubber tread stolen from a 1/8-scale tank toy and refitted over a plastic base.

FINDING THE RIGHT TANK TREADS

The first order of business is to locate a suitable tread or track for the tank-style robot. Common tread materials are rubber, plastic, and metal.

- Rubber treads are perhaps the most common, found (for example) on many tank and earthmover toys. You can rob the toy of its treads and other parts and use them on your robots.
- Plastic treads are made of rigid segments, linked together using pins or rivets. Several companies (Lynxmotion, Vex, JohnnyRobot) make plastic tracks for the express use as robot treads.
- Metal treads are found on high-end die-cast toys, as well as snowmobiles. These are heavy and expensive and are ideally suited for larger bots.

Regardless of the material, the treads work in the same way: a drive sprocket positively engages with matching teeth or indentations in the tread. The tread is laid out along more or less the full length of the robot. Nonpowered idler sprockets or untoothed wheels keep the tread in place.



Each kind of track has its own unique method of engaging with its drive sprocket. Whenever possible, always purchase (or rob from a toy) a track with its corresponding sprockets and idlers.

BOTS WITH FLEXIBLE RUBBER TREADS

As noted, one of the best sources for inexpensive rubber tracks is toy tanks. These are sold in different scales, from about 1:64 (miniature) to upward of 1:10 or even 1:6. (The scale is the ratio of the size of the model to its original. A scale of 1:24, for example, means the model is 1/24 the size of the original. Most toy tanks are in the range of 1:24 to 1:32 scale.)

The robot in Figure 26-14 was constructed using rubber treads from a 1/10 (approximately) scale tank toy. The full circumference of the treads is about 26", large enough to allow the unusual arrangement you see in the picture. The outside of the treads extends beyond the body of the robot, allowing it to flip over and still keep going. The robot can start to climb up a wall, and when it flips over, the motors immediately change direction. The robot doesn't have to stop, back up, and steer around the wall.

Look for a toy where the track is not too elastic and where, at a minimum, the drive sprocket and idler rollers can be removed and placed on your own custom base. Some toy

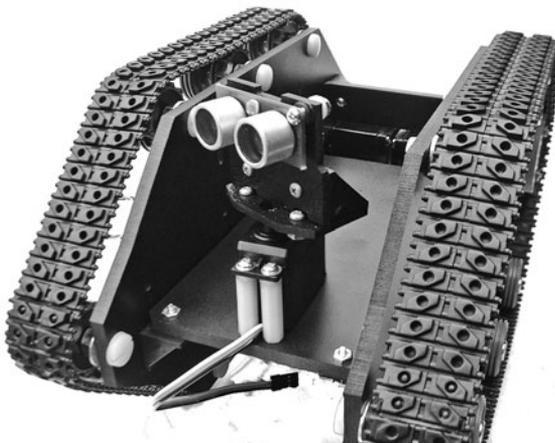


Figure 26-14 An *invertible* tank-type robot, capable of operating right side up or upside down. The treads and idler wheels were pulled from a remote control tank toy.

tanks offer easier hacking, where you can simply remove the turret and top of the vehicle, and replace the electronics with your own microcontroller and H-bridge. For these, you don't need to build a body for your robot, as you have one ready-made in the toy itself.



A good source for smaller rubber treads is LEGO Technic sets. The sets come with suitable sprockets that are made to engage with the teeth on the inside of the treads.

USING THE TAMIYA TRACK AND WHEEL SET

A commonly used track for robots is made by Tamiya and sold by itself as Tamiya Track and Wheel Set (item #70100). A number of online sources, such as Tower Hobbies and Hobby-linc, offer this set (see Appendix B, “Internet Parts Sources”). The track is also included in a few other Tamiya products, as the Tamiya Tracked Vehicle Chassis Kit and the Tamiya Remote Control Bulldozer Kit. These also come with motors.

The Tamiya track is rubber and comes in segments of various lengths. You put the segments together to build a track. Sprocket and idler rollers are included (see Figure 26-15). Pick out the parts you need. The segments connect using a little nub on the edges of the track. Despite how it sounds—or even looks—the tracks are fairly robust and seldom break apart unless forced.



In a pinch, you can glue the pieces together with a flexible adhesive, such as silicone caulk. Make sure the adhesive doesn't seep into the part of the track that engages with the sprocket and that the seam is smooth.

Build an All-Purpose Tracked Robot Base

You can construct a practical and sturdy tracked robot base using two motors (DC gear or R/C servo) and a Tamiya Track and Wheel Set (#70100). The finished base is shown in Figure 26-16. Ideal construction materials are 1/4" aircraft-grade birch plywood or 6mm expanded PVC.

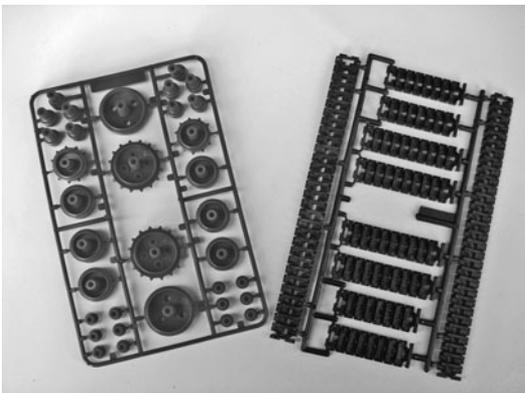


Figure 26-15 Tamiya Track and Wheel Set, showing both plastic parts and rubber tread. You connect the tread in various lengths to suit the size of your robot.

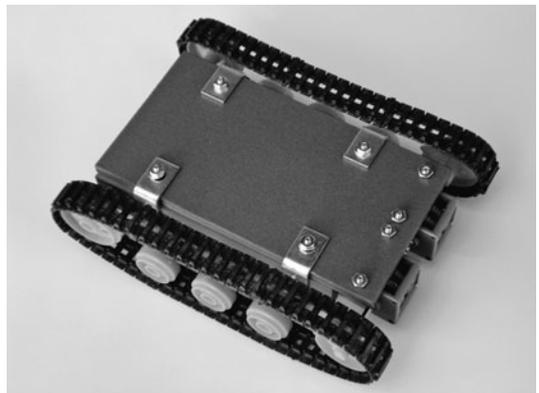


Figure 26-16 The All-Purpose Tracked Robot Base, using a couple of Tamiya gear motor kits and a Tamiya Track and Wheel Set. Construction is easy, and the whole thing costs under \$25.

In addition to the base material you need:

1	Tamiya Track and Wheel Set (#70100)
2	Tamiya 3-Speed Crank-Axle Gearbox kit (#70093), or equivalent
8	4-40 × 1" machine screws
8	4-40 nylon insert locking nuts
18	#4 washers
6	4-40 × 1/2" machine screws, nuts
4	3/4" × 1/2" wide corner angle brackets

Begin by cutting out the base deck and side rail pieces as shown in Figure 26-17. Refer to Figure 26-18 for the drilling details for the two side rails (make two of these). Drill the holes using a 1/8" bit. Hole placement is fairly critical, especially the distance between the two mounting holes for each motor, so use care.



Note the alternative mounting holes. These holes are for providing a higher tension for the treads. See the section "Tread Assembly" for more info.

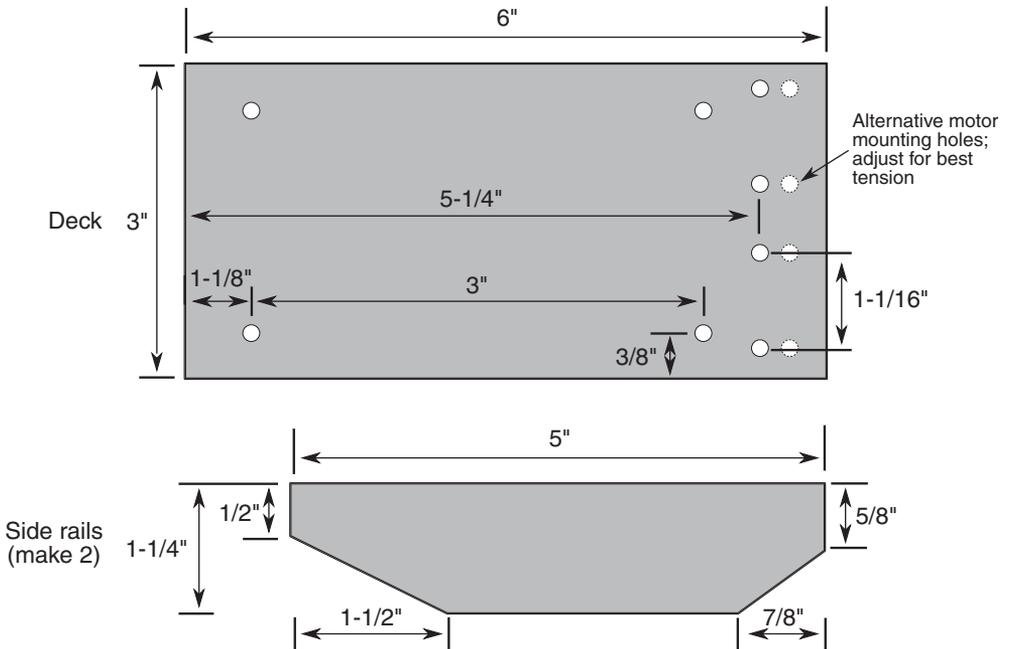


Figure 26-17 Cutting and drilling guide for the All-Purpose Tracked Robot Base. All holes are 1/8".

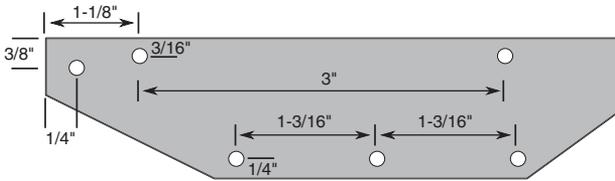


Figure 26-18 Drilling guide for the side rails for the All-Purpose Tracked Robot Base. All holes are $1/8"$. Hole placement is fairly critical in order to maintain proper tension of the track. (Even so, you may need to re-tension the track as detailed in the "Tread Assembly" section.)

I've used a pair of Tamiya 3-Speed Crank-Axle Gearbox kits (#70093), as they are inexpensive and offer a choice of three different gear ratios: 17:1, 58:1, and 204:1. Normal speed is 204:1, but if you want a really fast tank, pick the 58:1 ratio.

Before construction, use a hacksaw or heavy-duty lineman's pliers (be sure to wear eye protection!!) to cut the hex shaft to a length of $2-1/4"$. Complete the assembly of the motors as described in the instruction sheet that comes with them. You'll need a small Phillips screwdriver. Be sure to use the same gear ratio for both motors.

Prior to mounting the motors, use the included hex wrench to lightly tighten the setscrew that secures the driveshaft to its output gear. You need to make a "right" and a "left" motor, as shown in Figure 26-19. Position the drive shaft so there's no more than $1/4"$ protruding from the side of the motor.

Assembling the Side Rails

Refer to Figure 26-20 for the following. The side rails are what the tracks are attached to. Let's begin with the left side rail.

1. Use a $4-40" \times 1"$ screw, washer, and locking nut to attach the large idler wheel to the hole on the far left side of the rail. The wheel should face you. Tighten the locking nut so that the wheel just begins to stop turning freely, then back off about $1/8$ of a turn. When you rotate the wheel there should be no drag. Neither should there be any kind of excessive wobble.
2. Use three more screws, washers, and nuts to attach the small idler wheels to the three holes along the bottom of the rail. Tighten them as you did the large wheel.

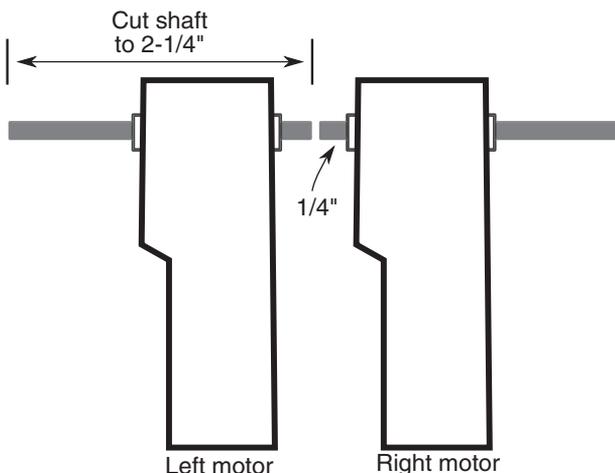


Figure 26-19 Motor construction and drive axle shaft placement for the All-Purpose Tracked Robot Base. You need to cut the axle shaft to length, using a saw or pair of heavy-duty lineman's pliers (be sure to wear eye protection!).

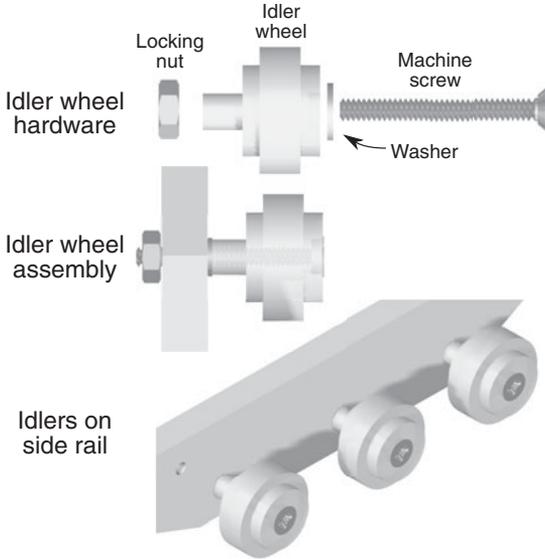


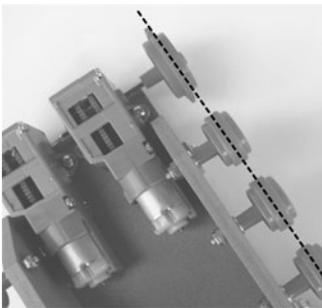
Figure 26-20 Construction detail for the idler wheels. Tighten the locking nut so the wheel just starts to drag, then back off the screw 1/8 turn.

- Using 4-40 × 1/2" screws, washers, and nuts, attach two corner angle brackets as shown in the illustrations.

Construct the right side rail in the same manner, but in mirror image.

Attaching the DC Gear Motors

Use 4-40 × 1/2" screws and nuts to attach the two motors to the base. The short ends of the driveshafts should face each other.



Assemble the side rails to the base, using 4-40 × 1/2" screws, washers, and nuts—see Figure 26-21. The motors are located at the rear of the base; the front should be flush with the leading edge of the right and left side rails.

Temporarily attach the large sprockets to the driveshafts. Using the hex key included with the motor, slightly loosen the setscrew that holds the driveshaft in place. As needed, move the driveshaft in the motor so that the sprocket lines up with the idler wheels in the side rail. Even a slight misalignment can cause the track to pop off when the robot turns, so try to be as accurate as possible. When done, tighten the setscrew on each motor.

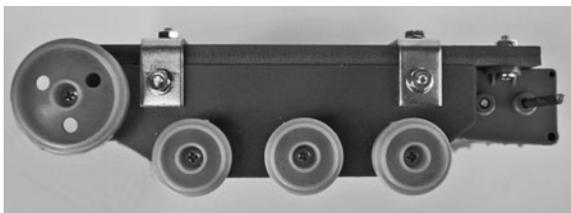


Figure 26-21 Edge view of the idler wheels attached to the All-Purpose Tracked Robot Base.

Tread Assembly

Assemble the treads from the Track and Wheel Set using the following lengths for each side:

- 1 each 30-links segment
- 2 each 10-links segments
- 1 each 8-links segment

The segments interlock into one another. Use only your fingers; avoid using mechanical force or tools to link the segments together, or else the rubber may get torn. It can take a few tries to get the hang of it. Refer to the instructions that come with the set for details.

To install the tread, loop over the four idler wheels. Wrap the tread around the teeth of the drive sprocket, and carefully push the sprocket onto the motor shaft. Note: Do not overstretch the tread or it may become unlinked.

The design of the All-purpose Tracked Robot Base allows for adjusting the tension of the track—important, because if the treads are too loose, they'll pop off easily. Remove the motors, and drill new holes about 3/16" away from the original set. Remount the motors.



You can create a slot for adjusting the track tension by drilling several holes close together in line. Use a thin rat-tail file to remove the material between the holes, thus making a slot.

Replacement FA-130 Motors

The miniature Mabuchi FA-130 motor included with the 3-Speed Crank-Axle Gearbox kit is not very efficient, consuming several amps when the motor is stalled (prevented from turning, but still powered). If you use electronic control to operate the motors, be sure it is rated for at least 2 amps, if not more.

Optionally, you can substitute the FA-130 for a higher-voltage, lower-current version. Pololu and several other online resources offer 130-size replacements that consume much less current and can be used with small motor bridge circuits. Consult Appendix B, "Internet Parts Sources," for Web links.

Optional: Using Servo Motors

If you prefer, you may use an R/C servo to drive the tracked base. Mount the motor to the robot base using your favorite technique (see Chapter 24 for some suitable mounts you can make yourself). Then, interface the sprocket to the motor with these steps:

1. Cut off the molded-in cap on the side of the drive sprocket.
2. Drill out the center using a 1/4" bit.
3. Drill out two holes in a small round servo disc to match two of the holes in the drive sprocket. It turns out the holes match exactly when using the small round servo disc that comes with most Futaba and Futaba-style servos.
4. As shown in Figure 26-22, attach the servo disc to the sprocket using a pair of 4-40 × 3/8" machine screws and nuts.
5. Use the screw included with the servo to mount the drive sprocket to the motor.

Adding a Second Deck

Need more space? You can add a second deck to the tracked base by removing the nuts on the top of the corner angle brackets and replacing them with 4-40 threaded hex standoffs.

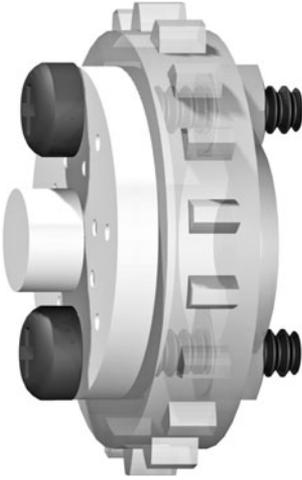


Figure 26-22 Hardware detail for mounting a small round servo horn onto the drive sprocket that comes in the Tamiya Track and Wheel Set.

Cut out the wood or plastic for the top deck, and secure it in place over the hex standoffs using $4-40 \times 1/2$ " screws.

BEST STEERING FOR TRACKED ROBOTS

Because a tank track exposes a considerable amount of its surface onto the ground at any one time, in a turn the tracks must actually slip, or skid, over the earth. The part of the track farthest from the midpoint of the vehicle skids the most.

Unlike a differentially steered two-wheel bot, where it is possible to turn by simply stopping one wheel, this is not advisable with a track drive. The track exposes too much surface area to the ground, which greatly increases friction. The stopped track will skitter over the ground (and possibly come off), and turning is harder to control.

SPECIAL CONSIDERATIONS FOR RUBBER TREADS

Because of size, cost, and weight concerns, the track material on most robot tanks is rubber. Rubber has a higher compliance than plastic or metal. If the robot is operated over a surface that is also fairly *compliant* (means having resiliency or “give”), turning may be difficult for the little tank.

Another potential issue of using a rubber tread is static friction, or *stiction*. (There may be other frictional components involved, but we’ll bypass them for this discussion.) With stiction, a rubber tread may have difficulty skidding over a highly polished material, such as a glass tabletop or hardwood floor.

There are numerous techniques to reduce the steering problems inherent in all treaded vehicles. One is to use a less compliant tread material. Not all rubber compounds are equally elastic. A good rubber tread for a tank design exhibits only limited elasticity (stretch). The surface of the rubber is smooth and may have molded-in “cleats” that reduce the surface area of rubber touching the ground at any one time. With less surface area, there is less rubber to skid.

USING PLASTIC TREADS

An alternative to rubber treads is tracks of hard plastic. An example is the track for the Vex Robotics Design System. The kit, which is designed for the Vex line of robots but can be adapted to other applications, consists of a series of plastic links that you put together.

Another example of hard plastic tracks comes from an outfit named JohnnyRobot and is sold by a number of specialty online robotics sources (see Appendix B for Web sites). These tracks are composed of ABS plastic links, connected by miniature stainless steel rods. You connect the links together to make a track any size you want.

Plastic sprockets and idlers are also available to make a complete tracked system. Drive sprockets are available for Futaba and Futaba-style servo motors, Solorobotics, and other DC motors that use 7mm double-flatted driveshafts. Figure 26-23 shows a complete track drive subsystem that uses the JohnnyRobot links, ready to be mounted on a base. Of course, you need two of these drive subsystems to make a completed robot.

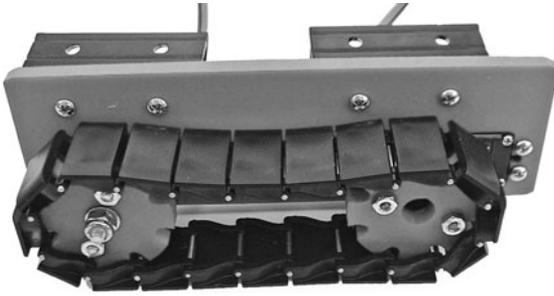


Figure 26-23 All-plastic treads on a custom-made tank-style drive subsystem. Just attach the subsystem to a base to complete the robot.



If there is a disadvantage to hard plastic tracks, it's that the plastic may slip over hard surfaces—the exact opposite of rubber treads. Depending on the design of the track, you may be able to overcome this by applying small pieces of rubber material over the track segments. This provides enough compliancy to improve locomotion and steering.

DEALING WITH DETRACKING

Rubber and plastic tracks (or metal tracks, for that matter) differ in their resistance to *detracking*—also called *derailing*, or “throwing a track.” Detracking occurs mostly when negotiating a turn. This is when the frictional forces acting against the track are at their highest. As the vehicle attempts to turn, heavy sideways pressure is exerted at the front and back of the track. If the pressure is great enough, the track may come off its drive sprocket or guide rollers.

Detracking rears its ugly head the most when using highly elastic rubber treads. The more elastic the track material, the more readily it will stretch during a turn. The problem is magnified if the tank is loaded down with weight. The heavier the vehicle, the more likely you'll have a thrown track. To limit this problem:

- Reduce the weight on the vehicle.
- Make slower turns.
- Try to find a rubber tread that doesn't stretch as much. The lower the elasticity, the less likely the tread will pop off.
- As necessary, tighten the track by adjusting the distance between the drive sprocket on one end and the idler roller on the opposite end. This limits the track from stretching too much more. Avoid overtightening, which can deform the tread and place excessive stress on the drive components.
- Decrease the surface area of the tread on the ground. You may do this by changing the elevation of the idlers toward the front and back.
- Experiment with the width between the tracks. Longer, narrower track widths resist turning more than shorter, fatter widths.
- Add “keepers” to the idlers that don't touch the ground. The keepers are like oversized rims that keep the track in place.

By their nature, plastic and metal tracks don't stretch, so, assuming they are placed snugly onto the sprocket and idlers, detracking is rare.

Build Robots with Legs

Do you enjoy challenges? I mean the really tough nuts that are hard to crack, the stuff that keeps you up at night as you try to work through the problems? If so, then maybe you're ready to build a walking machine.

Legs are biologically inspired solutions to that old problem of how to get your robot from here to there. And not only as a means to mobilize your creation, but to step over common obstacles like yesterday's lunch bag, your dirty socks, and the family turtle—stuff that might confound the typical robot with wheels or treads.

In this chapter you'll learn about the role of legs in creating mobile robots, and the special requirements and limitations they impose. You'll read about ready-made solutions, as well as several designs for making your own six-legged bots on a budget and from scratch.

An Overview of Leggy Robots

With legs, a robot can live among humans, ideally without any kind of special adjustments or alterations to the environment. Ramps, curbs, steps, stairs, and cracks in the sidewalk all pose no more of a problem for the robot than they do for any other ambulatory human being.

As of this writing, human-size legged robots that can go anywhere and do anything are still the province of science fiction. Less ambitious are the full-scale walking robots of industrial and educational research that show promise but cost a fortune and are known to fall over on their batteries now and then.

Small-scale legged robots are another matter. With a modest inventory of servo motors and some specially crafted brackets, you can construct walking bots with two, four, six, and even more legs. Though they're more expensive and more demanding to build than robots with wheels or tank treads, constructing a programmable walkerbot is well within the reach of the garage-shop tinkerer.



Legged robots aren't for beginners. If you're just starting out, first hone your skills by constructing a couple of wheeled bots. This applies whether you buy a ready-made kit of parts or build it from scratch. Walking robots require greater precision and attention to detail—when improperly constructed, they rattle apart, may simply not work, or fall over in a crashing heap.

NUMBER OF LEGS

This concept was introduced in Chapter 2, “Anatomy of a Robot,” and Chapter 20, “Moving Your Robot,” but it bears repeating here. The most common forms of legged bots are:

Bipeds, which walk on two legs. In a true bipedal robot, one leg lifts up to make a step. Balance can be tricky when the robot has just one foot on the ground, making this type among the hardest to master. The alternative is the “shuffling” walking robot common in toys whose legs don't actually lift up and down.

Quadrupeds, meaning four legs. The more sophisticated four-legged bots demonstrate a variety of walking styles, some mimicking animals. These can be tricky to build, because each leg needs three separate joints—otherwise known as degrees of freedom, or DOF—in order to keep balance and make turns.

Hexapods, for which six legs provide excellent balance and mobility. Among walking robots, they're the most common and, despite the extra legs, the easiest to build.

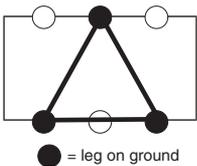
And, of course, there are examples of robots with other appendage counts. One-legged robots, or hoppers, look like pogo sticks without a rider. They move around by bouncing.

Robots with eight or more legs, or multipods, include a wide variety of wild and crazy designs. These include segmented bots, which behave (and somewhat look) like snakes, worms, or caterpillars. Each segment is controlled by one or more motors not unlike the joints in a walking robot.

A few experimental walking robots employ an odd number of legs. If four are too few, and six too many, why not build a bot with five legs? And finally, hybrid robots may combine legs with wheels or tracks. An example is a 4WD robot that has two legs in the front. The legs can help to lift the robot up steps and over obstacles and, if jointed properly, can serve as manipulators. Pretty rad stuff.

STATIC VERSUS DYNAMIC BALANCE

Balance is the ability of the robot to remain upright when it's standing on its legs. Balance can be static or dynamic.



- *Static* balance means the legs provide a natural stability, using any (or a combination) of several techniques. In four- and six-legged bots, the most common static balance comes from always having at least three legs on the ground at the same time, forming a tripod stance. With all types, static balance is improved by having a low center of gravity; much of the weight is at least 50 percent below the overall height of the robot.
- *Dynamic* balance means the robot uses sensors to keep itself upright. When the bot feels it's starting to tip over, the sensors activate one or more motors to shift the robot's weight one way or another. As the weight shifts, the tilt is negated, and the robot is kept upright.

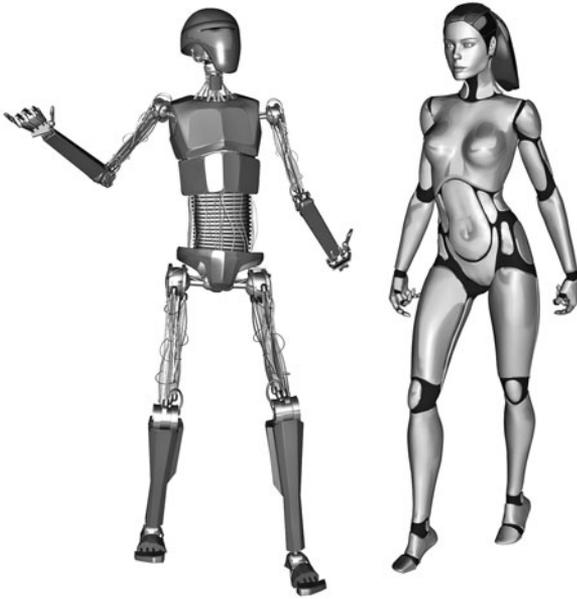


Figure 27-1 Bipeds in both android and humanoid forms.

BIPEDS: ANDROID OR HUMANOID

The terms *android* and *humanoid* are used to describe robots that are modeled after the human form (Figure 27-1). There's a head, torso, two legs, and at least one arm. Though usage varies, the terms don't mean exactly the same thing. An *android* is a robot designed to look as much like a human being as possible, either male, female, or an *androgynous* mix of the two (androgyny is where we get the term andr-oid, meaning "like male and female"). The robot has eyes where we have eyes, a nose, mouth, ears, and other things that make us look like we do.

Conversely—and somewhat confusingly—a humanoid robot is one that shares the basic architecture of a human. It has two legs, with a head at the top, and two arms at the side. Rather than duplicate the appearance of people, it's meant more to replicate human ability, such as being able to walk through a hallway meant for humans or sit down in a chair meant for humans.

NUMBER OF JOINTS/DEGREES OF FREEDOM

Leg motion is provided by a series of joints. Some joints allow the leg to swing back and forth, while others permit side-to-side motion. The number of joints that provide motion is called the *degrees of freedom*, or *DOF*. For the most part, the more DOF, the more agile the robot. For amateur robots the 2- and 3-DOF bot designs are more common.

Except for some unique designs that use clever mechanical linkages, each leg DOF needs a separate motor to control the joint. The more motors, the more expensive the robot; it's heavier and larger, too. So consider the number of leg DOF when picking a walking robot design. That hexapod with 3-DOF legs looks mighty impressive; just remember it needs 18 motors. It's not unusual for these kinds of robots to cost upward of \$600, and that's before any microcontrollers or other electronics.

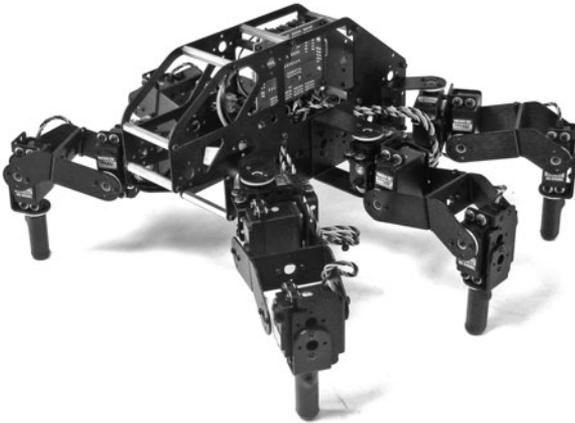


Figure 27-2 Commercially manufactured hexapod kit featuring 3-DOF legs. The extra degree of freedom in the legs provides a more defined stepping action. (Photo courtesy Lynxmotion.)

OPERATING TERRAIN

Bots with 2 DOF per leg work best on smooth, unobstructed surfaces—a kitchen floor or a hallway with wood laminate are good choices. The surfaces shouldn't be too slick, unless you add rubber pads on the robot's feet. Otherwise, the contraption will tend to skitter on the floor as it frantically moves its limbs. It's comical but not too productive.

The more sophisticated 3-DOF robots with four- or six-legs can be used on more challenging terrain. This is because they're endowed with an additional articulation that permits the legs to literally step up and down rather than just swing in and out, which is typical of the simpler 2-DOF designs. The full leg-lift helps the robot clear low objects and even thick carpet nap.

Many of the more elaborate ready-made walking robot kits are designed with this feature. Figure 27-2 shows an example: its third DOF permits the legs to lift up vertically, clearing obstacles at least 1/2" to 1" in height. Of course, these designs are more expensive because they require three servos per leg and additional leg hardware.

Selecting the Best Construction Material

Walking robots need to be strong yet lightweight. The heavier the robot, the less likely it can stand on its own two feet, so to speak. Pine lumber and soft plywood are pretty much out, because they're too bulky and heavy for the strength they offer.

FYI

There's lots more about wood, plastic, and metal in the chapters in Part 2 of this book. Be sure to see Chapter 7, "Working with Wood," Chapter 9, "Working with Plastic," and Chapter 11, "Working with Metal."

Materials choice, in order of preference, is:

- *Aluminum*, cut and drilled to shape. For walking robots under 1 foot high, thickness ranges from about 20 gauge (0.0320") to 8 gauge (0.128"). Premade parts are often cut by a powerful waterjet machine controlled by a computer, then bent using special jigs. When building your own, the likely tools are hacksaw, drill press, and bench vise.

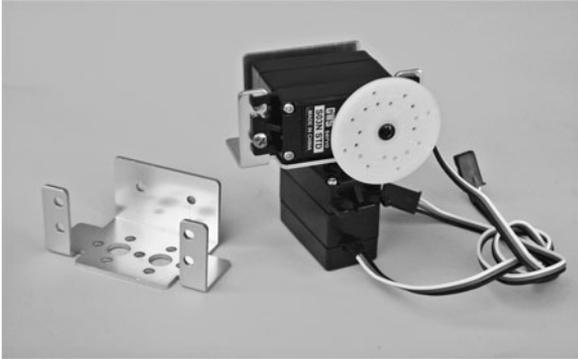


Figure 27-4 A precut and preformed aluminum X-Y joint, available from a number of online specialty robotics retailers.

Plans for the joint parts, which can be constructed using nothing more than hand tools, appear later in this chapter. These same joints are then used to create a fully functional and robust six-legged walking robot, which is provided as a free bonus project on the RBB Online Support site.

BUILD FROM PREMADE PARTS

Numerous sources such as Lynxmotion and Pitsco provide premade parts for X-Y joints. Prefabbed aluminum brackets like the one in Figure 27-4 come cut and drilled to work with most any standard-size servo motor. The bracket weighs about the same as the PVC X-Y joint (half an ounce), but the PVC version is considerably cheaper to build.



Various types of these aluminum brackets can be combined to create different arrangements. For example, you can create an X-Y bracket with an L bracket in order to reconfigure the orientation of the servo motors used for the joint. Among other things, this can be used to provide a more streamlined shape or to create mechanisms for 3-DOF joints.

Some kits with premade parts use modular metal and/or plastic construction and are designed as development systems for building walking and rolling bots. Example: The Bioloid robot sets offer numerous permutations including sophisticated bipedal designs. The kits, which are designed for advanced study, are available in different parts assortments.

BUILD FROM A MIXTURE OF PARTS

There's nothing stopping you from combining your own homemade parts with prefabbed ones, mixing and matching as needed. For example, you might combine a PVC X-Y joint with a premade L bracket. You might also provide your own chassis for the robot, leaving just the leg pieces as store-bought.

MOTOR SHAFT SUPPORT AND YOKE BRACKETS

The simplest of servo brackets, like the one in Figure 27-3, lack a means to support the load placed on the servo motor by the weight of the leg pieces. This isn't a major problem for

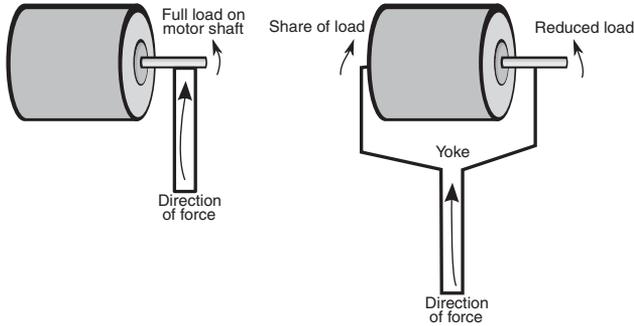


Figure 27-5 A yoke mechanism distributes weight or force so that it's not all against the shaft of the motor. This helps the motor work more efficiently and last longer.

robots that see only occasional use and aren't too heavy. But it's something you'll want to consider as your walkerbots gain weight or are frequently used for show-and-tell.

The most common way to alleviate the strain on the motor is by using a yoke or double-sided bracket (see Figure 27-5). Here, the load on the motor is distributed evenly between the motor shaft and a secondary (and passive) shaft. The two shafts are in line with one another. The second passive shaft can be a steel or aluminum rod, or even a machine screw. The shaft may use ball bearings or bushings to ensure unimpeded rotation.

Note that the X-Y joint plans that appear later in this chapter are sans support. You can add a yoke-style arrangement with nothing more than a third piece of plastic (or wood if that's your material of choice) that screws onto the bracket opposite the motor shaft.

Leg Power

When it comes to the muscles of a walking robot, radio-controlled servos are the preferred choice; they're compact and widely available, and they require no special interfacing or drive electronics connected to your robot's central computer.

Where rolling robots can make do with just about any standard-size R/C servo, legged robots need a bit more attention to the details. Specifically, you need servo motors that provide enough torque to lift the legs and move the robot. It's not unusual for a six-legged hexapod to weigh several pounds. With underpowered servos, the robot will just sag to the ground, unable to sustain its own weight.

CONTROLLING SERVOS

Recall from Chapter 23, "Using Servo Motors," that servos require a special pulse signal in order to operate. Typical operating voltage for servos is between 4.8 and 6.0 volts, though many robot builders push to the edges of the envelope and supply 7.2 volts—this increases the torque of the motor by as much as 30 or 40 percent.

Overvolting an R/C servo motor has the potential of burning it out, so you must exercise care if you wish to experiment with this technique. Some brands, and even models within each brand, are more tolerant of the higher voltages than others, so you need to experiment. When using a higher-than-normal voltage with your servos, give them a periodic touch test to make sure they aren't overheating. Immediately disconnect the servo if it seems to be getting too hot or is putting out an unusual smell.

Because of the heavy demands of the motors in a walking robot, you need to run them

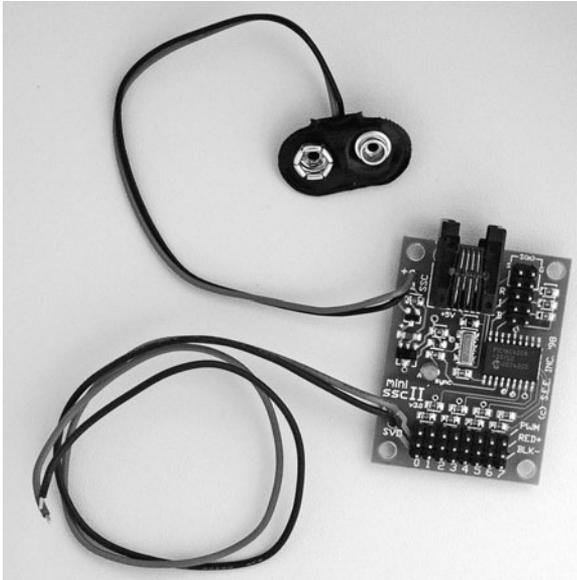


Figure 27-6 The Mini SSC II, a popular serial servo controller capable of operating up to eight R/C servo motors simultaneously. (Photo courtesy Scott Edwards Electronics.)

from a separate battery supply. When using this arrangement, be absolutely sure that the ground connections of the two battery supplies are connected, or your robot will not function properly.

USING A DEDICATED SERVO CONTROLLER

Legged robots require lots of servos. Rather than hogtie the robot's main processor with the job of running them all, you can hand it off to a coprocessor instead. That's the idea of the *serial servo controller*, or SSC (see Figure 27-6). These compact circuit boards are designed to receive one-time instructions from your robot's microcontroller on which servos to activate and what position to move them to. The SSC then independently controls the servos without any further intervention by the robot's controller.

It's called a *serial* servo controller because the microcontroller communicates with the SSC via a simple serial communications link. The link can be one-way or two-way. Most microcontrollers provide a simple command structure for sending serial data to another device. For example, the BASIC Stamp offers the *serout* command, and the Arduino has a SoftwareSerial library that can use any I/O pin as a serial line.



Check out the RBB Online Support site (see Appendix A for details) for several hands-on examples of using popular SSCs with the Arduino and other microcontrollers.

ANALOG VERSUS DIGITAL SERVOS

As you read in Chapter 23, the most common radio-controlled servos are analog. Digital servos use onboard microcontrollers to enhance their operation. Because of the way digital servos work, they tend to provide more torque and are naturally better suited for use on walking robots. Digital servos are more expensive (sometimes *much* more expensive), so you'll

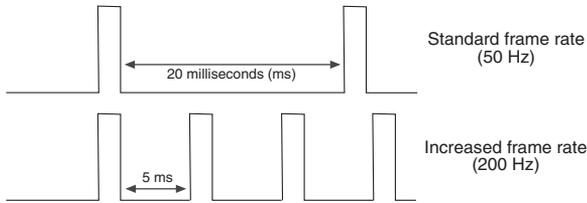


Figure 27-7 The frame rate of pulses applied to an analog servo directly affects its power. Increasing the frame rate can deliver more torque. Not all analog servos can tolerate a frame rate above a certain threshold.

want to carefully select the servos you use based on the weight and other design factors of your walkerbots.

Why are digital servos more powerful? Part of the answer lies in how the motor inside the servo gets juice, and knowing this answer can help you determine if cheaper analog servos might fit the bill.

In an analog servo, the control pulses sent to it act as momentary jolts of current, each jolt energizing the motor and making it go. As shown in Figure 27-7, the normal repetition rate of the pulses is around 20 milliseconds, which equates to 50 Hz (50 times a second).

If the repetition rate—or “frame rate”—is slowed down too much, the motor gets weak and may not even function. Conversely, with a higher frame rate the motor receives more pulses, and its torque increases.

The increase in frame rate is exactly what happens in a digital servo. Even though the servo may get the normal pulses at 50 Hz, internally the electronics in the servo energize the motor at rates in the range of 200 Hz or more.

Most analog servos cannot tolerate a frame rate of 200 Hz . . . even 100 Hz may be too high. It all depends on the brand and model of servo. If your microcontroller allows you to specify how often the servo is pulsed, you can experiment with different values to find the maximum rate before the motor either stops functioning or becomes erratic.



Carefully monitor the operation of your servo when altering any of its control signal characteristics. Analog servos that are fed a high rate of pulses may run at a higher current, which causes the motor to dissipate (give off) more heat. If the current and heat get too high, the motor may be permanently damaged.

Beyond a faster rate of pulsing, many digital servos also offer other torque-enhancing features, such as higher-efficiency motor windings and more sophisticated control electronics. The servo itself tends to be better made, able to withstand the higher current, temperatures, and torque exhibited by the motor. Many of the better digital servos use metal gears, which are preferred when the motor has to push heavy loads.

REMEMBER SERVO TORQUE RATINGS

Digital or analog, when selecting a servo for your walking robot take note of its torque rating, which will be listed at either 4.8 or 6 volts (or both). As you learned in Chapter 23, “Using Servo Motors,” the higher the torque, the more power is delivered by the motor. The typical standard servo provides under 50 or 60 oz-in of torque, which is acceptable for desktop robots and smaller walking robots.

For anything larger, you want a servo that delivers 90 oz-in or above. There are a number of standard-size analog and digital servos that offer this torque rating. Very large and heavy walkers need 200+ oz-in of torque, which is practical only with the much more expensive digital servos.

Walking Gaits for Legged Robots

Gait refers to the pattern of leg movement as an animal or insect walks. Gaits can and do differ depending on the speed of travel—a running gait is wholly different from a walking gait. The leg motions are distinctive in each one. So, too, for robots, though almost all are restricted to walking gaits. There aren't too many legged robots that can run, at least not without falling over and making fools of themselves.

Figure 27-8 shows most common gait of a six-legged robot that has independent control of each leg. This gait is often referred to as an “alternating tripod gait,” because at least three legs are always touching the ground at the same time, providing static balance. (A similar gait, also an alternating tripod, is used when the robot uses legs that are linked together. This gait is described in more detail in “Build a 3-Servo Hexapod,” next.)

The alternating tripod gait goes through a number of sequences, as shown in Figure 27-8. For each side of the tripod, the legs are lifted, lowered, and swept forward or backward in unison. This provides for a reasonably fast walk while still maintaining good static balance as the weight of the robot is shifted from one side to the other.

The legs act either to lift or to power.

- Legs are *lifted* to orient them into a new position. During lift, the leg does not provide propulsion, nor does it contribute to the balance of the robot.
- Legs that *power* propel the robot in the opposite direction of the movement.

To be sure, there are other gaits for six-legged creatures, living or robotic. They include metachronal (or wave) and ripple. Space forbids me from detailing each one of these, but you can learn more about these and others with a Web search.

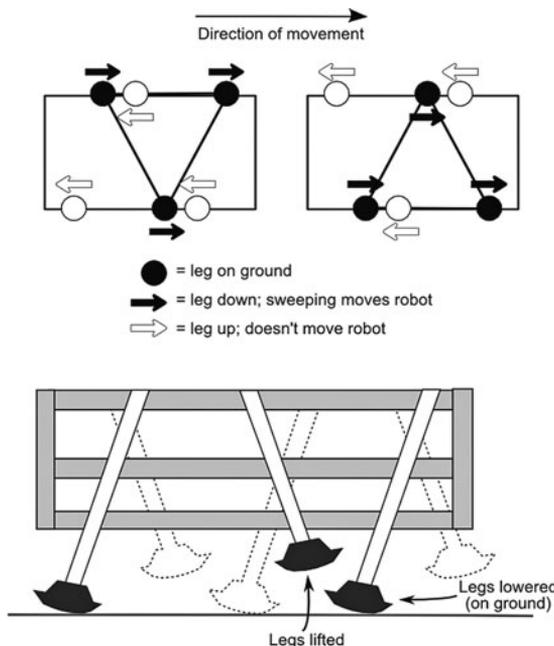


Figure 27-8 The alternating tripod gait common in hexapod robots. Three legs always touch the ground in a tripod arrangement. The robot moves by alternating the side of the triangle that's on the ground and “sweeping” those legs in the opposite direction of intended travel.

Build a 3-Servo Hexapod

Using simple linkages, you can construct a fully functional hexapod walking robot powered by just three servos. Figure 27-9 shows the completed Hex3Bot, which measures 7" by 10", and stands 3-1/2" tall. Weight is only 11.5 ounces when constructed out of 6mm expanded PVC, the recommended material. (You can also use 1/4" aircraft-grade plywood, but this will make the robot weigh a bit more.)

Like any hexapod robot with static balance, the Hex3Bot keeps from falling over by always having at least three legs—in a tripod arrangement—on the ground at any time. Walking is accomplished by sweeping the front and rear legs on each side in alternating cycles. The middle legs, which only lift up and down, act to “rock” the robot from side to side, providing the third leg of the tripod.

The walking gait is composed of a three-step sequence, shown in Figure 27-10:

1. Lift right or left middle leg. Only one leg is down at any time. The robot tilts to the side opposite the middle leg that is down.
2. Power sweep the front/rear legs that are touching the ground. The robot propels forward.
3. Non-power sweep the front/rear legs that are not touching the ground. The robot doesn't move in this step; it merely positions the legs for the next sequence.

PARTS YOU NEED

In addition to fastener hardware (see the section “Assemble to Complete the Hex3Bot” for a list), you need the following stuff to build this robot:

- 1 piece of 12" × 12" 6mm-thick expanded PCV (preferred), or 1/4" birch or other hardwood aircraft-grade plywood
- 3 standard-size servo motors.
- 2 12" lengths of 1/2"-wide by 0.016"-thick brass strips (available at hobby and craft stores, such as K&S Metals, #1412110231)

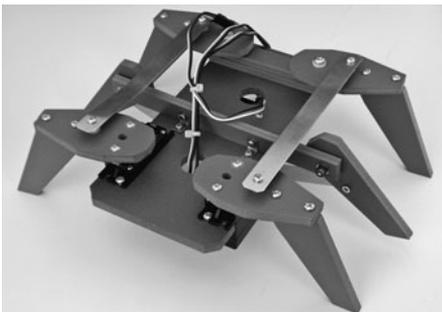


Figure 27-9 The completed Hex3Bot walking robot, which uses three R/C servo motors to get around. The Hex3Bot is an example of a hexapod design that uses linked legs.

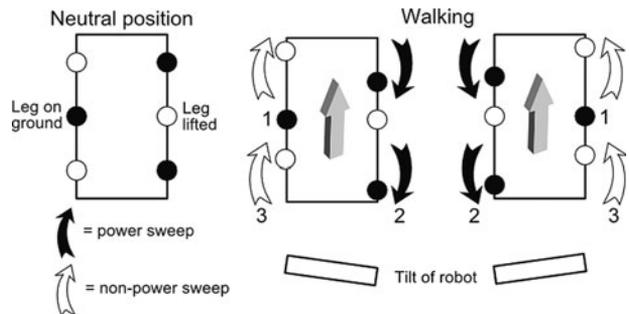


Figure 27-10 Walking gait of the Hex3Bot. The center legs serve to tilt the robot to one side or another. Motion is accomplished by pushing the left or right legs when they are on the ground. The numbers 1, 2, and 3 are the sequence of steps used when walking.

- 2 six-arm (“star”) servo hubs (they usually come with the servo)
- 1 large round servo hub (usually comes with the servo)

You need the following fasteners to complete the Hex3Bot (get extras in case you lose any during construction):

4	3/8" × 3/8" miniature L bracket, Keystone 633
8	#4 × 1/2" sheet metal screws
18	4-40 × 1/2" machine screws
8	4-40 × 7/16" machine screws (okay to substitute 1/2" length)
2	4-40 × 7/16" flathead machine screws (okay to substitute 1/2"-length pan head)
6	4-40 × 3/4" machine screws
2	4-40 × 3/8" machine screws
2	6-32 × 1-1/2" machine screws
2	#6 nylon washers (okay to substitute #6 metal washers)
8	#6 nylon spacer, 1/8" thick (check the special parts drawers at the better hardware stores, or you may substitute a stack of two or three nylon washers to build a spacer that is a total of 1/4" long)
6	#6 metal washers
6	#4 metal washers
2	6-32 locking nuts
10	4-40 locking nuts
24	4-40 hex nuts

Machine screws are pan or round head, unless otherwise noted.

CUT AND CONSTRUCT THE LEGS



All holes are 1/8" unless otherwise noted.

Begin by constructing the leg pieces; a cutting and drilling template is provided in Figure 27-11. The four front and rear legs are composed of two pieces—upper and lower. The middle legs have just one piece. The “feet” (bottoms) of all the leg pieces are cut at a 30° angle. Use a protractor to mark the angle. It’s okay if it’s a degree or two off, but try to be as accurate as possible to match all cuts.

Assemble the upper leg and lower leg pieces using #4 sheet metal (*not* wood) screws (see Figure 27-12). Using a pencil and the top leg as a template, mark the placement for the mounting in the edge of the bottom leg. Prepare a pilot hole for the screw with a small 1/16" bit. It doesn’t need to be a deep hole. Fasten the leg pieces using the sheet metal screws. They’ll self-tap as you tighten them.

Notice that there are two styles of legs: the rear legs, which include a hub for connecting

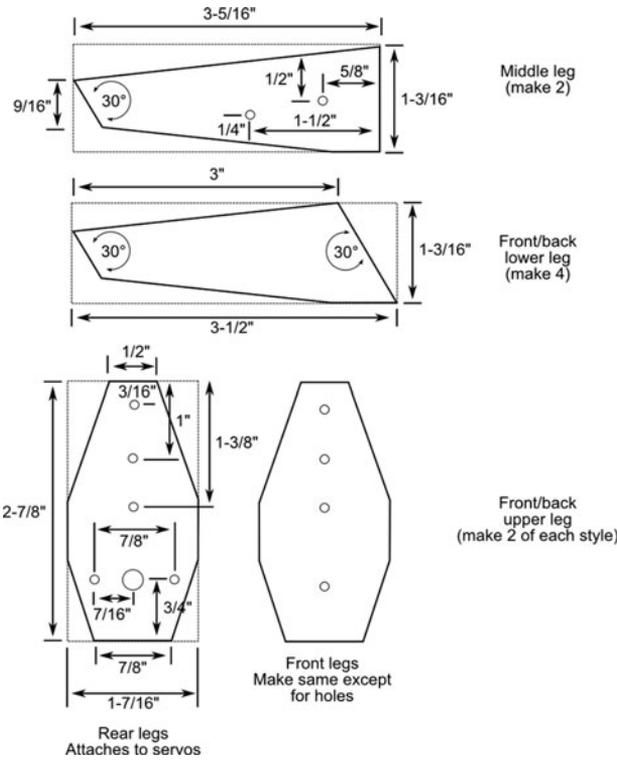


Figure 27-11 Cutting and drilling template for the Hex3Bof's legs. All holes are 1/8" except as noted.

to a servo, and the front legs, which attach to the robot using a screw and lock nut. You will make two of each style. I've set the hole spacing for a six-arm "star" servo hub, common on Futaba and Futaba-style standard-size servos. If you use a different hub, you'll need to adjust the spacing of the holes. Drill out the holes in the hub to accommodate the 4-40 machine screws.

Use two 4-40 × 1/2" machine screws and nuts to attach the servo hub to the underside of the upper leg piece. See Figure 27-13 for a pictorial view of the leg pieces, separate and assembled. Note the extra chamfering added to the corners of the upper leg pieces. This isn't required, but it makes the legs look a bit more streamlined.

CUT THE BASE PIECES

Follow the cutting and drilling guide in Figure 27-14 to prepare the base, middle spar, middle leg servo mount, and the two front leg spacer spars. Be careful when drilling the holes for mounting the servos; these require a fair amount of accuracy.

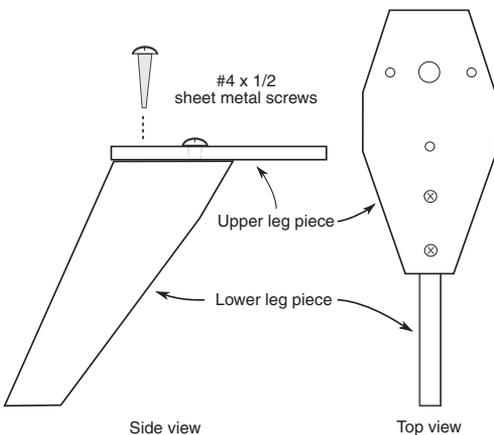


Figure 27-12 The legs are assembled by attaching the upper leg piece to the lower piece, using #4 sheet metal screws.



Figure 27-13 The front and rear legs, as cutout pieces and assembled. The rear legs have a six-arm "star" servo horn attached to them.

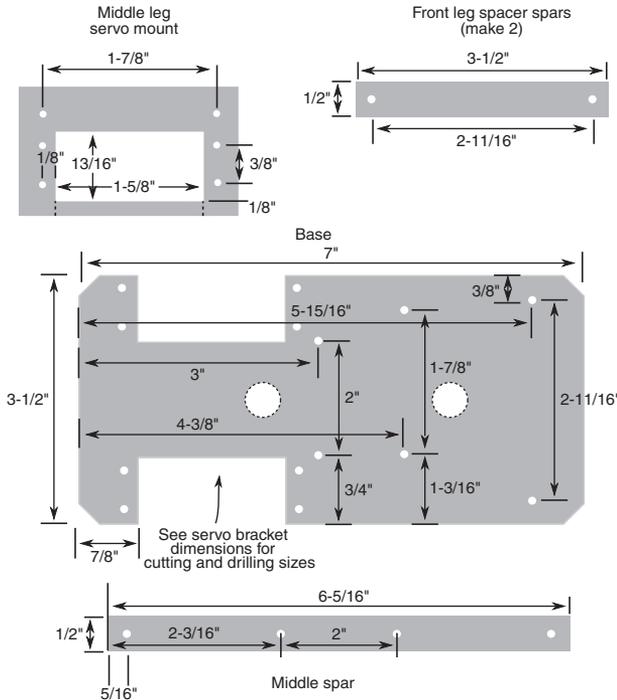


Figure 27-14 Cutting and drilling template for the base pieces for the Hex3Bot. All holes are 1/8" except as noted in the text.

The two large holes down the middle of the base are for passing wires through. The holes can be most any size and shape—1/2" is handy—as long as they don't interfere with the construction of the bot.

Use a 9/16" bit (to accommodate a 6-32 screw) for the following:

- Two holes on the far right of the base
- Two holes on the front leg spacer spars

PREPARE THE LINKAGES



The Hex3Bot uses locking nuts for its moving links. The nuts should not be overtightened on their screws, or the robot may have difficulty in moving; nor should they be too loose. Use a screwdriver and wrench (or nut driver) to tighten the locking nuts so that they just begin to impede free movement of the linkage. Then back off about 1/8 to 1/4 turn.

Using two 1/2"-wide by 0.016"-thick brass strips, cut and drill the four linkages as shown in Figure 27-15. From each of the 12" lengths of brass, cut one strip to 5-1/2" and another to 3-5/8". Use a hacksaw with an 18- or 24-tooth-per-inch blade. Use a file to smooth the cut edges, and file down the sharp corners.

- The longer links connect the front and rear legs together.
- The shorter links connect the middle leg servo to the middle legs.

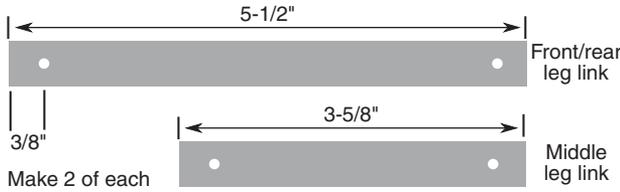


Figure 27-15 Linkages are made from 1/2" brass strips, cut to length. The holes on either end are 1/8". Make two of each length.

BUILD THE FRONT/REAR LEG ASSEMBLY



Prepare the left front/rear leg assembly using one rear leg and one front leg, plus one long linkage strip. Use 4-40 × 3/4" machine screws, nylon spacers, and 4-40 locking nuts, as shown in Figure 27-16. Be sure not to overtighten the locking nuts. Repeat for the right front/rear leg assembly. Make this one in mirror image to the first.

BUILD THE MIDDLE LEG ASSEMBLY



Prepare the middle leg assembly by attaching the legs as shown in Figure 27-17. Use the middle spar, large round servo hub, short linkage strips, and fasteners as noted. You will need to drill out two of the holes in the servo hub to accommodate the screws. The other end of the linkages connects to the large servo hub. Drill out two holes at the 5 and 7 o'clock positions on the hub, as shown.

ASSEMBLE TO COMPLETE THE HEX3BOT

Remember the note about the locking nuts. The nuts should not be overtightened on their screws, or the robot may have difficulty in moving. Use a screwdriver and wrench to tighten the nuts so that they just begin to impede free movement. Then back off about 1/8 to 1/4 turn.

1. Begin with the servo that operates the middle legs. Attach the servo mount to the underside of the base using a pair of 3/8" × 3/8" miniature L brackets and 4-40 × 7/16" machine screws and 4-40 nuts.

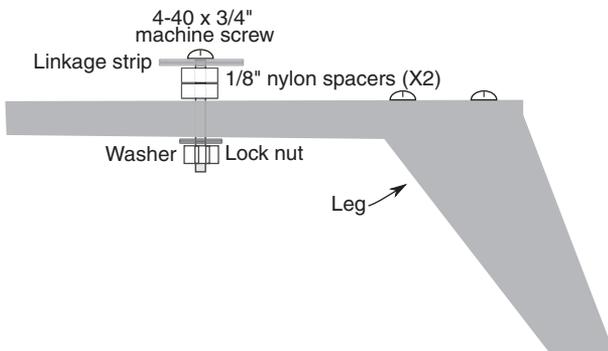


Figure 27-16 Side view of the legs and linkages, showing the hardware for assembly. The locking nut should not be so tight that it prevents the linkage from moving freely.

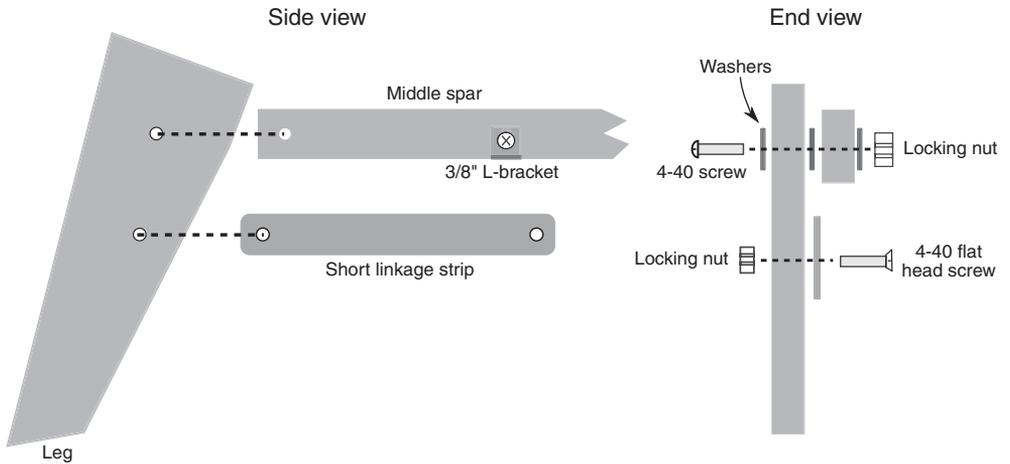
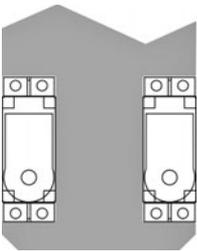


Figure 27-17 Construction detail of the middle leg assembly. Assemble both the right and the left middle legs, making sure they are in mirror image to one another.



2. Fasten the servo into the mount using two or four 4-40 × 1/2" machine screws and 4-40 nuts. (When using only two screws, mount them on opposite corners of the servo.)

3. Mount two servos to the base using 4-40 × 1/2" machine screws and nuts. The driveshaft of the motors should be closest to the rear of the base and should be on the side opposite the middle leg servo.

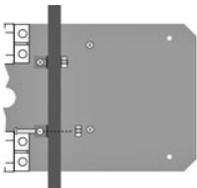


4. Attach the middle leg assembly to the top of the base using a pair of 3/8" × 3/8" miniature L brackets and 4-40 × 7/16" machine screws and 4-40 nuts.

5. Connect each servo to your microcontroller (or similar circuit), and apply 1.5-millisecond pulses. This centers the servo to its middle, or neutral, position.

6. Attach the large round servo horn to the middle leg servo using the screw included.

7. Attach the left leg assembly to the left servo motor using the screw included with the servo.



8. Use a 6/32 × 1-1/2" machine screw, washers, and 6/32 locking nut to attach the front left leg to the base. Place the two front leg spacer bars between the bottom of the leg piece and the base. Don't forget the washers, as shown in Figure 27-18.

9. Repeat steps 7 and 8 for the right leg.

Refer to Figure 27-19 for an underside view of the completed Hex3Bot.

OPERATING THE HEX3BOT

```
101010
010101
101010
010101
```

See the RBB Online Support site (refer to Appendix A) for working microcontroller code for operating the Hex3Bot.

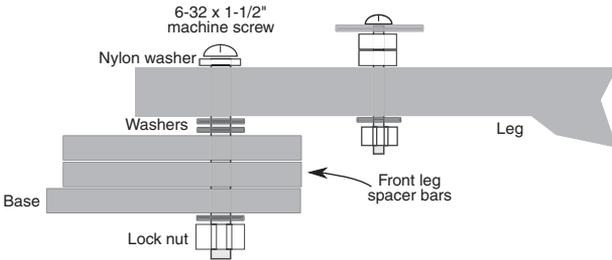


Figure 27-18 Assembly detail for attaching the front legs to the Hex3Bot base. Assemble both the right and the left front legs in the same fashion.



Figure 27-19 Under view of the completed Hex3Bot, showing how the middle legs are articulated using the servo.

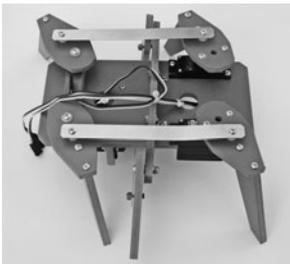
Refer again to Figure 27-10 for the walking gait of the Hex3Bot. You need a microcontroller or serial servo controller to run the Hex3Bot's servos. The walking sequence is straight-forward:

1. Begin by rotating the middle leg servo so that one of the center legs (let's say the left leg) is on the ground. Experiment to find a rotation angle that lifts the front/rear legs on the same side completely off the ground.



In the prototype Hex3Bot this angle was about 60° from center—the exact angle on your Hex3Bot may vary slightly depending on the particular measurements of the parts you built. You should expect some variance, and be ready to compensate for it in your program code.

2. Sweep the right front/rear legs back—toward the end with the servos—to push the robot forward. (It will actually move forward and turn to the left at the same time; this “waddling” is common with this form of hexabot).
3. Sweep the left front/rear legs forward to move them into position.



4. Rotate the middle leg servo so that the opposite (right) leg is on the ground and the right front/rear legs are lifted.
5. Sweep the left front/rear legs backward to propel the robot forward.
6. Sweep the right front/rear legs forward to move them into position.
7. Repeat these steps to continue moving. Optionally, you can add a speech synthesizer that says, “I’m walkin’ here, I’m walkin’ here!”

- To make the robot go in reverse, have the front/rear legs push in the opposite direction to that described previously.
- To make the robot turn, have only one side of the front/rear legs push.

MOUNTING ELECTRONICS AND ACCESSORIES

There's very little room on the top of the Hex3Bot for mounting batteries or electronics. To make space you'll probably want to add a second deck, separating the deck with the base using standoffs or risers of some type or another. The standoffs should be at least 1-1/4" long, in order to clear the mechanical movement of the legs.



In lieu of standoffs you can construct your own risers using a suitably long machine screw, some aquarium tubing, and a couple of nuts. Start by drilling matching holes in the Hex3Bot base and the second deck. For whatever riser length you want, add $3/4$ " to accommodate the thickness of the base and deck, plus extra for securing a nut on top. In the case of a $1-3/4$ " riser, select a $2-1/2$ " screw, and cut the aquarium tubing to $1-3/4$ ".

Creating X-Y Servo Joints

With two servos (any size) and some basic parts, you can construct an articulated X-Y joint for use in multilegged robots, as well as servo turrets, gripper arms and wrists, and a variety of other mechanisms.

There are literally hundreds of ways to construct an X-Y joint using R/C servos. The one outlined here (maybe it's #113, who knows?) is designed to be easy to construct using ordinary shop tools. It lacks some features such as supporting shafts on the side opposite the servo, which is useful when lifting heavier weight.

But given the typical robotic application, you'll probably find this method more than adequate. (Of course, feel free to modify the design to incorporate any improvements you see fit. A couple of suggestions follow the how-to construction guide.

CUTTING THE PARTS



The sizing dimensions that follow assume you're using standard-size servos. If you're using larger or smaller servos, you'll want to adjust the dimensions accordingly.

Using 6mm expanded PVC sheet or $1/4$ " aircraft plywood, begin by cutting a strip measuring $2-1/2$ " \times $6-1/2$ ". This is enough to construct a pair of X-Y joints. If you need six pairs—you're building a six-legged hexapod, for example—cut three of these strips.

As shown in Figure 27-20, next cut up the strip into four $1-1/2$ " segments. Depending on the kerf width of your saw, the last segment may be wider than $1-1/2$ "; just cut it to the proper size. You will end up with:

- Two servo mounts. You'll mount the servos to these.
- Two solid plates. You'll mount servo horns to these.

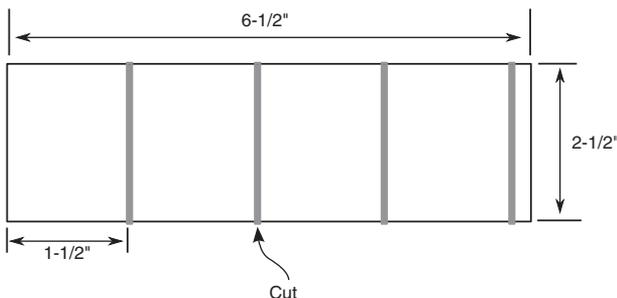


Figure 27-20 Cutting guide for making the X-Y joint bracket pieces. Use a piece of $1/4$ "-thick wood or plastic that's $2-1/2$ " wide, and cut into $1-1/2$ " slices.

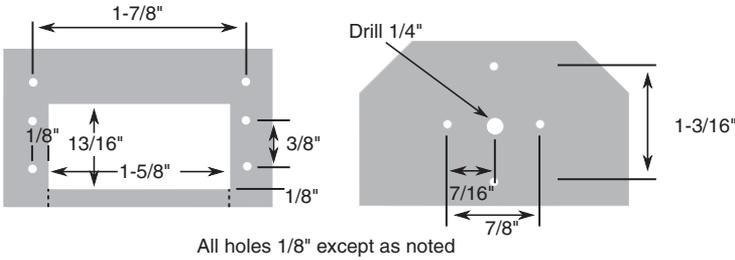


Figure 27-21 Final cutting and drilling guide for the X-Y joint bracket and solid plate. Cut on the dotted lines if making an enclosed rectangle proves difficult. If this is done, be sure to always mount the servo using all four screws.

Finally, on the two solid plates, chamfer (lop off) about $1/2''$ on two corners, as shown. You can cut the corners off or grind them down if you have a power sander.

DRILLING THE PARTS

Using the drill patterns in Figure 27-21, drill the mounting holes in the plates and servo mounts. Important! The dimensions on the plates indicate candidate holes for matching with the servo horns you're using. Different brands (e.g., Hitec and Futaba) have horns with different hole spacing around their circumference. You need only drill two holes opposite one another to match two of the holes in the servo horn.

All holes are $1/8''$, except the hole in the center of the solid plate, which is $1/4''$.



Printable drill patterns are available at the RBB Online Support site; see Appendix A for details. The patterns are in PDF format. Open the PDF and print out the pattern on regular paper. Cut the pattern pieces to size, then tape or lightly glue (use a glue stick) the pattern onto the wood or plastic pieces. Also on the RBB Online Support site are links to precut parts for building the X-Y joints detailed here.

Exercise care when drilling the four servo mounting holes. These need to be accurately placed.

CUTTING THE SERVO POCKET

Also on the drill pattern is a cutout guide for the servo. Using a coping saw with a general woodworking blade, carefully cut out the pocket for the servo, being careful not to take away too much material close to the mounting holes. You may wish to cut away too little, then come back with some rough sandpaper or a woodworking file and remove any extra so your servo will fit.

CONSTRUCTING THE X-Y JOINT

Refer to Figure 27-22 as you follow these steps to construct an X-Y joint. You will need two servos: one that mounts from the body (or some other part of your robot) and connects to the solid plate and the other that attaches to the servo mount. In addition to the servos, you need the following hardware for *each* X-Y joint:

- Six 4-40 \times $1/2''$ machine screws and nuts
- Two #4 \times $1/2''$ wood screws

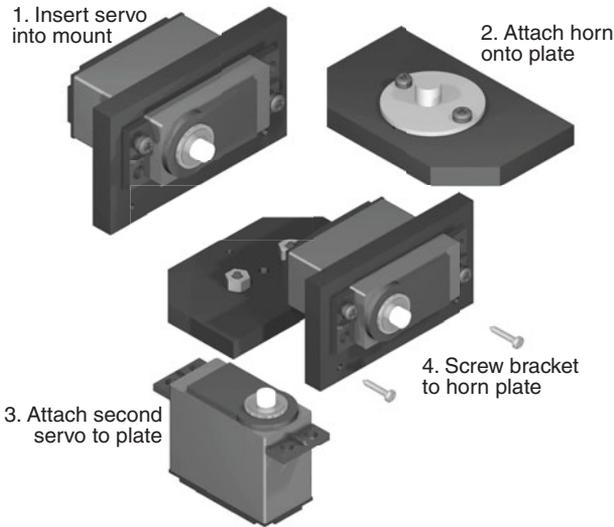


Figure 27-22 Basic assembly steps for constructing a generic X-Y joint with servos.



Figure 27-23 Complete X-Y joint, with servos.

1. Using 4-40 \times 1/2" machine screws and nuts, secure a servo into the servo mount. Use four machine screws for each servo.
2. Center the servo shaft to its midpoint. (You can temporarily mount a long-arm-style horn to the servo so that you can turn the servo shaft. Rotate the shaft slowly so that the internal gears are not damaged.)
3. Using 4-40 \times 1/2" machine screws and nuts, attach a disc-shaped servo horn onto a solid plate. You'll probably need to drill out the hole on the servo horn in order to pass the screw through.
4. Dry-fit the edge of the solid plate without the chamfered corners against the two holes of the servo mount. Use a pencil or small nail to mark the position of the holes in the edge of the plate.
5. Using a center punch or nail set, create a shallow hole at the marks you scribed in step 4. Be sure the hole is in the middle of the thickness of the material. Be careful to avoid cracking off any of the material. You just need a light, shallow hole as a screw starter.
6. Attach the servo from your robot body (or arm, wrist, or whatever) to the servo horn you mounted in step 3. Insert the horn retaining screw (it comes with the servo) through the plate and into the servo output shaft. Do not over-tighten.
7. Complete the X-Y joint by using the two 4-40 wood screws to secure the servo mount to the solid plate.

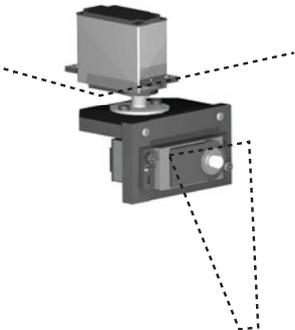


Figure 27-23 shows the completed X-Y joint.

USING AS A LEG JOINT

You can create a two-axis leg joint for a walking robot by mounting a servo from the underside (or topside) of your robot chassis and connecting

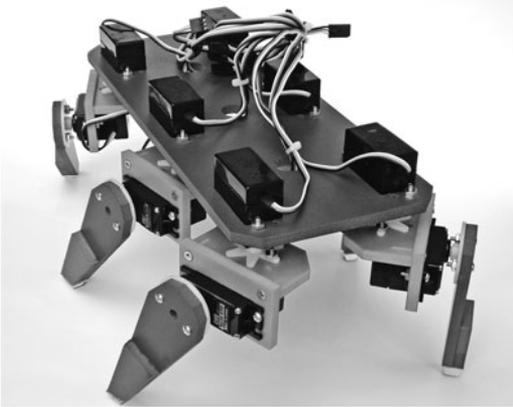


Figure 27-24 Completed Hex12Bot, a hexapod robot with six individually manipulated legs. Each leg uses two servos, and therefore has 2 DOF.

the servo to the horn on the solid plate. See the section “Build a 12-Servo Hexapod” later in this chapter that uses these joints for legs.

USING AS A SENSOR TURRET

Turrets use motors to scan a sensor, or a group of sensors, back and forth. A single turret made with a standard R/C servo can provide near-180° coverage, sweeping back and forth looking for objects nearby and far. An X-Y joint allows you to sweep the sensor(s) horizontally, as well as tilt up and down. Mount a servo pointing upward on the robot chassis, and attach it to the horn on the solid plate. This provides the side-to-side motion of the turret.

Then attach a bracket of your own design to the other servo. This provides the up-and-down tilt motion. Sensors tend to be very lightweight, so they don’t need a heavy-duty turret structure. That saves on expense and weight.

USING AS PART OF A WRIST

By stringing X and Y segments together you can create a 3-DOF robotic wrist, similar to the joints of the human wrist and forearm, which can be used with a variety of grippers. See Chapter 29 for ideas on making robotic grippers.

Bonus Project: Build a 12-Servo Hexapod

Using the X-Y joints described in the previous section you can construct a hexapod with 2-DOF legs. The robot uses 12 servos, 2 per leg. Besides the X-Y joints already discussed, and the servos (of course), you need only six simple leg pieces and a base to mount everything on.

Figure 27-24 shows a completed 12-servo hexapod, the Hex12Bot, constructed with six X-Y joints. The robot measures 11-1/2” by 7”, and stands over 6”. Full construction plans, with cutting and drilling templates, programming examples, and parts lists and sources, is provided as a free bonus project on the RBB Online Support site.

Experimenting with Robotic Arms

Robots without arms can't "reach out and touch someone." Arms extend the reach of robots and make them more like humans. For all the extra capabilities arms provide a robot, it's interesting that they aren't difficult to build. Your arm designs can be used for factory-style, stationary "pick-and-place" robots, or they can be attached to a mobile robot as an appendage.

This chapter deals with the concept and design theory of robotic arms. Incidentally, when we speak of arms, we will usually mean just the arm mechanism minus the hand (also called the gripper). Chapter 29, "Experimenting with Robotic Grippers," talks about how to construct robotic hands and how you can add them to arms to make a complete, functioning appendage.

The Human Arm

Take a close look at your own arms for a moment. You'll quickly notice several important points. First, your arms are amazingly adept mechanisms, no doubt about it. Each arm has two major joints: the shoulder and the elbow (the wrist, as far as robotics is concerned, is usually considered part of the gripper mechanism). Your shoulder can move in two planes, both up and down and back and forth. The elbow joint is capable of moving in two planes as well: back and forth and up and down.

The joints in your arm, and your ability to move them, are called *degrees of freedom* (DOF). Your shoulder provides 2 DOF in itself: shoulder rotation and shoulder flexion/extension (shoulder flexion is motion upward to the front; shoulder extension is motion downward to the rear). The elbow joint adds a third and fourth degree of freedom: elbow flexion/extension and elbow rotation.

Robotic arms also have degrees of freedom. But instead of muscles, tendons, ball-and-socket joints, and bones, robot arms are made from metal, plastic, wood, motors, solenoids, gears,



Figure 28-1 Robotic arm with three joints that each provide a degree of freedom.

pulleys, and a variety of other mechanical components. Some robot arms provide but 1 DOF; others provide 3, 4, and even 5 separate DOF.

Degrees of Freedom in a Typical Robotic Arm

Human anatomy offers an inexact comparison with robotic arm systems. Our bone and muscle structure provides movement in a way that is seldom duplicated in robot arms. For example, out of simplicity, most robotic arms don't use a ball joint for the shoulder. In the human arm, this joint provides multiple degrees of freedom. In the robot version, shoulder motion is duplicated with two and sometimes three separate joints.

Figure 28-1 shows a representative robotic arm—it happens to be attached to a mobile base, which provides yet an additional degree of freedom, though many robot arms are stationary. The arm provides 3 degrees of freedom; additional freedoms are provided by the “wrist” and gripper.

- DOF #1 is rotation of the arm at its base. The base may rotate up to 360°, depending on design, though it's more common to limit rotation to about 180°. This represents the mechanical extents of the typical R/C servo, which is often used to move the joints in a low-cost robotic arm.
- DOF #2 and DOF #3 are essentially the shoulder and elbow joints, respectively. Together they allow the arm to lift and lower, and to position its gripper at the height and distance to grasp an object in front of it.

As noted, the wrist mechanism of this arm adds 3 DOF of its own. These include two joints that form the wrist: it moves the gripper up and down and rotates it to position the gripper fingers to best grasp the object. The third DOF is the finger mechanism.

All the joints of the arm, including those in the gripper section, work in tandem to locate, grasp, and move objects. By pitching the shoulder, elbow, and wrist joints forward, the arm can reach down and pick up something on the ground.

Arm Types

Robot arms are classified by the shape of the area that the end of the arm (where the gripper is) can reach. This accessible area is called the *work envelope*. For simplicity's sake, the

work envelope does not take into consideration motion by the robot's body, just the arm mechanics.

The human arm has a nearly spherical work envelope. We can reach just about anything, as long as it is within arm's length, within the inside of about three-quarters of a sphere. Imagine being inside a hollowed-out orange. You stand by one edge. When you reach out, you can touch the inside walls of about three-quarters of the orange peel.

In a robot, such a robot arm would be said to have revolute coordinates. The three other main robot arm designs are polar coordinate, cylindrical coordinate, and cartesian coordinate. You'll note that there are 3 DOF in all four basic types of arm designs. Let's take a closer look at each one.

REVOLUTE COORDINATE

Revolute coordinate arms, such as the one depicted in Figure 28-2, are modeled after the human arm, so they have many of the same capabilities. The typical robotic design is somewhat different, however, because of the complexity of the human shoulder joint.

The shoulder joint of the robotic arm is really two different mechanisms. Shoulder rotation is accomplished by spinning the arm at its base, almost as if the arm were mounted on a record player turntable. Shoulder flexion and extension are accomplished by tilting the upper arm member backward and forward. Elbow flexion/extension works just as it does in the human arm. It moves the forearm up and down. Revolute coordinate arms are a favorite design choice for hobby robots. They provide a great deal of flexibility, and, besides, they actually *look* like arms. See later in this chapter for details on how to construct a revolute coordinate arm.

POLAR COORDINATE

The work envelope of the *polar coordinate* arm is the shape of a half sphere. Next to the revolute coordinate design, polar coordinate arms are the most flexible in terms of the ability

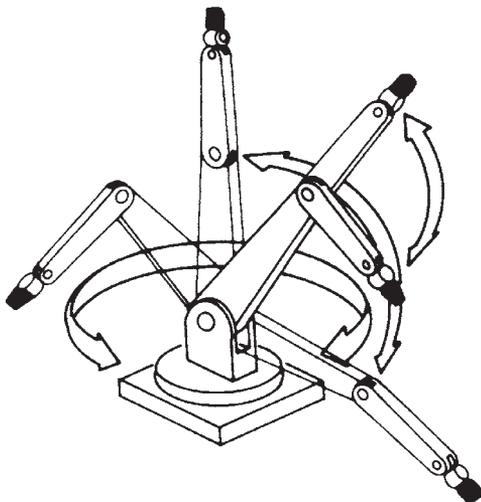


Figure 28-2 Revolute coordinate arm. This is among the most common of arm designs for robotics.

to grasp a variety of objects scattered about the robot. Figure 28-3 shows a polar coordinate arm and its various degrees of freedom.

A turntable rotates the entire arm, just as it does in a revolute coordinate arm. This function is akin to shoulder rotation. The polar coordinate arm lacks a means for flexing or bending its shoulder, however. The second degree of freedom is the elbow joint, which moves the forearm up and down. The third degree of freedom is accomplished by varying the reach of the forearm. An “inner” forearm extends or retracts to bring the gripper closer to or farther away from the robot. Without the inner forearm, the arm would only be able to grasp objects laid out in a finite two-dimensional circle in front of it.

The polar coordinate arm is often used in factory robots and finds its greatest application as a stationary device. It can also be mounted to a mobile robot for increased flexibility.

CYLINDRICAL COORDINATE

The *cylindrical coordinate* arm looks a little like a robotic forklift. Its work envelope resembles a thick cylinder, hence its name. Shoulder rotation is accomplished by a revolving base, as in revolute and polar coordinate arms.

The forearm is attached to an elevator-like lift mechanism, as depicted in Figure 28-4. The forearm moves up and down this column to grasp objects at various heights. To allow the arm to reach objects in three-dimensional space, the forearm is outfitted with an extension mechanism, similar to the one found in a polar coordinate arm.

CARTESIAN COORDINATE

The work envelope of a cartesian coordinate arm (Figure 28-5) resembles a box. It is the arm most unlike the human arm and that least resembles the other three arm types. It has no rotating parts. The base consists of a conveyer belt-like track. The track moves the elevator column (like the one in a cylindrical coordinate arm) back and forth. The forearm moves up and down the column and has an inner arm that extends the reach closer to or farther away from the robot.

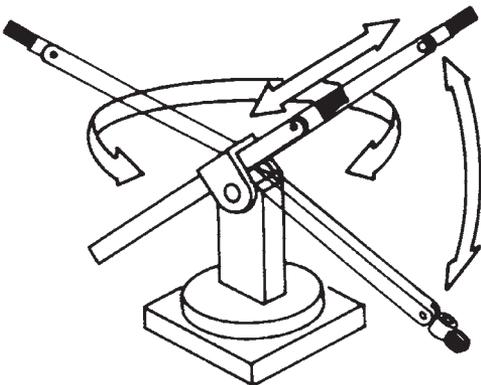


Figure 28-3 Polar coordinate arms are typical in manufacturing and industrial environments.

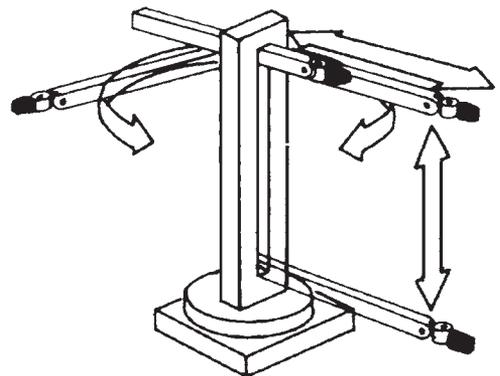


Figure 28-4 A cylindrical coordinate arm. It works like a forklift and can rotate on its base.

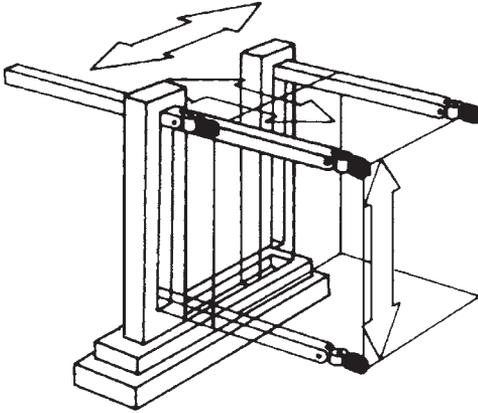


Figure 28-5 A cartesian coordinate arm.

Actuation Techniques

There are three general ways to move the joints in a robot arm: electrical, hydraulic, and pneumatic. For small robots, electrical actuation is the method of choice; it's the least expensive and the easiest to implement. I'll briefly describe the other two for the sake of completeness.

ELECTRICAL ACTUATION

Electrical actuation is done with motors, solenoids, and other electromechanical devices. It is the most common and easiest arm type to implement. The motors for elbow flexion/extension, as well as the motors for the gripper mechanism, can be placed in or near the base. Cables, chains, or belts connect the motors to the joints they serve.

Electrical actuation doesn't always have to be via an electromechanical device such as a motor or solenoid. Other types of electrically induced actuation are possible using a variety of technologies. One of particular interest to hobby robot builders is shape memory alloy, or SMA, as discussed in Chapter 25, "Robot Movement with Shape Memory Alloy." When electrical current is applied to the wire, it contracts.

HYDRAULIC ACTUATION

Hydraulic actuation uses oil-reservoir pressure cylinders, similar to the kind used in earth-moving equipment and automobile brake systems. The fluid is noncorrosive and inhibits rust: both are the immediate ruin of any hydraulic system. Though water can be used in a hydraulic system, if the parts are made of metal they will no doubt eventually suffer from rust, corrosion, or damage by water deposits. For a simple homebrew robot, however, a water-based hydraulic system using plastic parts is a viable alternative.

PNEUMATIC ACTUATION

Pneumatic actuation is similar to hydraulic, except that pressurized air is used instead of oil or fluid (the air often has a small amount of oil mixed in it for lubrication purposes). Both hydraulic and pneumatic systems provide greater power than electrical actuation, but they are

more difficult to use. In addition to the actuation cylinders themselves, a pump is required to pressurize the air or oil, and valves are used to control the retraction or extension of the cylinders. For the best results, you need a holding tank to stabilize the pressurization.

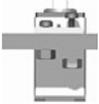
Build a Robotic Wrist

Sometimes a whole arm isn't required. All the bot really needs is a hand (gripper) on the end of a wrist-like mechanism that provides basic movement. The human wrist has 3 degrees of freedom: it can twist (rotate) on the forearm, it can bend up and down, and it can rock from side to side. You can add some or all of these degrees of freedom to a robotic hand.

You can create a basic 3-DOF wrist using the same X and Y components described in Chapter 27, "Build Robots with Legs"; see the section "Building X-Y Servo Joints." Plans are provided for making compact parts for creating X-Y (pan/tilt, up-down/left-right) joints. By stringing together a sequence of these joints you can build a compact wrist that provides a remarkable amount of dexterity.



You can attach a variety of grippers to the wrist mechanism described here. See Chapter 29, "Experimenting with Robotic Grippers," for some ideas and hands-on examples. Look especially at the easy-to-build gripper described in "Tool Clamp Gripper" in Chapter 29.



Parts you'll need:

- X-Y pieces cut to size, according to the plans in Chapter 27. (See step 1, below.)
- 16 4-40 × 1/2" machine screws and nuts
- 4 #6 × 3/4" wood screws



1. Begin by cutting two pairs of X and Y pieces, as described in Chapter 27. To complete the wrist you'll need to make one more servo bracket. You'll end up with a total of five pieces: three servo brackets and two solid plates.



2. Using 4-40 × 1/2" machine screws and nuts, secure each of the three servos onto servo mounts. Observe the orientation of the servo output shaft, along with the remaining two mounting holes of the mount. When viewed from the front, the shaft should be on the right and the two mounting holes should be on the bottom. When finished, you will have three servos in three mounts.



3. Using 4-40 × 1/2" machine screws and nuts, attach a disc-shaped servo horn on a solid plate. Repeat for the second plate. When finished, you will have two servo horns on two plates.



4. Take one mounted servo and orient as shown (holes facing the top). Center the servo shaft to its midpoint. (You can temporarily mount a long-arm-style horn to the servo so that you can turn the servo shaft. Rotate the shaft slowly so that the internal gears are not damaged.) Note the orientation of the servo horn mount. The chamfered side should be on the left. Insert the horn retaining screw (it comes with the servo), through the horn mount, and into the servo output shaft. Do not overtighten. This is the *wrist rotator*.



5. Repeat step 4 (including centering the servo), except orient the servo so that the holes in the mount are on the bottom, rather than the top. Attach the solid plate so that the chamfered side faces bottom. This is the *wrist flexor/extender*.

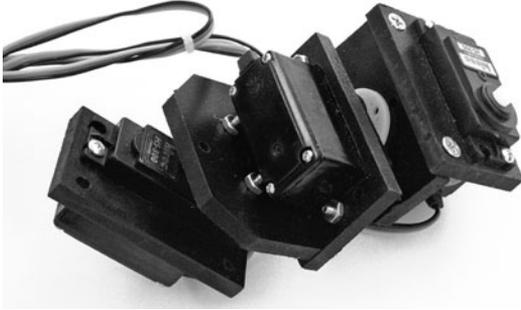


Figure 28-6 The completed wrist mechanism provides 3 degrees of freedom. It can be constructed using ordinary plastic or wood parts, and three R/C servo motors.



6. Attach the wrist rotator and the wrist flexor/extender subassemblies using two #6 x 3/4" self-tapping screws. The screws are flat-head and should be tightened so that they are fairly flush with the surface of the mount.



The completed wrist is shown in Figure 28-6. When controlling the servos via a microcontroller or other means, be careful to limit the movement to the mechanical extents of the pieces. For the most part, wrists don't have massive extents of movement—a little goes a long way.

Build a Functional Revolute Coordinate Arm

You can build your own revolute coordinate robotic arm with about \$20 in parts—not including servo motors. The size of the arm is scalable; the prototype shown in Figure 28-7 stands about 8" tall and has a reach (without gripper) of over 9-1/2". It provides 4 degrees of freedom, including a rotating base and shoulder, elbow, and wrist joints.



Figure 28-7 Completed robotic revolute coordinate arm. Attach the arm to a base or robot and add a gripper mechanism.

You will need four standard-size R/C servo motors. For added strength, select servos with a torque of no less than 45 oz-in; 65 to 85 oz-in is preferable. On most servos, the higher the torque, the slower the servo turns, so bear this in mind when selecting the motors for your arm.

To complete the arm, you'll need:

- 1 12" length of 3/8" U-channel extruded aluminum
- 1 3" ball bearing turntable
- 1 small piece (about 12" × 8") 1/4" aircraft-grade plywood or PVC plastic
- 1 pair of small 3/4" corner angle brackets
- Small assortment of 4-40 machine screws and nuts

MAKE THE SERVO MOUNTS

The robotic arm uses two types of servo mounts and one general-purpose (and optional) solid plate for attaching to the shaft of the servo. The smaller mount and plate are described in the "Building X-Y Servo Joints" section of Chapter 27, "Building Robots with Legs." You need two small mounts and one solid plate. You also need one slightly larger version of the servo mount, shown in Figure 28-8. It's virtually identical to the mount detailed in Chapter 27, except the top flange is larger and the spacing for the two holes is different. You can construct these parts with 1/4" aircraft-grade plywood or PVC plastic.



The spacing of the two holes in the top flange of the larger mount depend on the horns that come with your servos. The dimensions shown are for the large circular horn that comes with most Futaba and Futaba-style servos. If you use a different servo and horn, the hole spacing may be slightly different; adjust the spacing of the holes in the top flange accordingly.

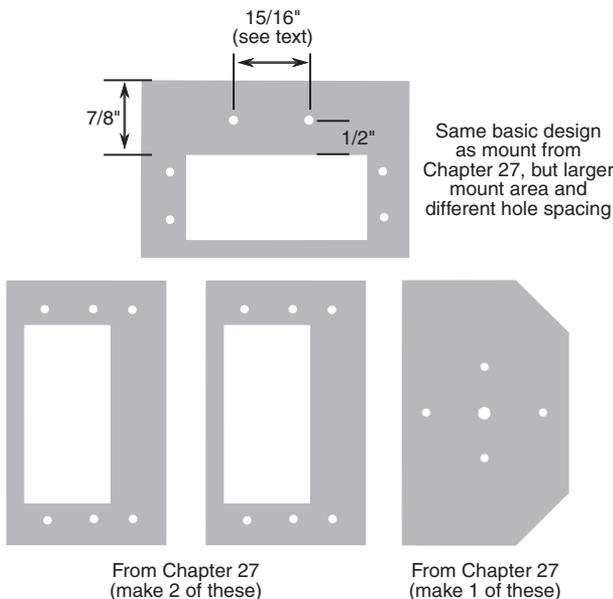


Figure 28-8 Cutting and layout guides for the servo mounts used in the revolute coordinate arm. Basic construction of the mounts is detailed in Chapter 27.

BUILD THE BASE

The base of the arm consists of a 3" ball bearing turntable ("lazy Susan") mounted on 4-1/2"-diameter round plastic, which serves as a bottom plate (this plate can be square if that's easier for you to cut). You can get this size turntable at better stocked hardware stores and home improvement outlets, or via mail order. Search for 3" *lazy susan*—I found mine for under \$2 sold through Amazon.



1. Begin by drilling four holes to mount the turntable. Use the turntable itself to mark the holes in the center of the base. Once marked, drill the holes with a 1/8" bit. Don't mount the turntable just yet.



2. Using the largest horn that comes with your servos, find two holes opposite one another on the horn that are about 3/4" to 1-1/4" apart. Use a 1/8" bit to drill these out.



3. Place the horn in the center of the 4" bottom plate. Insert two 4-40 × 1/2" machine screws through the horn and plate.

4. On the other side of the bottom plate, thread the screws through a pair of 3/4" metal corner angle brackets. Secure the screws with 4-40 nuts.

5. Flip the bottom plate over again and mount the turntable using at least two screws on opposite corners. To insert the screws you'll need to spin the turntable so that all the flanges (top and bottom) are exposed.



These construction plans don't include mounting the fourth servo under the bottom plate. I'm leaving that up to you, based on where you want to put your arm. You can mount the arm on top of a mobile robot (it should be at least 8" to 10" in size), or you can build a stationary arm that operates within a confined space.

MOUNT THE JOINT SERVOS



You need to attach three servos into their mounts. Two sizes of mounts are used, as mentioned in "Building the Servo Mounts": two regular-size mounts and one with a larger flange. Secure the servos to their mounts using at least two screws, one on each corner of the motor.

Each of the servos needs to be oriented a certain way within its mount. Refer to Figure 28-9 for a guide. When viewed from the front of the servo, the output shaft of the motor should be located as shown. Secure the servo within its mount using at least two 4-40 × 1/2" machine screws and nuts.

MOUNT SHOULDER SERVO TO BASE

Use the servo in the larger of the two mounts for the shoulder joint. This servo attaches to the two 3/4" corner angle brackets using 4-40 × 1/2" machine screws and nuts. The servo is

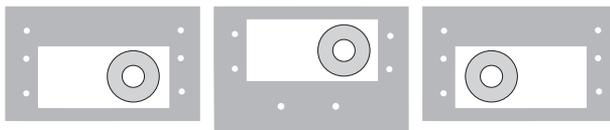


Figure 28-9 Orientation of the servo motors in the mounts. Attach the motors to the mounts with their output shafts oriented as shown.

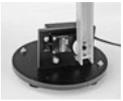
oriented so that its mounting flange points toward the base. The backside of the motor is over the centerline of the base.

CONSTRUCT UPPER ARM

Make an upper arm (the part of the arm between shoulder and elbow) from a 6" length of 3/8" aluminum U-channel. Cut with a hacksaw, and file down the ends to remove any burrs. Then:

1. Using the small round or double-arm horn that comes with your servos, use a 1/8" bit to drill out two holes on opposite sides for mounting screws (see Figure 28-10). Once drilled out, mark three holes on each end of the upper arm: the two holes you just drilled and the center hole for the servo screw. Place the marks on the flat ("bottom") of the U in the U-channel.
2. Use a 1/8" bit to drill each of the marked holes (for best results, use a center punch to begin the hole—see Chapter 11, "Working with Metal," for more tips and tricks).
3. Use a 1/4" bit to drill out the center hole on each end of the upper arm. This makes the hole large enough for you to insert the servo horn screw.
4. Use 4-40 × 3/8" or 4-40 × 1/2" machine screws and nuts to secure the servo horns to the upper arm.

ATTACH UPPER ARM TO SHOULDER



Manually (and slowly) rotate the shoulder joint motor so that it is at its approximate center position. Use the screw supplied with the servo to attach the shoulder joint motor to the bottom of the upper arm.

Point the arm piece straight up when attaching the motor, so that the joint will rotate equally in both directions. Don't overtighten the screw or else it might strip the output shaft of the servo.

ATTACH SERVOS TO FOREARM

Using 4-40 × 1/2" machine screws and nuts, attach the two servos (the ones in the smaller mounts) to the forearm. Be sure to orient the servos so that one faces the "front" of the arm, and the other the back, as shown.

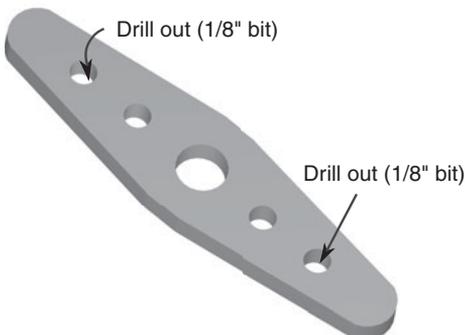


Figure 28-10 Drill out the holes in a servo horn for the base servo motor. For best results, pick holes in the horn that are about 1" apart.



- The output shaft of the servo that faces the front (same side as the open part of the U-channel) should be on the right side.
- The output shaft of the servo that faces the back (flat part of the U-channel) should be on the left side. Refer to the completed pictures of the arm to see how the servos are oriented.

MOUNT UPPER ARM TO ELBOW



Manually (and slowly) rotate the shoulder joint motor so that it is at its approximate center position. Use the screw supplied with the servo to attach the elbow joint motor to the top of the upper arm. Don't overtighten the screw or else it might strip the output shaft of the servo. As you did with the shoulder motor, mount the arm piece pointing straight up. That way the joint will rotate equally in both directions.

ATTACH WRIST

Attach the solid plate to the wrist joint motor. From this plate you can mount a gripper to the arm. (If your gripper already has a means to attach directly to the servo motor you can dispense with the solid plate.)

See Figure 28-11 for another view of the finished arm. Note that you'll need to extend the length of the wires from at least the two servos mounted on the forearm in order to reach the control electronics. You can get servo extensions in 12" lengths (and longer) at hobby stores specializing in radio control parts, and through online and mail-order outlets that sell servos.

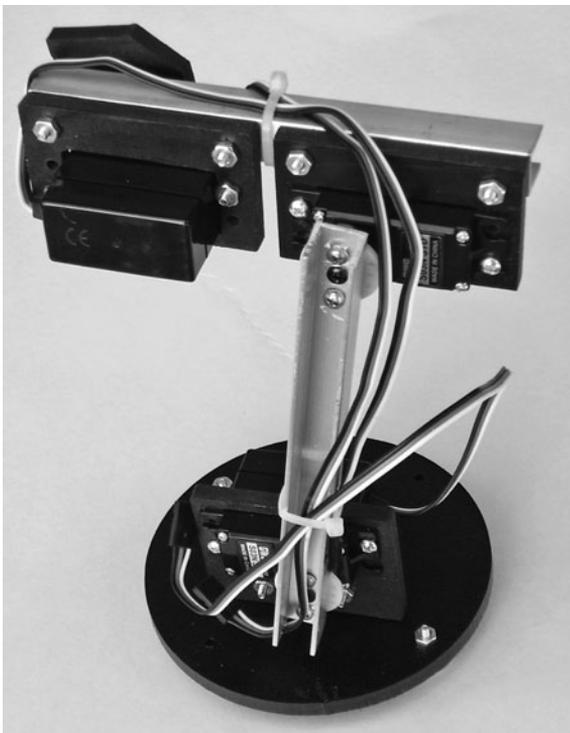


Figure 28-11 The reverse side of the revolute coordinate arm.

Build a Robotic Arm from a Kit

The revolute coordinate arm described in the preceding section is designed around simple components that can be constructed using ordinary tools and limited precision. If you want something more elaborate but don't want to build it from scratch, you can choose from among a number of specialty robotic arm kits or use custom construction set parts

ARMS FROM ROBOTICS CONSTRUCTION SET PARTS

Think of these as Erector Sets for grown-ups—they're parts of various shapes and sizes, with holes already drilled in them for attaching to R/C servos, motors, and other robotic components. One popular robotics construction set, the TETRIX Robotic Design System from online educational outlet Pitsco. Several sets are available, each with a different assortment of parts. You can also purchase individual parts as needed. Most construction sets are made with stamped aluminum, though they may also contain plastic pieces.

Even with the variety of parts that come in the typical robotics construction set, you're not limited to using just those components. Unless there's a restriction otherwise (you're building a robot for a school competition, for example), you should feel free to add your own bits and pieces.

For example, you can combine a TETRIX U-channel with your own homebrew servo mounts to make two of the joints (shoulder, elbow) of a robot arm. The TETRIX U-channels measure 1-1/4" in each of the three faces and are available in different lengths. I've used the 160mm (about 6-1/4") channel in the upper arm mechanism shown in Figure 28-12.

ARMS FROM SPECIALTY KITS

Thanks to the popularity of amateur and educational robotics, there are plenty of specialty kits that are constructed just for the purpose of building a robot arm. Some are low-cost and meant for casual experimentation using manual switch control, but others, like the arm in Figure 28-13 from Lynxmotion, use R/C servo motors and microcontrollers to precisely posi-

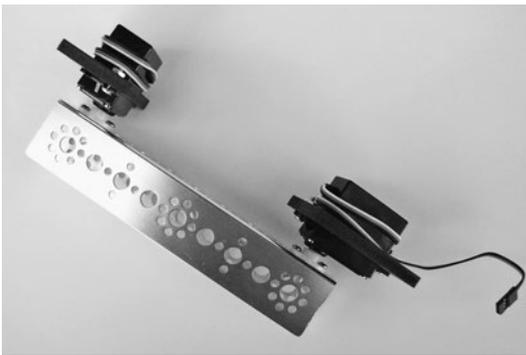


Figure 28-12 Two servo motors, with homebrew mounts, attached to a TETRIX U-channel beam. Use a combination of ready-made and home-built parts for constructing arm arms.



Figure 28-13 Robot arm kit, shown with control electronics and gripper. (Photo courtesy Lynxmotion.)

tion the arm components. Such arms may or may not come with a gripper; you can add one you made yourself or get a gripper kit and build it from premade parts.

This particular arm has 4 degrees of freedom:

- Shoulder rotation, using a rotating base. This allows the arm to pick up objects in a circular arc of about 90° to each side.
- Shoulder flexion/extension, using a servo mounted near the base.
- Elbow flexion/extension, using a servo mounted off a yoke from the shoulder joint.
- Wrist flexion/extension, using a servo mounted at the end of the forearm.

In almost all cases, arm kits are designed for use with standard- and mini-size R/C servos. The kits are available with and without servo motors, in case you already have some in your parts bin. If you supply your own servos, take note of any special requirements that are listed in the documentation for the arm. Some or all of the servos may need to have a minimum torque, for example.

Experimenting with Robotic Grippers

Arms aren't much good without hands. In the robotics world, hands are usually called *grippers* (also *end effectors*) because the word more closely describes their function. Few robotic hands can manipulate objects with the fine motor control of a human hand; they simply grasp or grip the object, hence the name gripper. Never sticklers for semantics, I'll use the terms *hands* and *grippers* interchangeably.

Gripper designs are numerous, and no single design is ideal for all applications. Each gripper technique has unique advantages over the others, and you must fit the gripper to the application at, er, hand. This chapter outlines a number of useful gripper designs you can use for your robots. Most are fairly easy to build; some even make use of inexpensive plastic toys. The gripper designs encompass just the finger or grasping mechanisms.

Concept of the Basic Gripper

In the world of robotics there are hundreds of ways to make a gripper. The designs tend to be application-specific: like Captain Hook in *Peter Pan*, a metal hook offers advantages a normal hand cannot. A perfectly useful gripper might be designed for a single task, like collecting Ping-Pong balls or picking up chess pieces—whatever the robot is made to do.

Figure 29-1 shows a typical robotic gripper, depicted in three different states: all the way open, halfway open, and all the way closed. This style of gripper uses “fingers” that stay parallel as they open and close. The design allows the gripper to apply even pressure on either side of an object and close in on the object without pushing it away. The mechanism is not difficult, but you can see that it adds a bit of complexity to the construction of the gripper.

To be useful, most grippers are attached to the end of an arm. If the arm is on a wheelbase, the bot can steer around the room looking for things to pick up and examine. By orienting the gripper vertically or horizontally via a rotating wrist mechanism, the robot can grasp all manner of objects.

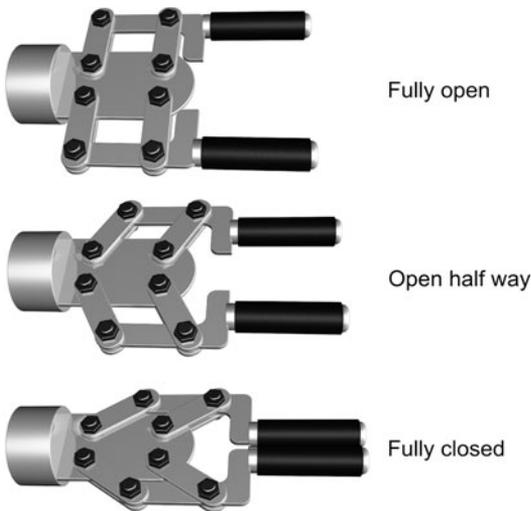


Figure 29-1 Robotic gripper in three states: all the way open, closed, and midpoint. In each case the “fingers” of the gripper remain parallel.

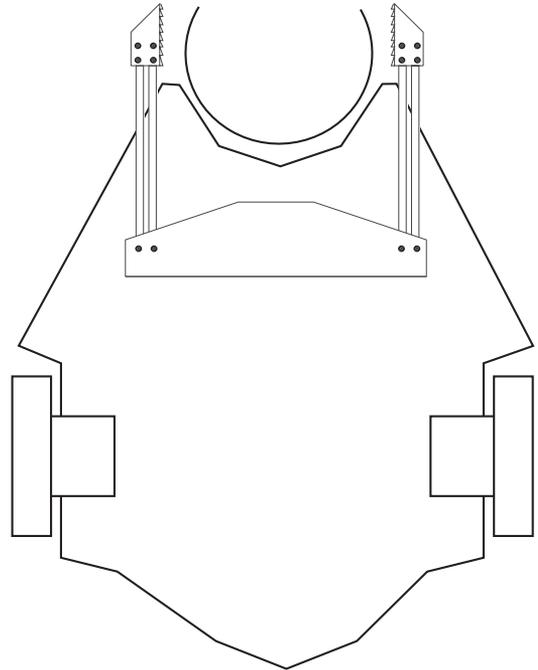


Figure 29-2 Grippers may be mounted directly on the base of a robot and used in specialty applications, such as collecting and holding balls, cans, or eggs.

There’s no rule that says the gripper must be attached to an arm; it can be connected directly to the robot itself. Figure 29-2 shows a “two-finger” gripper at the front of a robot base that collects balls. A cup indentation at the front of the robot serves to capture the ball, and a gripper holds the ball in place as the bot steers around the room.

Two-Pincher Gripper

The two-pincher gripper consists of two movable fingers, somewhat like the claw of a lobster. The steps for constructing several models are described in this section.

BASIC MODEL

For ease of construction, the basic two-pincher gripper is made from extra Erector set parts (the components from a similar construction kit toy may also be used). Cut two metal girders to 4-1/2” (since this is a standard Erector set size, you may not have to do any cutting). Cut a length of angle girder to 3-1/2”. Use 4-40 or 6-32 × 1/2” machine screws and nuts to make two pivoting joints. Cut two 3” lengths and mount them (see Figure 29-3). Nibble the corner off both pieces to prevent the two from touching one another. Nibble or cut through two or three holes on one end to make a slot. As illustrated in Figure 29-4, use 4-40 or 6-32 × 1/2” machine screws and nuts to make pivoting joints in the fingers.

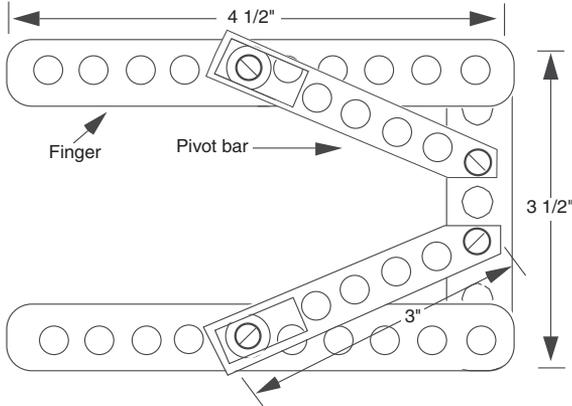


Figure 29-3 Construction detail of the basic two-pincher gripper, made with Erector set (or similar) parts.

The basic gripper is finished. You can actuate it in a number of ways. One way is to mount a small eyelet between the two pivot joints on the angle girder. Thread two small cables or wire through the eyelet and attach the cables. Connect the other end of the cables to a solenoid or a motor shaft. Use a light-compression spring to force the fingers apart when the solenoid or motor is not activated.

You can add pads to the fingers by using the corner braces included in most Erector set kits and then attaching weather stripping or rubber feet to the brace. The finished gripper should look like the one depicted in Figure 29-5.

ADVANCED MODEL

You can use a readily available plastic toy and convert it into a useful two-pincher gripper for your robot arm. The toy is a plastic “extension arm” with the pincher claw on one end and a hand gripper on the other (see Figure 29-6). To close the pincher, you pull on the hand gripper. The contraption is inexpensive—usually under \$10—and it is available from many online toy stores.

Chop off the gripper 3” below the wrist. You’ll cut through an aluminum cable. Now cut off another 1-1/2” of tubing—just the arm, but not the cable. File off the arm tube until it’s straight, then fashion a 1-1/2” length of 3/4”-diameter dowel to fit into the rectangular arm. Drill a hole for the cable to go through. The cable is off-centered because it attaches to the pull mechanism in the gripper, so allow for this in the hole. Place the cable through the hole, push the dowel at least 1/2” into the arm, and then drill two small mounting holes to keep the dowel in place (see Figure 29-7). Use 6-32 × 3/4” machine screws and nuts to secure the pieces.

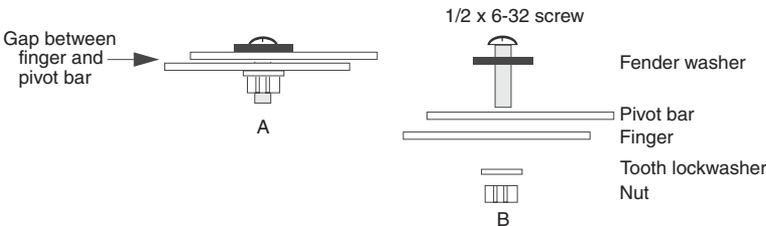


Figure 29-4 Hardware assembly detail of the pivot bar and fingers of the two-pincher gripper: (A) assembled sliding joint; (B) exploded view.

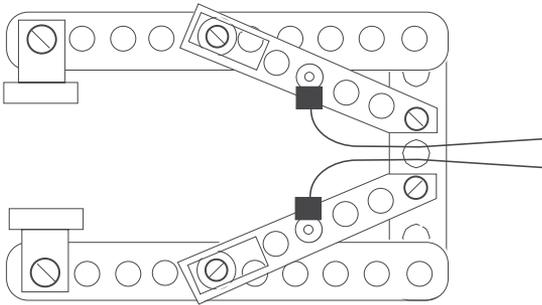


Figure 29-5 The finished two-pincher gripper, with fingertip pads and actuating cables.



Figure 29-6 A commercially available plastic two-pincher robot arm and claw toy. The gripper can be salvaged for use in your own designs.

You can now use the dowel to mount the gripper on an arm assembly. You can use a small 3/4" U-bolt or flatten one end of the dowel and attach it directly to the arm. The gripper opens and closes with only a 1/2" pull. Attach the end of the cable to a clevis (available at hobby stores), then connect that to a servo horn (Figure 29-8).

CONSTRUCTING PARALLEL FINGER GRIPPERS

Figures 29-9 through 29-12 show another approach to constructing two-pincher grippers. By adding a second rail to the fingers and allowing a pivot for both, the fingertips remain parallel to one another as the fingers open and close. You can employ several actuation techniques with such a gripper. Only basic plans are provided here to give you an idea of how these grippers work. Feel free to experiment to come up with unique designs of your own.

Tool Clamp Gripper

If figure if someone else has gone to the trouble and expense of making a product that just happens to work in robot projects, the least we can do is take advantage of their kindness!

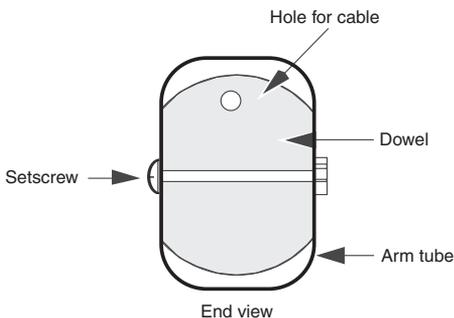


Figure 29-7 Assembly detail for the claw gripper and wooden dowel. Drill a hole for the actuating cable to pass through.

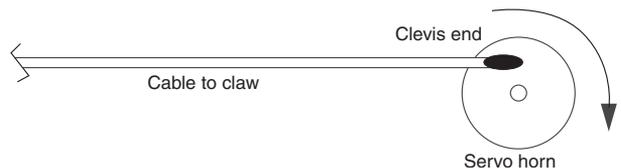


Figure 29-8 One method for actuating the gripper: Attach a solid aluminum cable from the claw to a clevis end (available at hobby stores) and servo horn. An R/C motor turns the horn, pulling or pushing the cable and opening or closing the gripper.

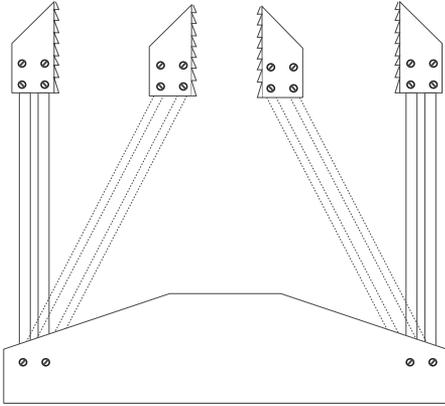


Figure 29-9 Adding a second rail to the fingers and allowing the points to freely pivot causes the fingertips to remain parallel to one another.

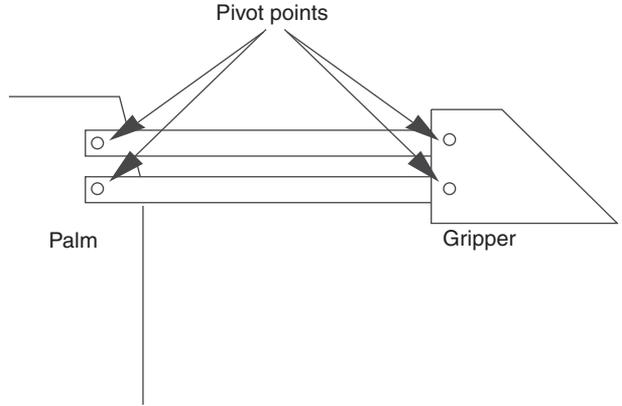


Figure 29-10 Close-up detail of the dual-rail finger system. Note the pivot points.

In this project you use an ordinary plastic tool clamp, available at dollar stores and discount tool outlets, as a motorized robotic gripper. An R/C servo motor opens and closes the clamp. This gripper is simple to build and surprisingly strong.

Plastic tool clamps come in a variety of shapes and sizes. You want one that's as close as possible to the clamp in Figure 29-13. The clamp measures 5" in overall length and is about

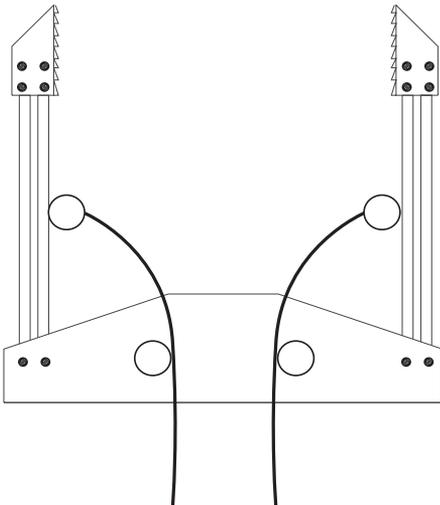


Figure 29-11 A way to actuate the gripper. Attach cables to the fingers and pull the cables with a servo motor. Fit a tension spring (a thin metal strip) along the fingers and palm to open the fingers when power is removed from the motor.

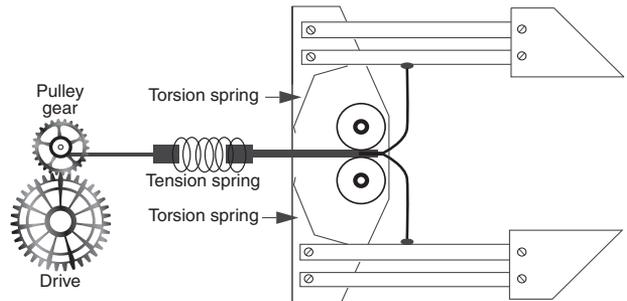


Figure 29-12 Actuation detail of a two-pincher gripper using a motor and gears. The tension spring prevents undue pressure on the object being grasped. The torsion springs are thin pieces of brass that act to reopen the gripper fingers when the motor shuts off.



Figure 29-13 Small plastic tool clamp used in the tool clamp gripper.

1-1/2" wide when the clamp is closed. The clamp is fitted with a plastic locking mechanism, which you'll remove as part of the construction steps outlined as follows.

CUT OUT GRIPPER MOUNT

Using 6mm expanded PVC or 1/4" hobby or aircraft plywood, cut the mount of the gripper as detailed in Figure 29-14. The only true dimension-sensitive cut is the inside of the servo mount.

This cut is best made by drilling a hole at one corner, then threading a thin (woodworking) coping saw blade through the hole. Attach the blade to the coping saw, and carefully cut the rectangular hole for the servo. Be sure not to cut away too much, or you won't have room for the servo mounting holes.



An alternative is to saw a pocket for the servo by cutting at the dotted lines, as noted in Figure 29-14. This kind of cut can weaken the mount because there's supporting material on just three sides rather than four. But all or most of the strength will return when the servo is mounted in the pocket. Use all four servo mounting screws in this case.

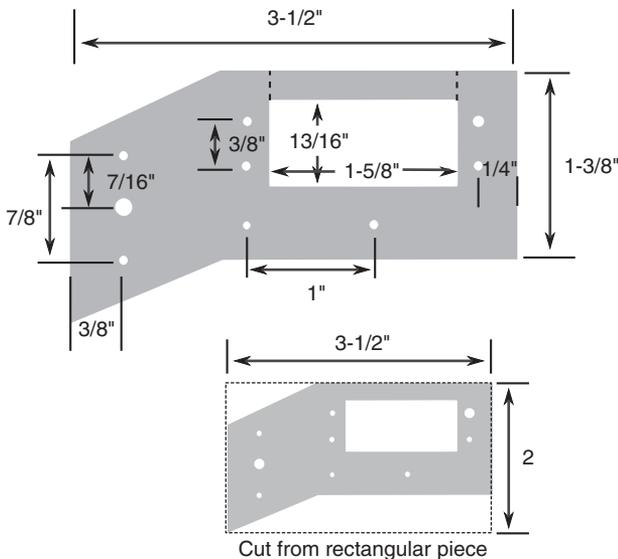


Figure 29-14 Cutting and drilling layout for the mount for both R/C servo motor and tool clamp.

ATTACH SERVO TO THE MOUNT

Using 4-40 \times 1/2" machine screws and nuts, attach a standard-size servo to the gripper mount, as shown in Figure 29-15. The angled portion of the mount should face toward the right. The servo shaft should be closest to the angled portion of the mount.

If you'd like to add a side-to-side wrist joint to the gripper, attach a double-arm servo horn (it should come with the servo) to the angled portion of the mount, also using 4-40 \times 1/2" machine screws and nuts. The nuts should be on the "top" of the mount (same side as the servo output shaft), and the horn should be on the bottom.

PREPARE AND MOUNT THE CLAMP

Prepare the clamp by first removing the plastic locking mechanism. I've found the best way is to simply use brute force: start by releasing the lock and opening the clamp all the way. Using a pair of heavy-duty pliers, grip the locking mechanism and literally tear it out. If any pieces of the locking mechanism remain inside, nudge them out with a small flat-bladed screwdriver.



You may ruin the first clamp you try to modify this way, so better get two at the store just in case. They're not that expensive, and, besides, in many instances they sell a pair of clamps in one package. That's how I've always bought them.

Drill three 1/8"-diameter holes, as noted in Figure 29-16. On one handle you drill two holes; the spacing must match the holes in the mount. The other hole is for the servo linkage, and its exact position isn't critical. Make sure you drill all the holes in the approximate middle of the clamp handles.

Drilling complete:

1. Attach the clamp to the gripper mount using 4-40 \times 3/4" machine screws and nuts.
2. Attach a double-arm horn or adjustable arm horn to the servo. (When using a double-arm horn, cut off the opposite arm.)
3. Using your fingers, slowly center the gripper servo to its midpoint.

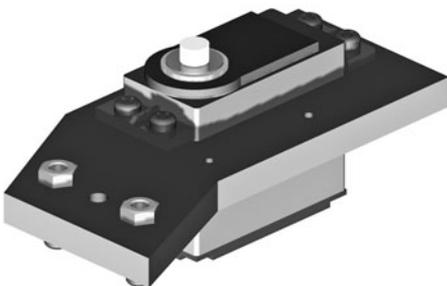


Figure 29-15 R/C servo motor attached to the mount. Note the orientation of the output shaft of the servo motor.

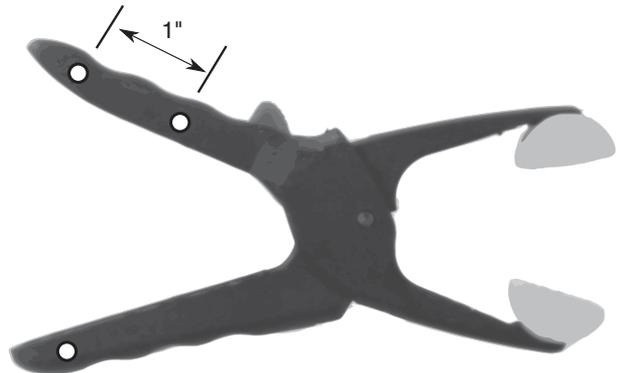


Figure 29-16 Drill three holes for mounting the tool clamp and for attaching the activating rod from the servo motor.



Figure 29-17 Finished tool clamp gripper, showing the activating rod between the clamp and the R/C servo motor.

4. Cut a 1" (approximate) length of solid music (also called piano) wire, available at the hardware store. Actually, I just use a piece of wire cut off from an extra-large metal baby safety pin—you can get a dozen for just a dollar or two at the local discount store. A pair of lineman's pliers easily cuts the wire.
5. Using pliers, bend 1/4" or so at the ends of the wire.
6. Place one bent end into a hole in the servo horn and the other into the empty hole you drilled in the clamp handle.
7. Once you're sure the wire is the right length (move the servo back and forth a few times), use a pair of pliers to cinch the wire securely in place. You don't need or want to clamp the wire shut, just close it up a bit so it won't easily fall out.

The completed tool clamp gripper is shown in Figure 29-17. At the base of the gripper is another double-arm servo horn, for attaching to a servo that's mounted on the robot. This servo lets the gripper scan back and forth for its prey. You don't need to use this servo horn if you're planning on bolting the gripper directly to your robot.

On the Web: More Gripper Plans

Visit the RBB Online Support site (see Appendix A) for bonus gripper plans, including a solenoid-driven "clapper" two-pincher gripper, fingers that open and close using a homemade worm gear system, and ideas on how to construct flexible fingers that have a human-like *compliant* grasp.

This page intentionally left blank

Robot Electronics

This page intentionally left blank

Building Robot Electronics—the Basics

In previous chapters you learned all about the mechanics of robots, including their construction, motors and wheels, and power systems. In this chapter you'll discover the electronics that endow bots with the appearance of life. In this and future chapters you'll find out ways to use modern (but still inexpensive) advances in electronics to create fully programmable robots able to truly think on their own. It's an exciting endeavor, so let's get started.

FYI

If you're absolutely new to electronics you may wish to first read the lessons in *My First Robot* (see Appendix A, "RBB Online Support," for details). You'll learn the very basic concepts of electricity and electronics as you build a simple robot pet that explores its surroundings by touch.

Tools for Electronics You Should Have

Compared to mechanical construction, you need relatively few tools to build the electronic centerpieces of your robots. You can always spend lots and lots of cash for all kinds of testing equipment and specialized electronic gear, but what follows are the basics that will get you started.

MULTIMETER

A *multimeter*, also called a *volt-ohm meter*, *VOM*, or *multitester*, is used to test voltage levels and the resistance of circuits—among other things. This moderately priced tool (see Figure 30-1) is the basic prerequisite for working with electronic circuits of any kind. If you don't already own a multimeter, you should seriously consider buying one. The cost is minimal, considering its usefulness.

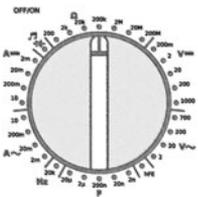


Figure 30-1 A digital multimeter checks resistance, voltage, and current. This model also performs simple checks of a number of common electronic components, including capacitors, diodes, and transistors.

There are many multimeters on the market today. A meter of intermediate features and quality is more than adequate. Meters are available at RadioShack and most online electronics outlets. Shop around and compare features and prices.

Digital or Analog

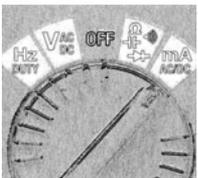
There are two general types of multimeters available today: digital and analog. The difference is not in the kinds of circuits they test, but in how they display the results. *Digital* multimeters, which are now the most common, use a numeric display not unlike a digital clock. *Analog* meters use the more old-fashioned—but still useful—mechanical movement with a needle that points to a set of graduated scales.



Automatic or Manual Ranging

Many multimeters require you to select the range before it can make an accurate measurement. For example, if you are measuring the voltage of a 9-volt battery, you set the range to the setting closest to, but above, 9 volts. With most meters it is the 20- or 50-volt range. Because you need to select the range, there are lots of options on the dial, but in reality these options are really just variations on a theme. The meter is easier to use than it looks.

Autorange meters don't require you to do this, so they are inherently quicker to use. When you want to measure voltage, for example, you set the meter to volts (either AC or DC) and take the reading. The meter displays the results in the read-out panel.



For the sake of completeness, the examples in this book that explain how to use a meter assume you have a manual (*nonautomatic*) ranging model. If yours has automatic ranging, then just skip the step that says to dial in the upper range of your expected measurement.

Accuracy

The accuracy of a meter is the minimum amount of error that can occur when taking a specific measurement. For example, the meter may be accurate to 2000 volts, plus or minus 1 percent. A 1 percent error at the kinds of voltages used in robots—typically, 5 to 12 volts DC—is only 0.1 volts. Not enough to quibble about.

Digital meters have another kind of accuracy: the number of digits in the display determines the maximum resolution of the measurements. Most digital meters have three and a half digits, so they can display a value as small as 0.001—the half digit is a “1” on the left side of the display.

Functions

Digital multimeters vary greatly in the number and type of functions they provide. These functions are selectable by rotating a dial on the front of the meter. At the very least, all standard multimeters let you measure AC and DC voltage, DC amperage, and resistance.

The maximum ratings of the meter when measuring volts, milliamps, and resistance also vary. For most applications, the following maximum ratings are more than adequate:

DC voltage	1000 volts
AC voltage	500 volts
DC amperage	200 milliamps
Resistance	2 megohms (2,000,000 ohms)

One *very important* exception to this is when you are testing the amount of current draw from motors. Many DC motors draw in excess of 200 milliamps.

Better multimeters have a separate DC amperage input that allows readings of up to 10 amps (sometimes as high as 20 amps). If you have the budget for it, I highly recommend that you get a meter with this feature. In most cases, it’s a separate input on the front of the meter and is clearly labeled, like that in Figure 30-2.

The high-amperage input may or may not be fuse-protected; if it is fuse-protected and you exceed the current rating for the input, a fuse will blow and you’ll have to get it replaced. On some inexpensive multimeters the inputs are not fused, and exceeding the maximum ratings could result in permanent damage to the device. So be careful!

Meter Supplies

Multimeters come with a pair of test leads, one black and one red. Each is equipped with a pointed metal probe. The quality of the test leads included with the multimeter is usually

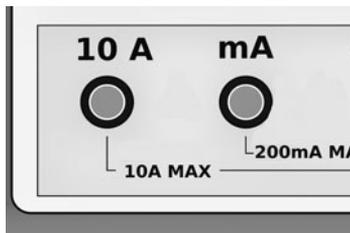


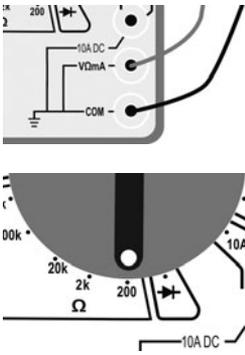
Figure 30-2 For robotics work you’ll want a multimeter with a high amperage (10 amps or higher) input. You use this to easily test the current draw of motors, among other tasks.

minimal, so you may want to purchase a separate set that’s better. The coiled kind is handy; the test leads stretch out to several feet, yet recoil to a manageable length when not in use.

Standard point-tip leads are fine for most routine testing, but some measurements may require that you use a *clip lead*. These attach to the end of the regular test leads and have a spring-loaded clip on the end. You can clip the lead in place so your hands are free to do other things. The clips are insulated with plastic to prevent short circuits.

Using the Meter: The Basics

To use your multimeter, first set it next to whatever circuit you’re testing. Make sure it’s close enough so the test leads reach the circuit without any risk of pulling either the meter or the circuit into your lap. Then start with this:



1. Plug in the test leads; the black lead of your meter goes into the – or COM jack, and the red lead of your meter goes into the + or labeled function jack (the label may be something like VΩmA, which represents the kinds of tests you can do when the lead is inserted into that jack—in this case, voltage (V), resistance (Ω), and low-milliamp (mA) current testing.
2. Check for proper meter operation by doing a *continuity test*. This involves selecting one of the following operating modes. Depending on the features of your meter, choose Resistance (Ω), Diode check, or Continuity. If using Resistance and the meter is not autoranging, choose the lowest Ω setting.

Touch the metal tips of the test probe together. If the meter is functioning properly—the battery is good, the test leads are not broken—the results should be as shown here:

Meter Setting	Good	No Good
Resistance (Ω)	Zero or nearly zero resistance	Infinite* or very high resistance
Diode check†	Good	Infinite value*
Continuity†	Good	Infinite value*

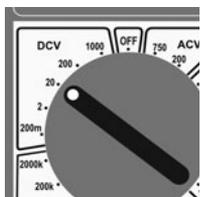
* Meters show infinite value differently, but most display a blinking “1” on the left side of the display.
 † Many digital multimeters provide an audible beep when using the Diode check or Continuity settings, and the continuity test is good.

Once the meter has checked out, select the desired function and range, and apply the leads to the circuit under test.

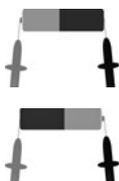
What?!? The meter doesn’t pass its simple continuity test? The reasons could be simple and easy to fix: check that the internal battery is good. Replace as needed. Inspect the test leads for breaks. If the metal prongs of the leads are old, they could be corroded or rusted. Clean or replace. And finally, if the meter is internally fused, the fuse could be blown. Try the spare.

Using the Meter: Testing a Battery

You can use your multimeter to test batteries and other low-voltage DC power sources. Merely as an example to get you started, here are the steps:



1. With the meter on, dial to the DCV (DC volts) setting. If your meter is not autoranging, select a range that is one step higher than the expected voltage; for most robot tasks this will be 20 volts or less.
2. Ensure the test leads are plugged into the proper jacks, as detailed in the previous section “Using the Meter: The Basics.”
3. Touch the black lead to the negative (-) terminal of the battery or pack; touch the red lead to the positive (+) terminal.

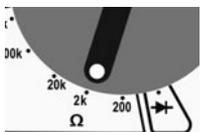


4. Note the value displayed by the meter. If the battery (or pack) is good, the voltage should be close to the expected value—for example, an AA alkaline battery should be about 1.5 volts, give or take 0.1 or 0.2 volts.
5. For grins, switch the test leads so that black goes to the + battery terminal and red goes to -. Note the voltage again; it should now be a negative value, indicating that the polarity of the test connection to the battery is reversed.

Using the Meter: Verifying the Value of a Resistor

Another common use of a multimeter is verifying the value of a resistor. Here’s how.

1. From your parts bin, select any four-banded resistor with the color brown as its third band. This will ensure the resistor is between 100 and 990 ohms. (Resistors and their values and markings are discussed in more detail in Chapter 31.)
2. Refer to the resistor color code table in Appendix D, “Electronics Reference,” to look up the value of the resistor. For example, if the first three color bands are orange-orange-brown, the indicated value of the resistor is 330 Ω.



3. With the meter on, dial to the Ω setting. If your meter is not autoranging, select a range that is one step higher than the expected reading. For instance, if the ranges are 2, 20, 200, 2000, and so forth, select 2000 (2k), as it is one step higher than the expected value of 330 Ω.

4. Ensure the test leads are plugged into the proper jacks, as detailed in the previous section “Using the Meter: The Basics.”
5. With the resistor resting on the table or workbench, apply the test leads to either side of the resistor. Be sure not to touch the metal of the test leads, or else the natural conductivity of your skin will influence the result.
6. Read the value on the meter. See Figure 30-3 for an example:



So the resistor doesn’t read exactly what it should? There’s no cause for alarm. The fourth band on a resistor with four color bands indicates its tolerance, or how far off the printed value it can be from the actual value. A gold band indicates 5 percent tolerance; a silver band, 10 percent tolerance. If the resistor is 330 Ω with 10 percent tolerance, its reading on the multimeter can range from about 300 to 360 Ω.

Meter Safeguards, Good Operating Habits

When using a meter, even on low-voltage circuits, observe these best operating habits:

- Never hold the test probes by their metal part. Not only can this give you a nasty shock when testing AC household current, but your skin resistance can alter the test results.
- On each use of the meter, test its basic operating status with the continuity check detailed previously.

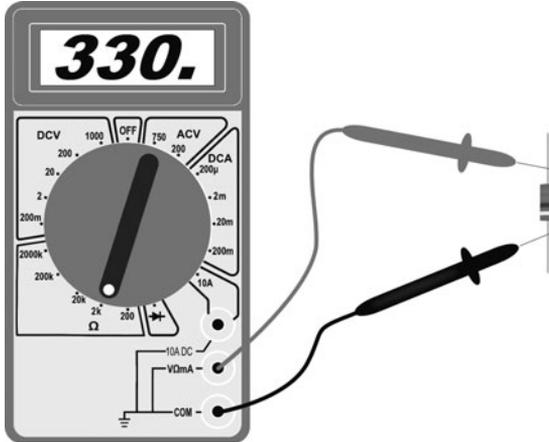


Figure 30-3 Reading the value of a resistor. Be sure not to touch the test probes, or the resistance of your skin will influence the result. When using a meter without autoranging, adjust the range dial just above the expected value.

- None of the projects in this book involve working directly with AC household current—everything is battery-powered. Should you wish to use your multimeter with an AC circuit, be sure to consult the instruction manual that came with the meter for important safety precautions.
- Be very careful when selecting the operating mode of the meter. Never accidentally set the meter to read resistance and then test a voltage source. Your meter could be damaged otherwise or, at the least, burn out a fuse if it is so equipped.
- Turn off the meter when you're done with it. This preserves battery life.

ON THE WEB: USING A LOGIC PROBE

Another handy tool for testing electronic circuits is the logic probe, so called because it verifies signals used in logic circuits (anything that deals with digital 0s and 1s, LOWs and HIGHs). These kinds of circuits include microcontrollers. I've prepared a free "Logic Probe 101" article on the RBB Online Support site (see Appendix A) that provides more information.

SOLDERING PENCIL

You can build robots without owning a *soldering pencil*, but it's darned difficult to do anything more advanced than just put together basic kits. Even if you never plan to make your own circuit boards for your robots, you still need a soldering pencil for basic electronic chores, such as attaching wires to motors. My trusty soldering pencil is shown in Figure 30-4. I've had it for years, and I have built many robots with it.

Note that I've called it a soldering pencil, not the more old-fashioned "soldering iron." For most electronics work these days, you want a slim-line modular soldering pencil. It's smaller than the soldering iron your dad (or granddad) used to build the family's Heathkit color TV kit way back when, and it's designed specifically for the more delicate components common in modern electronics.

Not only do you want a soldering pencil, you really want the kind that lets you change the tip and heating element. Why? These are the parts that, over time, wear out. Rather than buy a whole new soldering pencil, you only have to buy replacement parts.



Figure 30-4 Soldering station with adjustable heat output. This model does not have a temperature indicator, but you can infer it by adjusting the output to where it just begins to melt solder. Remember that different compositions of solder have different melting points. (Photo courtesy Cooper Tools)



Figure 30-5 Trio of the most used tools for working with electronics: a vise or “third hand” (magnifier optional, but handy), flush wire cutters, and wire strippers.

For routine electronic work, you should get a soldering pencil with a 25- to 30-watt heating element. Anything higher may damage electronic components. You can use a 40- or 50-watt element for wiring switches, relays, and power transistors. If you can afford it, opt for a model with a temperature dial. They cost a bit more, but they’re far more flexible.

See the section “How to Solder,” later in this chapter, for a step-by-step guide on soldering.

HAND TOOLS FOR ELECTRONIC CONSTRUCTION

You need just a few hand tools for electronic construction. The ones described here will fill your electronic toolbox quite nicely. All of these tools are inexpensive. The first three on the list, which are among the most used tools for electronic construction, are shown in Figure 30-5.

- *Flush wire cutters*, sometimes referred to as “nippy” cutters. These let you cut off wire flush with the surface of a circuit board.
- *Wire strippers* for smaller-gauge wire. Be sure it can handle between 18- and 26-gauge wire (the higher the number, the smaller the diameter of the wire). Most electronic hookup wire is 22 gauge. The stripper should have a dial that lets you select the gauge of wire you are using.
- *Solder clamp or vise*. The clamp or vise serves as a “third hand,” holding together pieces to be soldered, so you are free to work the soldering pencil and feed the solder. One with a built-in magnifying glass is nice. The ones with simple alligator clips are the least expensive, but they do the job.
- Set of *flat-bladed* and *Phillips screwdrivers*, including sizes #1 and #0.
- Small *needle-nose pliers*.
- *Dental picks*. These are ideal for scraping, cutting, forming, and gouging into things. You can buy these surplus.



Always wear eye protection when using flush wire cutters or, for that matter, any wire-cutting tool. It's quite common for wires to literally shoot out at high velocity when cut. You don't want anything flying into your eyes.

Making Electronic Circuits—the Basics

You have at your disposal numerous ways to construct the electronic circuits for your robots. Those designs involving only switches and batteries and motors can simply be wired together, one to the other, and there is no need to centralize the components in a single place. Options include:

- *Solderless breadboard.* Quickly and easily construct circuits by plugging components into sockets on a plastic board. No soldering necessary. See Chapter 32, “Using Solderless Breadboards” for more information.
- *Permanent circuit board.* Select from among several methods for soldering parts to build a permanent circuit. You can use generic boards that accept common components, or design your own printed circuit board (PCB). See Chapter 33, “Making Circuit Boards” for details.
- *Wire wrapping.* Use a low-cost tool to interconnect electronic components with very fine wire. See how in Chapter 33.

Understanding Wires and Wiring

Almost every electronic circuit uses wire of one kind or another. Wiring is a science all to itself, but we'll concentrate just on three main aspects: insulation, gauge, and conductor type.

INSULATION

Most of the wire used in building robot electronics is insulated with a plastic covering. This keeps one wire from touching another and causing a short circuit. Apart from esoteric aspects about insulation, the most important is its color. Get into the habit of using different-color wiring to denote what it's being used for in your circuit. For example, red wire is often used for the + (positive) battery connection; black wire for the – (negative) connection.

GAUGE

The thickness, or *gauge*, of the wire determines its current-carrying capabilities. Generally, the larger the wire, the more current it can pass without overheating and burning up.

See Appendix D, “Electronic Reference,” for common wire gauges and the maximum accepted current capacity, assuming reasonable wire lengths of 5 feet or less. When you are constructing circuits that carry high currents, be sure to use the proper gauge of wire. Conversely, there's no need to use wires that are way too large for their job. That just makes things bulkier and harder to solder together.

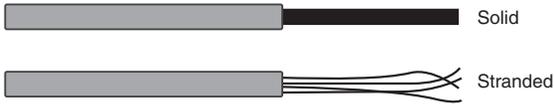


Figure 30-6 Types of hookup wire: solid and stranded. Each has its place. Solid conductor wire is ideal for use with solderless breadboards. Stranded wire is good for general jobs.

CONDUCTOR TYPE

Wire is made of one or more strands of metal, as depicted in Figure 30-6.

- Single-strand wire has just one metal conductor. It's said to be *single stranded*, or *solid conductor*.
- Multiple-strand wire has many conductors and is said to be *stranded*. For any given wire gauge the conductors in stranded wire are small. When banded together, the individual strands make up the gauge of the wire.

Which is better? Both—it depends on the application. Solid wire is commonly used when building circuits using a solderless breadboard. It's cheaper to make, so the wire is less expensive. It's also easier to solder. Stranded wire is more flexible and doesn't break as easily when it's repeatedly flexed. It can also carry a bit more current, per gauge, than solid conductor wire.

How to Solder

Few electronic projects can be assembled without soldering wires together. Soldering sounds and looks simple enough, but there's a bit of science to it. If you are unfamiliar with soldering, or you need a quick refresher course, read the primer on soldering fundamentals provided in this section.

SOLDERING TOOLS YOU NEED

Good soldering means having the proper tools. If you don't have them already, you can purchase them at RadioShack or most any electronics store. Let's review the soldering-related tools you need.



We've already introduced the soldering pencil earlier in the chapter. See Figure 30-7 for a description of the pencil's main parts. Be sure to get one with a three-prong power cord. This provides important grounding of the tool, which is needed for safety.

Stand

If your soldering pencil doesn't come with a *stand*, be sure to get one. They're used to keep the soldering pencil in a safe, upright position. You should *never* simply lay a hot soldering pencil down on your work table.

Sponge

Keep a damp (never dry) sponge by the soldering station. Be sure to keep it wet. Use the sponge to wipe off globs of solder that may remain on the tip. Otherwise, the glob may come

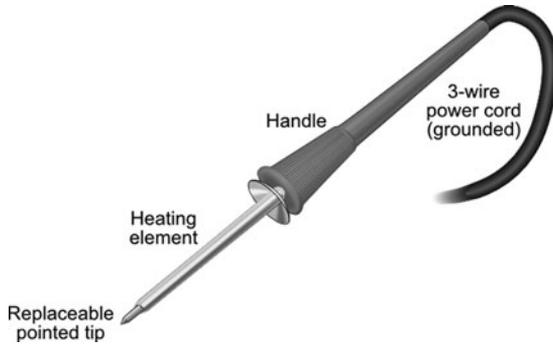


Figure 30-7 Parts of the soldering pencil. Be sure yours is a three-wire grounded model. Don't use an ungrounded soldering tool for electronics work. The replaceable tip is a good feature. When it's worn, you can get a new tip rather than a whole new soldering pencil.

off while you're soldering and ruin the connection. In a pinch, you can substitute a wetted and folded-up paper towel or napkin. Be sure it stays wet—you don't want it to catch fire when you try to clean the soldering tip against it!

Solder

Use only rosin core solder approved for use in electronic circuits. It comes in different thicknesses. For best results, use the thin type (0.050") for most of your electronics. Don't use acid core or silver solder on electronic equipment. (Note: Certain "silver-bearing" solders are available for specialty electronics work, and they are acceptable to use.)

For noncommercial applications, you have the option of lead-bearing or lead-free solders. Both are safe to use when handled properly, but even the lead-free type can be toxic if ingested. Lead-bearing solder is a bit cheaper and easier to work with, as it has a lower melting temperature.

Miscellaneous Soldering Tools

And there are a few more soldering tools worth mentioning:

- A *heat sink* looks like a small metal clamp. It's used to draw heat away from components during soldering.
- Ordinary household *isopropyl alcohol* makes a good, all-around soldering cleaner. After soldering, and when the components and board are cool, clean the board with *rosin flux remover*.
- A *solder vacuum* (or "solder sucker") is a suction device used to remove excess solder. It is often used when desoldering—that is, removing a wire or component from the board, so that you can fix a mistake.

CLEANING THINGS PRIOR TO SOLDERING

Before soldering, make sure all parts of the connection are clean. If you're soldering a component onto a printed circuit board, clean the board first with warm water, a kitchen scouring powder, and a nonmetallic scrubbing pad. Rinse thoroughly, and let dry.

Next, wet a cotton ball with normal household isopropyl alcohol and wipe off all the connection points. Wait a minute for the alcohol to *completely* evaporate, then start soldering.

SETTING THE CORRECT SOLDERING TEMPERATURE

A soldering tool with a temperature control is often referred to as a *soldering station*. If that's what you've got, and you're using traditional lead-bearing solder, dial the station to between 665° and 680°F (352° to 360°C). This provides maximum heat while posing the minimum danger of damage to the electronic components.

If your soldering pencil/station has just the control and lacks a heat readout, initially set it to low. Wait a few minutes for it to heat up, then try one or two test connections. Adjust the heat control so that solder flows onto the connection in under 5 seconds.



To do its job, your soldering tool needs to be significantly hotter than the melting point of solder. Most lead-bearing solders have a melting point of about 362°F (183°C). For lead-free solder, the range is much wider, but in general their melting point is 40° to 70°F higher. Increase the temperature of the soldering tool accordingly.

STEPS FOR SUCCESSFUL SOLDERING

The idea behind successful soldering is to use the soldering tool to heat up the work—whether it is a component lead, a wire, or whatever. You then *apply the solder to the work*. Don't apply solder directly to the soldering pencil. If you take that shortcut, you might end up with a “cold” solder joint. A cold joint doesn't adhere well to the metal surfaces of the part or board, so electrical connection is impaired.

Here are the steps to solder a component onto a circuit board:

1. Use small needle-nose pliers to bend the leads of the component to match the spacing of the holes in the circuit board (see Figure 30-8). Eyeball the correct distance; you'll get more accurate at judging where to place the bends as you gain experience.
2. Insert the component leads through the holes in the circuit board. The component should rest fully against the board, or be very close to it.
3. With your fingers, gently bend the leads of the component to the sides to prevent the component from falling out.
4. Place the board so that the side you are soldering faces you. Apply the tip of the soldering pencil against both the component lead and the “pad” around the hole.

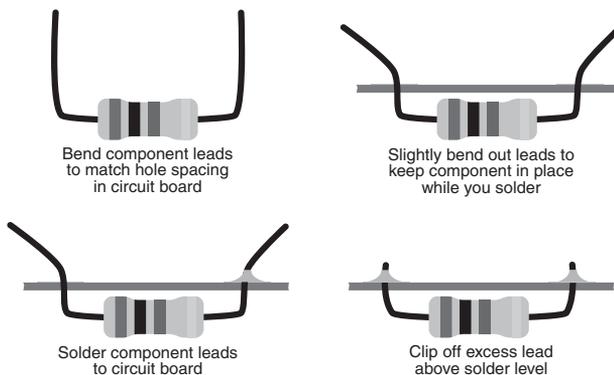


Figure 30-8 Process of soldering a component to a circuit board. Begin by bending the wire leads. Apply solder to one pad and lead at a time. Clip off any excess lead above the solder level. (A short “whisker” jutting out is fine.)

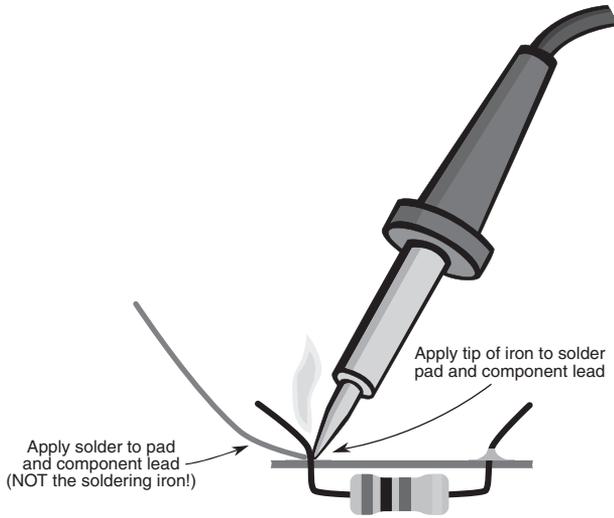


Figure 30-9 Remember to always apply the tip of the soldering tool to the work, not the solder. (Exception: You may apply the solder directly to the tool when *finning* the tip. Do this periodically after wiping off the tip against a damp sponge.)

5. Wait 3 to 5 seconds, and then touch the end of the solder against the opposite side of the solder pad. The solder should begin to flow into the pad and around the component lead (Figure 30-9). Allow just enough solder to flow into the joint, then immediately remove both the solder and soldering pencil.
6. If the solder does not flow, wait a few seconds more and try again. If it's still not flowing, the soldering pencil may not yet be hot enough. Remove the tip from the work, and wait another minute for the soldering pencil to reach proper operating temperature.
7. It takes several seconds for the solder to cool. During this time, be absolutely sure not to disturb the solder joint, or it could be ruined.



Don't apply heat any longer than necessary. Prolonged heat can permanently ruin electronic components. A good rule of thumb is that if the soldering tool is on any one spot for more than 5 seconds, it's too long.

Finishing Up

When soldering on printed circuit boards, you'll need to clip off the excess leads that protrude beyond the solder joint. Use a pair of diagonal or flush cutters for this task. You don't need (or want) to cut off any part of the solder, so don't be too aggressive with the nippers; it's okay if a little stubble of wire still sticks out.

Be sure to protect your eyes when cutting the lead; a bit of metal could fly off and lodge in an eye. Not fun.

Tips for Better Soldering

- If at all possible, you should keep the tool at a 30° to 40° angle for best results. Most tips are beveled for this purpose.
- Apply only as much solder to the joint as is required to coat the lead and circuit board pad. A heavy-handed soldering job may lead to solder bridges, which is when one joint melds with joints around it. That can cause a short circuit.

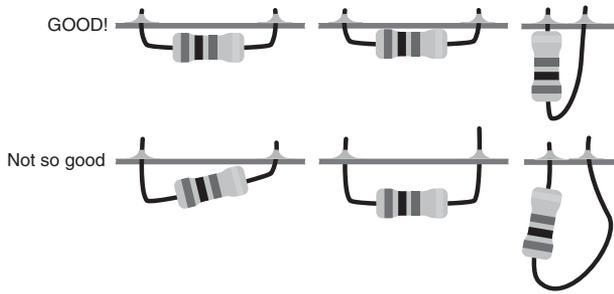


Figure 30-10 Good and not-so-good examples of soldering. When soldering components to a circuit board, be sure the parts are in fully and straight.

- When the joint is complete and has cooled, test it to make sure it is secure. Wiggle the component to see if the joint is solid.
- Be sure to insert the component completely through the circuit board, and check that it isn't crooked (see Figure 30-10). Excessive component lead length can cause short circuits if they touch other exposed parts of the board.

SOLDER TIP MAINTENANCE AND CLEANUP

As the soldering tool comes to temperature, clean off the tip by wiping it against a damp sponge. As you work, periodically repeat this step to keep the tip free of excess solder.

The tip should always be *tinned*, meaning it should have a very light coat of solder on it. Tinning involves cleaning off the tip with the damp sponge, then directly applying a bit of solder to the tip (this is one instance when applying the soldering tool directly to the solder is allowed). Remove any excess solder by wiping again with the damp sponge.

After soldering, let the tool cool down for at least 10 minutes before putting it away.

After many hours of use, the soldering tip will become old, pitted, and deformed. This is a good time to replace the tip. Old or damaged tips impair the transfer of heat, and that can lead to poor soldering joints. Be sure to replace the tip with one made specifically for your soldering tool. Different brands of tips are generally not interchangeable.

Using Headers and Connectors

Most robots are constructed from subsystems that may not be located on the same circuit board. So you need to connect these subsystems together using some kind of wiring system. For very simple connections, you can directly solder wires between boards and other components. But as the electronic systems of your bot get more complex, such direct connections make it harder to experiment.

The solution: Use connectors whenever possible. In this approach, you connect the various subsystems of your robot together using wires that are terminated with a connector of some type or another. The connectors attach to mating pins on each circuit board.

MAKING YOUR OWN MALE CONNECTORS

You don't need fancy cables and cable connectors for your robots. In fact, these can add significant weight to your bot. Instead, use ordinary 20- to 26-gauge wire, terminated with single- or double-row plastic headers.

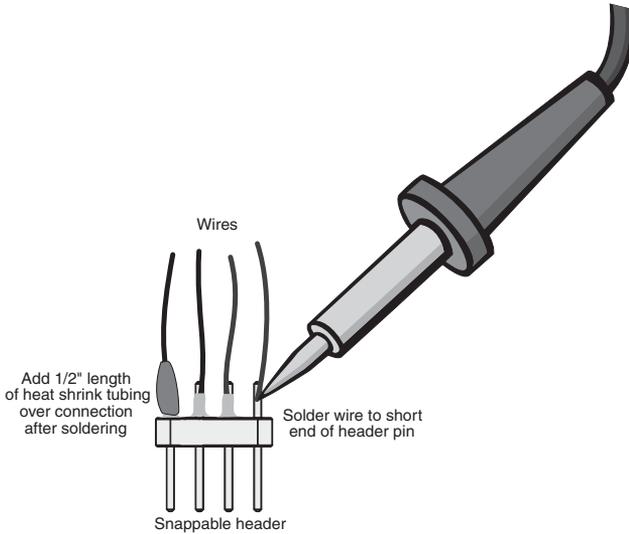


Figure 30-11 Make your own male header connectors by soldering wires to the short end of the header pins. For a professional look use heat shrink tubing over the solder joint.

You can make these yourself using breakaway header pins. You buy them in lengths of 10 or more pins, then break off as many as you need. Solder wires to the underside of the pins. Insert the top side into sockets on your circuit boards. Figure 30-11 shows how. The optional heat shrink tubing is applied by cutting a small piece of the tube and slipping it over the soldered wire. Shrink the tube using a heat gun or hair dryer set on high.

PREPARING FEMALE CONNECTORS

If you're connecting to a circuit board that already uses male pins, you'll need to use a female connector to hook things up. You can make these by purchasing a connector set that's designed for the pins you're plugging into. These vary by the number of pins and the pin spacing, so be sure to get the right set. Most pin headers use 0.100" spacing. There are smaller and larger spacings, but these are obvious just by looking at them.

Referring to Figure 30-12 as a guide, to make a connector prepare the end of a wire by removing about 1/4" to 3/8" of insulation. When using stranded conductor wire, twist the strands together. Insert the end of the wire into a crimp-on connector piece. Secure the wire in the connector by crimping it; a crimping tool made for the job works best. You then insert the connector piece into the plastic shell. The connector snaps into place in the shell.

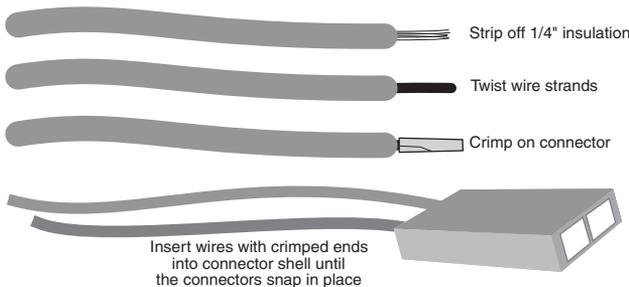


Figure 30-12 Make your own female connectors using a connector shell and crimp on sockets.

BEST CONNECTIONS

Use plastic ties to bundle the wires together. This keeps them all tidy. Or, instead of using individual lengths of insulated wire, use ribbon cable, which is made of many pieces of wire bonded together as a unit.

When making interconnecting cables, cut the wires to length so there is a modest amount of slack between subsystems. You don't want the wire lengths so short that the components are put under stress when you connect them together. But don't go overboard; you also don't want, or need, gobs of excess wire.

Using Clip-on Jumpers

Another kind of cable is used when experimenting with and testing your robot electronic circuits. These are clip-on jumpers. The jumper is made with flexible insulated wire, where both ends have some kind of spring-loaded clip. You attach the clips to wires, components, or another part of the circuit to make temporary connections.

You should get at least one set of clip-on jumpers for routine testing and experimenting. Jumpers are available with three basic kinds of clip-on ends:

- Small alligator clamp, useful for smaller components and wires. Don't use these to attach to an individual pin on an integrated circuit (IC). They're too large and will cause a short.
- Large alligator clamp, good for motors and other, bigger components (the wiring is heavier, too).
- Push-in hooks, ideal for use when connecting to the individual integrated circuit pins.

All three kinds are shown in Figure 30-13.

Good Design Principles

While building circuits for your robots, observe the good design principles described in the following sections, even if the schematic diagrams you are working from don't include them.

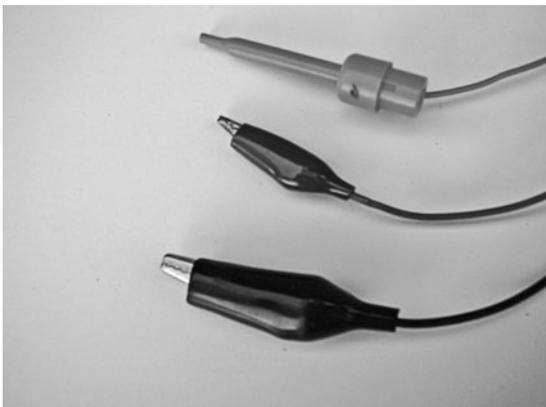


Figure 30-13 Assortment of common jumper cables: hook, small alligator, and large alligator. They come in various colors and lengths. Always use the jumpers that have fully insulated (plastic-covered) clamps. They help prevent short circuits.

FYI

Some of these concepts assume you already know what a resistor and capacitor are. If you don't, then no worries; these topics are covered in Chapter 31. This is just a review of ways to improve the functionality of your robot circuits.

USE PULL-UP/PULL-DOWN RESISTORS

When something is unplugged in your robot, the input voltage might waver back and forth. This can influence the proper functioning of your robot. Use pull-up or pull-down resistors on any circuit inputs where this could be a problem. A common value is 10 k Ω ; 10,000 ohms). In this way, the input always has a “default” state, even if nothing is connected to it.

A pull-up resistor is connected between the input and the + (positive) power supply of the circuit; a pull-down resistor is connected between the input and – (negative or ground), as shown in Figure 30-14.

TIE UNUSED INPUTS LOW

Unless the instructions for a component say otherwise, tie unused inputs to ground to keep them from “floating”—*floating* means an indeterminate voltage state. A floating input can cause the circuit to go into oscillation, rendering it practically unusable.

USE DECOUPLING CAPACITORS

Some electronic components, especially fast-acting logic chips and the venerable LM555 timer IC, generate a lot of electrical noise that can spread through the power supply connections. You can reduce or eliminate this noise by using *decoupling* (also called *bypass*) capacitors, like that in Figure 30-15.

These aren't specific types of capacitors; rather, “decoupling” refers to the job they perform. The value of the capacitor isn't supercritical. I like to use 1 μ F to 10 μ F (1 to 10 microfarad) tantalum electrolytic capacitors positioned between the positive and ground terminals

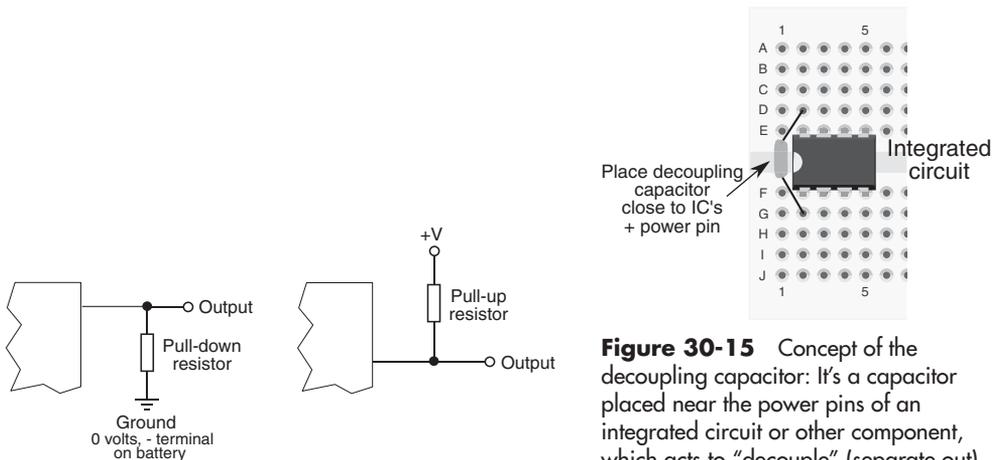


Figure 30-14 Concept of the pull-down and pull-up resistor. A common value of the resistor is 10 k Ω .

Figure 30-15 Concept of the decoupling capacitor: It's a capacitor placed near the power pins of an integrated circuit or other component, which acts to “decouple” (separate out) any noise that may be in the power supply.

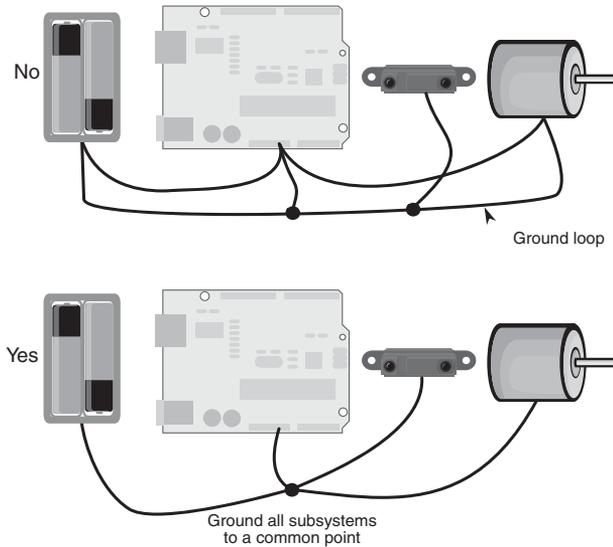


Figure 30-16 A ground loop is when there is more than one path for the ground connection. Ground loops can cause erratic behavior in circuits. Always connect the ground leads of components to one central point.

of the noisy component. Some designers like to use a decoupling capacitor on every integrated circuit, while others place them beside every third or fourth IC on the board.

It's also a good idea to put decoupling capacitors between the positive and ground connections of any circuit at the point of entry of the power supply wires. Many engineering texts suggest the use of 1 μF to 100 μF tantalum capacitors for this job. Remember that tantalum capacitors are polarized—they have a + and a – side. Be sure to properly orient the component in the circuit, or the capacitor (and maybe some other parts) will be ruined.

KEEP LEAD LENGTHS SHORT

Long wire leads on components can introduce electrical noise in other parts of a circuit. The long leads also act as a virtual antenna, picking up stray signals from the circuit, from overhead lighting, and even from your own body. When designing and building circuits, try to keep lead lengths as short as you can for everything. When soldering, this means soldering the components close to the board and clipping off any excess lead length.

AVOID GROUND LOOPS

A ground loop is when the ground wire of a circuit comes back and meets itself. The positive and ground connections of your circuits should always have “dead ends” to them. Ground loops can cause erratic behavior and excessive noise in the circuit. See Figure 30-16 for a visual depiction of a nasty ground loop that almost guarantees problems.

RoHS Demystified

RoHS Ever wondered about that “RoHS” thing that you see with many other electronic parts these days? *RoHS* stands for *Restriction of Hazardous Substances* directive, a

worldwide effort to reduce the amount of toxic materials in commercially produced electronic devices. These materials—which include lead, mercury, and cadmium—present significant health hazards as tons of discarded electronic circuit boards are dumped into landfills. *Yecch!*, as Alfred E. Neuman would say.

In following various international laws, many sellers of electronic components and completed circuits list whether a product is RoHS compliant. In the case of electronic components, RoHS compliance typically indicates that the use of lead is either reduced or eliminated. Not being in compliance does *not* mean the product is inferior, just that its manufacture does not yet meet the very strict RoHS standards.

Common Electronic Components for Robotics

Components are the things that make your electronic projects tick. Any given robot project might contain a dozen or more electronic components of varying types, including resistors, capacitors, integrated circuits, and light-emitting diodes.

In this chapter, you'll learn about the components commonly found in electronics for robotics, and the roles they play in making the circuits work. We've got a lot of ground to cover, so let's get started.

But First, a Word about Electronics Symbols

Electronics use a kind of specialized road map to tell you what components are being used in a circuit and how they are connected together. This pictorial road map is called the *schematic*; it is a blueprint that tells you just about everything you need to know to build the circuit.

Schematics are composed of special symbols that are connected with intersecting lines. The symbols represent individual components, and the lines are the wires that connect these components together. The language of schematics, while far from universal, is intended to enable most anyone to duplicate the construction of a circuit with little more information than a picture.

Forget what you might have read in some of the other books—learning how to read a schematic isn't hard. Especially as it relates to robotics, all it takes is learning the meaning of a few basic symbols.

In this chapter you're introduced to the symbol for each of the primary electronic components. These symbols are combined with those shown in Figure 31-1 that show how the various components are wired together. Purely for example purposes, Figure 31-2 shows how some symbols are interconnected to build a simple circuit—push the switch and the light-emitting diode turns on.

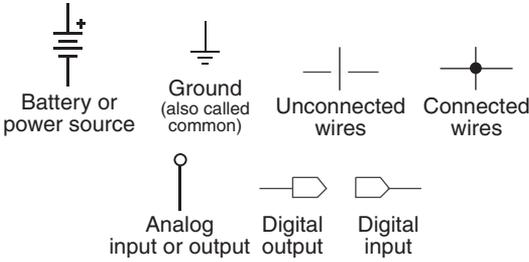


Figure 31-1 Basic schematic symbols used for wiring circuits. There are variations of those shown here, but these are the wiring symbols used throughout this book.

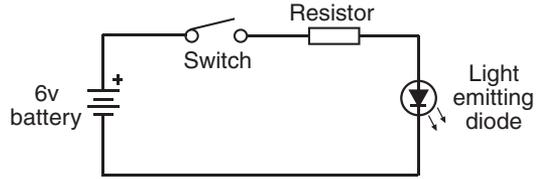


Figure 31-2 Example of a basic circuit, shown in schematic form. It consists of a battery, switch, resistor, and light-emitting diode. In many instances, the symbols are self-explanatory or pictorially descriptive.

There are hundreds of different electronic components, but only a small handful see regular use in electronic circuits for robotics. So instead of listing them all, I'll cover only those that you're most likely to encounter and use.

Fixed Resistors

Besides wire, *resistors* are the most basic of all electronic components. A resistor opposes the passing of current through it. It's like squeezing down a rubber hose to keep water from flowing through. Fixed resistors (there are also variable resistors, discussed next) apply a predetermined resistance to a circuit.

By using resistors of different values in a circuit, different parts get varying amounts of current. The careful balance of current is what makes the circuit work.

HOW RESISTORS ARE RATED

The standard unit of value of a resistor is the *ohm*, represented by the symbol Ω . The higher the ohm value, the more resistance the component provides to the circuit. The value on most fixed resistors is identified by color-coded bands, as shown in Figure 31-3 (you can't see the colors here, but you get the idea). The color coding starts near the edge of the resistor and comprises four, five, and sometimes six bands of different colors. Most off-the-shelf resistors for amateur projects use standard four-band color coding. They're the easiest to find, and the least expensive.

To read the value, start with the band closest to the edge and decode the colors using

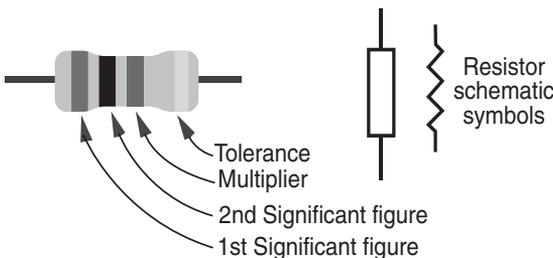


Figure 31-3 Component outline and schematic symbols for a resistor. In this book the hollow rectangle is used for resistors, but you may encounter the "sawtooth" symbol used in schematics elsewhere.

Color	1st Digit	2nd Digit	Multiplier	Tolerance
Black	0	0	1	—
Brown	1	1	10	± 1%
Red	2	2	100	± 2%
Orange	3	3	1,000	± 3%
Yellow	4	4	10,000	± 4%
Green	5	5	100,000	± 0.5%
Blue	6	6	1,000,000	± 0.25%
Violet	7	7	10,000,000	± 0.1%
Gray	8	8	100,000,000	± 0.05%
White	9	9	—	—
Gold			0.1	± 5%
Silver			0.01	± 10%
None				± 20%

Table 31-1. (For your convenience, the same chart is duplicated in Appendix D, “Electronic Reference.”)

The first and second colors are the first and second *digits* of the value; the third color is the *multiplier*, the number of zeros you need to add. For example, if the first color is brown, the second color is red, and the third color is orange:

Brown = 1

Red = 2

Orange = Multiply by 1000 (the same as simply adding three zeros)

which gives you 12,000.

The fourth band in a four-band resistor is the *tolerance*, which is the amount (in percentage form) that the resistor may actually vary from its printed value. For example, a silver band means the resistor has a 10 percent tolerance; assuming a value of 12,000, 10 percent is 1200. That means the actual value of the resistor can be anything between 10,800 and 13,200.

Note the \pm in front of the tolerance values in Table 31-1. That means + or – the indicated percentage. It can go both ways.

So-called precision resistors have a tolerance of less than 1 percent. Generally, these have five or more bands. The color coding is the same, but the extra bands provide more accurate numbers.

High-precision resistors are virtually never needed in the typical robotics circuits, and you can basically forget about them. The information is provided here so you know about the variations you can encounter when buying resistors new and as surplus.



The ohm value of resistors can vary from very low to very high. To make it easier to notate higher resistance values, resistors employ a common shorthand.

- The letter k (or K) is used to denote 1000. So a resistor with a value of 5 k Ω is the same as 5000 Ω . Sometimes the Ω symbol is dropped, because it's understood. So you may also see the resistor noted with just 5k.
- The letter M is used to denote one million, 1,000,000. A resistor with a value of 2 M Ω has a value of two million (2,000,000) ohms, or 2 *megohms* for short.
- Resistor notations for decimal values can vary depending on the country of origin. In the United States, a 4.7 ohm resistor is notated simply as 4.7 Ω . But some countries, like the United Kingdom and Australia, often use a different system, where the letter R replaces the decimal point, as in 4R7. Similarly, a 4.7 k Ω resistor is shown as 4k7.

Resistors are also rated by their *wattage*. The wattage of a resistor indicates the amount of power it can safely pass through its body without burning up—the correct term for this is *power dissipation*. Resistors used in high-load, high-current applications, like motor control, require higher wattages than those used in low-current applications. The majority of resistors you'll use for hobby electronics will be rated at 1/4 or even 1/8 of a watt.

TESTING THE VALUE OF A RESISTOR

You can readily test the value of any resistor, as shown in Figure 31-4.

1. Dial the multimeter to read ohms. If your meter is not autoranging, select a maximum range just above the marked value of the resistor. (If you don't know the value, select a high-resistance range to start.)
2. Connect the black (– or COM) lead to one end of the resistor; connect the red lead to the other end.
3. Read the result on the multimeter. If not using an autoranging meter, try a lower-resistance range to improve the accuracy of the measurement. If the meter shows over range (*Over range* indication, or the meter flashes 1.--) go back up one range.

COMMON APPLICATIONS FOR RESISTORS

Of the myriad uses of resistors in electronic circuits, two stand out as among the most common. We'll concentrate on those.

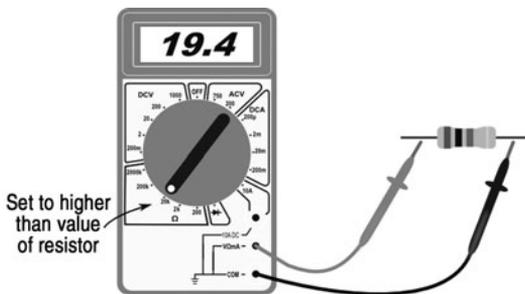


Figure 31-4 How to check the value of a resistor using a multimeter. Dial the meter to read ohms, and, if the meter is not autoranging, select a range just higher than the expected resistance value.

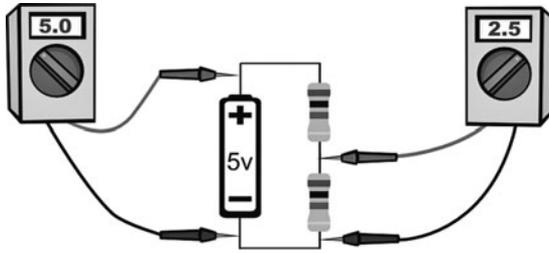


Figure 31-5 Resistors are commonly used to alter the voltage levels in a circuit. Two resistors connected in series as shown form a voltage divider. The actual voltage between the resistors depends on the values of the resistors. The readings shown here assume two resistors of the same value.

Using Resistors to Divide a Voltage

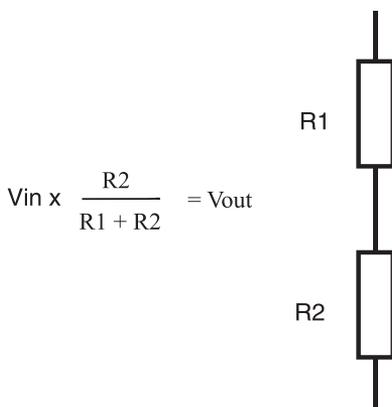
Remember that a resistor literally resists current flowing through it. In a working circuit this feature can be used to control voltage, since current, voltage, and resistance are all related—learn more about this later in “Understanding Ohm’s Law.”

Picture two resistors strung together like that in Figure 31-5. This is called a *series* connection, because the two resistors are in series with one another. (If they were side by side in the circuit, they’d be said to be in *parallel*. We don’t need to investigate this connection scheme right now, so we’ll move on.)

The circuit shown is powered by a 5-volt supply. The voltage at the point where the two resistors are connected in the middle will be somewhere between 0 and 5 volts. Exactly what that voltage is depends on the values of the resistors.

- If both resistors are of equal value, the voltage at the center is exactly one-half the supply voltage, or 2.5 volts.
- If the resistors are not the same value, the voltage is a ratio of the difference of their resistance. For example, if the top resistor is 5 kΩ and the bottom resistor is 10 kΩ, the voltage at the center is 3.33 volts.

How do you come up with these voltages, other than testing them with a multimeter each time? All it takes is some simple math. See Figure 31-6; the top resistor is referred to as R1, and the bottom is R2.



$$V_{in} \times \frac{R_2}{R_1 + R_2} = V_{out}$$

Let’s test this formula by plugging in the 5 kΩ and 10 kΩ values of the resistors and the 5 volts from the power supply. The formula becomes:

$$5 \times \frac{10,000}{15,000} = 3.33$$

or, to simplify:

$$5 \times 0.66 = 3.3$$

Figure 31-6 The basic (and simple) formula for calculating the divided voltage, when two resistors of unequal value are wired in series.

Using Resistors to Limit Current

Many electronic components, notably light-emitting diodes and transistors, will suck up as much current as the power supply will provide. This is bad because these components will

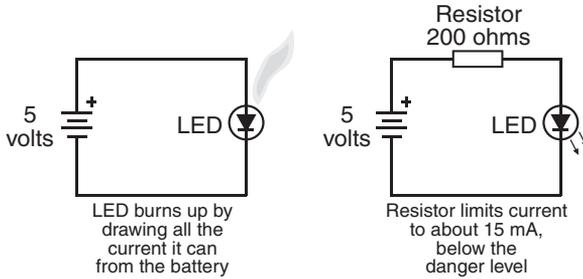


Figure 31-7 Another common use of resistors is to limit current; such a resistor is used to prevent a light-emitting diode (LED) from drawing too much current from its power source. Without a resistor to limit current, the LED will quickly burn out.

burn out if they receive too much current. They're made to handle only a certain amount of current, and beyond that they are permanently damaged.

By stringing a resistor in series with these other components you can limit the amount of current they receive. This, after all, is the main purpose of a resistor . . . to resist current.

Figure 31-7 shows a very typical wiring diagram of a battery illuminating a light-emitting diode, also known as an LED. To prevent the LED from frying because it's consuming too much current, a resistor is placed between it and the positive side of the battery. The circuit uses a 200 Ω resistor to limit the current. But how do we arrive at this value?

Again, all it takes is a little bit of math, plus knowing some things about the typical LED.

- First, most LEDs will burn out if they consume—also referred to as *draw*—more than about 30 milliamps (30 mA). So we want to make sure the LED gets less, and perhaps substantially less, than this amount of current. For example purposes, we wish to have the LED receive no more than 15 mA.
- Second, you need to know the *forward voltage drop* across the LED. This is literally the amount of voltage that is lost when current is passed through the component. The typical voltage drop of an LED is 1.5 to 3.0 volts, though this can vary pretty widely when you start using some of the specialty LEDs. For our purposes, we'll assume 2.0 volts for the drop.

Apply this simple formula to determine the value of the resistor:

$$R = \frac{V_{\text{in}} - V_{\text{drop}}}{\text{mA}}$$

V_{in} is 5, and V_{drop} (the voltage drop) is 2.0. We want to limit current to 15 milliamps, so the formula becomes

$$200 = \frac{5 - 2.0}{0.015}$$

or

$$200 = \frac{3.0}{0.015}$$

Notice that the current draw for the LED is a decimal fraction; gotta do it this way because the formula assumes amps, not milliamps (there are 1000 milliamps in 1 amp).

Resistors with four color bands come in only specific standard values, and, as it happens, 200 ohms is a standard value. When your calculation results in a nonstandard resistor value,

pick the next-higher standard value. This way, the worst thing that'll happen is that the LED won't glow quite as brightly.



In addition to calculating the resistance value, you often need to come up with the wattage value, too. For most circuits you'll be fine with the standard 1/8- and 1/4-watt resistors, but how do you know if you need a bigger wattage? It's easy when you use Ohm's law, presented next.

UNDERSTANDING OHM'S LAW

In the early 1800s, German physicist Georg Ohm experimented with the relationships between voltage, current, and resistance. He came up with a method of accurately calculating these relationships, and this became Ohm's law.

Figure 31-8 shows the basic triad that makes up the law—it's called the Ohm's law triangle. The triangle is pretty clever, because it indicates the math you do to calculate one value when you know the other two. In all cases, you use either multiplication or division, also shown in the figure.

V stands for voltage (note: in some texts describing Ohm's law, voltage is shown as E).

R stands for resistance.

I stands for current (it's not C , as that stands for the capacitance of a capacitor, as in flux capacitor).

Example Ohm's Law Calculation

Let's just take one of the formulas, the one for calculating V (voltage). For that, you need to know two of the other elements of the triangle, I (current, in amps) and R (resistance). The formula is:

$$V = I \times R$$

Suppose the current is 1.2 amps, and the resistance is 50 ohms. Simply multiply 1.2 times 50; the result is 60, for 60 volts.

Calculating Power (Watts)

You can use an extension of Ohm's law to calculate power dissipation in a circuit. This is helpful to ensure that the wattage of the resistors you choose is high enough. Higher-wattage resistors are bigger and can handle more power passing through them.

The extension isn't part of the simple Ohm's law triangle, but is a more complicated variation involving a wheel. If you're interested in learning more about calculating power and watts,

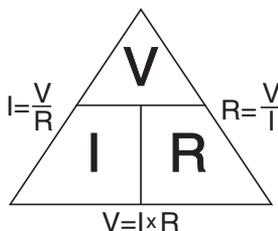


Figure 31-8 The Ohm's law triangle, a mnemonic diagram that shows how to calculate the missing value when two other values are known. You can calculate for resistance, voltage, or current (current is referenced as the letter I).

refer to Appendix D, “Electronic Reference,” for a brief discussion of using the Ohm’s law wheel. For now, though, we only need to apply one of the 12 formulas of the wheel.

Let’s return to the example of selecting a resistor to limit current to an LED. The formula to calculate power is very simple:

$$V \times I = W$$

- V = voltage through the LED. This includes the voltage drop through the LED; from the previous example this voltage is 3.0.
- I = current to the LED. From the previous example this is 15 mA, or .015.
- W is the power dissipation, in watts. The resistor needs to be rated to dissipate at least this amount of power.

Substituting the actual values, you get

$$3.0 \times 0.015 = 0.045$$

The answer is in whole watts; 0.045 is less than 1/20 of a watt, so the standard 1/8-watt resistor is more than enough.

Potentiometers

Potentiometers are technically variable resistors. They let you “dial in” a specific resistance. The actual range of resistance is determined by the upward value of the potentiometer, and this upward value is how the potentiometer is marked. As with fixed resistors, the values are in ohms. For example, a 50 k Ω potentiometer will let you dial in any resistance from 0 ohms to 50,000 ohms.

Potentiometers (or *pots* for short) are of either the dial or the slide type. The dial type shown in Figure 31-9 is the most familiar and is used in such applications as radio volume and electric blanket thermostat controls. The rotation of the dial is nearly 360°. In one extreme, the resistance through the potentiometer is zero; in the other extreme, the resistance is the maximum value of the component.



Figure 31-9 A potentiometer is a variable resistor. Turn the shaft of the potentiometer and its resistance changes.

LINEAR OR AUDIO TAPER

As you turn the dial of a potentiometer, the resistance varies from the lower extreme—usually 0 ohms, or very close to it—to the indicated value of the pot. The scale of the change is dependent on the internal construction of the component. There are two scales: linear and audio (also called logarithmic). The scale is referred to as the *taper* of the potentiometer.

- With *linear taper*, the most common, the value changes in proportion to the setting of the dial. For example, with a 10 k Ω pot, turning it a quarter of the way will yield 1/4 of the full scale, or 2.5 k Ω . For nearly everything you do in robotics you’ll want a linear taper potentiometer.

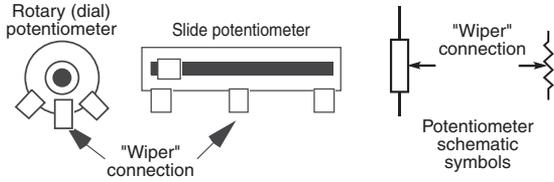


Figure 31-10 Component outline and schematic symbols for a potentiometer (or “pot” for short). The wiper is the center connection of the pot. It’s the wiper connection that provides the varying resistance value.

- With *audio taper*, the value of the potentiometer is a logarithmic function of the position of the dial. Given a 10 kΩ pot, the component still varies from 0 Ω to 10 kΩ; however, the change is not a straight line but a curve that’s especially steep. Audio taper pots are a fairly common find in the surplus market. You don’t want one of these unless you’re working on an audio project.

USING A POTENTIOMETER

Most pots have three connections (see Figure 31-10), which basically form two resistors in series. In fact, potentiometers behave just like two resistors in series, and they can be used for the same kinds of things; for example, as voltage dividers. The ratio of the values of the two resistors used in the divider determine the voltage.

As shown in Figure 31-11, the two terminals on either side of the potentiometer function like the top of the fixed resistor R1 in Figure 31-6, and the bottom of the fixed resistor R2. The center terminal, called the *wiper*, is the connection between R1 and R2. As you turn the dial of the pot, you vary the ratio between the two resistances.

You can quickly test the operation of a potentiometer by connecting it to a multimeter (see Figure 31-12).

1. Dial the multimeter to read ohms. If your meter is not autoranging, select a maximum Ω just above the marked value of the potentiometer.
2. Connect the black (– or COM) lead to the center wiper terminal of the pot, and connect the red lead to either of the end terminals.
3. Slowly rotate the pot in one direction or the other, and watch the resistance go up and down.

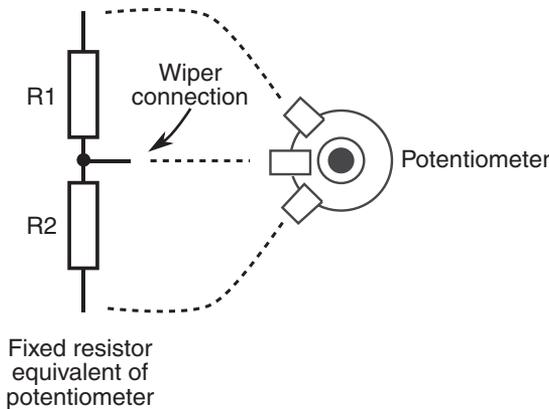


Figure 31-11 A potentiometer is basically two resistors wired in series, like that in Figure 31-5. Except in a pot, the values of the two resistors are constantly changing as you rotate the dial.

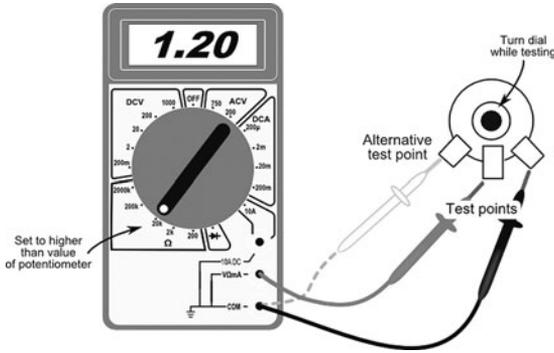


Figure 31-12 How to check the value of a potentiometer using a multimeter. The concept is the same as when testing a resistor. Turn the dial as you test to watch the resistance value change.

OTHER TYPES OF VARIABLE RESISTORS

Potentiometers are the most common type of variable resistor, but there are several other kinds you'll encounter in your robot-building lifetime. The two most often used are photoresistors and the force-sensitive resistor.

- *Photoresistors* are sensitive to light. Their value changes as the intensity of the light varies. Photoresistors often go by the names *photocell* and *CdS cell*; the CdS comes from cadmium sulfide, the mixture of chemicals that make the component sensitive to light.
- *Force-sensitive resistors*, or FSRs, register force or pressure against them. The resistance of the component goes up or down as the force/pressure changes. There are numerous kinds of FSRs used for various kinds of sensing jobs. For example, a flex sensor provides a varying resistance as it's twisted or bent. For others, resistance changes as any part of the membrane of the component is pressed against.

FYI

Read more about photoresistors in Chapter 44, "Robotic Eyes," where you'll learn how to use CdS cells to make your robot respond to light. Be sure to check out Chapter 42, "Adding the Sense of Touch," for additional details about FSRs.

READING POTENTIOMETER MARKINGS

Unlike fixed resistors, potentiometers aren't marked with a color code. Instead, their value is marked either directly—for example, 10,000 or 10K—or indirectly using a *decade* numbering system. In this system the value is a three-digit number such as 503, which means 50, followed by three 0s, or 50,000. The number is given in ohms, meaning the pot is 50 kΩ.

Capacitors

After resistors, *capacitors* are the second most common component found in the average robotics electronic project. Capacitors serve many purposes. They can be used to delay the action of some portion of the circuit or to remove bothersome electrical noise within a circuit. These and other applications depend on the ability of the capacitor to hold an electrical charge for a predetermined period of time.

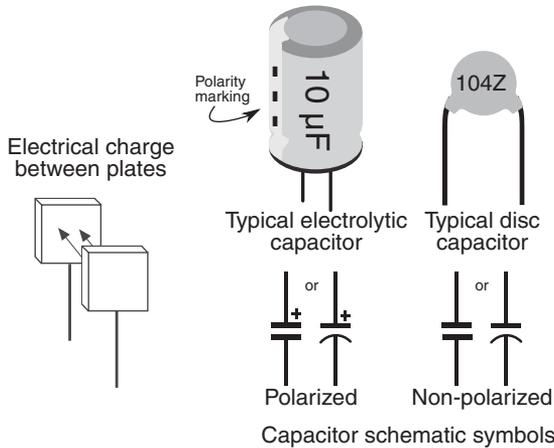


Figure 31-13 Component outline and schematic symbols for two popular styles of capacitors. Capacitors may be polarized or nonpolarized. When using polarized capacitors, there will be a polarity marking on the body of the component. Be sure to properly orient the capacitor in the circuit.

Capacitors come in many more sizes, shapes, and varieties than resistors, though only a small handful are truly common. However, most all capacitors are made of the same basic stuff: two or more conductive elements are separated by an insulating material called the *dielectric* (see Figure 31-13).

This dielectric can be composed of many materials, including air, paper foil, epoxy, plastic, even oil. When you select a capacitor for a particular job, you must generally also specify the dielectric. The most common are summarized in Table 31-3, later in this chapter, along with their common uses.

HOW CAPACITORS ARE RATED

Capacitors have two important ratings:

- **Capacitance.** Capacitance is the ability of the component to hold a charge. The larger the capacitance, the longer the charge is retained.
- **Dielectric breakdown voltage.** At higher voltages the dielectric becomes partially or completely electrically conductive and the capacitor no longer functions as it should. The capacitor must be used below this voltage.

Capacitance is measured in *farads*. The farad is a large unit of measurement, so the bulk of capacitors available today are rated in microfarads; one microfarad is a millionth of a farad.

When the capacitor is under 1 microfarad, its value may be shown as a decimal point number—for example, 0.1 for one-tenth of a microfarad. Or it may be shown as a nanofarad. A nanofarad is a thousandth of a microfarad—that 0.1 microfarad capacitor is instead listed as 100 nanofarad. Same value, different way of expressing it. An even smaller unit of measure is the picofarad, or a millionth of a microfarad.

The “micro-” in the term *microfarad* is most often represented by the Greek “mu” (μ) character, as in 10 μ F, or 10 microfarads. Keeping up with the shorthand, the nanofarad is nF, and the picofarad is pF.



In older books and magazines on electronics you may see microfarad shortened to *mfd*. Example: 10 *mfd* is 10 microfarads (10 μF). *Mfd* isn't used as much today, but it's good to know what it means in case you want to try out an older circuit you find at the library.

HOW CAPACITORS ARE MARKED

Capacitors are routinely marked with at least their capacitance value; many capacitors are also marked with value tolerance notation and a breakdown voltage. Additionally, capacitors that are polarized—they have a + and a – lead—also carry a polarization marking.

Capacitance Value

The capacitance values for some capacitors are printed directly on the component. This is true of larger capacitors with values of 1 μF or higher, if for no other reason than that their larger physical size allows the manufacturer to directly print the value on the component.

But for other capacitors, things aren't always so simple. Smaller capacitors often use a common three-digit *decade* marking system to denote capacitance. The numbering system is easy to use if you remember it's based on picofarads, not microfarads.

A number such as 104 means 10, followed by four zeros, as in

100,000

or 100,000 picofarads. To make the conversion, move the decimal point to the left six spaces: 100,000 becomes 0.1. Therefore, that 104 capacitor is 0.1 μF , or 100 nF. Note that values under 1000 picofarads do not use this numbering system. Instead, the actual value, in picofarads, is listed, such as 10 (for 10 pF). Table 31-2 provides a quick glance at how several common capacitor number markings convert to their μF (microfarad) equivalents.

Table 31-2 Capacitor Value Reference

Marking	Value (μF)	Marking	Value (μF)
xx (number from 01 to 99)	xx pF		
101	0.0001	331	0.00033
102	0.001	332	0.0033
103	0.01	333	0.033
104	0.1	334	0.33
221	0.00022	471	0.00047
222	0.0022	472	0.0047
223	0.022	473	0.047
224	0.22	474	0.47



As with resistors, what's printed on a capacitor may not exactly match its real value. The accuracy of the stated value can vary widely, typically much more than with resistors. There are several ways to indicate capacitor tolerance. Rather than take up the space here, I've uploaded a guide on capacitor selection to the RBB Online Support site, described in Appendix A.

Dielectric Breakdown Voltage Value

The dielectric breakdown voltage is specified only for certain capacitors. For those that have it, the voltage is marked directly, such as “35” or “35V.” Sometimes, the letters *WV* are used after the voltage rating. This indicates the *working voltage*. You should not use the capacitor in a circuit with a voltage that exceeds this value.

On capacitors without a breakdown voltage printed on them you must estimate the value based on the type of dielectric it uses. This is an advanced topic and not covered in this book; nevertheless, it seldom comes up in electronics for robotics because most circuits use 12 volts or less, and most capacitors have a rated breakdown voltage of 25 to 35 volts. As a safety margin, select a capacitor with a breakdown voltage at least double the operating voltage of the circuit.

Polarization Marking

Some capacitors are polarized. Markings on the capacitor indicate the + or the – terminal.

If a capacitor is polarized, it is *extremely* important that you follow the proper orientation when you install the capacitor in the circuit. If you reverse the leads to the capacitor—connecting the + side to the ground rail, for example—the capacitor may be ruined. Other components in the circuit could also be damaged.



By convention, the polarizing mark on aluminum electrolytic capacitors is typically the – (negative) lead. The polarizing mark on tantalum electrolytic capacitors is typically the + (positive) lead.

UNDERSTANDING CAPACITOR DIELECTRIC MATERIAL

Capacitors are classified by the dielectric material they use. The most common dielectric materials are listed in Table 31-3. The dielectric material used in a capacitor partly determines which applications it should be used for.

COMMON APPLICATIONS FOR CAPACITORS

Unlike resistors, for which it's easy to demonstrate practical applications in a circuit, capacitors are a bit more nebulous. They tend to work by interacting with other components, rather than doing things just on their own.

Capacitors are often used to filter, or remove, the rapid variations of an input voltage, leaving only a steady voltage. This is quite useful in all kinds of electronics, because some components produce large, instantaneous “glitches” of voltage. These glitches, referred to as power line noise, may disrupt neighboring components, especially integrated circuits.

Capacitors may also be used with resistors as part of a timing circuit. The value of the capacitor determines how long an event lasts—timing is controlled by how long it takes for the capacitor to charge or discharge its current. In all of the applications in this book involving

Table 31-3 Capacitor Dielectric Materials (and Their Uses)

Dielectric	Typical Applications	Polarized*
Aluminum electrolytic	Power supply filtering; electrical noise filtering (decoupling)	Yes
Tantalum electrolytic	Same as aluminum electrolytic, but smaller and more compact; more expensive; breakdown voltages may not be as high	Yes
Ceramic	Most common; inexpensive, general purpose; often referred to simply as a “disc capacitor” due to its typical shape	No
Silver mica	High precision, typically used for high-frequency applications; expensive	No
Polyester (or Mylar) polypropylene	Low-power, low-frequency signal applications where ceramic capacitors are not adequate; more stable than ceramic capacitors	No
Paper foil	Your grandpa’s Heathkit radio	No (but better ask Grandpa)

* Typical. There are exceptions to everything, including whether a capacitor is polarized. So be on the lookout for any polarization markings, or the lack thereof. For example, some aluminum electrolytic capacitors are nonpolarized. These are usually marked NP, and are designed for special applications, such as use in stereo speaker systems.

the ubiquitous LM555 timer IC, for example, there’s always a capacitor that—when combined with at least one resistor—determines the duration of the timer.

Diodes

The diode is a rudimentary form of semiconductor. Diodes (see Figure 31-14) are used in a variety of applications, and there are numerous subtypes. Here is a list of the most common that are used in the field of robotics (many of these topics are covered more fully in the projects found in Part 8):

- *Rectifier*. Let’s call this the “average” diode. It’s so called because one of its principal uses is to rectify AC current to provide DC only. It’s used for many other things, too, and is frequently employed in robotics circuits for motor control applications.
- *Schottky*. Special kind of diode that has improved performance. Used for applications where speed, low voltage drop, and “snap action” are needed.
- *Zener*. This diode limits voltage to a predetermined level. Zeners are used for low-cost voltage regulation.
- *Photo*. All semiconductors are sensitive to light, but photodiodes are made especially for the task. Photodiodes and their close cousin, the phototransistor, are frequently used as light sensors in robotics projects.

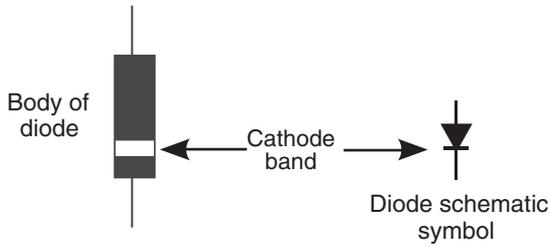


Figure 31-14 Component outline and schematic symbols for a diode. Diodes are polarized, as noted by a colored band. The band denotes the cathode, which is the negative (–) connection.

- *Light-emitting.* These diodes emit infrared or visible light when current is applied.
- *Laser.* The now-common penlight lasers use a specially constructed diode that emits single-color laser light. They can be used for visual effects and some sensor applications.

HOW DIODES ARE RATED

Diodes carry three important ratings: peak inverse voltage, current, and forward voltage drop (there are others, but these are the main ones).

The peak inverse voltage (PIV) rating roughly indicates the maximum working voltage for the diode. In the case of the common rectifier diodes, this maximum voltage is 50, 100, even 1000 volts, depending on the component. These voltages are well beyond what you'll typically find in electronics for a robot, giving you a wide variety of components to choose from.

The current rating is the maximum amount of current that can be passed through the diode without risk of overheating and eventual self-destruction. Current ratings are in amps or, in the case of small diodes, milliamps. This rating is very critical in robotics applications, where it's common to have circuits that require more current than what the average diode can handle.

Forward voltage drop is the amount of voltage that is essentially lost when it passes through the diode. The voltage drop affects the overall behavior of the diode; for example, with a very low drop, the diode is faster-acting.

UNDERSTANDING DIODE POLARIZATION

All diodes are polarized, and they have positive and negative terminals. The positive terminal is called the *anode*, and the negative terminal is called the *cathode*.

You can readily identify the cathode end of a diode by looking for a colored band or stripe near one of the leads. Figure 31-14 shows a diode that has a band at the cathode end. Note how the band corresponds with the heavy line in the schematic symbol for the diode.

EXPLORING THE COMPOSITION OF DIODES

Among other variations, diodes are available in two basic flavors, germanium and silicon, which indicate the material used to manufacture the active junction (the part that conducts current) within the diode. The two types of materials also have an effect on the forward voltage drop of the device: about 0.7 volts for silicon and 0.3 volts for germanium.

Since the voltage drop can change the behavior of the circuit, you should always be careful which of these two kinds you use. If the circuit doesn't say, a silicon diode is probably fine; if a germanium component is called for, be sure not to substitute a silicon type.

COMMON APPLICATIONS FOR DIODES

Like all other electronic components, diodes are used for a fantastically wide array of applications. Two serve as good examples of the beneficial role diodes play in circuits for robotics control: incremental voltage drop and reverse polarity protection.

In both of these examples, no attention is given to ensuring that the diodes selected for the task can adequately handle the current demands of the circuits. In an actual circuit you'll need to determine the total current draw through the diode and make sure you pick one that is rated for that current.

Incremental Voltage Drop

Recall that diodes exhibit a forward voltage drop, whereby a certain amount of voltage is lost as current passes through the device. With silicon diodes, this drop is about 0.7 volts. You can use this "feature" of a diode to incrementally decrease a voltage in specific steps. Note that this isn't real voltage regulation; if the voltage from the power supply goes up or down, so will the voltage on the other side of the diodes.

Reverse Polarity Protection

Connecting a battery backward to an electronic circuit can easily damage the circuit. You can prevent such damage by putting a diode in series with the positive power supply connection. This technique works because the diode passes current in one direction only: from anode to cathode. It won't allow current to flow the other way.

Light-Emitting Diodes (LEDs)

All semiconductors emit light when an electric current is applied to them. This light is generally very dim and only in the infrared region of the electromagnetic spectrum. The *light-emitting diode* (LED; see Figure 31-15) is a special type of semiconductor that is expressly designed to emit copious amounts of light. Most LEDs are engineered to produce a specific color of light, as well as infrared and ultraviolet. Red, yellow, and green LEDs are among the most common, but blue, violet, and even white light (all-color) LEDs are available.

LEDs carry the same specifications as any other diode. The *typical* LED has a maximum current rating of 30 milliamps or less, though this varies greatly, and depends on size, type,

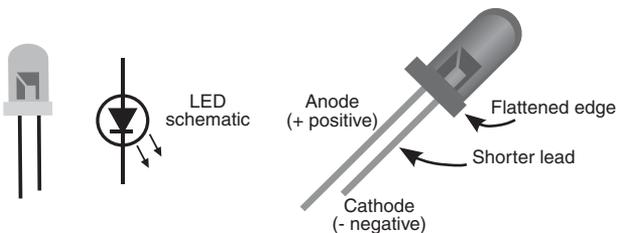


Figure 31-15 Component outline and schematic symbols for a light-emitting diode (LED). Diodes are polarized, and, depending on the component, the polarization may be noted by a flattened edge and/or a shorter lead. These denote the cathode, which is the negative (-) connection.

even color. And like a diode, all LEDs exhibit a forward voltage drop—only it's often much higher than that of a standard diode. Depending on the LED (and often related to its color), expect a voltage drop of between 1.5 and 3.5 volts. Some specialty high-brightness LEDs have even higher drops.

And, as with a standard diode, the terminals on an LED are its anode (+) and cathode (-). Rather than a white or colored stripe, most LEDs distinguish the two using other methods, as shown in Figure 31-15. Not all LEDs follow the same marking conventions, so you may need to experiment. (Usually nothing bad happens if you connect an LED backward—that is, if you switch the anode and the cathode—but the LED will not light.)

POWERING LEDs

LEDs are most often used in low-power DC circuits and are powered with 12 volts or less. Always remember that the component can be ruthlessly damaged if you expose it to currents exceeding its maximum rating. So, unless the LED has a built-in resistor, you always need to add one to limit the amount of current that flows through the LED. See the application examples for the resistor, mentioned previously in the chapter, which demonstrate this process.

SHAPE, SIZE, AND LIGHT OUTPUT

Light-emitting diodes come in all kinds of shapes and sizes. The most common are cylindrical and shaped with a domed top. Popular sizes are:

- T1, or miniature: 3mm in diameter
- T1-3/4, or standard: 5mm
- Jumbo: 10mm

While the most common LED is round, there are also square, rectangular, even triangular LEDs. The shapes are handy for different kinds of applications—the triangles look like arrows, so they can be used to show direction.

Multi-LED Displays

LEDs can come one in a package or as part of a larger package with other LEDs. Each individual LED in the package can be individually lit. Three common variations on the multi-LED theme are:

7-segment display: Has seven individual LEDs in special shapes to form a large numeral.

By controlling which LEDs are on and off, the display can show numbers 0 through 9.

Bar graph display: Typically contains 10 miniature rectangular LEDs.

Dot matrix display: Contains rows and columns of dots; any number, letter, or special character can be created by lighting up certain dots.

LED COLORS

Most LEDs emit a single color, but others are designed to produce two or three colors. You can control which color is shown by applying current to various terminals on the LED (Figure 31-16).

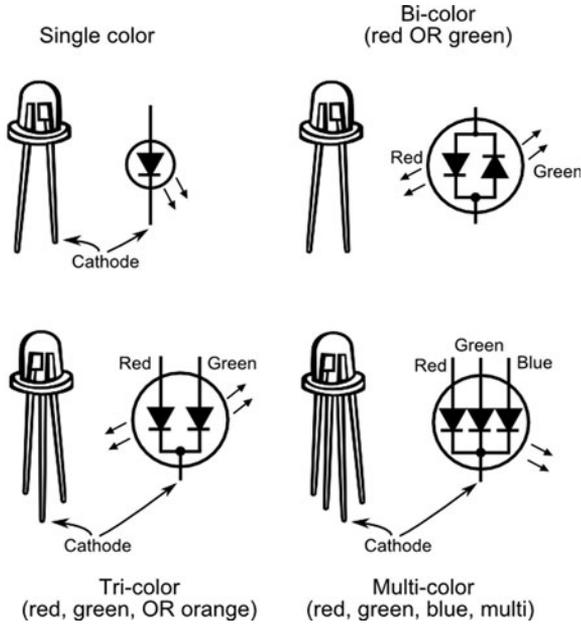


Figure 31-16 LEDs that display more than one color have additional connection points. The wiring of the connections depends on the type of LED. Shown here are several of the most common.

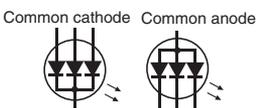
- A two color (or *bicolor*) LED contains red and green LED elements (other colors are possible too, but the red/green combination is the most common). You control which color is shown by reversing the voltage to the LED. You can also produce a yellowish-orange by quickly alternating the voltage polarity.
- A *tricolor* LED is functionally identical to the bicolor LED, except that it has separate connections for the red and green diodes. You can produce the intermediate color of yellowish-orange by turning on both the red and green diodes at the same time.
- A *multicolor* LED contains red, green, and blue LED elements. You control which color to show by individually applying current to separate terminals on the LED. These are also called RGB LEDs.



There's a bit of confusion as to what, exactly, constitutes bicolor and tricolor LEDs. Both can produce three colors: red, green, and yellowish-orange. The way they do it differs. Compounding the confusion is that some sources call a multicolor LED a tricolor LED, because it (rightly) contains three colors.

Common Anode or Cathode

To reduce the number of terminals coming out of the multidiode LED, all of the anodes, or all of the cathodes, of each diode in the device are wired together. When all the anodes are combined, the LED is said to be *common anode*. And when all of the cathodes are combined, it's *common cathode*.



Which one you choose depends on what's available and how the circuit is designed. Common cathode is more common among multiple-color LEDs; both common anode and common cathode are used in 7-segment and other LED displays. Be sure to match the device with the circuit plans you're using.

Transistors

Transistors (see Figure 31-17) were designed as an alternative to the old vacuum tube. They're used for many of the same things, either to amplify a signal or to switch a signal on and off. At last count there were *several thousand* different types of transistors available.

Transistors are divided into two broad categories: signal and power. You can usually tell the difference between the two merely by size.

- *Signal.* These transistors are used with relatively low current circuits, like radios, telephones, and some hobby electronics projects. They are available in either plastic or retro-looking metal cases. The plastic kind is suitable for most uses, but some precision applications require the metal variety. No projects in this book require great precision, so the cheap plastic signal transistor is good enough.
- *Power.* These transistors are used with high-current circuits, like motor drivers and power supplies. Power transistors come in metal cases, though a portion of the case (the back or sides) may be made of plastic.

Figure 31-18 shows several common varieties of transistor cases and how they are referred to, such as *TO-92* and *TO-220*. The signal transistor is rarely larger than a pea and uses slender wire leads. The power transistor has a large metal case, to help dissipate heat, and heavy, spokelike connection leads.

HOW TO IDENTIFY A TRANSISTOR

Transistors are identified by a unique code, such as 2N2222 or MPS6519. Refer to a data book to ascertain the characteristics and ratings of the particular transistor you are interested

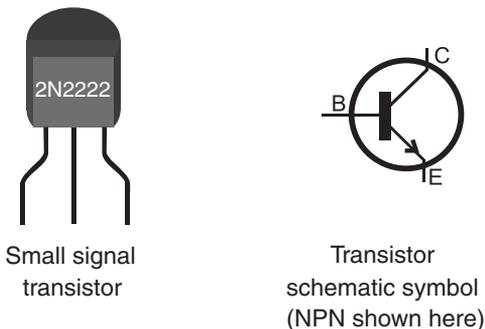


Figure 31-17 Component outline and schematic symbols for a transistor. The outline view is of a small signal transistor.

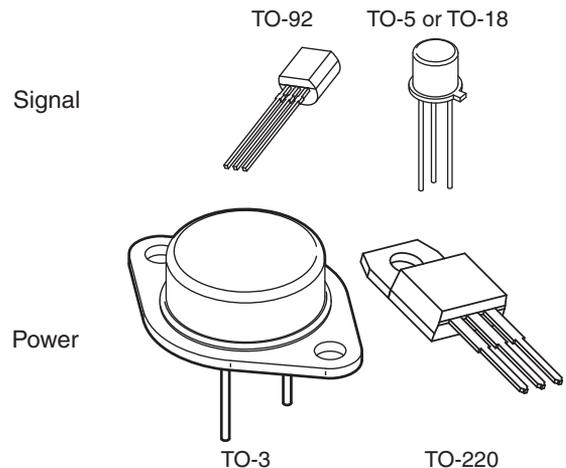


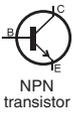
Figure 31-18 There are many shapes and sizes of transistors, but a common trait of most of them is that they have three connection wires. Larger transistors are generally used for applications involving high currents, such as powering motors. Shown are four popular transistor cases.

in. Transistors are rated by a number of criteria, which are beyond the scope of this book. None of these ratings are printed directly on the transistor.

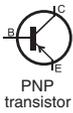
Transistors have three- or four-wire leads. The leads in the typical three-lead transistor are *base*, *emitter*, and *collector*. (A few transistors, most notably certain types of the field-effect transistor—or FET, for short—have a fourth lead. None of the circuits in this book use this type, which are not common anyway.)

NPN, PNP—TWO SIDES OF THE SAME COIN

Transistors can be either NPN or PNP devices. This nomenclature refers to the sandwiching of semiconductor materials inside the device. You can't tell the difference between an NPN and a PNP transistor just by looking at them.



NPN transistor



PNP transistor

However, the difference is indicated in the catalog specifications sheet as well as by the schematic symbol for the transistor. In an NPN device, the arrow is shown leaving the transistor; in a PNP device, it's the opposite. This differentiation helps you to quickly tell whether you should use an NPN or a PNP type transistor for the circuit.

ENTER THE MOSFET

Some semiconductor devices look and act like transistors and are actually called transistors, but in reality they use a different underlying technology. So far I've been talking about a class of transistor called the *bipolar junction transistor*, or *BJT*. These are by far the most common. Another form of transistor is the MOSFET. This collection of alphabet soup stands for metal-oxide semiconductor field-effect transistor. It's often used in circuits that demand high current and high precision.



N-channel MOSFET



P-channel MOSFET

Wouldn't you know it, but just to complicate things, MOSFET transistors don't use the standard base-emitter-collector connections you just read about. Instead, they call them *gate*, *drain*, and *source* connections. And note, too, that the schematic diagram for the MOSFET is different from that of the standard transistor.

Like the bipolar junction transistor, MOSFETs come in two varieties: N-channel and P-channel. And, as before, you can't tell the difference between an N-channel and a P-channel MOSFET just by looking at it.



The schematic symbols for the two types of MOSFETs are only ever-so-slightly different, and not all schematic diagrams bother to show which one is used in the circuit. You *cannot* substitute N-channel for P-channel, so when using these devices, be absolutely sure you've got the right ones in your hands.

Integrated Circuits

The *integrated circuit*, colloquially referred to as an *IC* or *chip*, forms the backbone of the electronics revolution. The typical integrated circuit comprises many transistors and other components. As its name implies, the integrated circuit is a discrete and wholly functioning circuit in its own right. ICs are the building blocks of larger circuits. By merely stringing them together you can form just about any project you envision.

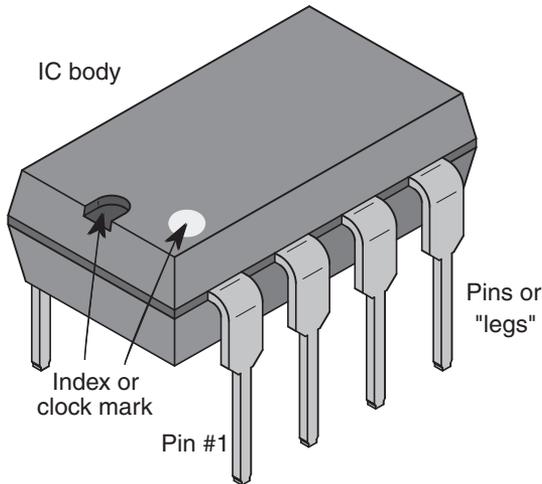


Figure 31-19 Integrated circuits come in a plastic rectangular body, with four or more connection pins, or legs. A printed index mark or notch shows the “top” of the component. Pin 1 is always the first pin when going counterclockwise.

Integrated circuits are enclosed in a variety of *packages*. The actual integrated circuit itself is just a tiny sliver inside this package. For the hobbyist, the easiest-to-use IC package is the *dual in-line package* (or *DIP*), like the one in Figure 31-19. The illustration shows an 8-pin DIP, but other sizes are common, too.

IDENTIFYING INTEGRATED CIRCUITS

As with transistors, ICs are identified by a unique code printed on the top. Codes may be a simple number sequence such as 7400 or 4017. This code indicates a type of device that is made by many different manufacturers. You can use this code to look up the specifications of the IC in a reference book.

More than likely, the number identifying the IC will contain letters that further distinguish it from other ICs of the same family, that is, ICs that do pretty much the same thing but have different operating characteristics or manufacturing technologies. The differences among these IC families are quite complex and well outside of what this book is about, but to get you started, let’s take the 7400 as an example.

The 7400, which dates back to the swingin’ 1960s, contains four digital NAND gates; NAND gates are one of several common forms of logic circuits used to create computers. Variations of this chip include:

- 7400—Base chip as originally manufactured.
- 74ALS00—Advanced low-power Schottky; enhanced lower-power version.
- 74HCT00—CMOS version of the 7400 compatible with older devices.

And there are *many* others. When a circuit specifies just the base chip—the 7400 rather than a specific family member—it usually means that the circuit isn’t particularly picky, and you can use most any IC in that family.

Many ICs also contain other written information, including manufacturer catalog number and date code. Do not confuse the date code or catalog number with the code used to identify

the device. Date codes can look like part numbers—9206 might mean the IC was made in June 1992.

MICROCONTROLLERS AND OTHER SPECIALTY ICs

Chips like the 7400 (and all its kin) are considered standard building blocks and are available from a variety of manufacturers. Add to these the thousands of specialty ICs that are unique to specific chip makers and perform unique tasks.

For robotics, the most common of these specialty ICs is the microcontroller, a type of all-in-one computer that is designed to directly connect with external inputs and outputs. Microcontrollers are more fully detailed in Chapters 35 through 40.

Still other specialty ICs might perform any of the following tasks:

- Sense the tilt of your robot to let you know if it's fallen over (accelerometer)
- Detect small changes in temperature and convert this information to a signal that can be read by a microcontroller (temperature sensor)
- Convert one voltage to another, with very high efficiency (switching mode regulator)
- Create sound effects and intelligible speech (sound and speech synthesizer)
- Control the operation and speed of a DC motor (H-bridge motor driver)

and, of course, many others.

As with standard-building-block ICs, specialty chips carry markings that indicate the manufacturer name and part number, date code, and other important information.

Switches

Most robots use at least one switch—to turn it off or to reset the thing when its programming goes bonkers. A lot of robots use multiple switches, for things like setting operating modes or detecting when the bot has bumped into a wall, chair leg, or person.

Switches come in a variety of shapes and sizes, but their overall functionality is universal: switches are composed of two or more electrical *contacts*. In one position of the switch, the contacts are not connected, and current doesn't flow through the device. The switch is said to be *open*. In the other position, the contacts are connected together, and current flows. The switch is said to be *closed* (see Figure 31-20).

Poles: The most basic switch has a single pair—or *pole*—of contacts. But some switches are composed of several poles, so there are extra pairs of contacts—one switch functions as multiple switches all connected together. Multiple poles allow the switch to control multiple circuits at the same time.

Throws: The basic switch has two positions, on or off. But some switches have more positions. Each position is called a *throw*. Instead of just on or off, a two-throw switch is on-on. Each “on” can connect to a different part of the circuit. Most often, multiple throws are combined with multiple poles to provide all manner of choices.

Momentary position: Switches that are spring-loaded are called *momentary*, because when you release the switch it automatically goes back to its normal position. The most common type of momentary switch is the push button: press it, and the switch closes; release it, and the switch returns to open all on its own.

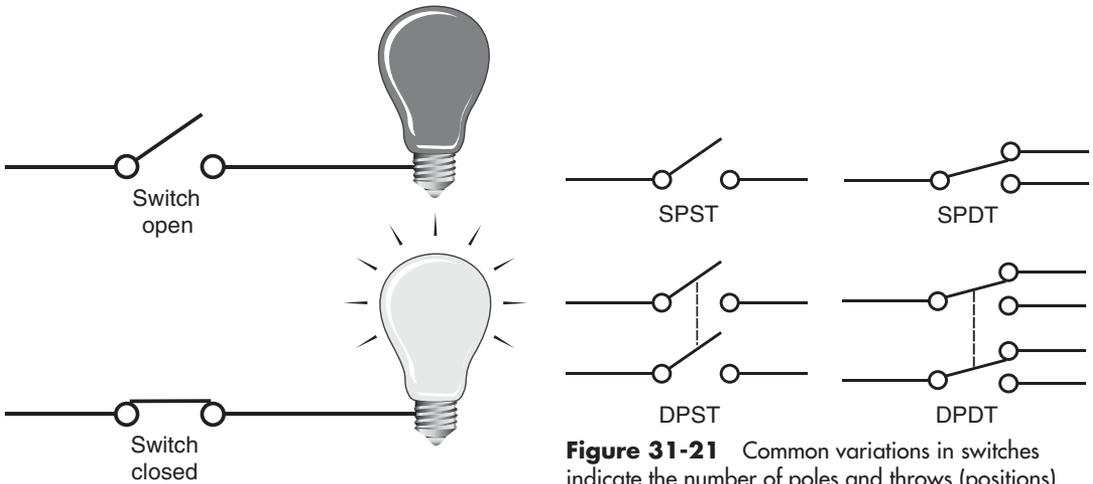


Figure 31-20 Basic function of a switch: When open, the light is off (unpowered). When closed, the switch completes the circuit, and the light turns on.

Figure 31-21 Common variations in switches indicate the number of poles and throws (positions). Common switches for use in robotics include single-pole, single-throw (SPST), and double-pole, double-throw (DPDT).

Switches are further defined by how they are operated. There are four principal types, and each is self-explanatory by its name: toggle, push-button, slide, and rotary.

MIXING AND MATCHING

Poles, throws, and momentary position can all be mixed together in various combinations. Switches follow a common nomenclature to describe how they function. Here are just a few of the combinations you'll likely encounter (see Figure 31-21).

- Single-pole, single-throw (SPST) has one set of poles and one throw. It's the basic, no-frills switch.
- Single-pole, double-throw (SPDT) has one set of poles and two throws.
- Double-pole, single-throw (DPST) has two set of poles and just one throw. You can think of it as two SPST switches combined into one.
- Double-pole, double-throw (DPDT) has two set of poles and two throws.

MOMENTARY AND CENTER-OFF

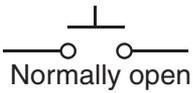
Add to the three main pole/throw combinations spring-loaded momentary and center-off positions. A DPDT switch with a center-off position is said to be on-off-on. One or both of the on positions may be momentary, that is, spring-loaded. Release the switch, and it returns to its off position.

When you read specifications about a switch, you may see something like this:

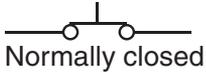
(on)-off-(on)

It describes a switch with a center-off position. The parentheses mean that the position is momentary.

NORMALLY OPEN, NORMALLY CLOSED



The contacts of a momentary switch can be either normally open or normally closed. The “normally” has to do with the position of the contacts when the switch is not being depressed.



In a normally open (NO for short) switch, the circuit is normally broken. Depressing the push button closes the contacts. It’s just the opposite with a normally closed (NC) switch: depressing the button opens the contacts.

Relays

Relays work like switches, but they are changeable under electronic control. Instead of a human (or dog, or whatever) pressing a switch to change the contact settings, in a relay it’s all done via electrical signals. Figure 31-22 shows the basic construction of a relay and how it works. As shown, the two basic parts of the relay are the *coil* and the *contacts*. In operation, when the coil is energized, the magnetic field that is created activates the switch contacts.

Because relays are essentially switches, they are defined in the same way: the poles and throws indicate the number of contacts. But unlike switches, most (but not all) relays are by their nature momentary. The switch contacts in a relay are activated when the relay is energized. Remove the juice from the relay, and the spring-loaded contacts go back to their original position.

COMMON RELAY TYPES

There are lots of different kinds and styles of relays out there, but most fall into one of the following three types (see Figure 31-23):

- The SPST (single-pole, single-throw) relay has four connections: two for the coil and two for the switch contacts. The contacts may be normally open (the most common) or normally closed.
- The SPDT (single-pole, double-throw) relay has five connections: two for the coil, as usual, and three for the switch contacts. The contacts are often labeled as Common, NC, and NO. You wire your circuit to Common and either the NC or the NO connection.
- The DPDT (double-pole, double-throw) relays are an extension of the SPDT. They have a separate set of pole contacts, for a total of eight connections.

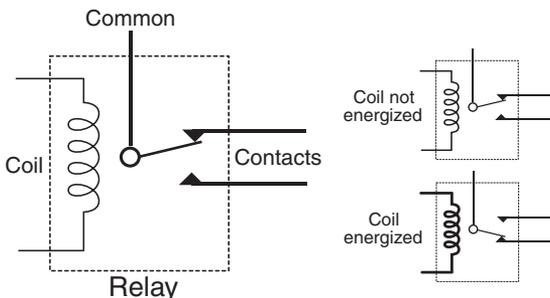


Figure 31-22 Relays are electrically controlled switches. Energizing a coil with current activates the switch contacts. Many relays have two contacts, in addition to a common connection. These contacts are marked normally open (NO) and normally closed (NC). The notation refers to when the relay is not energized.

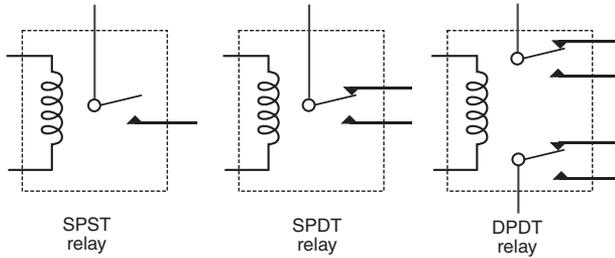


Figure 31-23 Three typical relay types used in robotics: single-pole, single-throw; single-pole, double-throw; and double-pole, double-throw.

SIZES OF RELAYS

Relays are “rated” by the amount of current that can be passed through their switch contacts. The more current, the larger the relay. For the typical desktop robot application, the smallest of relays are ideal: these are about the same size and shape as an integrated circuit.

Larger currents (loads) require bigger contact points, so if you’re operating a robot with a huge motor, you’ll need something more. As the relay gets bigger, properly operating it becomes much harder. For one thing, you need more drive current from your circuit to activate the relay. The standard setup of using a pea-sized signal transistor between your microcontroller and relay won’t cut it.

High currents can cause arcing and other unpleasant artifacts whenever the relay is energized and deenergized. Careless use of a heavy-duty relay can even result in the electrical contacts becoming welded together! These issues aren’t a problem as long as you stick with small relays and small motors—the stuff of the average desktop robot. Operating the motors of a big-brute combat robot is beyond the scope of this book, but there are guides and online sites that cover this topic. Do a Web search and visit your local library.

RELIABILITY OF RELAYS

Relays are electromechanical, and, because of this, over time they could wear out and fail. But, in truth, for the average small relay used on the average bot, you’d have to operate your little robotic pet for several *years* before the relay would give out.

Most relays are rated for 100,000 or more switches. The trick is to make sure your circuit doesn’t exceed the current rating of the relay contacts. The less complicated the construction of the relay, the longer it’ll last (and the cheaper it is to replace). The lowly reed relay (so called because the contact is a simple metallic reed inside a glass ampoule) can last for years and years.

. . . And the Rest

In the first season of the old TV show *Gilligan’s Island*, the characters of The Professor and Mary Ann didn’t even rate being mentioned in the opening theme song. Instead, they were referred to as “and the rest.”

The components that follow are part of “and the rest” when building robot electronics. These are important in their own right, to be sure, but are better left to examples in other chapters that show them in actual use. They include:

Speakers, to better hear your robot.

Microphones, to better your robot's hearing you (is that even a proper sentence?). See Chapter 46, "Making and Listening to Sound," for more on speakers and microphones.

Light-sensitive resistors, transistors, and diodes, to give your robot the gift of simple sight. These components are detailed more fully in Chapter 44, "Robotic Eyes."

LCD displays, to let your robot communicate with you in words. Read more about how to use these in Chapter 47, "Interacting with Your Creation."

On the Web: Stocking Up on Parts

So what parts should you get for your robotics lab? How many resistors, and what values, should you stock up on? That can be a tough question if you're just starting out. The details would kill too many trees, so as a bonus to the readers of this book, I've outlined some suggestions (with suppliers and parts numbers) on the RBB Online Support site, described in Appendix A.

Using Solderless Breadboards

Solderless breadboards let you experiment with circuits without having to solder together components. You literally plug in the parts and string some wires to complete the connections. You save time, and you get to play with your creation that much sooner.

And, just as important, you can easily change the value of components to see if another one works better. Just unplug the old, and switch in the new.

While solderless breadboards are intended for experimentation and testing, there's no law that says you must eventually solder your circuit together to make it permanent. Many robot builders construct the solderless version only and use it that way for weeks, months, even years.

But, of course, you always have the option of someday transforming your brightest *sans solder* ideas into gleaming soldered-together circuits. In this chapter you'll learn how to best leverage solderless breadboards in your robotics projects. The next chapter is devoted to making circuit boards using soldering and other more permanent electronics construction techniques.

Anatomy of a Solderless Breadboard

A solderless breadboard (see Figure 32-1) is made of plastic and is composed of many columns and rows. The columns are wired inside the board so that anything attached to that column shares a single electrical connection (see Figure 32-2). To use the board, you start by plugging in ICs, resistors, and other components. You then establish how they're interconnected by stringing wires from one column to another.

SIZES AND LAYOUTS

All solderless breadboards consist of holes with shared internal contacts. These contacts are spaced one-tenth of an inch apart—just right for integrated circuits, most transistors, and

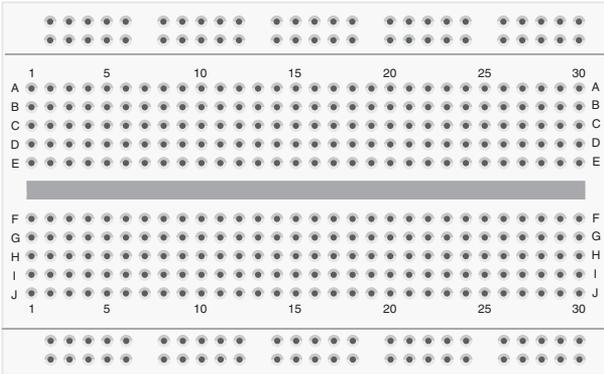


Figure 32-1 Solderless breadboard with 400 tie points.

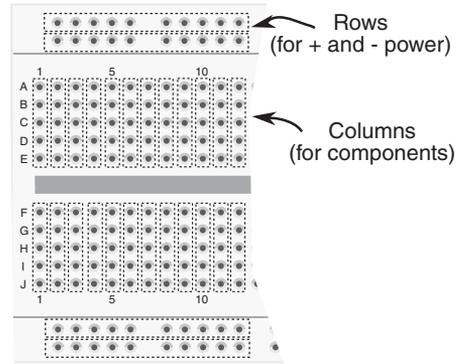


Figure 32-2 Underneath the holes in the breadboard are strips of conductive metal with spring contacts. Plugging a wire into a hole pushes the wire into the contacts.

discrete components like capacitors and resistors. The holes, called contact points or tie points, are the proper size for 22-gauge solid conductor wire. On each column you can connect together as many as five wires or components.

In addition to the column connections, many solderless breadboards have long rows on the top and bottom. These provide common tie points for the positive and negative power supply.

But from here, the size and layout of breadboards can differ greatly. Some measure a few inches square and contain only 170 holes. The 170-tie-point boards are made for simple circuits with one or two small ICs or a handful of small components. For bigger circuits you need a bigger breadboard. You can pick among models with 400, 800, 1200, even 3200 tie points.

Which size should you get? I believe in starting out small, the extra benefit being that small breadboards cost less. They're usually more than adequate for testing the typical robot control or sensor circuit. You can always get a larger board as your needs grow.

MAKING CONTACT

The contact points are usually made from a springy metal coated with nickel plating. The plating prevents the contacts from oxidizing, and the springiness of the metal allows you to use different-diameter wires and component leads without seriously deforming the contacts. However, the contacts can be damaged if you attempt to use wire that's larger than 20 gauge. (Remember: Wire gets bigger with *smaller* gauges.)



The more you use your solderless breadboard, the faster it'll wear out. After a while, the springy metal isn't so springy. Dust can settle inside the contact points and decrease the electrical contact. This is another reason not to invest in a large and expensive breadboard. The cheap ones don't cost as much when they need to be replaced.

Things you should *not* try to plug into a solderless breadboard contact point include:

Stranded conductor wire, which won't work reliably, even if you twist the strands.

Larger than 20-gauge wire or fat component leads.

Smaller than 26-gauge wire or really skinny component leads (the electrical contact will be iffy, at best).

High-voltage sources of any kind—these include wires from an AC wall socket or any circuit that uses high voltage. *Solderless breadboards are for low-voltage DC circuits only.*

CONNECTING WIRES FOR YOUR BREADBOARD

No solderless breadboard is complete without wires, but you can't use just any wire. The best wire for solderless breadboards is:

- 22-gauge
- Solid conductor
- Plastic insulated

You want wires of different lengths, with about 1/2" of the insulation stripped off each end. These *jumper wires* are available premade, or you can make them yourself. I prefer the pre-made kind.

If you decide to make your own set of breadboard wires, look for wire spool assortments with different colors. For starters, cut the wires into the following lengths:

Total Wire Length	Jumper Length	Quantity
1-1/4"	3/4"	10
1-1/2"	1"	15
1-3/4"	1-1/4"	15
2"	1-1/2"	15
2-1/2"	2"	10
3"	2-1/2"	10
3-1/2"	3"	5
4-1/2"	4"	5
6-1/2"	6"	5

Note: Jumper length assumes 1/2" of insulation is stripped off the wire on each end.

1. Start by cutting the wires to the total wire length, as indicated.
2. Use a pair of wire strippers to remove 1/2" of insulation off each end, as shown in Figure 32-3. While stripping the insulation, insert one end of the wire into the stripping tool (if it's adjustable, dial it for 22-gauge) and hold the other end with a pair of needle-nose pliers—the kind *without* serrated jaws is the best.
3. After stripping the insulation, use your needle-nose pliers to bend the exposed ends of the wire at 90°, as shown.

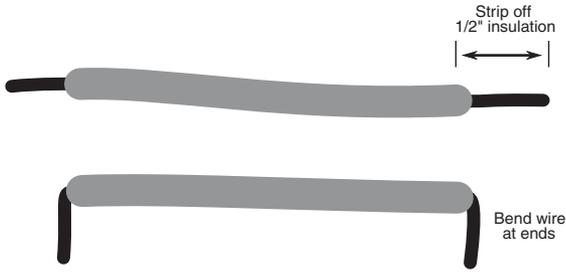


Figure 32-3 To make your own breadboard jumpers begin by stripping of 1/2" of insulation from both ends, being careful not to nick the wire (that weakens it and can cause the wire to break off easily). Finish by bending the wires at the ends.

USING PIN JUMPERS

Solid conductor wire can break when it's used too much. Due to metal fatigue, the wire just snaps off—sometimes right inside the breadboard contact point (use a small needle-nose pliers to remove these).

For longer-lasting wires, you want to purchase or make a set of jumpers made with stranded conductor wire, with soldered-on machine pin ends. These last much longer than regular solid conductor jumpers, though they're a lot more expensive if you buy them ready-made.



Check out the RBB Online Support site for a step-by-step pictorial on making your own pin jumpers. See Appendix A for more details about the support site.

MAKING PIN HEADERS FOR OFF-BOARD COMPONENTS

There are times when you want to use components—speakers, switches, potentiometers—that just won't fit into the holes of the breadboard. You can make pin jumpers for these, too. But instead of soldering a pin to each end of the wire, you solder a pin to one end, and on the other end you connect to the component. Figure 32-4 shows the general idea.

For components that you don't want to permanently solder to, use just a single header pin and an alligator or pushpin jumper wire. The pushpin type is smaller and wraps around the small-diameter pin a little better. Then connect the other end of the jumper to the component.

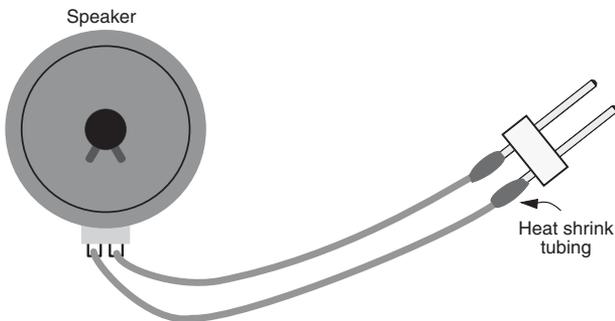


Figure 32-4 Example of a pin header permanently soldered to a component (a speaker shown here) that cannot by itself plug into the breadboard.

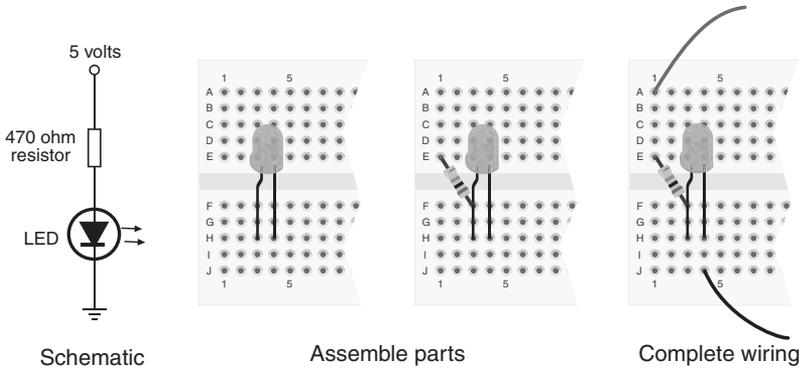


Figure 32-5 Steps for assembling a solderless breadboard circuit from a schematic. Start with the largest components, and insert them into the breadboard. Add smaller components, then attach connecting wires.

Steps in Constructing a Solderless Breadboard Circuit

Building a solderless breadboard project is merely a matter of placing the electronic components into the tie points and then completing the circuit using jumper wires. Start with a schematic or other plan, as shown in Figure 32-5.

1. Identify the components of the circuit and collect them before you begin. There's nothing worse than starting to build a circuit, only to discover you're missing some important part.
2. Visualize the best placement for the components. On simple circuits this is easy to do, as you'll have plenty of room. On more complex projects you'll want to carefully plan the best spot for each part, so that everything will fit.
3. Place the parts into the tie points of the breadboard starting from one end of the schematic and proceeding to the other. For the example circuit, the "ends" are the + and - power leads, but on others it could be a signal input or a sensor output. I've elected to start with the LED, as it's the largest of the components.
4. Finish the construction by adding any jumper wires that are needed, including the power leads.



Though the breadboard is constructed with columns and rows at right angles, there's no rule that says you must place parts in such rigid rank-and-file order. Feel free to turn parts this way and that, to take best advantage of the tie points.

Making Long-Lasting Solderless Circuits

Of course, you already know solderless breadboards are not "intended" for permanent circuits. But if your aim is to build a circuit on a solderless breadboard and keep it that way, you'll want to ensure a solid construction. Bird's-nest wiring and loose components just won't cut it.

Use a new board, one where the spring contacts aren't already getting loose. The low price of a small breadboard means you can keep a stock of these and use them as needed.

Firmly seat all components, including resistors, capacitors, and diodes. This means you need to trim the lead lengths so that the body of the component sits flush with the surface of the board.

Cut jumpers to length, and carefully route them around components and other jumpers so that none will accidentally pull out. Push the jumpers flush against the board.



To help prevent parts and wires from coming loose, strap them in using some rubber bands. Avoid the use of things like cling plastic wrap for storing food. This generates lots of static electricity when you apply and remove it, possibly ruining your circuit.

Mounting the Breadboard to Your Robot

If you're using the breadboard directly on your robot, you'll want to securely mount it to keep it in place. Otherwise it may fall off the next time your bot does a pirouette turn, and all your hard work will be ruined as parts scatter onto the floor.

If the breadboard is readily accessible—so you can work on it—you can simply use double-sided foam tape to secure it to the robot (see Figure 32-6). Otherwise, use Velcro or other hook-and-loop material, so you can peel the board off to work on it.

When double-sided tape and Velcro won't work, use long cable ties to hold the board in place. Use two wraps, one at each end. Cinch the tie for a secure fit. If you need to later remove the board (so you can work on it on your bench), just cut the ties and discard. Use new cable ties when you're ready to remount the board on your robot.

Some breadboards, such as the Global Specialties EXP-350, have mounting holes in the corners. The holes are usually pretty small, so you need 4-40 or maybe 2-56 miniature screws and nuts. The EXP-350 accepts 4-40 machine screws and nuts; use flathead screws for a flush look. Or it accepts 6-32 screws from the back, which tap straight into the plastic.

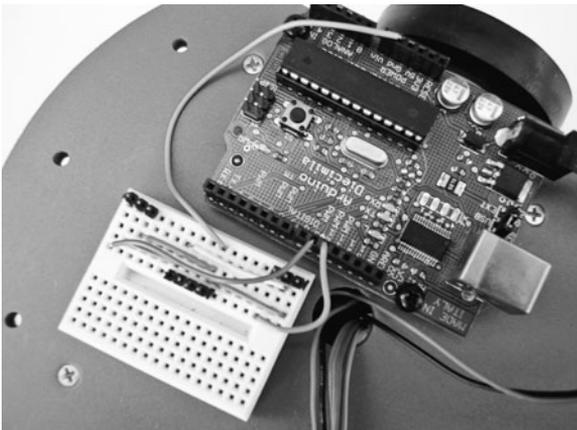


Figure 32-6 A breadboard mounted on top of a robot, alongside a microcontroller. Jumper wires connect the microcontroller to the breadboard, which serves as a way to quickly and easily experiment with different robot sensors.

Tips for Using a Solderless Breadboard

Here are some tips for solderless breadboard success:

- When using static-sensitive components like CMOS integrated circuits or MOSFET transistors, build the rest of the circuit first, using “dummy” parts to make sure you wire everything properly. When you are ready to test the circuit, remove the dummy chip and replace it with the CMOS parts.
- When inserting wire, use a small needle-nose pliers (no serrated jaws please) to plug the end of the wire into the contact hole. Use the pliers to gently pull it out of the hole when you are done with the breadboard.
- Never expose a breadboard to high heat, or the plastic could be permanently damaged. ICs and other components that become very hot (because of a short circuit or excess current, for example) may melt the plastic underneath them. Check the temperature of all components while the circuit is under power.
- Use a chip inserter and extractor to implant and remove ICs. This reduces the chance of damaging the IC during handling.
- Avoid building a bird’s nest, where the connection wires are routed carelessly. This makes it harder to debug the circuit and it greatly increases the chance of mistakes.
- Remember! Solderless breadboards are designed for low-voltage DC circuits *only*. They are not designed, nor are they safe, for carrying AC house current.

Making Circuit Boards

You've built your robot, and it's ready to rumble. You've tested its control electronics on a solderless breadboard, and everything's a go. But now you want to commit the breadboard electronics to a permanent circuit board. Your bot will be more ruggedized, and it'll withstand your nephew's pawing (don't blame him for loving robots as much as you do!).

When it comes to making circuit boards, you've got plenty of choices. In this chapter you'll learn about the most popular (and most affordable) circuit-board-making techniques. All are suitable for the amateur and educational robot builder, though those involving dangerous chemicals are ill-advised for children, even with adult supervision.

Overview of Your Primary Circuit Board Options

Before getting down to the nitty-gritty here's a quick overview of your best options:

- *Solder breadboards.* These mimic the solderless breadboards you read about in Chapter 32, but these are made for permanent soldering.
- *Unplated perforated boards.* Old-fashioned but still useful for very simple circuits, these are boards with holes already drilled in them. You wire up components directly.
- *Plated stripboards.* Same idea as above, but these are plated in different grid styles to avoid wiring directly to component leads. Easy to work with and cheap. Lots of variety.
- *Quick-turn printed circuit boards.* Design a printed circuit board (PCB) on your computer, then send it out for manufacturing. Less expensive than you may think.
- *Home-etched PCBs.* Using a strong chemical, you can etch the pattern of your own PCB onto a copper-plated board. Time-consuming and messy, but can be a good learning experience.

- *Custom prototyping boards.* Some components with “rock star” status (most notably, microcontrollers like the PICAXE, AVR, and PICMicro) have various custom prototyping boards available for them. Take your pick.
- *Wire wrapping.* Semipermanent construction using very thin wire between components. You can undo the wires to make changes.

There are other methods that I’ll be skipping here. Many are out of date, are specific to one manufacturer, require hard-to-find parts, or may require special tools.

Clean It First!

Circuit boards use a thin layer of copper to form *traces*, the wires of the circuit. Copper can oxidize and get dirty over time, both of which can lead to a poor solder joint. No matter which circuit-board-making method you use, prior to any soldering *be sure to thoroughly clean the board* using warm water and ordinary kitchen cleansing powder. You can scrub the board using your fingers, a folded-up paper towel, or a nonmetallic scouring pad.

After cleaning, rinse all of the cleanser off the board. Pat it dry with a paper towel, then let it air-dry—it takes a couple of minutes. For a superclean board, wet a cotton ball with household isopropyl alcohol and give the metal a final wipedown. Allow all of the alcohol to evaporate.

Making Permanent Circuits on Solder Breadboards

Akin to the solderless breadboard is the *solder breadboard*, where you can make permanent any design you create on a solderless breadboard.

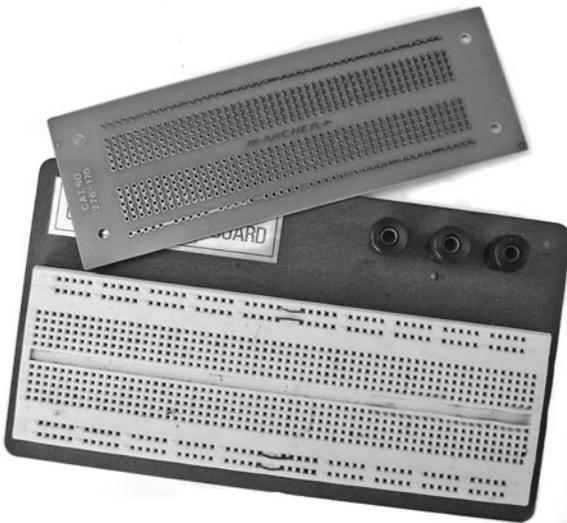


Figure 33-1 A solderless breadboard and a companion solderboard. Test your circuits on the solderless breadboard, and once the design is finished, debugged, and working, transfer the components to the solderboard following the same arrangement.

The solderboard comes pre-etched with 550 tie points. Circuits may be designed on a solderless breadboard, then transferred to the solder board when you're sure everything is working to your liking. Simply solder the components into place following the same design you used on the solderless board. Use jumper wires to connect components that can't be directly tied together. Figure 33-1 shows both a solderless and a solder breadboard.

Small circuits take up only a portion of the solder breadboard. You can cut off the extra using a hacksaw or razor saw. (But beware of the "sawdust" from these boards; it's not healthy for you, so don't inhale or ingest any of it.) Leave space in the corners of the board to drill new mounting holes, so that you can secure the board inside whatever enclosure you are using. Alternatively, you can secure the board to a frame or inside an enclosure using double-sided foam tape.

If you're constructing small circuits, one solderboard will last for several projects. In time, you'll learn how to conserve space to make good use of the real estate on a solderboard.

Using Point-to-Point Perforated Board Construction

An alternative to the solder breadboard is *point-to-point perforated* (or "perf") circuit board construction. This technique refers to mounting the components on a predrilled board and connecting the leads together directly with solder. Perf boards are basically just blank pieces of phenolic or other plastic, with holes drilled every 0.1". This is the correct spacing for standard integrated circuits, and it works well for other components.

For robot electronics, point-to-point perf board construction is best used—if at all—for very simple circuits containing just a few components. You use the board as a kind of structure for the electronic parts.

BUILDING A CIRCUIT ON A PERF BOARD

Figure 33-2 shows the concept behind using a perf board—again, it's merely a board with holes already drilled into it. Basic construction goes like this:

1. Cut the board to the size you'll need for the circuit. You'll need to estimate the amount of board space.
2. Insert one component at a time through the holes of the perf board. Bend the leads on the opposite side to keep the component from falling out.

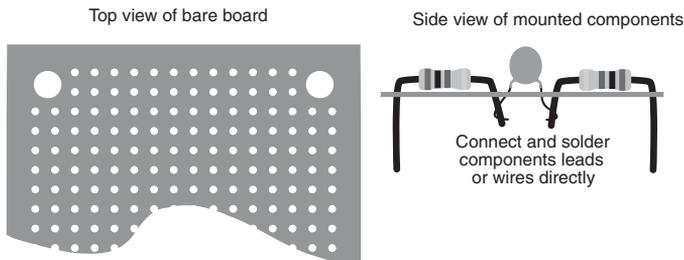


Figure 33-2 When using a perforated (perf) board, you directly connect components together or solder component leads with insulated jumper wires.

3. For components that are right next to one another, wrap leads that connect. One or two turns is enough.
4. For connecting components that are more than an inch or two from one another, use 30-gauge wire meant for wire wrapping (see the section on wire wrapping, later in the chapter), strip off about 1/2" of the insulation from the end of the wire, and hook it around the component lead.
5. Use a soldering pencil to flow some solder to the wires.
6. Repeat until all the components are mounted and wired together.
7. Trim off any excess wire lengths to avoid short circuits. Double-check your work.



Perf board wiring is often referred to as *bird's-nest* construction—you can guess why. For circuits using more than a couple of components, use a copper layout board, detailed next. Since circuits on point-to-point perf boards are delicate, mount them on your robot using one or two small fasteners or a cable tie. Avoid having the board just flop around loose.

ALTERNATIVES TO PERF BOARD MOUNTING

A variation on the theme of point-to-point wiring on a perf board is to simply do away with the perf board. Over the years, a variety of techniques have been developed; here are some of them:

- *Dead bug* wiring is sometimes used with a single IC connecting to just a few components. It's called "dead bug" because when the chip is turned upside down, and things are soldered to its pins, it looks a bit like a little, black, dead bug. When soldering on resistors, capacitors, and similar components, cut the lead to length, then bend the end to make a U shape. Solder the U directly to the IC pin.
- *Wire wrap* sockets have extra-long pins for attaching to wires. The IC itself plugs into the socket after construction is complete. As with dead bug wiring, you can form fairly strong solder joints by bending the end of the component lead into a U and hooking it around the socket pin.
- *Lead-to-lead* construction is suitable when you're soldering the leads of one discrete component to another—for example, a resistor to an LED. Prepare the leads of both components by cutting them to the desired length, then form small hooks to make a good mechanical joint. Solder the two together.

Using Predrilled Stripboards

A stripboard has holes drilled in it, just like a perforated board. But on at least one side of the board is a series of copper metallic pads and/or strips that run through the center of the holes. These boards come in a variety of sizes and styles. All are designed for use with ICs and other modern-day electronic components. One application of the stripboard is circuit construction using wire wrapping, as detailed later in this chapter. But many circuits can be soldered directly onto the board.

Choose the style of grid board depending on the type of circuit you are building. Figure 33-3 shows a selection of basic grid layouts. Variations include:

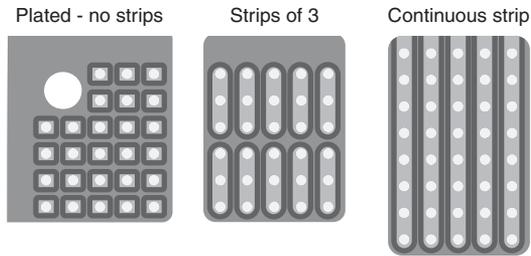


Figure 33-3 Example grid layouts for perforated stripboard. Select the layout that best matches the circuit you're building and your construction preferences.

Plated holes without strips: These are used like perforated boards, previously described, but you can solder component leads to the copper plating around the holes. You then complete the board by attaching wires to the component leads.

Continuous strips that run the length of the board: To make a circuit, solder in components and wires, then use a sharp knife (or a specialty *spot face cutting* tool) to break the strip after all the connections are made for that particular part of the circuit.

Strips in groups of three to five holes: Rather than have you cut a long strip into smaller segments, these boards have done it for you. Each segment spans from three to five holes (sometimes more). Some boards have additional strips running perpendicular to the segments; these are common *busses* for easy connection to positive and negative terminals from the power supply.

Personally, I prefer boards with the three-hole segment plan. Components are tied together using three-point contacts. If you need to connect more components to a single point, you can link multiple segments together by bridging them with a piece of wire. Those boards with the extra strips for positive and negative terminals are ideal for circuits that use many integrated circuits, as it simplifies connection to power. The interleaving of the power supply rails also helps reduce electrical noise.



Always use either the plated copper segments to bridge components together or else a length of bare wire. Avoid the temptation of using a big blob of solder to bridge segments together.

Creating Electronic Circuit Boards with PCB CAD

Thinking about making your own printed circuit boards? Here's one way: start by laying out your circuit on paper, then transferring that as a *photoresist* to a piece of copper-cladded fiberglass sheet. You then mix up a nasty (and toxic) soup of ferric chloride and dip the board into the liquid for 20 minutes, hoping none of it gets on your clothes. You then painstakingly drill all the holes and—voila!—your printed circuit board (PCB) is ready for soldering.

Or you could just prepare the board using free design software, then click a Send button. In a few days you get back a professionally produced single- or double-sided PCB through the mail. For just a few dollars more, your custom PCB board can even include silkscreen printing, where all the parts are indicated in white ink.

Automated printed circuit board manufacturing is a kind of quick-turn prototyping. In Chapter 15, "Drafting Bots with Computer-Aided Design," you learned how quick-turn pro-

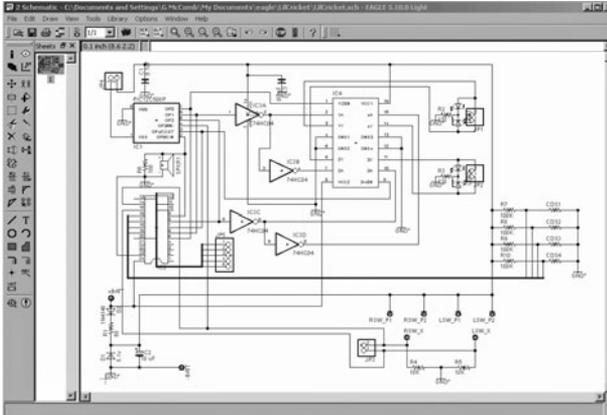


Figure 33-4 Schematic for *L'il Cricket*, drawn and executed in CadSoft Eagle.

otyping can build mechanical parts out of metal or plastic. When the concept is applied to circuit boards, you get a two-dimensional PCB.

Many PCB service bureaus (companies that make PCBs for you) rely on their own proprietary electronic computer-aided design (ECAD) software to lay out the board. If the service bureau doesn't offer an ECAD program you can use, you'll need to get one compatible with their system. The free version of CadSoft's Eagle program is one popular option; it's limited to laying out boards measuring up to 4" by 3.2". If you need to build larger PCBs, opt for the full version of Eagle.

Figure 33-4 shows the schematic for *L'il Cricket*, a buglike robot designed in Eagle. The robot measures about 5" × 3", where its PCB is also its body.

Figure 33-5 shows a completed PCB layout. I placed the components where I wanted them to be, and Eagle *autorouted* (figured things out by itself) the connecting points between everything, based on the schematic that I previously prepared. This is a two-sided board, so there are copper traces connecting components on both the top and bottom. Note that the PCB view also includes a *silkscreen* layer, which contains the outlines and labels of the parts. Read more about this feature in the following section.

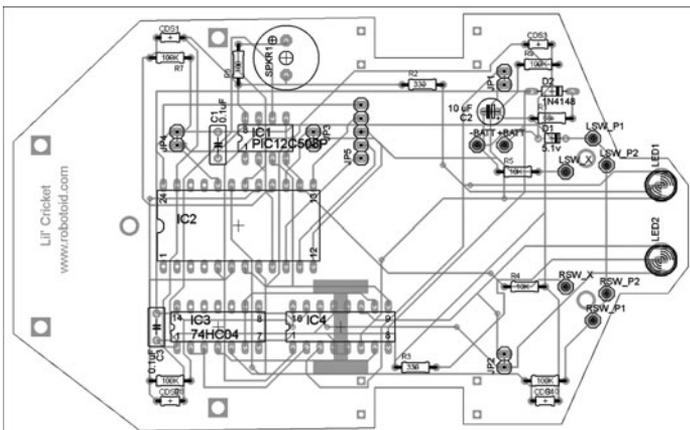


Figure 33-5 PCB layout for *L'il Cricket*, made by Eagle. The layout is ready to be sent to a PCB service bureau for production. To create the layout, the components were placed where I wanted them, and then Eagle autorouted the connections following the schematic I had already prepared.

Whether you use Eagle or some other software, here is the basic process for designing and making your printed circuit board:

1. Create a schematic of your circuit. The schematic uses symbols for all electronic parts and lines to indicate how everything is connected together.
2. Create the PCB layout by dragging the outlines of your parts onto a template of the size of board you want.
3. Convert the schematic and the PCB layout. Finish the PCB design by linking components that need to be connected.
4. If you are having a service produce the board, order the board by submitting it. Payment is collected up front. Most services will prepare two or three boards for the base price.

PRINTED CIRCUIT BOARD DETAILS

Following are concepts you should know about when making printed circuit boards:

Pads, traces: PCBs are composed of pads and traces. The metal leads of the components are soldered to *pads*, and the pads are connected together via *traces*. The traces are the interconnecting wires of a PCB.

One or two sides: All printed circuit boards have two sides, but you may elect to have pads and traces on one side only. It's a little cheaper that way.

Layers: Each side of the PCB can be separated into individual layers. The layers are insulated from one another by a protective film, so that the pads and traces in one layer don't short-circuit into those of another layer. Your quick-turn PCB may have two, four, six, even eight layers. The more layers, the more expensive it is.

Soldermask: If you choose to order it as an option, a soldermask applies an insulating lacquer over your board that covers everything except the pads to which components are to be soldered.

Silkscreening: Another option is the *silkscreen legend*, which marks where the components go. You can include the outline shapes of the components, component values, part number references, even limericks. It's completely up to you.

Producing Arduino-Specific Boards with Fritzing

Fritzing (available at www.fritzing.org) is a nifty, free ECAD program designed for creating printed circuit boards. What sets it apart is that it is designed from the ground up to support the open-source Arduino, one of the key microcontrollers used in projects throughout this book.

By connecting wires on a virtual solderless breadboard, your circuit designs are automatically converted to schematics *and* printed circuit board layouts. These layouts are meant to be Arduino *shields*, circuit boards that attach on top of the Arduino and extend its functionality. (The Arduino and shields are detailed in Chapter 37, "Using the Arduino.")

Figure 33-6 shows one of the example projects that come with the Fritzing software, a step-motor experimenter board. Simply by clicking tab buttons inside the Fritzing program, you can view the circuit as a schematic (see Figure 33-7) and even as a Arduino shield PCB.

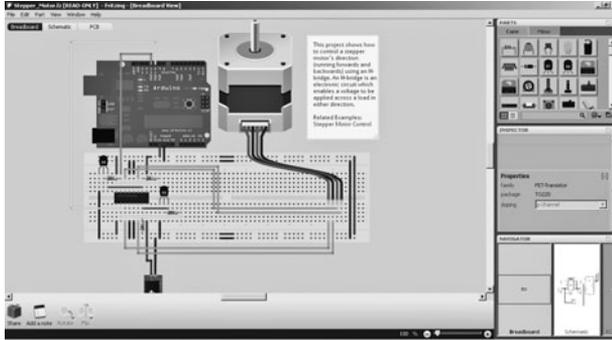
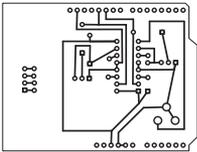


Figure 33-6 Virtual breadboard view of an Arduino project created in the Fritzing ECAD software. This is one of the projects that comes with the free Fritzing software.

You can export the PCB layout as an *etchable PDF*, then print the PDF using special film that can then be transferred to a bare copper-clad circuit board. This involves immersing the prepared board in a caustic chemical to remove the extra copper, and then drilling out holes for the components.



Other board-making options using Fritzing include exporting the design to an Eagle file for use with the CadSoft Eagle PCB software, mentioned earlier. Or you can convert the layout to a series of *Gerber files* that contain all the information that an automated PCB quick-turn service can use to prepare your board. The files generated by Fritzing specify the actual pattern layout, soldermask, silkscreen, and hole sizes that are to be used.

Fritzing will even generate a “bill of materials” (BOM) file for you, listing all the components used in the project. The BOM file is stored as text, but it’s in Unicode format. Microsoft Word can read a Unicode-encoded file, but Windows Notepad and many other text-only editors won’t know what to make of it. It’ll just look like gibberish.

On the Web: Etching Your Own Printed Circuit Board

So far, I’ve talked about designing a PCB using a layout program, then shipping the files off to a service that makes the board for you. By far, this is the most convenient approach; but it’s also the more expensive route. It’s cheaper (in terms of money) if you make the board yourself.

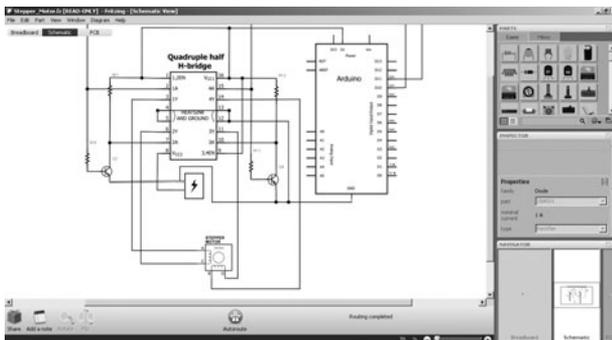


Figure 33-7 From the breadboard view, Fritzing can automatically generate a schematic diagram. You can adjust the drawing to fine-tune it.

The process of making your own printed circuit boards is not complex or even difficult, but it is time consuming, and it does involve using some nasty chemicals that cause stains on clothing and burns on skin. With the advent of low-cost PCB service bureaus, making your own circuit boards is something of a dying artform. If you're interested in the techniques used to make your own printed circuit boards, check out the bonus tutorials about it on the RBB Online Support site (see Appendix A).

Using Custom Prototyping Boards

The popularity of several brands of microcontrollers has spawned a kind of cottage industry in prototyping (or *proto*) boards custom-made for them. Proto boards are empty PCBs with layouts and predrilled holes to accommodate a wide variety of projects.

You start by soldering the microcontroller into its spot (better yet, use an IC socket, then plug the controller into the socket), then complete the board by adding other components. Many prototyping boards have a reserved section for general experimenting, where you can add your own circuits to the basic one already there. Some proto boards are even made with mini solderless breadboards stuck to them.



Some proto boards already have the microcontroller soldered to them. In this case, the controller may be in a more compact format than the typical dual in-line package (DIP) you're most familiar with. This makes using the board very convenient, as the controller is already built in.

The disadvantage is that if the MCU is not in a socket, any damage to the chip means the entire prototype board is a loss. My preference is to use a prototyping board that accepts the standard DIP IC packages, and then use sockets to allow easy swapping of controllers.

Making Semipermanent Circuits with Wire Wrapping

With wire wrapping there's no soldering (or at least only very little); instead, plug a special socket into a perforated board, and then wrap thin wire around the pins of the socket using a special tool. The advantage of wire wrapping is that it's relatively easy to make changes. Just unwrap the wire and reroute to another post.

Wire wrapping is commonly used in IC-intensive circuits. The construction technique doesn't promote fast results, but because soldering is usually not involved it's a safer bet for kids interested in learning electronics. And, of course, mistakes are easier to correct than when using solder construction.

HOW WIRE WRAPPING WORKS

To wire wrap, first mount a wire wrapping socket into the perforated board. Sockets are available for all the common sizes of DIP-style ICs. I prefer to use a perf board with copper pads around the holes so that I can apply a tiny bit of solder to at least one of the pins of the socket. This keeps the socket from falling out while I'm working on the board.

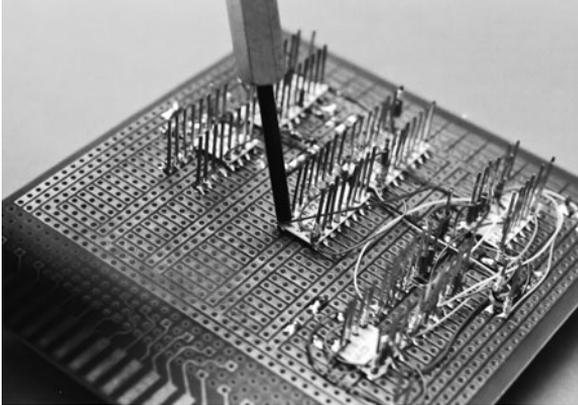


Figure 33-8 Wire wrapping creates circuits by literally wrapping wire around metal posts.

Wire wrapping sockets use extra-long square-shaped pins (also called *posts*) that can accommodate up to about five wrapped wires. Using a special tool, as shown in Figure 33-8, each wire is wrapped around the pin like figures on a totem pole. Once the wire is wrapped about a half-dozen times around the pin, the connection is as solid and secure as a soldered joint.

CHOOSING A WIRE WRAPPING TOOL

You need a special tool for wire wrapping. A manual wire wrapping tool is shown in Figure 33-8. To use it, you insert one end of the stripped wire into a slot in the tool, then place the tool over a square-shaped wrapping post. Give the tool 5 to 10 twirls, and the connection is complete. The edges of the post keep the wire anchored in place. To remove the wire, use the other end of the tool and undo the wrapping.

Other tools are motorized, and some automatically strip the wire for you, which frees you of this time-consuming task (or the need to purchase the more expensive prestripped wire). I recommend that you start with a basic manual tool. You can graduate to other tools if you find wire wrapping suits you.

Successful wire wrapping takes practice. Before you build your first circuit using wire wrapping techniques, try your hand on a scrap socket and board. Visually inspect the wrapped connections and look for loose coils, broken wires, and excessive uninsulated wire at the base of the post. Most wire wrap tools are designed so one end is used for wrapping wire and the other end for unwrapping. Undo a connection by inverting the tool, and try again.

WIRE: SPOOLS OR PRECUT

You can purchase 30-gauge wire for wire wrapping by the full spool, but unless you're going into full-fledged production, a far easier method is to get precut wire in a couple of different lengths—2" and 4" lengths make for a good start (that's the length of the wire not counting the bare ends).

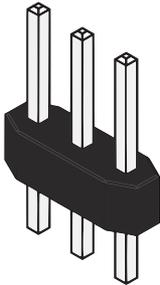
When using spools, you must cut the wire to length, then strip off the insulation using the stripper attached to the wrapping tool. A stripper for regular wire does a poor job because the insulation on wire wrapping wire is extra tough. These steps greatly add to the time it takes to make a single wire wrapped joint.

When you use the precut or prestripped packages, the work is already done for you. Buy a selection of different lengths, and always try to use the shortest length possible. Precut and prestripped can be expensive (\$5 or more for a canister of 200 pieces), but it will save you a great deal of time and effort.

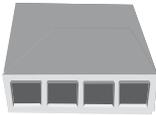
Effective Use of Plug-in Headers

Nothing in life is certain but death, taxes, and robot circuit boards with too many wires on them. You can take the easy route and just solder all the wires directly to the robot's circuit board, but a better approach is to take a cue from automobile manufacturers: use lots and lots of connectors.

These three methods are the most popular:



Use male pin headers on the board, and attach female connectors to the incoming wires. This works best when you already have wires with the requisite connectors on them; otherwise, you need to make the connectors, which requires a special crimping tool.



Use female pin headers on the board, and attach male connectors to the incoming wires. With this method you can make all kinds of connectors for plugging in just about anything.



Terminal blocks are specifically designed to carry more current than pin headers. Solder the block to the board, then securely attach wires to the block using the included screws. You can get blocks with one or more terminals on them. Use these for attaching to large batteries, heavy-duty motors, and any other component that requires the use of heavier-gauge wires.

Computers and Electronic Control

This page intentionally left blank

An Overview of Robot “Brains”

Brain, brain, what is brain?” If you’re a Trekker, you know this is a line from one of the original *Star Trek* episodes of the 1960s, entitled “Spock’s Brain.” The quality of the story notwithstanding, the episode was about how Spock’s brain was surgically removed by a race of women who needed it to run their air-conditioning system. Dr. McCoy rigged up a gizmo to operate Spock’s brainless body by remote control.

“Brains” are what differentiate robots from simple automated machines—brainless Spocks who might as easily crash into walls as move in a straight line. The brains of a robot process outside influences, such as sonar sensors or bumper switches; then, based on their programming or wiring, they determine the proper course of action.

A computer in one form or another is the most common brain found on a robot. A robot control computer is seldom like the PC on your desk, though robots can certainly be operated by most any personal computer. And, of course, not all robot brains are computerized. A simple assortment of electronic components—a few transistors, resistors, and capacitors—are all that’s really needed to make a rather intelligent robot. Hey, it worked for Mr. Spock!

In this chapter we’ll review the different kinds of “brains” found on the typical amateur robot, including the latest microcontrollers—computers that are specially made to interact with (control) hardware. Endowing your robot with smarts is a big topic, so additional material is provided in Chapters 35 through 41, including individual discussions on using several popular microcontrollers, such as the Arduino and PICAXE.

Brains for the Brawn

Let’s start by reviewing the six principal ways to endow your Scarecrow robot with a brain—no Wizard of Oz required here; it’s all done with bits of wire and other parts.

Human control: Some very basic robots are controlled by human interaction. Switches on the robot, or in a wired control box, let you operate your creation by hand. Human

control is an ideal way to learn about how robots operate. Examples include robotic arms, where you control the movement of various arm joints.

Discrete components: In years past, the typical robot brains used basic electronic components like the resistor and transistor (and even before that, tubes!). Now, with inexpensive microcontrollers, these kinds of circuits aren't seeing as much use, but they're still ideal for simple robots with simple jobs to do.

Microcontrollers: A microcontroller is computer-on-a-chip, with a “thinking” processing unit, memory, and means to connect with the outside world. Microcontrollers are the ideal form of robot brain because they're simple, cheap, and easy to use. You program them from your personal computer.

Onboard computers: Some bots need more computational power than microcontrollers can provide. Most any computer that doesn't weigh more than you do and require massive amounts of power to operate is a candidate for use as a robot brain. Laptops, netbooks, and computers with compact main boards are among the best choices as Robo Brainiac.

Remote computers: While the typical amateur bot is self-contained, there's no technical reason it can't be controlled by a computer located someplace else. This is typical of “teleoperated” robots, like those used by the military or in police bomb disposal. The computer is connected to the robot via wire, radio waves, or some other means.

Smart phones, tablets, and PDAs: Some consumer gadgets like mobile phones and personal data assistants can be pressed into service as robot controllers. The ideal device has a USB or other standard communications interface and an open architecture to allow you to write programs for it. One possibility: the mobile phones that use the Android operating system.

Igor, Pull the Switch!

I'm a believer in starting out simple. And it doesn't get any simpler than using manually operated switches to control a bot. While this is not a true “robot” in the formal sense of the word, it's a useful way to discover how robots work. By manually operating the robot with your own hands, you learn how it has to be done via fully electronic control.

Adding switches to operate a basic robot is easy. I prefer putting the switches and battery power in the same handheld remote—fewer wires that way. The basic RBB Bot in the *My First Robot* lessons (see the RBB Online Support site) uses a piece of picture frame mat board to hold two switches and a standard battery holder. You can get fancier and build a control box out of a real box. There are a number of project boxes just the right size for use as a switch-operated remote.

In operation, the switches are wired so you can start and stop the motors and control their direction. By changing the direction of one or both motors, you learn how to maneuver the robot around a room.

Brains from Discrete Components

In the world of electronics, *discrete components* are parts like transistors, resistors, and basic-building-block integrated circuits. These components, used in some clever combination, can produce a working, thinking brain of a basic robot.

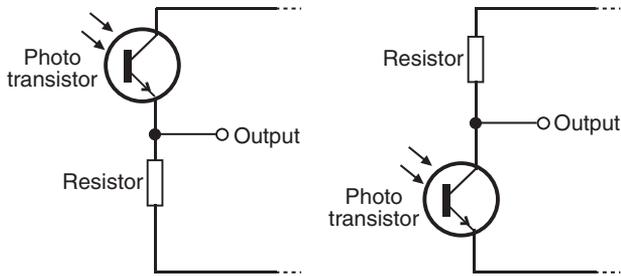


Figure 34-1 A few parts from basic electronic components form a workable robotic brain. Depending on how the parts are configured, the robot can display different “behaviors.” In one variation of this light-detection circuit, the robot responds to light; in the other variation, the robot responds to darkness.

Figure 34-1 shows a common form of robot brain made from simple parts. Wired one way, this brain makes the robot reverse direction when it sees a bright light. The circuit is simple, as is the functionality of the robot: light shining on the photodetector might turn on a relay or actuate some other circuit.

Variations of this circuit could make the robot stop when it sees a bright light. By using two sensors, each connected to separate motors, you can make the robot follow a bright light source as it moves. By simply reversing the sensor connections to the motors, you can make the robot behave in the opposite manner: it steers away from the light source, instead of driving toward it.

You could add additional simple circuitry to extend the functionality of robots that use discrete components for brains. For instance, you could use an LM555 timer as a time delay: trigger the timer and it runs for 5 or 6 seconds, then stops. You could wire the LM555 to a relay so it applies juice only for a specific amount of time. (This is the basic function of the enhanced RBB Bot in the *My First Robot* lessons.) In a two-motor robot, and using two LM555 timers with different time delays, your robot can be made to steer around things.

Programmed Brains

Perhaps the biggest downside of making robot brains from discrete components is that because the brains are hardwired as circuitry, changing the behavior of the machine requires considerable work. You need to either change the wires around or add and remove components. Using a solderless breadboard (see Chapter 32) makes it easier to try out different designs simply by plugging in components. But this soon gets tiresome and can lead to errors because parts can work loose from the board.

You can “rewire” a robot controlled by a computer simply by changing the *software* running on the computer. For example, if your robot has two light sensors and two motors, you don’t need to do much more than change a few lines of *programming code* to make the robot come toward a light source rather than move away from it.

TYPES OF PROGRAMMABLE GRAY MATTER

An almost endless variety of computers and computer-like devices can be used as a robot brain. The five most common are:

- *Microcontroller.* A microcontroller is programmed in either assembly language or a high-level language such as Basic or C. Figure 34-2 shows a Parallax BOE-Bot robot kit, which

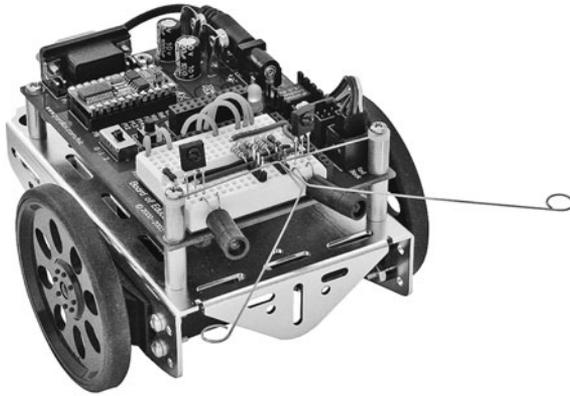


Figure 34-2 A programmable controller uses software rather than specific wiring to determine how the robot reacts. This BOE-Bot robot kit from Parallax uses the BASIC Stamp 2 microcontroller, which reads the value of sensors and applies power to the robot’s two motors accordingly. (Photo courtesy Parallax Inc.)

is driven by a BASIC Stamp microcontroller. See Chapter 35, “Understanding Microcontrollers,” for more information on this topic.

- *Single-board computer.* This is also programmed but it generally offers more processing power than a microcontroller. It is more like a miniature personal computer. For example, you can get a single-board computer based on the Intel Pentium and have it run versions of Windows or Linux.
- *Personal computer.* Old-school examples include the IBM PC and compatibles, but as these machines get older, they’re harder to find. Plus they consume lots of power because their electronics weren’t as efficient as they are today. Better choices are laptops and net-book computers.
- *Motherboard for a compact personal computer.* The main board, or *motherboard*, of a Mini-ITX PC measures under 7” square, yet it can run any modern operating system (OS) designed for the Intel processors, including Windows and Linux. The idea is that, rather than using a whole computer on your robot, you can attach just the motherboard.
- *Smartphone, tablets, and personal data assistants.* If you already have the processing power in a device you use every day, like your mobile phone, why not use it to make a robot? That’s the idea behind powering bots using personal consumer electronics. While great in theory, the practicality of pulling it off depends on the architecture of the device. Some products are more amenable for use as robot electronics than others.

As noted, microcontrollers are covered in their own chapter; let’s take a closer look at the other options.

SINGLE-BOARD COMPUTERS

Single-board computers (SBCs) are a lot like “junior PCs” but on a single circuit board. While there are some that can run versions of Windows (typically Windows CE or the embedded version of Windows), many are engineered for an operating system that consumes less disk and memory space, such as old-fashioned DOS or Linux.

SBCs are full-blown computers in every way, except that all the necessary components are on one board. Because of their design, SBCs can support megabytes, and even gigabytes, of program and data storage (most microcontrollers are limited to kilobytes of storage space). Whether you need a lot of storage depends on your application, but it’s nice to know the SBC can support it if you do.



If your SBC doesn't come with an operating system, you need to provide one. DOS is a good all-around choice for robots that don't need the bells and whistles of Windows. Microsoft no longer sells its once venerable MS-DOS, which at one time was packaged with almost all PCs. There are many free open-source alternatives, such as FreeDOS and DR-DOS, available for download from the Web.

You can also sometimes find surplus 3-1/2" diskettes of MS-DOS 5 and 6 in the dusty-shelf section of old surplus stores. You need a 3-1/2" disk drive to read the content of the disks and transfer it to the solid-state memory used by most SBCs.

SBC Form Factors

Single-board computers come in a variety of shapes and forms. A standard form factor supported by many manufacturers is *PC/104*, which measures about 4" square. PC/104 gets its name from "Personal Computer" and the number of pins (104) used to connect two or more PC/104-compatible boards together.

SBC Kits

To handle different kinds of jobs, SBCs are available in larger or smaller sizes than the 4" by 4" PC/104. And while most SBCs are available in ready-made form, they are also popular as kits. For example, the HandyBoard, designed by instructors at MIT, is a single-board computer based on the Motorola 68HC11 microcontroller. It is available already assembled or as a kit.

PERSONAL COMPUTERS

Having your personal computer control your robot is an admirable use of available resources, but it's not always practical if you're planning on mounting the thing on top of good old Tobor (that's robot spelled backward). The old-style desktop PC is simply too heavy, bulky, and power hungry to be an effective source of brains for your bot.

There are two ways to use a PC to control your robot:

- *Brains on bot.* Mount the computer on the robot. For a laptop, you can rely on its internal battery. But for a desktop PC meant to be plugged into the wall, you'll need to either run the computer using a large 12-volt battery and car power inverter or retrofit the computer with a power supply that can be juiced directly from the battery.
- *Brains off bot.* You use any kind of computer and link it to your robot via wires, radio frequency (RF) link, or optical link. This is common practice when using tabletop robotic arms; since the arm doesn't scoot around the floor, you can place it beside your PC and tether the two via wire. A USB connection is a favorite tethering technology—not to mention inexpensive and easy to use. There's also Bluetooth, Zigbee, and other types of radio links if you don't want the wires.

Using an On-Bot PC

You have quite a few options for mounting a PC on your robot.

Laptops and Netbooks

The ideal computer for onboard brains is a laptop running your favorite operating OS. Laptops carry their own rechargeable batteries and are made to be lightweight. You can use your regular laptop or else purchase one specially for your bot. You don't need a new laptop; find one used and save a few bucks.

A netbook is an even smaller lightweight alternative. Many run a version of Windows or else a proprietary operating system. Be careful with netbooks with a proprietary OS, as you may not be able to write programs for use with your robot. Like laptop PCs, netbooks are battery powered.

Mini-ITX PC

And yet another option is the Mini-ITX PC, so called because it uses a mini-ITX motherboard. Mini-ITX is not a brand but technically a “form factor” and design architecture. The board measures 6.7” × 6.7”, and many versions don’t need a fan for cooling.

If using a Mini-ITX PC in a case designed for AC operation, you can still give it power on your mobile robot by using a 12-volt battery and car power inverter. The inverter takes the 12 volts DC from the battery and generates 110 volts AC for the computer.

While this sounds like a roundabout way to go, it’s not quite as inefficient as it seems. Still, you need a strong battery to power the computer for any length of time. A 12-volt motorcycle battery is one option.

Mini-ITX Motherboard

While plopping a whole Mini-ITX PC on top of your robot is a quick and convenient way to endow it with smarts, the case and power supply add unnecessary weight that reduce battery life. One approach is to pull out the motherboard from the PC (Figure 34-3) and mount it directly onto your bot.

The design of mini-ITX motherboards incorporates all of the really important jacks and sockets directly on the board. You seldom need to add accessory boards to complete the system—jacks are provided for mouse, keyboard, monitor, USB, sound out, microphone in, and many others.

DC-to-DC power supply modules let you operate the motherboard directly from a 12-volt battery; no inverter is necessary. These modules are very small, highly efficient (90 percent and higher), and relatively inexpensive. You can match the DC-to-DC module with the power consumption of your motherboard. You can save money if the mini-ITX board consumes only 90 or 100 watts, as this means you can use a cheaper DC-to-DC power supply module.



Figure 34-3 Mini-ITX motherboard, able to run Windows, Linux, and many other modern operating systems. Its small size and miserly power requirements make the board a perfect contender for use in a medium- to large-size robot.

Communications Interconnectivity

Not long ago, computers came with a variety of external ports for connecting things to them: parallel ports for printers, serial ports for phone modems, and game ports for joysticks—among others. All these provided fairly easy ways to connect the computer to robot hardware.

Today, the typical PC lacks all three. Instead, it relies almost entirely on the all-purpose USB port. That's okay, as USB is a flexible system and cables for it are inexpensive. But it also means you need to get an adapter to convert the fairly sophisticated USB signals from your PC to a form you can readily use with your robot. These adapters change USB ports into parallel or serial ports. They're commonly available from online electronics parts outlets; refer to Appendix B, "Internet Parts Sources," for a list of popular Web-based stores.

Data Storage

Whatever your type of PC, you need some means to store your programs and other important data. Notebook and netbook computers already have this solved; they come with their own compact drives, either hard disk or solid state.

If using a mini-ITX motherboard by itself, you can pick from a variety of mass data storage options:

- *Flash drive.* This appears to the mini-ITX as a standard hard drive, but it actually contains no moving parts. It's all flash memory in there. Data capacities are somewhat low when compared to traditional hard drives, and the cost is more. But flash drives are quiet and weigh next to nothing, and they aren't damaged if the robot suddenly falls over.
- *Small-profile (2.5") hard drive.* When you need lots of data space, nothing beats a hard drive. These are compact models made to take up little room. Because they contain a spinning disk drive, you must be careful when using them on mobile robots.
- *USB hard drive or thumbdrive.* Via the USB port on the motherboard you can connect any of a number of mass media drives, including compact hard drives and small *thumb* drives. When selecting a compact hard drive, get one that derives its power from the USB port itself. You don't want the kind that needs to be plugged into a wall outlet.



In order to use a USB hard drive or flash drive, you must be sure the motherboard can boot (start) from the USB drive. Most can, but you'll want to make sure before you purchase any new board.

Using an Off-Bot PC

Tabletop and teleoperated robots can be controlled with a separate PC. An example of a tabletop robot is a stationary arm. These can readily be connected to the host computer via a USB cable. Unless the arm has its own USB jack on it, you'll need a USB-to-serial or USB-to-parallel adapter.

Teleoperated robots can also use wireless communication. There are literally dozens of off-the-shelf standardized solutions for connecting devices (including robots) through the airwaves, including *Bluetooth*, *802.15.4 Zigbee*, and *802.11 Wi-Fi*. From your desktop or laptop PC, you connect to a wireless transceiver through a standard USB port. That transceiver communicates with another transceiver on your robot. This second transceiver can then control motors, operate a remote camera, even send back video signals to your computer.

If going this route, you can choose between half-duplex and full-duplex communications. With half-duplex, only one side can talk at a time. For example, you can command your robot, but it can't send back information at the same time. For greatest flexibility, you want full-duplex

communications. You'll be able to send and receive simultaneously. If the data speed is fast enough, you can readily command the robot to move around a room and have it beam back pictures via its video camera.

Data speeds for wireless communications depend on the distance between sender and receiver, and on the technology. Wi-Fi is faster than Bluetooth, which is (usually) faster than Zigbee. At longer distances, data speeds are reduced to avoid errors. Most RF data communications systems can be used 50 to 100 feet from source to target. For infrared, the distance is much less.

SMARTPHONES, TABLETS, AND PDAS

Rounding out the discussion of brains for your robot are smartphones, computer tablets, and personal data assistants (PDAs). To be useful as a robot brain, the device:

- Should be user programmable. A PDA or smartphone that won't let you add your own programs is useless as a robot controller.
- Provides some kind of communications link between itself and the robot electronics. On many devices this is through Bluetooth, but on others you need to use USB, if available.

Microsoft, Google, and several others tout smartphones that allow you to write and upload your own programs. Example: Microsoft smartphones run Windows Mobile, a version of Windows tailor-made for use on small devices. You can write programs using Microsoft Visual Basic .Net or C#.Net (both free from Microsoft) for use on the phone. Phones that use the open-source Android operating system developed by Google offer similar programming features.

The disadvantage of these devices is the limitations inherent in their design as products that are made for something other than robot control. Their programming tools are not designed to control real-world devices, so developing a robot application tends to involve a lot of compromises.

Of Inputs and Outputs

The architecture of robots requires inputs—things like sensors and bumper switches. And then there's outputs, such as motor control, light, and sound. The basic input and output of a computer or microcontroller is a two-state voltage level (that is, off and on), which usually equates to 0 and 5 volts. For example, to place an output of a computer or microcontroller to HIGH, the voltage on that output is placed, under software control, to 5 volts.



In programming, LOW is equivalent to off, or binary 0. HIGH is equivalent to on, or binary 1. The LOWs and HIGHs are *bits*. Read more about basic programming for robotics in Chapter 36, “Programming Concepts: The Fundamentals.”

Inputs and outputs are colloquially referred to as *I/O*. In addition to standard LOW/HIGH inputs and outputs, there are several other forms of I/O found on single-board computers and microcontrollers. The more common are listed in the following sections, organized by type. Several of these are discussed in more detail in Chapter 40, “Interfacing Hardware with Your Microcontroller or Computer.”

SERIAL COMMUNICATIONS

Robot subsystems need a way to talk to one another. This is often done with a serial communications interface. With serial communications, data is sent one bit at a time. Sounds slow and tedious, but it's really not. The communications link needs just a few wires and can exchange data at speeds easily exceeding tens of thousands of bits per second.

The most common types of serial communications include the following:

I2C (inter-integrated circuit): also shown as I^2C . This is a two-wire serial network scheme that allows integrated circuits to communicate with one another. With I2C you can install two or more microcontrollers in a robot and have them talk to one another.

SPI (serial peripheral interface): This is a popular serial communications standard used by many electronic devices. SPI is most often used to interface with microcontrollers or microprocessor support electronics.

Synchronous serial port: This is a generic term for most any serial data link where information is transmitted one bit at a time, using (at least) two wires. One wire contains the transmitted data, and the other wire contains a clock signal. The term *synchronous* means the clock serves as a timing reference for the transmitted data. This is different from asynchronous serial communication (discussed next), which does not use a separate clock signal.

UART (universal asynchronous receiver transmitter): UARTs are more common in desktop computers, but they have applications in microcontrollers as well. *Asynchronous* means there's no separate synchronizing system for the data. Instead, the data itself is embedded with special bits (called start and stop bits) to ensure proper communication.

MIDI (Musical Instrument Digital Interface): MIDI is a fairly old standard that is found on most every digital keyboard and electronic music device. While you can use MIDI just for its serial communications protocol, you can also adapt it for robotics. Though it's beyond the scope of this book, it's possible—just as an example—to control a robot by playing notes on an electronic keyboard.

Microwire: This is a serial communications scheme used in National Semiconductor products, which is popular for use with the PICMicro line of microcontrollers from Microchip Technologies. It's similar to SPI. Most Microwire-compatible components are used for interfacing with microcontrollers.

PARALLEL COMMUNICATION

Parallel data communication is more straightforward than serial, but it's not necessarily easier to implement. With parallel data, you combine the values of two or more I/O lines of the computer or microcontroller. With eight I/O lines you can communicate 256 different messages; this is because there are 256 different ways to set the two possible states (0 and 1) of the lines.

An example of using parallel communication is displaying text on a liquid-crystal display (LCD) panel. While you can purchase an LCD panel that connects to your microcontroller via serial data, these are much more expensive. With just six I/O lines you can directly control a run-of-the-mill LCD panel.



On most microcontrollers, there is a critical shortage of I/O pins, so using parallel communications for everything uses up valuable data lines. To circumvent this, you can use simple and inexpensive electronics to convert serial data to parallel. You can also do the inverse. See Chapter 40, “Interfacing Hardware with Your Microcontroller or Computer,” for more details on using serial-to-parallel and parallel-to-serial conversion.

ANALOG AND DIGITAL CONVERSION

The command circuitry of your robot is *digital*; the world around us is analog (see Figure 34-4). Sometimes the two need to be mixed and matched, and that’s the purpose of conversion. There are two principal types of data conversion:

ADC: Analog-to-digital conversion transforms analog voltage charges to binary (digital).

ADCs can be outboard, contained in a single integrated circuit, or included as part of a microcontroller. Multiple inputs on an ADC chip allow a single IC to be used with several signal sources.

DAC: Digital-to-analog conversion transforms binary (digital) signals to analog voltage levels. DACs are not as commonly employed in robots, but that doesn’t mean you can’t be clever and think of a nifty way to use one.

PULSE AND FREQUENCY MANAGEMENT

Digital data are composed of electrical *pulses*, and these pulses may occur at a more or less even rate. Pulses and pulse rate (or frequency) are commonly used in robotics for such things as reading the value of sensors or controlling the speed of motors. The three major types of pulse and frequency management are:

Input capture: This is an input to a timer that determines the frequency (number of times per second) of an incoming digital signal. With this information, for example, a robot can differentiate between inputs, such as two different locator beacons in a room. Input capture is similar in concept to a tunable radio.

PMW: Pulse width modulation is a digital output where pulses have a varying duty cycle (that is, the “on” time for the waveform is longer or shorter than the “off” time). PMW is often used to control the speed of a DC motor.

Pulse accumulator: This is an automatic counter that counts the number of pulses received on an input over a period of time. The pulse accumulator is part of the architecture of the microcontroller and can be programmed to operate autonomously. This means the accumulator can be collecting data even when the rest of the microcontroller is busy doing something else.

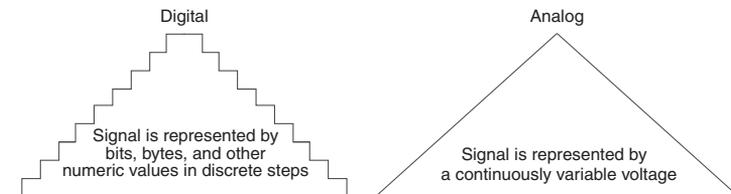


Figure 34-4 Comparison of digital and analog signals. Digital signals are in discrete steps and equate to numeric values. Analog signals are continuously variable.

Understanding Microcontrollers

Microcontrollers have become the favorite method for endowing a robot with smarts. And there's good reason: microcontrollers are inexpensive, have simple power requirements—usually just 5 volts—and most can be programmed using software on your PC. Once programmed, the microcontroller is disconnected from the PC and operates on its own.

A microcontroller is a computer-on-a-chip. It contains everything (or almost everything, depending on the exact model) that's needed to be a fully functional computer. It contains a central processing unit that does the thinking, memory for storing programs and data, and multiple connections that allow it to interface with external devices. In all, the perfect brain for robots.

In this chapter you'll learn about microcontrollers in general—what's inside, how they differ, and why you might want to choose one kind over another. Then in Part 7 you can learn specific details about three very popular microcontrollers used in robot projects: the Arduino, PICAXE, and BASIC Stamp. These chapters include sample projects and programming code you can try for yourself. Find more in Part 8 and on the RBB Online Support site (see Appendix A for details).

All about Microcontroller Categories

Microcontrollers, or MCUs for short, come in all sizes, styles, and categories. Some are for esoteric applications, like running a car's engine or controlling the operation of industrial furnaces. Others are general-purpose, designed for a wide variety of applications. The latter kind are the ones we're most interested in. Figure 35-1 shows a low-cost modular microcontroller designed for experimentation—it contains the MCU itself, plus a liquid-crystal display and other components for learning about programming microcontrollers.

Here are other ways microcontrollers differ and the benefits of each type.

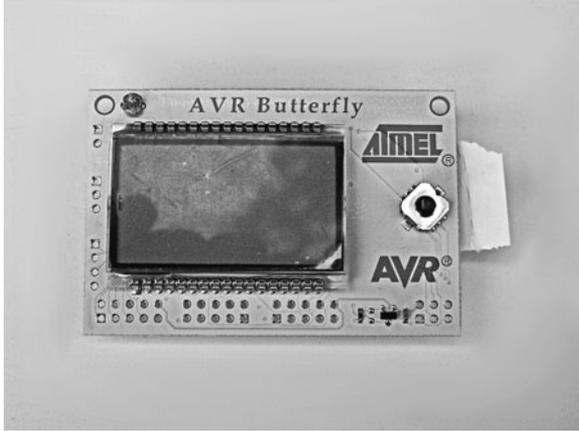


Figure 35-1 Example low-cost modular microcontroller, designed for learning and experimentation. It contains the microcontroller chip itself, plus a liquid-crystal display, button joystick, and other components.

8- 16-, OR 32-BIT ARCHITECTURE

As with your desktop computer, microcontrollers come with different data-processing architectures; these are denoted by the number of bits they process at one time.

- The oldest and simplest of microcontrollers used 4-bit processing. These are of little use to us today, and, besides, they're no longer common in the consumer chip-selling market-places.
- On the other end of the spectrum are MCUs that handle 64 bits at a time. These are specialty chips used for high-end applications and, as such, tend to be expensive and more difficult to program. (Of course, these limitations could and probably will change as time marches on. That's progress for you.)
- Middle-of-the-road microcontrollers handle 8, 16, or 32 bits at a time. The most common for amateur robotics is the 8-bit variety, which, despite handling "only" 8 bits at a time, is ideally suited to the vast majority of robot programming tasks. Eight-bit MCUs are the least expensive and the most widely available.



Recall that the *bit* is the smallest value that can be stored in the memory of a microcontroller or processed through its circuitry. A bit is either off or on. The two possible values for a bit are most commonly represented by the numerals 0 and 1. The value stored in the bit is not really a numeric "zero" or "one"; the 0 and 1 are used as a kind of shorthand. Other shorthands include LOW for 0 and HIGH for 1.

LANGUAGE PROGRAMMING

All microcontrollers need to be programmed for the task you want them to do. None "just work" out of the box (exception: they come preprogrammed with a demo).

There are two general methods of programming a microcontroller, which, for the lack of better terms, I'll call *low-level programmable* and *integrated-language programmable*. These loosely defined terms relate to how the programming is stored and executed in the controller. Both kinds of microcontrollers are fully programmable.

Low-Level Programmable

The traditional way to program a microcontroller has been with assembly language, using your PC as a host development system. Assembly language appears arcane to newcomers, and for many it proves to be a stopping point in further study of microcontrollers. Making matters more complex is that the assembly language for one brand of microcontroller is different from that for another.

Fortunately, most microcontrollers today can also be programmed with a more user-friendly language, such as BASIC, C, or Pascal. These languages are much more approachable, making them easier to learn. You might already know a little (or a lot) about one or more of these languages, in which case you're well on the road to programming a microcontroller. Software on your computer converts your program into so-called machine code, which can be directly read and used by the microcontroller (see Figure 35-2).

An example of a microcontroller that is low-level programmable is the Arduino, which actually uses the Atmel AVR (a popular 8-bit MCU) as its processor. As you'll read in Chapter 37, "Using the Arduino," this microcontroller is more than just a chip. It takes a holistic (whole system) approach that's become very popular among robot builders and electronics experimenters.

Integrated-Language Programmable

In this type of microcontroller, the chip itself contains a kind of language *interpreter*. A program on your computer, a compiler, converts your program into an intermediate language that uses "tokens" to represent actions. The interpreter inside the microcontroller finishes the job of translating the tokens to the low-level code needed by the chip.

Examples of microcontrollers that use this approach include the BASIC Stamp and the PICAXE. Both are based on 8-bit MCUs. These two controllers are further detailed in their own chapters. See Chapter 38, "Using the PICAXE," and Chapter 39, "Using the BASIC Stamp."

Choice of Programming Language

A low-level programmable microcontroller is basically a blank canvas; it's up to you what you put in it and how you do it. With these controllers you have an option of choosing the language you wish to use. These include:

BASIC: This language is popular with those just starting out in programming, as the language is designed for beginners—in fact, the B in BASIC stands for beginner. It's also a favorite among those already familiar with a BASIC language for the PC, such as Visual Basic. BASIC is a more forgiving language than the others; for example, it's not

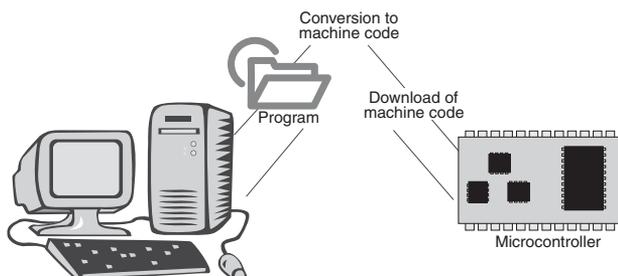


Figure 35-2 Programming cycle of a microcontroller. Programs are developed on a PC, where they are compiled to a machine-readable format, and then downloaded (usually via cable) to the microcontroller.

case-sensitive. It doesn't care if you use different capitalizations to reference the same things.

C: The language of choice for programming professionals, C has more strict syntax rules, so it's often considered harder to learn. (In a spoken language, *syntax* is how the parts of speech are strung together to form a coherent statement. It's the same in a programming language.) For all the bad rep C gets for being a stern schoolmaster, it's actually not that much more difficult to learn than BASIC, at least not when it comes to microcontrollers.

Pascal: Considered a blueprint for modern structured programming, Pascal is a lesser-used language when it comes to microcontrollers (so there aren't as many options for it), but it's known for being easy to learn.



There are several other programming languages used in microcontrollers, including Forth, Java, Python, and C#. The choice of which language to use depends greatly on which one you're more comfortable with and which offers the feature set you want to exploit in your robot work.

Three Steps in Programming a Microcontroller

No matter what system or language you use, there are three basic steps to programming a microcontroller. They are:

1. Use your personal computer to write your program with a text editor or other application. Many commercial programming languages come with a fancy editor, called an *IDE*, for integrated development environment. This single application combines the programming step with the other two that follow.
2. You then compile your program into a form of data that the MCU will understand. The microcontroller doesn't know an "If" statement from a hole in the ground, and the job of the compiler is to convert the human-readable code you just wrote to the machine language the microcontroller understands.
3. After being compiled, the translated program is downloaded to the microcontroller. This is most often done using a USB or serial cable. Once downloaded, the program is immediately ready to be run inside the microcontroller. In fact, it will usually start running the moment the downloading process is complete.

Microcontroller Shapes and Sizes

All microcontrollers are integrated circuits, with anywhere from 8 to over 128 connection pins, depending on complexity. But you don't always work with a microcontroller as a separate integrated circuit. Popular form factors of microcontrollers include (see Figure 35-3):

Chip-only: You work with the microcontroller at the bare IC level. The chip may or may not need any external components to operate. At a minimum, some need a voltage regulator and an oscillator or resonator as a clock source.

Carrier: In this variation the microcontroller IC is mounted on a carrier, which contains additional electronics. The carrier is itself the size and shape of a wide-profile integrated circuit: a 24-pin "double-wide" IC is common. This is the form factor of the BASIC Stamp and similar products. The carrier can be plugged into solderless breadboards for

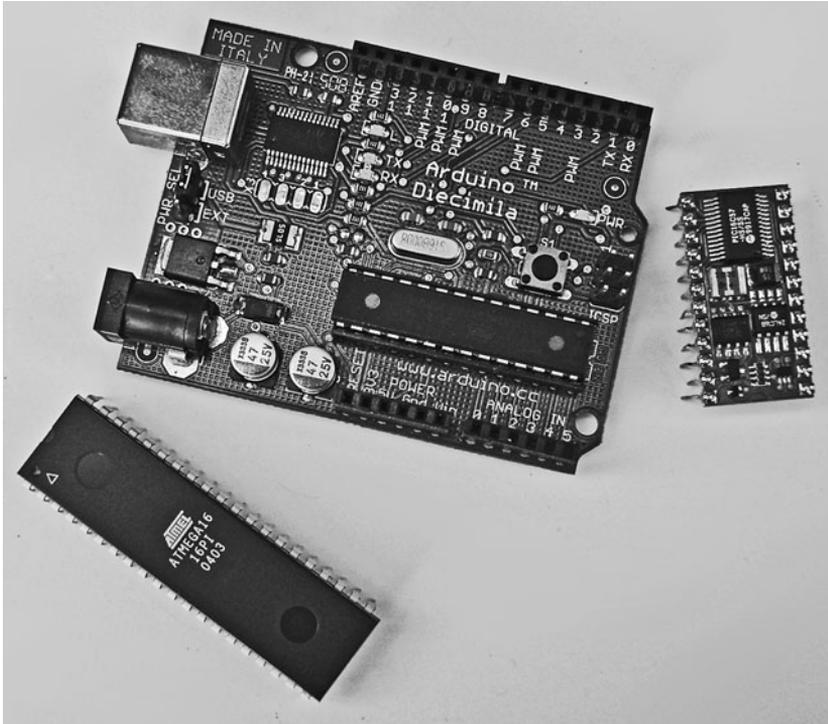


Figure 35-3 Three typical microcontroller form factors: bare integrated circuit (chip-only), carrier board, and development board.

further experiments. Not all carriers are shaped like ICs. Some look more like long sticks, with one or two rows of connection pins.

Integrated development board: Physically the largest of the bunch, integrated (all-in-one) development boards contain the microcontroller and support electronics, plus extras like LEDs, switches, and header connections for experimenting. Using jumper wires, you can connect the development board to a solderless breadboard, where you might attach sensors, servo motors, and other external components.



Separate development boards are commonly available for chip- and carrier-style microcontrollers. These offer the best of both worlds; you get a compact controller when you must conserve space, but, when needed, you can transplant the MCU to the board.

Under the Hood of the Typical Microcontroller Chip

A key feature of microcontrollers is that they combine a microprocessor with various inputs and outputs (called *I/O*) that are needed to interface with the real world. For example, the Atmel ATmega328 28-pin microcontroller (Figure 35-4) sports the following features, many of which are fairly standard among microcontrollers.

- *Central processing unit (CPU)*. This is the core of the microcontroller and performs all of the logic and arithmetic computations. The CPU takes your programming instructions and evaluates each one in turn.
- *23 input/output pins*. The I/O lines are the gateway of information in and out of the microcontroller. An I/O line can be set to act as an input, in which case it can accept data from the outside, or it can act as an output, where it can control some external component. The ATmega328 has 28 pins total, of which 23 may be used for input/output chores—it's said to have 23 I/O lines or I/O pins. The remaining five pins are used for power supply connections.
- *Built-in analog converter*. Six of the chip's input/output pins are connected to an internal analog-to-digital converter (ADC), which translates analog voltages to digital binary values.
- *Built-in analog comparator*. The internal analog comparator can be used for basic go/no-go evaluation of analog voltage levels.
- *Flash program storage*. Programs you write and download to the microcontroller are stored in 32K bytes of rewritable flash memory. This allows the controller to be reprogrammed over and over again. Note that some MCUs are program-once only. Read more about these later in this chapter.
- *RAM and EEPROM data storage*. Small amounts of RAM (2K bytes) and EEPROM (1K bytes) storage are provided for the data that the controller uses during operation. Data in RAM (random-access memory) is lost when the controller loses power. Data in EEPROM (electrically erasable programmable read-only memory) retains its value even when the microcontroller is no longer powered.
- *Hardware interrupts*. Microcontrollers are designed to interact with the outside world, and hardware interrupts allow the chip to be literally "interrupted" by some external stimulus. It's not unlike asking someone to pinch you to wake you up. Interrupts make many common robotic programming tasks easier and more elegant. The ATmega328 has two primary hardware interrupts that are connected to two of its pins.
- *Built-in timers and counters*. These all-purpose accessories of microcontrollers operate separately from the CPU and provide a wide variety of useful services. For example, under program control, you can command a timer to generate a pulse every second. Counters are likewise special accessories separate from the CPU and literally count individual signal events, like the number of times a robot's wheels have rotated.

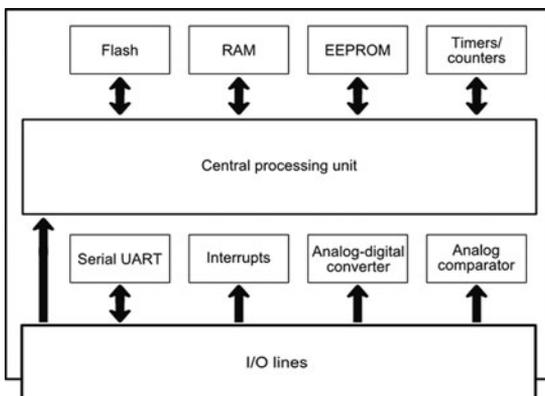


Figure 35-4 Basic block diagram of the Atmel AVR ATmega328 microcontroller. The central processing unit (CPU) forms the core of the controller. Additional built-in hardware provides special functions, such as timers, an analog comparator, and an analog-to-digital converter.

- *Programmable full-duplex serial port.* Serial data is a common means of communication with a microcontroller, not only for programming it, but also for the MCU to communicate with other devices.

PIN FUNCTIONS

Being integrated circuits, the pins on microcontrollers serve as the means to power the chip, as well as get signals in and out of the device. Figure 35-5 shows the pinout diagram of the Atmel AVR ATmega328 microcontroller and the functions of each pin. The labels for the pins have little meaning without the datasheet for the controller, but this diagram gives you an idea of the general layout you'll encounter.

As with many microcontrollers, the pins are identified with a default function, as well as alternative functions, if any. The alternative functions are shown here in parentheses. For example, pin 1 serves as the reset button for the chip. Press it and the microcontroller restarts its programming.

But pin 1 can also serve additional functions, if a reset isn't needed. It can be used as just another digital input/output pin (belonging, according to the diagram, to the Port C group; *ports* are convenient collections of I/O pins).

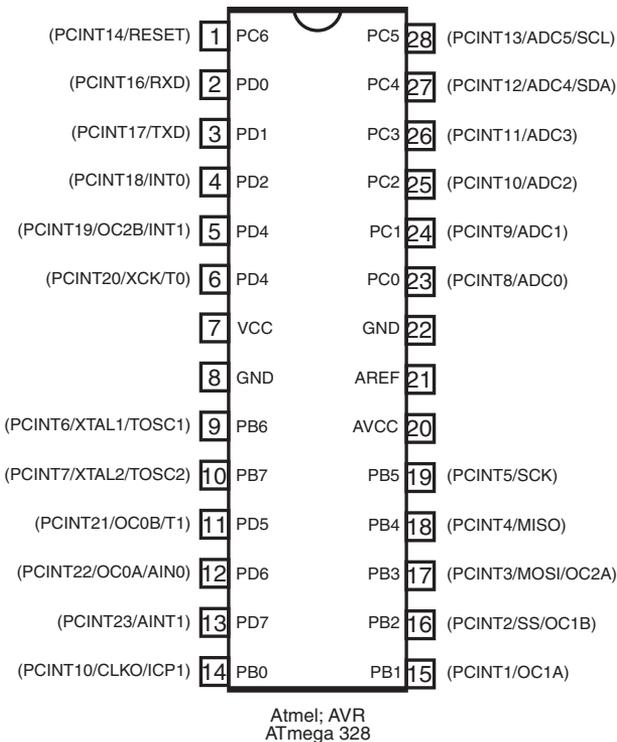


Figure 35-5 Pinout diagram of the ATmega328 microcontroller, showing the pin numbers (from 1 to 28), plus labels for the functions of each pin. Most pins have alternative functions, shown in parentheses.

MORE ON PROGRAM AND DATA STORAGE

At first glance, it looks like microcontrollers have somewhat limited memory space for programs. The typical low-cost microcontroller may have only a few thousand bytes (*yes, bytes, not megabytes*) of program storage. While this may seem terribly confining, in reality most microcontrollers are programmed to do single or well-defined jobs. These jobs may not require more than a few dozen lines of program code.

More elaborate microcontrollers may handle more program storage—8K or 32K are not uncommon, and a few can support well over a megabyte. Keep the program storage limits in mind when you're planning which brain to get for your robot. A single type of microcontroller is often available in slightly different versions; each version may accommodate a different amount of program data.

Data storage may seem even more restricted. The typical microcontroller may only provide 1K or 2K of RAM for data (EEPROM space, if provided, is even less). But again, most microcontroller applica-

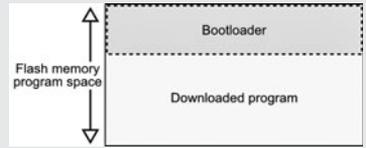
tions make efficient use of data space. As long as you follow good coding practice, you should seldom run out of data storage space.

ERASING AND STARTING OVER

As noted, most microcontrollers are meant to be programmed over and over again. Each time you download a program to it, the old one is erased, and the new one takes its place. The most common program storage used in modern microcontroller is flash, the same as that used in USB thumb drives, or the CompactFlash cards in a digital camera.



There is an exception to completely erasing the flash memory each time you download a new program. Some microcontrollers are set up to allow a special “bootloader” area in flash, which contains special setup code to make it easier to reprogram the chip without having to pull it out of its circuit.



IN-FIELD PROGRAMMING (AND REPROGRAMMING)

A key benefit of microcontrollers with flash memory is that they can be programmed and reprogrammed “in circuit”—that is, while the chip is still plugged into whatever circuit board home it’s living in.

This has enormous potential for use in your programmable robot. With in-field programming there is no need to remove the microcontroller chip from its circuit in your robot to reprogram it. Instead, you merely connect a cable from your PC and download the new program. Of course, this requires that the microcontroller have an onboard connector so it can be attached to your PC cable.

ONE-TIME PROGRAMMABLE

Less costly microcontrollers are made to be programmed only once, and are intended for permanent installations. These *one-time programmable* (OTP) microcontrollers are popular in consumer goods and automotive applications.

For robotics applications, the OTP is useful for dedicated processes, such as controlling servos or triggering and detecting a sonar ping from an ultrasonic distance measurement system. You’ll find a number of the ready-made hobby robotic solutions on the market today that have, at their heart, an OTP microcontroller. The microcontroller takes the place of more complex circuitry that uses individual integrated circuits.

Microcontroller Programmers

All microcontrollers need to be programmed. The complexity of the programming setup depends on the architecture of the microcontroller. For example, the PICAXE and BASIC Stamp controllers can be connected to a PC using a simple cable—typically just a serial cable that has the usual DB-9 connector on one end (for connection to the PC) and header pins on the other (for connection to the controller).

Chip-only microcontrollers without a built-in language need a programmer, a physical device that provides all the necessary power and signal connections to the chip. The program-

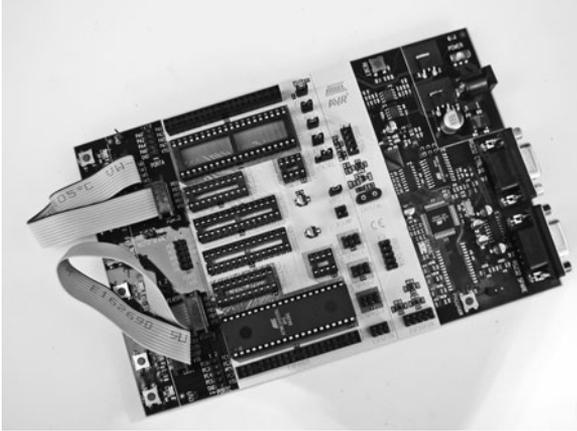


Figure 35-6 Microcontroller programmer, with sockets to accept controllers of different sizes. The programmer connects to a host computer via a cable.

mer connects to your PC via a serial or (more often) USB cable. Most programmers are designed for a certain brand of microcontroller—the Atmel AVR line or the PICMicro.

Figure 35-6 shows an STK500 programmer made for AVR microcontrollers. Like many of its kind, the STK500 has several sockets of different sizes for accepting the different members of the microcontroller family. To use, you plug an “empty” (unprogrammed) chip into its corresponding socket, connect the programmer to your PC, and download your application.

Some programmers are combined with development systems, which provide numerous standard accessories onboard. These allow you to design fully operational microcontroller solutions, without having to create elaborate breadboards or construct custom circuits. Everything (or nearly so) is already included in the development system.

All about Microcontroller Speed

If you have a personal computer, you probably know that its microprocessor runs at a certain speed. Older PCs were rated in megahertz (millions of cycles per second); the latest models operate in the gigahertz (billions of cycles per second) range.

Likewise, microcontrollers operate at set speeds. These speeds are rather low for a modern computational device—most MCUs operate at 4 MHz to 40 MHz. That’s along the lines of the first IBM PCs that came out in the early 1980s!

But again, the nature of microcontrollers doesn’t require superspeeds. For one thing, many microcontrollers are more efficient in how they execute their program code. Most microcontrollers can execute a single programming instruction in just one cycle of the clock. A controller operating at 20 MHz can therefore process about 20 million programming instructions each second.



Yet sometimes the speed of a microcontroller is not enough for the task you want to give it. Processing full-motion video is a good example. You probably wouldn’t want to task your 4-MHz MCU with reading a frame of video (there are 25 or 30 frames each second) and analyzing each pixel for motion.

Programming Concepts: The Fundamentals

Back in the “olden days,” you built a robot with a couple of motors, some tubes and a relay, and a big ol’ battery. Today, many robots, including the amateur variety, are equipped with a computational brain of one type or another, and the brain is told what to do through software programming.

Why is this progress over tubes and a big ol’ battery? The brain and programming are almost always easier and less expensive to implement than other methods. The nature of the programming depends on what the robot does. All its actions boil down to a relatively small set of instructions in whatever *programming language* you are using. If you’re new to programming or haven’t practiced it in several years, read through this chapter to learn the basics of programming for controlling your robots. It discusses rudimentary stuff so you can better understand the more technical material in the chapters that follow.

Of course, if you’re already familiar with programming, feel free to skip this chapter.



Unless otherwise noted, the programming examples used in this chapter are what’s called *pseudo-code*. It isn’t fully functioning code, and it’s highly unlikely it’ll work as is in whatever programming environment you’re using. So don’t try. Pseudo-code uses English-like phrases to demonstrate the overall intent of the programming.

If you want to see actual runnable code, refer to the chapters in parts 7 and 8. Additional programming examples are provided on the RBB Online Support site, detailed in Appendix A.

Important Programming Concepts

There are 11 important concepts in understanding programming, whether for robots or otherwise. In this chapter, we’ll talk about each of the following in greater detail:

- Flow control
- Routines
- Variables
- Expressions
- Strings
- Numerical values
- Conditional statements
- Branching
- Looping
- Inputting data
- Outputting data

GOING WITH THE FLOW

You can create simple one-job programs without a predefined blueprint or flowchart. For more complex programs, you may find it helpful to draw a programming *flowchart*, which includes the basic steps of the program. Each box of the chart contains a complete step; arrows connect the boxes to indicate the progress or sequence of steps throughout the program.

Flowcharts are especially handy when you are creating programs that consist of many self-contained “routines” (see the next section) because the graphical format of the chart helps you visualize the function and flow of your entire program.

THE BENEFIT OF A ROUTINE

Even the longest, most complex program consists of little more than bite-size segments, called *routines*. The program progresses from one routine to the next in an orderly and logical fashion.

A routine is any self-contained segment of code that performs a basic action. In the context of robot control programs, a routine can be a single command or a much larger action. Suppose your program controls a robot that you want to wander around a room, reversing its direction whenever it bumps into something. Such a program could be divided into three distinct routines:

- *Routine 1: Drive forward.* This is the default action or behavior of the bot.
- *Routine 2: Sense bumper collision.* This routine checks to see if one of the bumper switches on the robot has been activated.
- *Routine 3: Reverse direction and turn.* This occurs after the robot has smashed into an object, in response to a bumper collision.

Figure 36-1 shows how this program might be mapped out in a flowchart. While there’s no absolute need to physically separate these routines within a program, it often helps to think of the program as being composed of these more basic parts.



A routine by any other name might be a *routine*, or it might be a *subroutine*, *sub*, *method*, *class*, *function*, or a variety of other terms. It all depends on the programming language and the habit of the programmer. Different terms, but the basic concept is the same.

VARIABLES

A *variable* is a holding area for information, a kind of shoe box for data. In most programming languages you can make up the name of the variables as you write the program. The

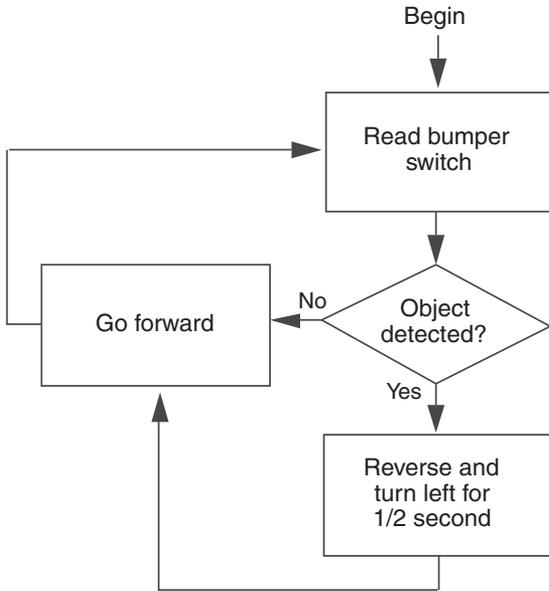


Figure 36-1 Example flowchart showing how a robot software program is broken down into separate actions and reactions. The robot follows either of two prescribed plans, depending on whether its bumper switch has been activated.

content of the variable is either specified by you or filled in from some source when the program is run. Since the information is in a variable, it can be used someplace else in the program as many times as you need.

Variables can hold different kinds of data, depending on the number of bits or bytes that are needed to store those data. A numeric value from 0 to 255, for example, requires 8 *bits*, equivalent to 1 *byte*, of storage space. This means the variable for such a value needs to be at least 1 byte large, or else the number won't be stored properly.

FYI

Bits, bytes, and other forms of data are discussed in more detail in “Understanding Data Types,” later in this chapter. For now, it's not critical if you aren't familiar with these terms.

EXPRESSIONS

An *expression* is a mathematical or logical “problem” that a program must solve before it can continue. Expressions are often simple math statements, such as $2 + 2$. When you ask a program to perform some type of calculation or thinking process, you're asking it to *evaluate an expression*.

One of the most common expressions is evaluating whether a statement is True or False (I've capitalized the words *true* and *false* to show that we're dealing with logical functions). Here's a good example of a True/False expression that must be evaluated by a program:

```
If Number = 10 Then End
```

This expression reads: “If the content of the *Number* variable is equal to 10, then end the program.” Before proceeding, your robot must pause, take a peek inside the *Number* variable, and apply it to the logical expression. If the result is True, then the program ends. If it's False (*Number* has a value other than 10), then something else happens.

STRINGS

A common term encountered in programming circles is *string*. A string is simply a sequence of number and letter characters. These characters are stored within the computer's memory one right after the other, like beads on a string—hence the name.

In the context of programming languages, strings are most often used to communicate information to human users, such as text on a liquid-crystal display (LCD) panel.

NUMERICAL VALUES

Computers, and the programs that run on them, are designed from the ground up to work with *numerical values*, ye olde numbers. Numbers differ from strings in one important way, however: the program can perform calculations on two numbers and provide you with the result.

CONDITIONAL STATEMENTS

Programs can be constructed so that they perform certain routines in one instance and other routines in another. Which routine the program performs depends on specific conditions, set either by the programmer, by some sensor on the robot, or by the contents of a variable.

A *conditional statement* is a fork in the road that offers the program a choice of two directions to take, depending on how it responds to a True/False question. The types and styles of conditional statements differ among robot programming languages, but they all have one thing in common: they activate a certain routine (or group of routines), depending on external forces.

The most common conditional statement is built using the *If* command. Here's an example: "If it's cold outside, I'll wear my jacket. Otherwise, I'll leave the jacket at home." The statement can be broken down into three segments:

1. The condition to be met (if it's cold).
2. The result if the condition is True (wear the jacket).
3. The result if the condition is False (leave the jacket).

BRANCHING

Akin to the conditional statement is the *branch*, where the program can take one of several paths, depending on external criteria. A good example of a branch is when a robot senses a collision while moving, as in our earlier example (see the section "The Benefit of a Routine").

The robot normally just drives forward, but many times each second its program branches off to a routine that checks to see if a collision sensor has been activated. If the sensor has not been activated, nothing special happens, and the robot continues its forward movement. But if the sensor *has* been activated, the program branches to a different routine and performs a special action to move the robot away from the obstacle that's just been struck.

LOOPING

A *loop* is programming code that repeats two or more times. A typical loop checks to make sure some condition is met, and if it is, the contents of the loop are processed. When the

program gets to the bottom of the loop, it goes back to the top and starts all over again. This process continues until the test condition is no longer met. At that point, the program skips to the end of the loop and performs any commands that follow it.

INPUTTING DATA

When you sit at a computer, you use a keyboard and a mouse to enter data into the machine. While some robots also have keyboards or keypads (and a few have mice), *data input* for automotons tends to be a bit more specialized, involving, for example, touch switches or a sonar ranging system. In all cases, the program uses the information fed to it to complete its task.

The reverse-on-collision robot described earlier is once again a good example. The data to be input is simple: it is the state of a bumper switch on the front of the robot. When the switch is activated, it provides a signal—“Hey, I think I hit something!” With that signal your bot can (for example) back up, get out of the way, and head toward some other wall to do it all over again.

OUTPUTTING DATA

In the realm of robot control programs, data output is most often used to turn motors on and off, to activate a sonar chirp for sensing distance, and to blink a light-emitting diode to communicate with you in a crude form of Morse code. (Or how about one flash for *Yes*, two flashes for *No*? Well, it worked for Captain Pike in *Star Trek*!) Data output provides a means for the robot to either interact with its environment or interact with you.

On robots equipped with liquid-crystal display panels, data output can provide a way for the machine to fully communicate its current condition. Simpler data output is possible with that Captain Pike light-emitting diode (LED) trick. Or even simpler still, maybe if the LED is on, it could mean your robot has run into trouble and needs your help. What the signals mean is completely up to you.

Understanding Data Types

At their most basic level, programs manipulate data. Data come in many forms, including a funny-looking guy with platinum-colored skin on *Star Trek: The Next Generation*. (Oh, no, not *another Star Trek* joke!) Actually, the kind of data we’re interested in are strictly numbers of one type or another. Those numbers might represent a value like 1 or 127, the numeric equivalent of a text character (65 is “A” using the ASCII standard), or binary 00010001, which could mean “turn on both motors” on your robot.

THE MOST COMMON DATA TYPES

In a program, data types can take the following common forms:

Literals: A literal is, literally, a value “hard-coded” into the program. For example, in the statement `MyVariable = 10`, the `10` is literal data.

Variables: We've already seen what a variable is: it's a place where data can be stored and referenced elsewhere in the program. It's the *MyVariable* in the statement *MyVariable = 10*.

Constants: Depending on the design of the programming language, a constant can be just another name for a literal or it can be a special kind of variable that—once set—is never meant to be changed.

Expressions: The result of a math or logical expression can “return” a data type. For example, the expression $2 + 2$ returns the value 4.

No matter what form the data type is in, the programming language expects to see its data follow predefined types. This is necessary so the data can be properly stored in memory. The most common data type is the 8-bit integer, so called because the value stores an integer (a whole number) using 8 bits.



A *bit* is the smallest value that can be stored in memory and processed by a computer. A bit is either off or on. The two possible values for a bit are most commonly represented by the numerals 0 and 1. The value stored in the bit is not really a numeric “zero” or “one”; the 0 and 1 are used as a kind of shorthand. Other shorthand terms include LOW (for 0) and HIGH (for 1).

With eight sets of bits, the program can work with a number from 0 to 255 (or -128 to $+127$, depending on how it uses the eighth bit, as discussed later). The basic data types are as follows:

- Single bit, on or off; HIGH or LOW; yes or no, 0 or 1
- 4-bit nibble (half a byte; sometimes called a nybble)
- 8-bit integer, or *byte* (can hold a number, a True/False value, or a string value)
- 16-bit integer, or *word*
- 32-bit integer, or *long* or *double word* (dword)
- 32-bit floating point, or *single* (“floating point” means a number with a decimal point)

SIGNED AND UNSIGNED NUMERIC VALUES

In many cases, programming languages provide for either (or both) “signed” and “unsigned” values. Signed means the number can be expressed as either a positive or a negative value; unsigned means the number can only be a positive value.

In these languages, the first bit on the left (called the *most significant bit*, or MSB; see Figure 36-2) is used to denote whether the number is positive or negative. If the MSB is a 0, it means the number is a positive value; if it's a 1, the number is negative. With an 8-bit unsigned integer, for example, the program can store values from 0 to 255. With an 8-bit signed integer, the program can store values from -128 to $+127$.

AT A GLANCE: NUMBER LIMITS BY DATA TYPE

Always remember that there are limits to how big a number can be when stored as any given data type. If the number you try to store exceeds the limits of the data type, your program

Unsigned MSB bit

1	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

MSB

10011010 binary = 154 decimal

Signed MSB bit

1	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

MSB

10011010 binary = -102 decimal

Figure 36-2 Numeric values are stored in a microcontroller as a set of bits, 8 bits making 1 byte. The *most significant bit*, or MSB, of this value is often used to determine if the number is positive only, or either positive or negative. Here, the bits 10011010 can represent the value 154 or -102, depending on how the MSB is treated.

could report an error (and perhaps stop), or at best the data will be inaccurate. The numeric limits for the main integer data types are:

Data Type	Unsigned	Signed
4-bit nibble	0 to 15	-8 to +7
8-bit byte	0 to 255	-128 to +127
16-bit word	0 to 65,535	-32,768 to +32,767
32-bit dword	0 to 4,294,967,295	-2,147,483,648 to +2,147,483,647



Over the years, programming languages have evolved and have greatly messed things up by playing loosey-goosey with terminology. A good example is the word *integer*. The term “integer” means—in Latin, no less—a whole number; yet many programming languages use it to refer to a data type of a specific size.

What’s worse, the size of the data type can and does change as the programming language evolves. In older versions of Visual Basic, for example, “integer” meant a 16-bit signed whole number. Now it means a 32-bit signed whole number.

Keep all this in mind as you’re learning a new programming language. You may see a data type called *Int*, *int*, *Integer*, or some other variation. Yes, it’s for storing an integer (whole number), but it’s also a data type whose size is specific to that language.

Lucky Seven Most Common Programming Statements

Here are seven of the most common statements you’ll encounter in most any programming language. The examples shown are those for the BASIC programming language, but the fundamentals are the same for other languages.

COMMENTS

Comments are used by a programmer as remarks or as reminders of what a particular line of code or routine is for. The comments are especially helpful if the program is shared among

many people. When the program is *compiled* (made ready) for the computer or microcontroller, the comments are left out. They appear only in the human-readable *source code* of your programs.

To make a comment using BASIC, use the ' (apostrophe) character. Any text to the right is treated as a comment and is ignored by the compiler. For example:

```
' this is a comment
```



The symbol used for comments differs among languages. In the C programming language, for instance, the characters `//` are used to denote a comment. C also offers the `/*` and `*/` characters to mark a *comment block*. Very handy.

IF

The *If* statement is used to build a *conditional* expression. It is called a conditional expression because it tests for a specific condition:

- If the expression is True, the program performs the commands following the If statement.
- If the expression is False, the program performs the commands after the Else statement, if any.

Here's a sample conditional statement in BASIC:

```
If ExampleVar = 10 Then
    Call (Start)
Else
    End
End If
```

The *If* statement evaluates a condition to determine if it's True or False. The most common evaluation is for *equality*. The = (equals) sign is used to test if one value, such as the contents of a variable, is equal to another value. However, there are other forms of evaluation, such as "not equal to," "greater than," "less than," and several others. See the "Variables, Expressions, and Operators" section later in this chapter for more information.



In the C language, equality is tested using two equals signs together: `==`. This is one of the most common mistakes beginning C programmers make. The statement `if x = 1` is wrong. It should be `if x == 1`.

SELECT CASE

The *Select Case* statement is used when you want to test a value (usually in a variable) against several possibilities. The *Select Case* statement lets you test each number individually and tell the program specifically what you want to happen should there be a match.

The basic syntax for the *Select Case* statement is:

```
Select Case (TestVar)
    Case x
        ' do if x
    Case y
        ' do if y
```

```

Case z
' do if z
End Select

```



In the C programming language, the *switch* statement takes the place of *Select Case*. In C, you still use the *case* statement to test each condition, but remember that in C, *case* is all lowercase.

TestVar is the test expression. It is evaluated against each of the *Case arguments* that follow. If the value in *TestVar* is equal to *x*, then the program performs the action that follows *Case x*. If the value in *TestVar* is equal to *y*, then the program performs the action following *Case y*, and so forth.

CALL

The *Call* statement tells the program to temporarily branch elsewhere in the program. This is a routine that is identified by name, using a *label*. The program expects to find a *Return* statement at the end of the routine. When it encounters the *Return* statement, the program jumps back to the *Call* statement and continues executing. A typical *Call* statement and label look like this:

```

Call (Loop)
' . . . additional program code here
Loop:
' statements to repeat go here
Return ' Program goes back to the Call, and continues

```

GO

The *Go* statement is used to jump to the specified label. In programming parlance, using *Go* to go to a label is called *unconditional branching*. The *Go* statement uses one *argument*, namely, the name of the destination label. For example:

```

Go (MyLabel)
' . . . some programming code here
MyLabel:
' Program doesn't go back to the Go

```

FOR/NEXT

The *For/Next* statement is actually a pair of commands. They repeat other programming instructions a specified number of times. The *For/Next* structure is perhaps the most commonly used of all programming loops. The *For* portion of the *For/Next* loop uses an expression that tells the program to count from one value to another. For each count, any programming code contained within the *For/Next* structure is repeated:

```

For x = 1 to 10
' . . . statements here repeated 10 times
Next

```

x is a variable that the program uses to keep track of the current loop iteration. The first time the loop is run, *x* contains 1. The next time through, *x* contains 2, and so on. When *x* con-

tains 10, the program knows it has run the loop 10 times and skips to the *Next* statement. From there, it executes any additional code that may be in the remainder of the program.

WHILE/WEND

The *While/Wend* statements also form a loop structure. But unlike *For/Next*, which repeats a set number of times, the loop of a *While/Wend* structure repeats while a condition is met. When the *While* condition is no longer met, the loop is broken and the program continues. Code that falls between the *While* and the *Wend* statements is considered part of the loop that is repeated.

```
While x
' . . . statements to repeat go here
Wend
```

x is an expression that is evaluated each time the loop is repeated. For instance, the expression might be *While switch = 0*, which tests to see if the *switch* variable contains a 0, signifying perhaps that some switch has not been activated. When *switch* is no longer 0, the loop breaks.



In many robot control programs, the *While/Wend* loop (and variations, such as *Do/Loop*) is set up to run indefinitely. This *infinite loop* repeats a process over and over again until the power to the robot is turned off.

Variables, Expressions, and Operators

Earlier in this chapter you read how variables are temporary holders of information. Placing data into a variable is referred to as *assigning*, or *assigning a value to a variable*.

ASSIGNING A VALUE TO A VARIABLE

The most common way to assign a value to a variable is to use the *assignment operator*. In most programming languages used for robot control, this is done with the = (equals) symbol. It is most often necessary (or at least advisable) to define the data type that will be stored in the variable before assigning a value to it. Here's an example for the BASIC language:

```
Dim X As Integer
X = 10
```

Dim stands for “Dimension,” which tells the program that you are defining the type of the variable you wish to use—allocating space (dimension) in memory for it. *X* is the name of the variable you wish to assign. The = (equals) sign is the variable assignment symbol, and *10* is the value you are placing inside the *X* variable.



Many languages restrict the names you can use for variables. Specifically, the variable name must start with a letter character, and it cannot contain spaces or other punctuation. *Reserved words*—special identifiers such as *If*, *While*, and *Select*—are also unavailable for use as variable names.

Once you define a variable as containing a certain kind of data, it's important that you do not then assign a different data type to the variable. For example, all of the following are wrong:

```
Dim X As Byte
X = 290                ' data overflow; byte range is 0 to 255

Dim X As String
X = 15                 ' 15 is not a string

Dim X As Integer
X = "hello"           ' "hello" is not an integer
```

Most robot control languages let you assign a variety of values to variables, as long as the values match the data type you are using. Among the most common value types assigned to variables are:

Literal values: As mentioned earlier, a literal value is a value you specify when you write the program. For example:

```
X = 15                ' X is the variable; 15 is the literal value
```

Contents of another variable: Here you copy the contents of one variable into another variable. For example:

```
X = Y                ' X is the newly assigned variable;
                    ' Y contains some value you're copying
```

(Careful with this one! In most languages, what gets copied is the *content* of the variable, not the variable itself. If you later change the content of Y, X stays the same. But some programming languages try to mess with your mind, letting you specify a "pointer" to the variable, rather than just its content. If you change Y, X changes, too.)

Memory location: Many programming languages let you reference specific portions of physical memory. For example:

```
X = Peek (1024)      ' read value of data starting at memory
                    ' location 1024
```

Register reference: A register reference is a value maintained by your robot's microcontroller. Registers are just like variables, but their names are built into the microcontroller hardware. For example:

```
X = b7                ' read value in register b7
```

Evaluated expression: The evaluated expression variable stores a value that is the result of an expression. For example:

```
X = 2 + 2            ' X holds 4
```

CREATING EXPRESSIONS

An expression tells the program what you want it to do with information given to it. An expression consists of two parts:

- One or more *values*
- An *operator* that specifies what you want to do with these values

In most programming languages, expressions can be used when you are defining the contents of variables, as in the following:

```
Test1 = 1 + 1
Test2 = (15 * 2) + 1
Test3 = "This is" & " a test"
```

The program processes the expression and places the result in the variable.

The following sections present the most common operators and how they are used to construct expressions. Some operators work with numbers only, and some can also be used with strings. The list is divided into two parts: math operators (which apply to number values only) and relational operators (which work with both numbers and in some programming languages' strings).

Math Operators

Operator	Function
-value	Treats the value as a negative number.
v1 + v2	Adds values v1 and v2 together.
v1 - v2	Subtracts value v2 from v1.
v1 * v2	Multiplies values v1 and v2.
v1 / v2	Divides value v1 by v2. Sometimes also expressed as v1 <i>DIV</i> v2.
v1 % v2	Divides value v1 by v2. The result is the floating-point remainder of the division. Sometimes also given as v1 <i>MOD</i> v2.

Relational Operators

Operator	Function
Not value	Evaluates the logical <i>Not</i> of <i>value</i> . The logical <i>Not</i> is the inverse of an expression: True becomes False, and vice versa.
v1 And v2	Evaluates the logical <i>And</i> of v1 and v2.
v1 Or v2	Evaluates the logical <i>Or</i> of v1 and v2.
v1 = v2	Tests that v1 and v2 are equal.
v1 <> v2	Tests that v1 and v2 are not equal.
v1 > v2	Tests that v1 is greater than v2.
v1 >= v2	Tests that v1 is greater than or equal to v2.
v1 < v2	Tests that v1 is less than v2.
v1 <= v2	Tests that v1 is less than or equal to v2.

Relational operators are also known as boolean or True/False operators. Whatever they test, the answer is either yes (True) or no (False). The expression $2 = 2$ is True, but the expression $2 = 3$ is False.

Using And and Or Relational Operators

The *And* and *Or* operators work with numbers (depending on the language) and expressions that result in a True/False condition. It’s often helpful to view the actions of the *And* and *Or* operators by using a *truth table* like the two that follow. The tables show all the possible outcomes given to values in an expression:

AND Truth Table		
0 means False, 1 means True		
Val1	Val2	Result
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table		
0 means False, 1 means True		
Val1	Val2	Result
0	0	0
0	1	1
1	0	1
1	1	1

USING OPERATORS WITH STRINGS

Recall that a string is an assortment of text characters. You can’t perform math calculations with text, but you can compare one string of text against another. The most common use of operations involving strings is to test if a string is equal, or not equal, to some other string, as in:

```
If "MyString" = "StringMy" Then
```

This results in False because the two strings are not the same. In a working program, you’d no doubt construct the string comparison to work with variables, as in:

```
If StringVar1 = StringVar2 Then
```

Now the program compares the *contents* of the two variables and reports True or False accordingly. Note that string comparisons are almost always case-sensitive:

String 1	String 2	Result
hello	hello	Match
Hello	hello	No Match
HELLO	hello	No Match

While the = (equals) operator is used extensively when comparing strings, in many programming languages you can use <>, <, >, <=, and >= as well. See “Creating Expressions,” previously in the chapter, for more details on these operators.

MULTIPLE OPERATORS AND ORDER OF PRECEDENCE

All but the oldest or simplest programming languages can handle more than one operator in an expression. This allows you to combine three or more numbers, strings, or variables to make complex expressions, such as $5 + 10 / 2 * 7$.

This feature of multiple operators comes with a penalty, however. You must be careful of the *order of precedence*, that is, the order in which the program evaluates an expression. Some languages evaluate expressions using a strict left-to-right process, while others follow a specified pattern whereby certain operators are dealt with first, then others. When the latter approach is used, a common order of precedence is as follows:

Order	Operator
1	– (unary minus), + (unary plus), ~ (bitwise <i>Not</i>), Not (logical <i>Not</i>)
2	* (multiply), / (divide), % or MOD (mod), DIV (integer divide)
3	+ (add), – (subtract)
4	<< (shift left) >> (shift right)
5	< (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to), <> (not equal), = (equal)
6	& (bitwise <i>And</i>), (bitwise <i>Or</i>), ^ (bitwise <i>Xor</i>)
7	And (logical <i>And</i>), Xor (logical <i>Xor</i>)
8	Or (logical <i>Or</i>)

The programming language usually does not distinguish between operators that are on the same level of precedence. If it encounters a + for addition and a – for subtraction, it will evaluate the expression by using the first operator it encounters, going from left to right. You can often specify another calculation order by using parentheses. Values and operators inside the parentheses are evaluated first.

On the Web: More Programming Fundamentals

There's more that I'd like to discuss about programming, but I'm running out of space. I've put together several short articles on the RBB Online Support site (see Appendix A) that provides additional programming fundamentals. Topics include:

- Using bitwise operators—work with the individual bits of a byte or larger value.
- Using operators with text strings—compare, match case, find specific text within a string.
- Additional programming statements and concepts.
- Techniques for writing your programs to reduce memory use.

This page intentionally left blank

Microcontroller Brains

This page intentionally left blank

Using the Arduino

Microcontrollers are now so commonplace that you have your pick of hundreds of makes and models, from the supersimple to the confoundingly complex. Somewhere in the middle is the Arduino, a small and affordable microcontroller development board that's fast becoming something of a superstar.

What's made the Arduino a darling of geeks the world over is this: both its hardware design and its software are open source. That means others are able to take the best ideas and improve on them, all without paying licensing fees. This has created something of a cottage industry of fans and third-party support.

In this chapter you'll read about what the Arduino is and how to apply it to your robotics chores. Be sure to see the chapters in Part 8 for numerous working examples of using the Arduino in real-world applications, and also check out the bonus programming examples on the RBB Online Support site. See Appendix A for details.

Arduino under the Hood

First introduced in 2005, the Arduino has gone through numerous iterations, revisions, and improvements. Figure 37-1 shows what might be called the main or core Arduino board design: it's a printed circuit board that measures 2-1/8" by 2-3/4", containing an Atmel ATmega microcontroller chip running at 16 MHz, a power jack for a 2.1mm (center positive) barrel connector, and a USB Type B jack for hooking up to a host computer. Example models of this design include the Uno and Duemilanove.

A series of 28 female pin headers allow connection of external devices to the Arduino. The headers are separated into three groups: power, analog input, and digital input/output.

Of the 28 pins, 20 are devoted to input and output. There are 6 analog input pins, which can also serve as general-purpose digital I/O. The 14 digital input/output pins include 6 that can be used to generate PWM (pulse width modulated) signals, useful for such things as con-

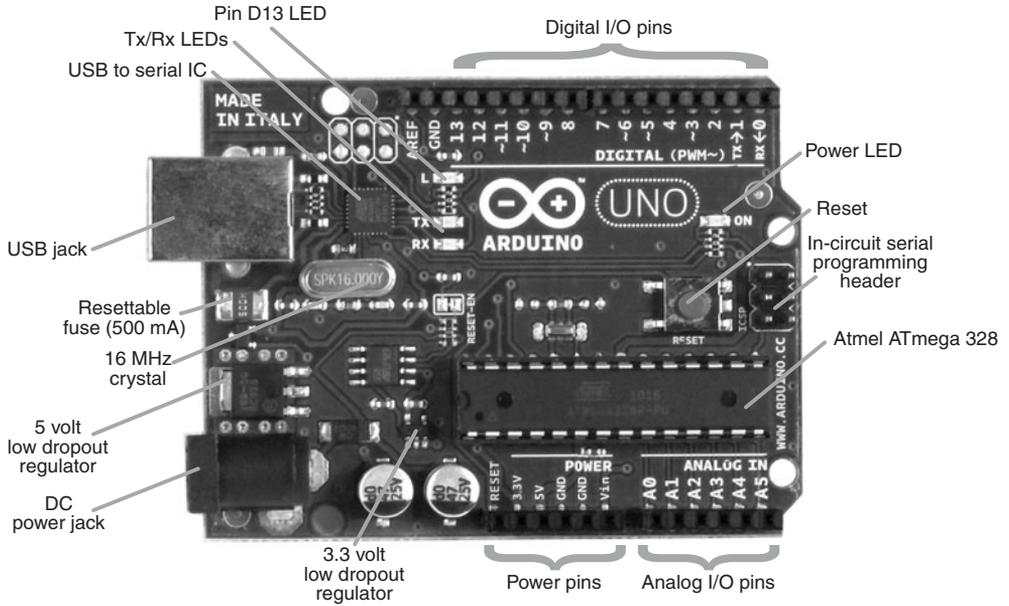


Figure 37-1 Points of interest on the Arduino board include the USB and power jacks, function and power LEDs, and rows of connection headers.

trolling the speed of motors. All I/O pins can be used as digital outputs and can sink or source up to 40 milliamps.

At the heart of the Arduino is an Atmel ATmega microcontroller. The exact version of chip depends on the version of the Arduino. For example, the older Diecimila used an Atmel ATmega168; later versions have used an ATmega328. The two chips are physically identical, but the '328 offers more memory space.



Let me pause here to point out that the microcontroller chips used on the Arduino are not “empty.” They, in fact, come preloaded with a small *bootloader* program for use with the Arduino development editor, described later in this chapter. The bootloader assists in the download process.

Many Variations on a Theme

The core board design, like that in Figure 37-1, is perhaps the most common and popular of the Arduinos, but there are numerous other variations. Here are just some of the standardized Arduino boards you’ll encounter—note that versions and names may change over time:

- The Arduino BT and Fio are intended for wireless applications. The BT contains a Bluetooth module; the Fio has a built-in Zigbee radio.
- The Nano is a compact stick-shaped board made for breadboard use. It has all the main features of the core boards (including built-in USB jack), but measures only 0.73” × 1.7”.

- The Mini is even smaller, and is ideal for very small bots with limited space. The Mini lacks its own USB jack and requires the use of a USB adapter or serial TTL connection to the host PC for programming.
- The Mega2560 is based on a larger Atmel chip, and it offers over twice the number of analog and digital I/O lines. Memory and program space are larger, too.

Several Arduino resellers offer their own custom offshoots of the Arduino—these typically go by different names, such as Boarduino or Freeduino, to differentiate them from the original Arduino designs.

Some variations of the Arduino depart from the standard form factor of the core Arduino boards and are not designed for use with expansion shields, discussed later in this chapter. A good example is the LilyPad, a special Arduino layout engineered for making—among other things—wearable microcontroller projects. Think Borg implants, only more friendly looking. The flower-shaped LilyPad has a flat profile and can be sewn into fabric. It has connection points on the ends of its 22 petals.

Ready Expansion via Shields

The Arduino itself has no breadboard area, but it's easy enough to connect any of the inputs or outputs to a small breadboard via wires. For an application like robotics, you'll want to expand the Arduino I/O headers to make it easier to plug in things like motors, switches, and ultrasonic or infrared sensors.

One method is to use an add-on expansion board known as a *shield*. Shields stick directly on top of the core board and Mega designs. Pins on the underside of the shield insert directly into the Arduino's I/O headers. Two popular expansion shields are the solderless breadboard shield (see Figure 37-2) and the proto shield; both provide prototyping areas for expanding your circuit designs.

Of course, you don't absolutely need a shield to expand the Arduino. You can place a breadboard—solderless or otherwise—beside the Arduino and use ribbon cables or hookup wire to connect the two together.

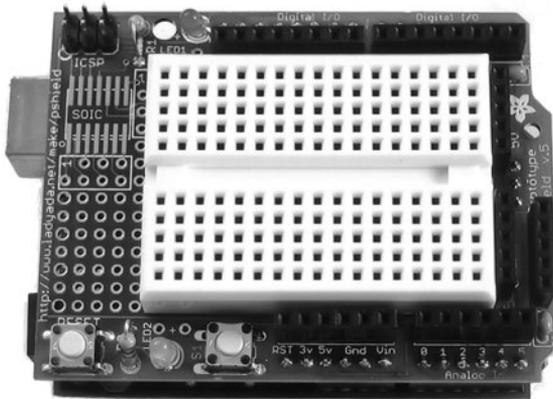


Figure 37-2 Expansion shield for the Arduino, offering a small solderless breadboard for experimenting. Use short jumper wires to connect the Arduino pins (top and bottom) to the contact points in the breadboard. (Photo courtesy Adafruit Industries.)

USB Connection and Power

To allow the easiest possible means of programming, all the core Arduino boards support USB. You need merely to connect a suitable USB cable between the Arduino and your computer. The necessary USB drivers are provided with the Arduino software. In most cases, installation of the drivers is not fully automatic, but the steps are straightforward, and the Arduino support pages provide a walk-through example.

The USB jack provides communications, both for downloading programs from your PC (discussed later), and for serial communications back to the PC. The USB link includes a 500-mA resettable fuse to guard against possible damage caused by a wayward Arduino to the USB ports on your PC. When plugged into a USB port, the Arduino takes its power from it. With USB 2.0, drive current is limited to 500 mA, depending on the port design.

Operating voltage of the Arduino circuitry is 5 volts, which is supplied either by the USB cable when it's plugged into a USB port on your computer or by a built-in low-dropout linear voltage regulator when the board is powered externally. The regulator is intended to be powered by 7 to 12 vdc; a 9-volt battery is ideal. Anything higher than 12 volts is not recommended, as it could cause the regulator to overheat.



The Arduino is also equipped with a 3.3-volt low-dropout voltage regulator. Depending on the version of the Arduino board, the regulator is either built into the USB-to-serial chip or separate. In either case, maximum current output is rather low, on the order of 50 mA. The 3.3-volt regulator is good for powering small electronics that require the lower voltage. These include certain types of accelerometers and gyroscopes.

For robotics I think it's best to power the Arduino from its own battery, and use different batteries for the motors. You can make your own 9-volt battery to 2.1mm barrel connector jumper cable or purchase one ready-made (see Figure 37-3).

Indicator LEDs are provided on the Arduino for testing and verification. One LED shows power; two other LEDs show serial transmit and receive activity, and should flash when the board is being programmed from your computer. A fourth LED is connected in parallel with digital I/O line 13 and serves as a simple way to test the Arduino and make sure it is working properly.



Figure 37-3 Ready-made cable for connecting a 9-volt battery to the Arduino 2.1mm power jack. (Photo courtesy Adafruit Industries.)

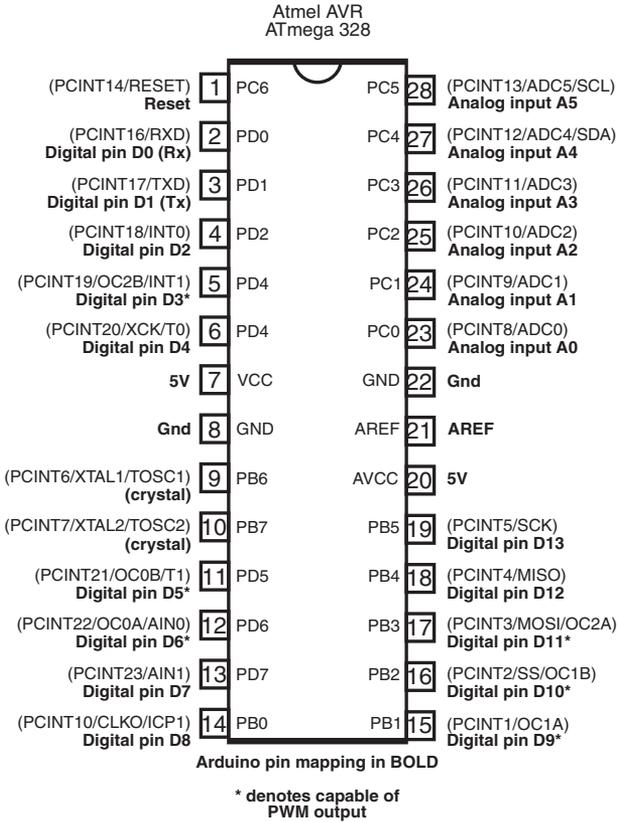


Figure 37-4 Pinout diagram of the Atmel ATmega328 chip, with the pin mapping to the Arduino I/O lines.

Arduino Pin Mapping

The Arduino uses its own nomenclature for its I/O pins, and the names and numbers of its pins don't correlate to those on the ATmega microcontroller. This can cause some confusion if you're already familiar with working with the bare ATmega chips.

Figure 37-4 shows the pinout diagram of the 28-pin ATmega328 microcontroller chip. The labels on the inside of the chip are the primary function names for each of the pins. The labels outside in parentheses are alternative uses, if any, for the pins.

Also shown in Figure 37-4 is pin mapping between the Arduino and the ATmega328. It's important to remember that the pin numbers are not the same between the two: pin 12 on the ATmega328 is actually mapped to digital pin D6 on the Arduino, for example. Pin mapping is not something you need to worry about in typical Arduino programming, but it's nice to know what leads to where.

Programming the Arduino

Microcontrollers depend on a host computer for developing and compiling programs. The software used on the host computer is known as an *integrated development environment*, or *IDE*. The Arduino language is based on good old-fashioned C. If you are unfamiliar with this language, don't worry; it's not hard to learn, and the Arduino IDE provides some feedback when you make mistakes in your programs.

To get started with programming the Arduino using its IDE, first go to www.arduino.cc, and then click on the Download tab. Find the platform link (PC, Mac, Linux) for your computer, and download the installation file. Step-by-step instructions are provided in the Getting Started section of the Arduino Web site. Be sure to read through the entire set of instructions.

Once installation is complete, you're ready to try out your Arduino. Start by connecting the board to your PC via a USB cable. If this is the first time you've used an Arduino on your PC, you must install the USB communications drivers, as detailed in the Getting Started guide.

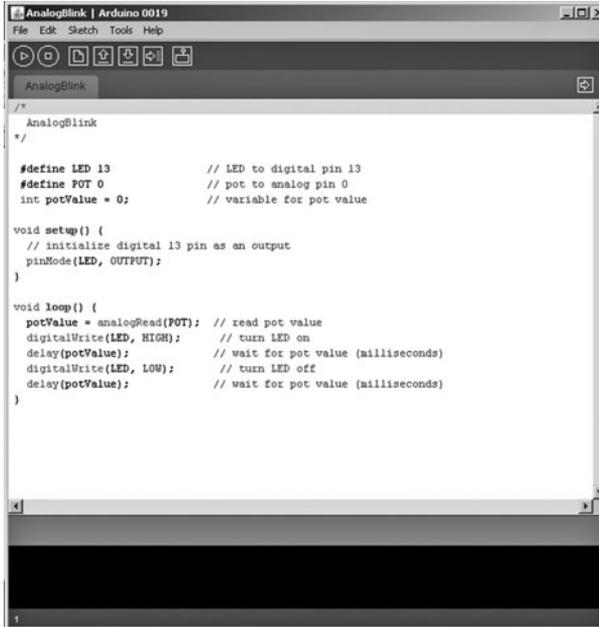


Figure 37-5 The Arduino integrated development environment (IDE) provides a centralized place to write, compile, and download programs to the Arduino board.

USING THE ARDUINO IDE: THE BASICS

Using the Arduino programming environment is simple. First-time use of the environment requires you to specify the Arduino board you are using and, as necessary, the serial port that is connected to the board (the Arduino’s USB connection looks like a serial port to your computer). You may then open an existing example program and download it to your board. Or you may write your own program in the IDE editor. Figure 37-5 shows the Arduino IDE with a short sketch in the main window.

The next step is to compile the program—called “sketches” in Arduino parlance. This prepares the code for downloading to the Arduino. At the bottom of the text editor is a status window, which shows you the progress of compiling. If the program was successfully compiled, it can then be downloaded to the Arduino, where it will automatically run once the download is complete.

A WORD ABOUT IDE VERSIONS

The Arduino IDE and the standard programming statements and libraries often undergo changes with each new version. The Servo library discussed later in this chapter was introduced in version 0017 of the Arduino IDE. If you already have an installed version of the IDE and it’s old, you’ll want to fetch the newest version. You can keep multiple versions of the Arduino IDE on your computer and even switch between them as needed—though that should seldom be required.

All of the programming examples in this book require version 0017 or later of the Arduino IDE, so make sure yours is compatible. The Arduino IDE is set to always check for the latest updates.

Programming for Robots

The Arduino supports the notion of *libraries*, code repositories that extend core programming functionality. Libraries let you reuse code without having to physically copy and paste it into all your programs. The standard Arduino software installation comes with several libraries you may use, and you can download others from the Arduino support pages, and from third-party Web sites that publish Arduino library code.

A good example of a library you'll use with most any robot is *Servo*. This library allows you to connect one or more hobby R/C servos to the Arduino's digital I/O pins. The *Servo* library comes with the standard Arduino installation package, so adding it to your sketch is as simple as choosing Sketch->Import Library->Servo.

Structurally, Arduino sketches are very straight forward and are pretty easy to read and understand. All Arduino sketches have at least two parts, named *setup()* and *loop()*. These are called *functions*, and they appear in the sketch like this:

```
void setup() {  
}  
  
void loop() {  
}
```

- The (and) parentheses are for any optional arguments (data to be used by the function) for use in the function. In the case of *setup()* and *loop()*, there are no arguments, but the parentheses have to be there just the same.
- The { and } braces define the function itself. Code between the braces is construed as belonging to that function—the braces form what's referred to as a *code block*. There's no code shown here, so the braces are empty, but they have to be there.
- The *void* in front of both function names tells the compiler that the function doesn't return a value when it's finished processing. Other functions you might use, or create yourself, may return a value when they are done. The value can be used in another part of the sketch.
- The *setup()* and *loop()* functions are required. Your program must have them, or the IDE will report an error when you compile the sketch. These are programming functions that do what their names suggest: *setup()* sets up the Arduino hardware, such as specifying which I/O lines you plan to use. The *loop()* function is repeated endlessly when the Arduino is operating.

Many Arduino sketches also have a *global declaration* section at the top. Among other things, the declaration is where you put variables for use by the whole program—see the example in the next section. It's also a common place to tell the IDE that you wish to use an external library, like *Servo*, to extend the base functionality of the Arduino.

USING VARIABLES

Arduino uses *variables* to store information while your sketch is running. The platform supports a number of *data types* for holding variables of different types and sizes. Among the most common are

- *int*—holds a signed (can be positive or negative) whole number (integer), where the value can be from $-32,768$ to $32,767$.
- *unsigned int*—holds an unsigned (positive only) number from 0 to 65,535.
- *byte*—holds an unsigned number from 0 to 255.
- *boolean*—holds a true or false value.
- *float*—holds a floating-point value, where the digits to the right of the decimal can have up to 15 places.
- *String*—holds text, usually meant for display on an LCD panel or transmission back to the PC as a message.

To use a variable, you must first declare it. This may be done at the top of the sketch or anywhere within it. Where you declare a variable determines its *scope*: variables declared at the top of the sketch are global; that is, they can be used anywhere within the sketch. A variable declared inside a function can be used only within that function.

Declaring (or *defining*) a variable requires you to first specify its type. You then indicate its name, followed by an optional step of assigning a value to the variable just declared:

```
int myInt = 30;
```

declares an *int* variable named *myInt*, plus it assigns a value of 30 to it. This one line is equivalent to:

```
int myInt;  
myInt = 30;
```

Names for variables must contain only letters and numbers or the underscore (`_`) character. The name can't start with a number, and it can't contain a space. Capitalization matters, so *myInt* is distinctly different from *myint*.

Many programmers prefer a consistent naming convention for their variables. This includes using consistent capitalization. The common practice today is called *camelCase*—low on the ends, high in the middle.

An exception to this is when using constants, variables whose contents are defined once and never changed. The common practice here is to use all uppercase characters, as in

```
const int POT = A0;
```

for a constant variable named *POT*.

USING ARDUINO PINS

The input/output pins of the Arduino are referenced by number. There are two numbering sequences: one for the analog pins and another for the digital pins.

- Analog pins are referenced as *Ax*, where *x* is a number. For example, to reference analog pin number 0, you'd use *A0*.
- Digital pins are referenced in sketches just by their numbers. (Additionally, within this book the digital pins are described as *Dx*—example: *D13* or *D9*—to avoid any confusion about which pins to connect things to.)

In actual use, most of the Arduino programming statements are self-aware of the proper pins—*analog* or *digital*—to use. For example, when using the *analogRead* programming statement, which reads a voltage level at a pin, the compiler knows you're talking about an analog pin, as the digital pins don't support this feature.

By default, the digital pins are automatically considered as inputs, meaning they are set up to read a value, rather than set a value. At any point in a sketch you can inform the Arduino that you wish to use a pin as an output. The process is simple:

```
pinMode(PinNumber, Direction);
```

where *PinNumber* is the number of the pin you'd like to use, and *direction* is either *INPUT* or *OUTPUT* (these are predefined constants, by the way—that's why they're in uppercase). For example,

```
pinMode(13, OUTPUT);
```

makes pin D13 an output. Once made an output, it can do output-like things, such as light up an LED. Such an example follows later in this chapter.

Digital pins may be on or off, defined as 0 or *LOW* (off), or 1 or *HIGH* (on).

- Use *digitalWrite* when setting the value of a digital pin.
- Use *digitalRead* when testing the value of a digital pin. You typically use a variable to store the value of the pin.

For example,

```
digitalWrite(13, HIGH);
```

turns pin D13 on (sets it *HIGH*).

```
myVar = digitalRead(13);
```

reads pin D13, and assigns its value (either *LOW* or *HIGH*) to the *myVar* variable.

EXPERIMENT BY DOING

Program *arduino_test.pde*, shown below, demonstrates a few fundamental Arduino concepts useful in any robotics development, reading an analog sensor and providing visual feedback. I've taken one of the examples that come with the Arduino IDE and modified it slightly to conform to the style used throughout this book. It uses a 10 kΩ potentiometer to alter how fast Arduino's built-in LED flashes.

Check out Figure 37-6 for a schematic of the circuit used for the program listing. The potentiometer is connected to the board as a common voltage divider. That way, the Arduino detects the value of the pot as a variable voltage from 0 volts (ground) to 5 volts.



Be careful with how you wire the potentiometer. Be sure the wiper is connected to pin A0, and the legs of the pot are connected to 5V and ground. If you accidentally connect the wiper to 5V or ground, you'll create a dead short across the pot as you rotate it. The pot may burn out, and the Arduino will go into "fail-safe" protection mode by shutting down.

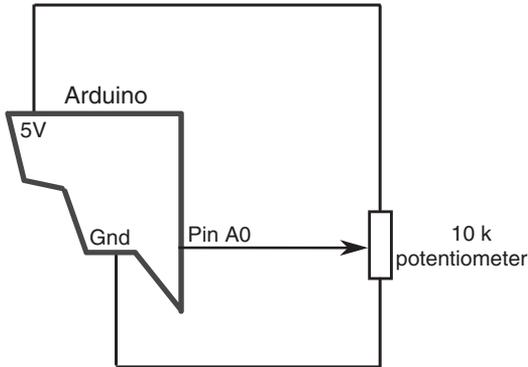


Figure 37-6 Schematic diagram for the *arduino_test.pde* sketch. The 10 k Ω pot is connected to the 5V and Gnd pins of the Arduino, as well as analog pin A0.

```

101010 arduino_test.pde
010101  const int LED = 13;           // LED to digital pin D13
101010  const int POT = A0        // pot to analog pin A0
010101  int potValue = 0;        // variable for pot value

void setup() {
  // initialize D13 as an output
  pinMode(LED, OUTPUT);
}

void loop() {
  potValue = analogRead(POT); // read pot value
  digitalWrite(LED, HIGH);    // turn LED on
  delay(potValue);            // wait for value (milliseconds)
  digitalWrite(LED, LOW);     // turn LED off
  delay(potValue);            // wait for value (milliseconds)
}

```

Here's how the program works:

The first two lines set variable constants, so hardware connected to the various I/O pins can be referred to by name, and not pin number. This is merely a convenience. The built-in LED is connected to pin D13, and the potentiometer—named POT in the program—is connected to analog pin A0.

Another variable is defined to hold the current value of the potentiometer, which will be a number from 0 to 1023. This number is derived from the Arduino's integrated 10-bit analog-to-digital (ADC) converter, and it represents a voltage level from 0 volts to 5 volts.

The `setup()` section gets the Arduino hardware ready for the rest of the program. When first powered on, all the I/O lines are automatically defined as inputs. But the LED pin needs to be an output, so that distinction is defined here.

The `loop()` section is automatically started the moment the program has been downloaded to the Arduino. Looping continues until either the board is unplugged, the Reset button on the Arduino is pushed, or a new program is loaded into memory. The loop begins by reading the voltage on analog pin A0. The program then turns the LED on and waits for a period of time defined by the current position of the potentiometer before turning the LED off again.

The waiting period is in milliseconds (thousandths of a second), from 0 to 1023, the range of values from the Arduino's ADC. Very fast delays of about 100 milliseconds or less will appear as a steady light.

Using Servos

The following program swings the servo motor in one direction, then the other, briefly pausing in between. You can use either an unmodified or a modified (continuous rotation) servo to see the code in action. Refer to Figure 37-7 for a diagram on hooking up the servo. Use a standard-size (or smaller) analog servo; stay away from larger or digital servos, as they may draw too much current for the USB port on your computer to handle.



In the following example, text after the double slash `//` characters means a comment. It's for us humans. During the compiling phase, comments are ignored, as they are not part of the functionality of the sketch.

```

101010 basic_servo_test.pde
010101 #include <Servo.h> // Use the Servo library, included with
101010 // the Arduino IDE (version 0017 or later)
010101
Servo myServo; // Create Servo object to control the servo
int delayTime = 2000; // Delay period, in milliseconds

void setup()
{
  myServo.attach(10); // Servo is connected to pin D10
}

void loop()
{
  myServo.write(0); // Rotate servo to position 0
  delay(delayTime); // Wait delay
  myServo.write(180); // Rotate servo to position 180
  delay(delayTime); // Wait again
}

```

The first line, `#include <Servo.h>`, tells the IDE that you want to use the Servo library, which is a standard part of the Arduino IDE installation. (Other libraries may require a separate download, but they are used in the same way.) The name of the main Servo library file is `Servo.h`, so that is what's provided here.

Servo is actually a name of a *class*; that's how Arduino uses its libraries. With a class you can create multiple instances (copies) of an object without having to duplicate lots of code. Note that `Servo` is the name of the class to use, and `myServo` is the name I've given to the object just created.

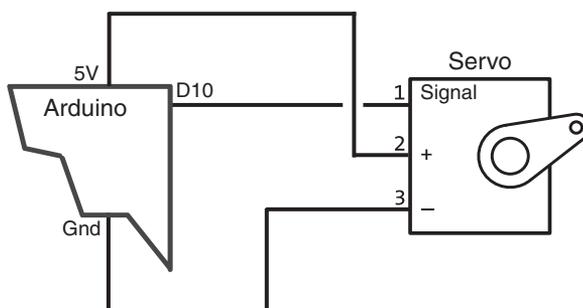


Figure 37-7 Connection diagram for testing servo functionality with the Arduino. Be absolutely sure of observing the correct polarity of the power and Ground connections to the servo.

The `setup()` function contains one statement, `myServo.attach(10)`. Here's what it all means:

- `myServo` is the name of the servo object that was defined earlier.
- `attach` is what's known as a *method*. Methods are actions that you use to control the behavior of objects. In this case, `attach` tells the Arduino that you have physically connected the servo to digital pin D10, and you want to activate it. A period separates the object name and method—`myServo.attach`.

Notice the `;` (semicolon) at the end of the statement. It's a statement terminator. This tells the compiler that the statement is complete and to go on to the next line.

The `loop()` function contains the part of the sketch that is repeated over and over again, until you download a new program or remove power from the Arduino.

- `myServo.write(0)` is another method using the `myServo` object. The `write` method instructs the servo to move all the way in one direction. When using a modified servo, this statement causes the motor to continually rotate in one direction.
- `delay(delayTime)` tells the Arduino to wait the period specified earlier in the `delayTime` variable, which is 2000 milliseconds (2 seconds).
- The two statements are repeated again, this time with `myServo.write(180)` to make the servo go in the other direction.

Here's an important note about capitalization of variables, objects, and statement names. Like all languages based on C, these names are case sensitive. This means that `myServo` is distinctly different from `myservo`, `MYSERVO`, and other variations. If you try to use

```
myservo.attach(10);
```

(note the lowercase "s") when you've defined the object as `myServo`, the Arduino IDE will report an error—"myservo not declared in this scope." If you get this error, double-check your capitals.

Creating Your Own Functions

The flexibility of any programming language, Arduino included, comes in the ways you can develop reusable code, such as creating user-defined functions. To create a user-defined function, you give it a unique name and place the code you want inside a pair of brace characters, like so:

```
void forward() {
  myServo.write(0);
  delay(delayTime);
}
```

All user-defined functions must indicate the kind of data they return (for use elsewhere in the sketch). If the function doesn't return any data, you use `void` instead. You must also include parentheses to enclose any parameters that may be provided for use in the function. In the case of the `forward` user-defined function, there are no parameters, but remember that you need the `(` and `)` characters just the same.

That defines the function; you need only call it from elsewhere in your sketch to use it. Just type its name, followed by a semicolon to mark the end of the statement line:

```
forward();
```

See *arduino_servo.pde* for a full demonstration of an Arduino sketch that runs a servo forward and backward, then briefly stops it, using the `detach` method. The effect of the sketch is most easily seen when using a servo modified for continuous rotation.

```
101010
010101
101010
010101
```

arduino-servo.pde

```
#include <Servo.h>

Servo myServo;           // Create Servo object
int delayTime = 2000;    // Standard delay period (2 secs)
const int servoPin = 10; // Use pin D10 for the servo

void setup() {           // Empty setup
}

void loop() {            // Repeat these steps
  forward();             // Call forward, reverse, servoStop
  reverse();             // user-defined functions
  servoStop();
  delay(3000);
}

void forward() {         // Attach servo, go forward
  myServo.attach(servoPin); // for delay period
  myServo.write(0);
  delay(delayTime);
  myServo.detach();     // Detach servo when done
}

void reverse() {         // Do same for other direction
  myServo.attach(servoPin);
  myServo.write(180);
  delay(delayTime);
  myServo.detach();
}

void servoStop() {      // Stop the servo by detaching
  myServo.detach();
}
```

On the Web: Operating Two Servos

The Arduino can control two servos with the same ease as one. All it takes is that you create a second instance (copy) of the `Servo` object, giving it a unique name. For example, in a two-wheeled differentially steered robot you might call one servo object `servoLeft` and the other `servoRight`.

As a bonus project, check out the RBB Online Support site (see Appendix A) for full details, hookup diagrams, and example code for using the Arduino with two motors.

On the support site you'll also find extended code and additional project ideas for a project I call the *ArdBot*, a low-cost and expandable platform for experimenting with Arduino robotics. The basic *ArdBot*, with Arduino and prototyping board, is shown in Figure 37-8.

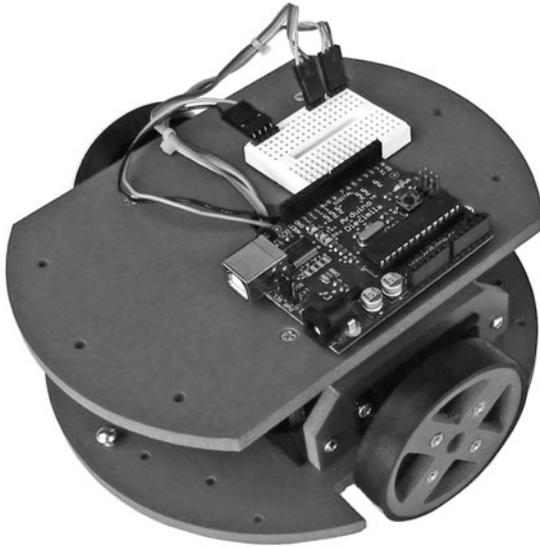


Figure 37-8 The “ArdBot,” with Arduino microcontroller and mini solderless breadboard for experimenting. Learn more about this robot on the RBB Online Support site.

Flow Control Structures

Flow control structures tell the sketch what conditions must be met to perform a given task. By far the most common flow control structure is the *if* command. It tests if a condition exists or doesn't exist, then branches off execution of the program accordingly. The *if* structure looks like this:

```
if(condition) {
  // True: condition is met
} else {
  // False: condition is not met
}
```

Condition is an expression, usually something that determines if A equals B, as in:

```
if(digitalRead(9) == HIGH)
```

This tests if digital pin D9 is HIGH. If it is, then the program performs the *True* code, because the condition is met. Otherwise, it performs the *False* code. A complete example goes like this, where the value of digital pin D9 makes the LED on pin D13 briefly flash:

```
pinMode(13, OUTPUT);
if(digitalRead(9) == HIGH) { // Remember: no semicolon here
  digitalWrite(13, HIGH);
  delay (1000);
  digitalWrite(13, LOW);
}
```

Note that in this case there is no code for when the condition is False. This is acceptable, and quite common. Because there is no code for when the condition is False, the *else* command is omitted, as are its corresponding brace characters.

Be mindful of the curly braces! You need a set of curly braces to enclose the code for any True condition.

```
if(blah-blah) {
  // True code here
}
```



And if you want separate code for when the condition is False, you must add the else command and its set of braces:

```
else {
  // False code here
}
```

Conditions for the if test typically test that something is equal to something else, but there are other possible comparisons. Here are the most common:

==	equal to
<>	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Take special note of double equals signs for *equal to*. Using = will not give you what you want. You need to use ==.

A practical example is testing if the voltage on an analog pin is above or below the halfway point—the halfway point being 511 (possible values are 0 to 1023). The following short example tests if the value at analog pin A0 is above the halfway point, or 512 and over. If it is, the LED on digital pin D13 is turned on.

```
if(analogRead(A0) >= 512) {
  digitalWrite(13, HIGH);
}
```

Using the Serial Monitor Window

The Arduino provides a quick and easy way to get feedback about any running program when it's tethered to its host PC via the USB cable. Built into the Arduino is the ability to talk to the PC via serial communications. You merely set up the serial link with

```
Serial.begin(9600);
```

and then send values or text to the PC with

```
Serial.println(value);
```

The result is shown in the Serial Monitor window, which you display by choosing Tools > Serial Monitor.

For example, suppose you want to see the digital value of a voltage applied to analog input pin A0. Set up the communications link in the `setup()` function, then repeatedly read the analog pin and return its value into the Serial Monitor window.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue, DEC);
  delay(100);
}
```



Note the optional data format in the `Serial.println` statement. The `DEC` tells the Arduino to return the value as a DECimal number. Also notice the delay statement. It's usually a good idea to include a slight delay to slow down processing when communicating back with your PC. The Arduino can process these commands faster than the serial link can accommodate, and there's a risk of losing data if the data is shuttled too quickly.

Some Common Robotic Functions

Arduino supports a number of built-in programming statements specifically useful for robotics. I'll briefly review just a few of them here, but you'll want to review these and others more fully in the Arduino language reference, available on the www.arduino.cc Web site. Examples of most of these statements can be found in Part 8 of this book.

- *tone*. Outputs a frequency on an I/O pin that, when connected to a piezo buzzer element, generates a tone. The tone to play is specified by its frequency, with a value of 440 being concert pitch A.
- *shiftOut*. Converts a byte (or more) of data to a stream of bits, which can then be sent one at a time to some external device, typically a *shift register* IC. The shift register receives the serial data and reconstructs it as parallel data outputs. The *shiftOut* statement is useful for minimizing the number of I/O pins used with external devices.
- *pulseIn*. Measures the width of a single pulse, from 10 μ s to 3 minutes. You can specify which I/O pin to use, whether you're looking for a LOW-to-HIGH or HIGH-to-LOW transition.
- *analogWrite*. Outputs a series of pulses whose duration, or *duty cycle*, is controlled via software—so-called pulse width modulation, or PWM. It's very commonly used to control the speed of a DC motor. The *analogWrite* statement may be used only with specifically marked digital pins. On the core board designs (Duemilanove, Uno, etc.), these pins are 3, 5, 6, 9, 10, and 11.

FYI

See Chapter 22, "Using DC Motors," for more information on PWM, especially as it relates to controlling the speed of motors.

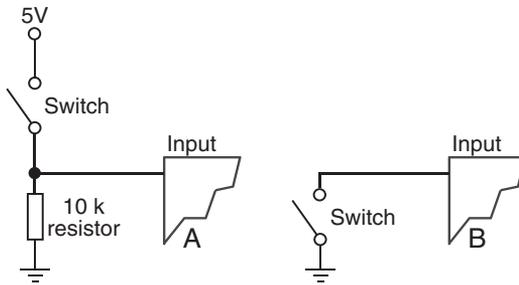


Figure 37-9 Standard connections for reading the value of a simple switch. In A, the switch is used with a 10 k Ω pull-down resistor. This keeps the value of the pin LOW until the switch is closed. In B, the switch relies on the Arduino's built-in pull-up resistor to keep the pin HIGH until the switch is closed.

Using Switches and Other Digital Inputs

Reading the value of a switch or other digital input is a straightforward affair with the Arduino. Simply use the `digitalRead` statement and indicate the digital pin you wish to use:

```
if(digitalRead(10) == HIGH) {
  Serial.println("Pin 10 is high.");
}
```

Figure 37-9A shows the traditional method of connecting a switch to a digital input pin. The 10 k Ω pull-down resistor ensures that the pin stays LOW as long as the switch is open. When the switch closes, the pin goes HIGH.

While resistors are the most common approach for wiring a switch to an Arduino, they are not strictly required, as the Arduino has its own pull-up resistors that can be turned on and off. When the pull-up resistors are engaged, the pins stay at a HIGH level, unless taken LOW by a switch or other input. To set the pull-up resistor, specify that the pin is an INPUT, then set its value to HIGH with `digitalWrite`.

```
pinMode(10, INPUT);           // set pin to input
digitalWrite(10, HIGH);       // turn on pull-up resistor
```

In this case, you can do away with the resistor, as shown in Figure 37-9B. Note that the built-in resistors are pull-ups, which means the switch goes LOW when activated. Your program code needs to be changed accordingly:

```
if(digitalRead(10) == LOW)
```

Interfacing to DC Motors

You've already seen how to use the Arduino with servo motors. You can also use the microcontroller with DC motors. The I/O pins on the Arduino can provide only about 40 milliamps of current, not enough to directly power the typical DC motor. But as described in Chapter 40, "Interfacing Hardware with Your Microcontroller or Computer," you can use a motor H-bridge or other driver to run most any size of DC motor with your Arduino.

- To turn the motor on, apply HIGH to the Enable/PWM line.
- To control the direction of the motor, apply HIGH or LOW to the Direction line of the H-bridge.

- To control the speed of the motor, use the *analogWrite* statement to send a PWM signal to the Enable/PWM line. (Remember: To do this, you must connect the line to one of the digital I/O pins on the Arduino that supports PWM.)

In the following sketch, a motor bridge module is connected to digital pins D11 (for Enable/PWM) and D12 (for Direction). After setting up the two pins as outputs, the sketch repeats the steps of turning the motor on fully, reversing its direction, then slowing it down to 50 percent.

```

101010  arduino_motor_control.pde
010101  int motDirection = 12;           // Direction line to pin D12
101010  int motEnable = 11;          // Enable/PWM to pin D11
010101

void setup() {
  // Pins 11 and 12 as outputs
  pinMode(motDirection, OUTPUT);
  pinMode(motEnable, OUTPUT);
}

void loop() {
  digitalWrite(motDirection, LOW);      // Set direction
  digitalWrite(motEnable, HIGH);       // Turn motor on
  delay(2000);                          // Wait 2 seconds
  digitalWrite(motDirection, HIGH);    // Reverse direction
  delay(2000);                          // Wait 2 seconds
  digitalWrite(motEnable, LOW);        // Turn motor off
  digitalWrite(motDirection, LOW);     // Set direction
  analogWrite(motEnable, 128);         // Set motor to half speed
  delay(2000);                          // Wait 2 seconds
}

```



The PWM pulsed output is disabled when you use *digitalWrite* on the same pin. In the example, *digitalWrite* is used to enable the motor at full speed, but you can also use *analogWrite* and set the PWM duty cycle to 255. However, *digitalWrite* is a more efficient use of the microcontroller when controlling the speed of the motor is not needed.

If your motor doesn't turn when its PWM duty cycle is set to 128 (50 percent), try a higher value. Not all motors will run at duty cycles below 50 percent.

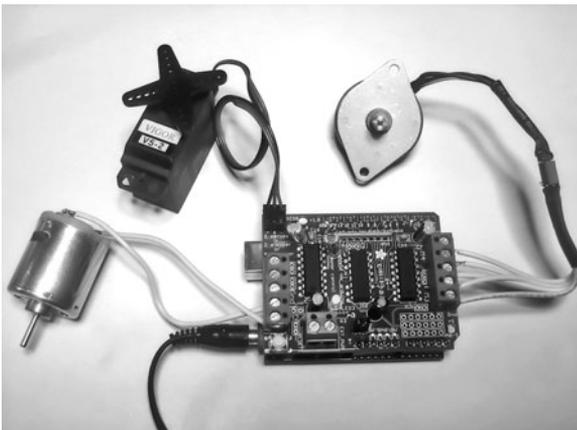


Figure 37-10 Expansion shield for testing and working with various kinds of motors. This model supports R/C servos, stepper motors, and DC motors. (Photo courtesy Adafruit Industries.)

Besides fashioning your own H-bridge circuit or board, you can use any of a number of third-party expansion shields that contain their own H-bridge modules. An example is shown in Figure 37-10, which offers connection points for working with two servos, two stepper motors, and up to four DC motors. The shield is designed so that you can provide separate power to the motors than what's supplied to the Arduino.

Using the PICAXE

Microcontrollers have become the favorite method for endowing robots with smarts. They're single-chip computers complete with their own input/output ports and memory. They're programmed from your PC, and, once programmed, they can be disconnected from the PC and operate on their own.

As detailed in Chapter 35, "Understanding Microcontrollers," they are available in two basic flavors, referred to in this book as low-level programmable and integrated-language programmable. These loosely defined terms relate to the programming of the controller.

The PICAXE series of microcontrollers are among the better known, and most widely used, integrated-language programmable devices available today (another is the BASIC Stamp, detailed in its own chapter, next). The PICAXE isn't just one controller; it's a series of them, each with distinct capabilities. All are very affordably priced, and they are designed to operate with a minimum of external parts.

As you'll read in this chapter, you can plug a PICAXE into a solderless breadboard and, with just a few resistors or so, have a complete microcontroller system ready for your use.

FYI

Be sure to see the chapters in Part 8 for numerous working examples of using the PICAXE in real-world applications. There are additional bonus examples and projects using the PICAXE on the RBB Online Support site. See Appendix A for details.

Understanding the PICAXE Family

PICAXE is a family of microcontrollers. Versions are available with as few as 8 pins and 5 I/O lines, to 40 pins and over 30 I/O lines. I won't talk about each one here but, rather, I'll take a look at two of the most commonly used devices.



The makers of the PICAXE, Revolution Education, update their product lineup from time to time, so you should always check their Web site at www.picaxe.co.uk for the latest. As of this writing, they have several series of PICAXE controllers, broadly classified into three segments: education, standard, and advanced.

They further categorize the PICAXE into M, M2, X, X1, and X2, with functionality varying between each. In this chapter you'll read about the 08M, the smallest and least expensive of the bunch, but very capable anyway, and the 18M2, a "middle-of-the-road" controller that's still very low in cost, offering additional features not found on its smaller siblings, and indeed, on many competing microcontrollers.

Figure 38-1 shows the pinouts for the 08M and 18M2. They differ not only in the number of physical pins on the chip, but in the ordering of the functions for the inputs and outputs. The 18M2 also supports some additional programming commands not found in the 08M. Several of these are outlined in this chapter, but the variations within the PICAXE chips—what features they support and what commands they'll run—can be a point of confusion. See the PICAXE documentation for a full understanding of the differences and capabilities within each chip in the family.

For all PICAXE chips, it's very important to remember the difference between the physical pins of the IC (called "legs" in the PICAXE documentation) and the names of the I/O lines. What's referred to as *Pin4* or *C.4* is the name given to an I/O line, for example *not* the actual pin on the chip.

The reason for this is consistency in sharing programs across the different PICAXE chips. The functions available at the different physical pins (legs) of the chip can vary, meaning that physical pin 4 on one PICAXE controller may have a completely different function than physical pin 4 on another PICAXE.

The variation in pin assignments is caused by the architecture of the microcontrollers that PICAXE chips are based on. Each PICAXE starts life as a Microchip PICMicro controller. It's

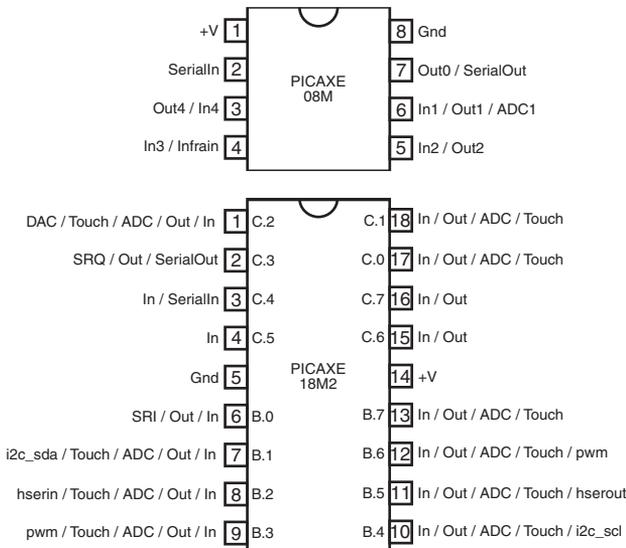


Figure 38-1 Pinout diagrams for the PICAXE 08M and 18M2 microcontrollers. There are other PICAXE chips available; refer to the PICAXE documentation for more details on the rest of the family.

made into a bona fide PICAXE by programming into it the language interpreter that takes your software and runs it once downloaded.

When you purchase a PICAXE, you're getting a PICMicro microcontroller that already has a core program included in it. When you download your own programs, they share space in the memory of the chip. When you delete your program, only your program is deleted; the language interpreter remains.

BASICS OF ALL PICAXE CONTROLLERS

All PICAXE chips require connections to power and ground. Except for some special low-voltage varieties, the PICAXE runs on power from 3 to 5.5 volts. It's common to power the chip with a set of two or three AA or AAA batteries. This provides 4.5 volts when using nonrechargeable batteries. If using rechargeable batteries rated at 1.2 volts per cell, you're better off using four cells, which provides a total of 4.8 volts.



Don't power the PICAXE with more than 5.5 volts. Otherwise, the chip will likely be destroyed within seconds.

The PICAXE is programmed via a serial port connection from your PC. Any standard serial port will do. If your computer lacks a serial port, you can use a USB-to-serial adapter cable. One made specially for the PICAXE experimenter boards is sold by Revolution Education (their part number AXE027) and has a USB Type-A connector on one end and a "stereo"-style 1/8" mini plug on the other.

All PICAXE controllers require at least two external resistors to function. The minimal circuit is illustrated in Figure 38-2, showing how the two resistors are used for the downloading process. Even when not downloading a program, the resistors must remain part of the circuit, or else the PICAXE will not behave correctly.

- The 22 k Ω series resistor limits current from the PC serial port to the chip.
- The 10 k Ω pull-down resistor prevents the SerialIn pin from floating, which can happen when the PICAXE is not actively connected to a serial port. If left floating, operation of the chip can become unstable.

Additionally, on PICAXE chips that have a Reset pin you need to connect a 4.7 k Ω resistor to the +V power, as shown in Figure 38-3 (the switch is optional, but recommended; pressing it resets the controller). Finally, the 28- and 40-pin PICAXE chips can use an external resonator, which occupies two pins on the IC. Use a three-leg ceramic resonator, as shown.

In this book, I/O designations are referred to in the following way:

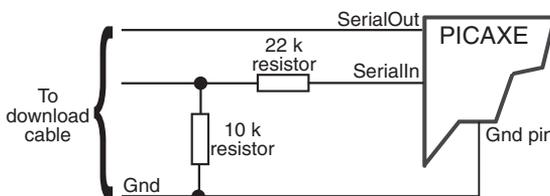


Figure 38-2 The minimal parts setup for all the PICAXE controllers consists of the microcontroller itself, plus two resistors used for downloading programs. The resistors must remain even when the download cable is removed.

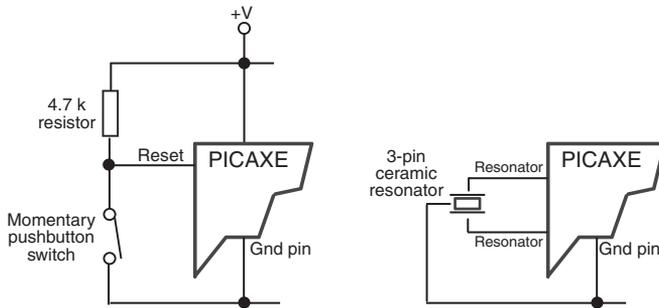


Figure 38-3 Additional components required on PICAXE chips that use a reset pin, and that require the use of an external resonator. The minimum operating speed is 4 MHz, but many PICAXE microcontrollers may be operated at between 8 and 64 MHz, depending on the chip.

- The physical pin numbers of the chips are largely ignored, unless otherwise specified. To indicate I'm talking about an actual pin on the chip, I'll refer to it as a *physical pin* or a *leg*. Note that the PICAXE documentation uses the term *leg*.
- The generic name for an I/O line is a *pin*, and they all have numbers. The way I/O pins are named differs depending on the version of the chip. The 08M uses the nomenclature *Pin1*, *Pin2*, and so on. (This is the more or less the same in the PICAXE documentation, except they use spaces between "Pin" and the number.) The 18M2 uses the nomenclature *B.1*, *B.2*, etc.
- The function of an I/O line is referred to by the same name as the PICAXE documentation, except I remove the space between the name and any number that follows—for the 08M there's *Out1*, *ADC2*, or *In3*. For the 18M2 I use the *port.number* nomenclature preferred for the M2 series: *B.1*, *C.2*, and so on.

Note that many pins can perform multiple functions, and these functions are set in the microcontroller hardware. For example, on the 18M2 physical pin 18, which is designated *C.1*, can serve as a digital input or output, an analog-to-digital conversion input, or as a capacitance touch sensor input.



Output pins on the PICAXE can supply up to 20 milliamps (mA) of current, but the whole chip is limited to under 100 mA or so combined. If you're working on a project using lots of light-emitting diodes, consider using a buffer driver IC, such as the ULN2003, between the PICAXE and the LEDs. See also Chapter 40, "Interfacing Hardware with Your Microcontroller or Computer," for other approaches.

Finally, the PICAXE chips operate out of the box at 4 MHz, but all of the currently offered controllers can be set to operate at a minimum of 8 MHz. The X1 parts can go to 20 MHz, and the more advanced M2 and X2 parts can run at 32 MHz and 64 MHz, respectively. Specifics on chip speed are provided in the PICAXE manual.

INSIDE THE 08M

Let's begin by taking a closer look at the 08M. Refer to Figure 38-1 for a pictorial layout of the pins.

- Three of its eight pins are dedicated for specific use—power, ground, and SerialIn—leaving the remaining five for input and output.

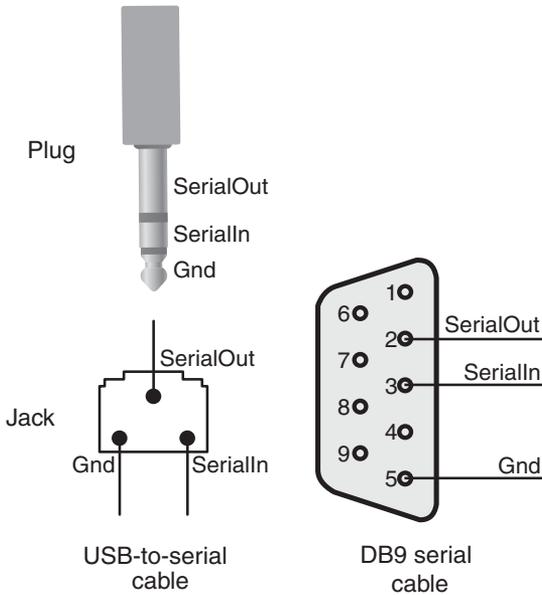


Figure 38-4 Download cable connections, for both the USB-to-serial cable and the standard nine-pin serial port PC connection. These match the connections in Figure 38-2.

- Pin3 (physical pin 4), labeled Infracin / In3, is an input only.
- Pin0 (physical pin 7) works as an output only. In addition to being a standard digital output (Out0), it is also used for special functions: SerialOut and Infraout. Both functions are detailed later in the chapter.
- The remaining pins can serve as either inputs or outputs. Their respective functions are listed.

The basic programming connection diagram for the PICAXE (any chip) is shown in Figure 38-4, for both the USB-to-serial and DB-9 (9 pin) serial ports. Note the wire from SerialOut, the link back to your PC’s serial port. During programming, this I/O line is occupied, but it can be used as another output if you disconnect the download cable.

The 08M has limited memory available for programs. It contains 256 bytes of EEPROM both to store your downloaded programs and for use as nonvolatile memory—the data stay even when you unplug the chip from its power. Depending on the programming commands you use, there’s enough room for about 40 lines of code.

INSIDE THE 18M2

The 18M2 is among a newer generation of PICAXE chips that support numerous new features. As a result, it uses a different nomenclature to denote its various I/O pins. More of the pins on the M2 can serve multiple purposes: input, output, and more. Each I/O is described using a letter, which indicates the *port*, and a number, which indicates the specific pin of that port. For example, *B.2* is port B, pin 2.

Refer to Figure 38-1 for a pictorial layout of the 18M2 pins. As with the 08M, the 18M2 has connections for power, ground, SerialIn, and SerialOut. All but a few of the I/O pins can be used as both inputs and outputs. This is set in software.

The 18M2 has about 2K bytes of space available for holding programs you download to it. Programs are stored in rewritable flash memory. Depending on the programming commands you use, there's enough room for about 600 lines of code. There's also an additional 256 bytes of EEPROM for nonvolatile data storage, handy for data-logging and other tasks.

Programming the PICAXE

All PICAXE controllers may be programmed using editor software provided for free by Revolution Education. Check their Web site at www.picaxe.co.uk for details. Versions of the program editor are available for Microsoft Windows (Windows 98 and later), Macintosh OS X, and Linux.

If you use a USB cable for programming, you will also need an appropriate USB driver for your operating system. These, too, are provided free of charge. Note that the USB or serial port connection does not provide power for the device.

The program editor, shown in Figure 38-5, supports the PICAXE BASIC language. Here you can write, save, and print your programs. PICAXE programs are stored as standard ASCII files, so they can be opened by any text editor. The default file extension is .bas.

The program editor has some nifty features you'll want to try out:

- Environment settings are made using the Options button, where (among other things) you specify the particular PICAXE microcontroller you're using and the serial port on your computer that's connected to the chip.
- The Flowchart tool lets you create programs visually. Clicking on the Flowchart button opens a separate editing window, where you build programs using graphical objects.
- Mistakes in your program are flagged using the Syntax checker. Errors are flagged by line, where you can make necessary corrections.
- Numerous wizards help in constructing several kinds of programming code. Examples include a tune wizard, used to build a progression of musical notes, a datalogger wizard for use in storing information collected over long periods of time, and others.
- The built-in simulator will run through your code and show you how it'll function inside the PICAXE. You see how memory is used and what effect your code has on the I/O pins of the controller.

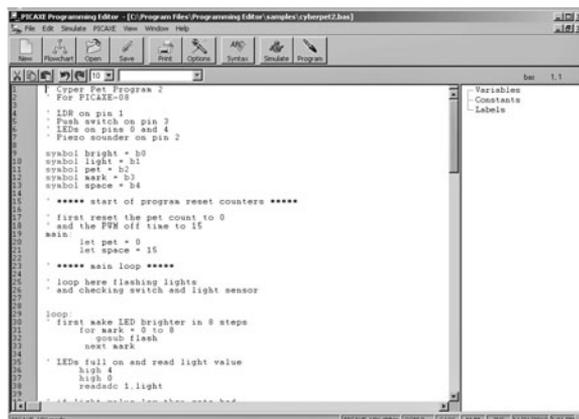


Figure 38-5 The PICAXE programming editor. From the editor you can write, edit, simulate, compile, and download your PICAXE programs.

Before you can send your programs to the PICAXE, they must be compiled. This is done in the editor by clicking on the Program button. Syntax errors are flagged, and if any are found, compiling stops. When you have successfully compiled the program, it is automatically downloaded to the PICAXE.

Core Language Syntax

The PICAXE uses its own unique flavor of the BASIC language, referred to as PICAXE BASIC. Like most any other programming language, it is composed of a series of *commands* used together in a specific, orderly way. The way the commands are assembled is called a *syntax*, and it's important that you use the correct syntax to form your PICAXE programs, or else they will not work.

COMMENTS

You can include remarks for yourself and others who read your code. The remarks are ignored when the program is compiled, so it doesn't matter how many you include. Commands are marked with the ' (apostrophe) character, as in:

```
' This is a comment  
b0 = 100 ' This is also a comment
```

VARIABLE AND PIN/PORT DEFINITIONS

PICAXE uses variables to store information during program execution. Most of the PICAXE chips provide several ways to store variables, chief among them a set of 14 (or more, depending on the chip) general-purpose variables, each of which can hold a byte of data.

Though the general-purpose variables are each byte sized, the 8 bits that make up each byte can be individually manipulated, allowing you to store simple on/off information much more efficiently. The PICAXE documentation provides details on how to do this for the various families of chips.

You can manipulate I/O pins as a group or individually. I'll defer to the PICAXE documentation for how to control groups of pins together (especially as it varies from chip to chip), but to do it for an individual I/O pin you specify the action you want, and the pin designation. For example, to make Out4 on the 08M HIGH (turn it on), you'd use

```
high 4
```

and for the 18M2, you'd use

```
high B.4
```

USING SYMBOLS

Another form of variable is the *symbol*. A symbol isn't a true variable—it's not used as temporary data space by the microcontroller; rather, it's a convenience during programming. It lets you refer to the various parts of the PICAXE architecture using your own terms, making your programs easier to read and understand. When the program is compiled, the PICAXE editor exchanges each instance of your term with the actual value.

To use a symbol, you first declare it at the top of your program. The syntax is simple

```
symbol yourname = PICAXE_name
```

where *yourname* is the term you want to refer to something by, and *PICAXE_name* is the actual name for a port or pin. Example:

```
symbol LED = 4
```

makes LED the same as Out4. You can then use the term LED in your program every time you want to do something to Out4. To make it HIGH, you'd use

```
high LED
```

That's the same as

```
high 4
```

PROGRAM CONSTRUCTION

The PICAXE doesn't have a lot of prerequisites for its programs. A program can be just one line long, though most use the following construction

```
main:
' . . . programming commands here
goto main
```

Main: is a *label*, a way for you to reference different parts of your program. Labels are formed using a single word, followed by a colon.

The preceding construction creates an endless loop, which is quite typical in robotics. The *goto main* command tells the PICAXE to pick up where it is and go to the identified label. Your programming commands, which are contained between the loops, are repeated over and over again.

It's also common to devise special *subroutines* in your program that do certain tasks, sometimes just once when the program first starts, or at specific times depending on the outcome of the computations made in the *main:* loop. For instance:

```
gosub init

main:
' . . . programming commands here
goto main

init:
' . . . do something first here
return
```

This code first branches off to a subroutine named *init*. Even though this subroutine appears at the bottom of the program, the *gosub* (which means goto subroutine) tells the program to first perform the steps in *init:*. Note the *return* command at the end of the *init:* subroutine. This informs the program that the subroutine is finished and tells it to go back to the line following the *gosub* command.

As before, the main body of the program is within the *main:* loop.

FLOW CONTROL

Flow control commands tell your program what to do next. You've already seen two forms of flow control: the *goto* and *gosub* commands. These are similar commands in that they both unconditionally branch execution to another part of the program.

Another type of flow control command provides conditional branching. It's the *if* command, which is used in conditional expressions that execute one part of the program if condition A exists, and another part of the program if condition B exists.

Using the *if* Command

The *if* command, which is always used in conjunction with the *then* keyword, conditionally branches execution depending on the outcome of an expression. The basic syntax is:

```
if Expression then Label
```

Expression is the condition that must resolve to a True or False statement, and *Label* is an identified label elsewhere in the program that the execution is to jump to. An example of a typical *Expression* is checking the value of an input pin against an expected value:

```
if Pin4 = 1 then flash
```

If the value of *Pin4* is equal to 1 (the expression is True), then the program is expected to jump to the *flash* label. As detailed in the previous section, this label is identified by the label name, followed by a colon, like this:

```
flash:
' . . . rest of the code goes here
```

The *if* expression can use a number of logical operators:

=	equal to
<>	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

PICAXE BASIC supports numerous variations of the *if* command. For example, you can insert the code to execute immediately after the *if* command using the format

```
if expression then
' . . . code if expression is true
endif
```

And by using the *else* keyword you can provide special instructions on what to do if the expression fails (is false):

```
if expression then
' . . . code if expression is true
```



```

else
' . . . code if expression is false
endif

```

Consult the PICAXE manual for these and other variations.

More on Using the *goto* and *gosub* Commands

Let's take a closer look at the *goto* and *gosub* commands. As we've seen, *goto* is most often used to create endless loops:

```

high 0
main:
  pause 1000
  toggle 0
  goto main

```

In this program, Out0 is set to 1 (HIGH). The program then pauses for 1000 milliseconds (1 second) and then toggles Out0 to its opposite state. The *goto* command makes the program jump back to the *main* label. Assuming the Out0 pin is connected to an LED, it would flash on and off.

Gosub is similar to *goto*, except that when the code at the label is done, the program returns to the command immediately after *gosub*. Here's an example:

```

high 1
low 2
main:
  gosub flash
  ' . . . some other code here
  goto main

flash:
  toggle 1
  toggle 2
  pause 500
  return

```

The program begins by setting I/O pin 1 to HIGH and I/O pin 2 to LOW. It then “calls” the *flash* routine, using the *gosub* command. The code in the *flash* routine toggles I/O pins 1 and 2 from their previous state, waits half a second, and then returns with the *return* flow control command.

Using the *for* Command

Another type of flow control command is *for*, and it is used with the *to* and *next* keywords. Used together, they control a counter that repeats your code a set number of times. The syntax for the *for* command is:

```

for Variable = StartValue to EndValue
' . . . more command
next

```

Variable is a variable that is used to contain the current count of the *for* loop. *StartValue* is the initial value to begin with; *EndValue* marks the value to count to. The loop is broken (and

the rest of the program can then run) when *Variable* exceeds *EndValue*. For example, if you use the following

```
for b0 = 1 to 10
```

the loop starts with 1 in variable *b0* and counts to 10. The *for* loop is repeated 10 times. You don't have to start with 1, and you can use an optional *step* keyword to tell the *for* loop that you want to count by 2s, 3s, or some other value:

```
for b0 = 5 to 7
```

counts from 5 to 7.

For loops are used to execute whatever programming lies between the *for* and *next* commands. Here's a simple example:

```
high 1
for b0 = 1 to 10      'repeat total of 10 times
  toggle 1
  pause 500
next
```

PICAXE Functions for Robotics

Much of the power of the PICAXE comes in its set of built-in commands that reduce the complexity of programming. Most are designed to control some activity of the chip, for example, to produce sound through one of the output pins, or to produce timed signals to control one or more R/C servo motors. I'll briefly review the special functions most useful for robotics, but you'll want to look over the PICAXE manual for more.

- *button*. The *button* command momentarily checks the value of an input and then branches to another part of the program if the button is LOW (0) or HIGH (1). The *button* command lets you choose which input pin to examine, the "target state" you are looking for (either 0 or 1), and the delay and rate parameters that can be used for such things as switch debouncing.
- *debug*. The PICAXE program editor has a built-in terminal that displays the result of bytes sent from the chip back to the PC. Of course, you need to have the serial/USB cable connected to receive anything. The *debug* command displays information that may be useful during testing.
- *infrain/infrain2/irin* and *infraout/irout*. These commands are used to receive and send (respectively) infrared remote control signals that use the Sony SIRC protocol. *Infrain*, *infrain2*, and *irin* are used in conjunction with an infrared receiver/modulator (see Chapter 41, "Remote Control Systems"), which detects the signals from the remote and converts them to on/off pulses. Conversely, *infraout* is used to produce Sony-compatible IR signals.
- *servo*. The *servo* command (and its lieutenant, *servopos*) allows you to control the operation of an R/C servo motor. With the command you specify the output port to use, along with the position of the motor, in tenths of a microsecond. Valid values are from 75 to 225 (note: you must use whole numbers only).
- *sound*. The *sound* command is used to generate tones primarily intended for audio reproduction. You can set the I/O pin, note frequency (in Hertz), and duration. You can string a

series of notes (each with a separate duration) to produce simple monophonic music or sound effects.

- *pause*. The *pause* command is used to delay execution by a set amount of time. To use *pause* you specify the number of milliseconds (thousandths of a second) to wait. For example, *pause 500* pauses for 500 milliseconds, or half a second.
- *pulsin*. The *pulsin* command measures the width of a single pulse with a resolution of 10 microseconds (10 μ s), when using the PICAXE at the default 4-MHz speed. You can specify which I/O pin to use, whether you're looking for a 0-to-1 or 1-to-0 transition, as well as the variable in which you want to store the result. *Pulsin* is handy for measuring time delays in circuits, such as the return "ping" of an ultrasonic sonar.
- *pulsout*. *Pulsout* is the inverse of *pulsin*: with *pulsout* you can create a finely measured pulse with a duration of between 10 μ s and 65,535 μ s (the resolution is increased when running the PICAXE at speeds over 4 MHz). The *pulsout* command is ideal when you need to provide accurate waveforms.
- *readadc*. The *readadc* command reads a linear voltage at any ADC (analog-to-digital converter) input pin. The voltage is then converted to an 8-bit (0 to 255) digital equivalent. A similar command, *readadc10*, provides a 10-bit (0 to 1023) resolution.
- *serin* and *serout*. The *serin* and *serout* commands are used to send and receive asynchronous serial communications using any output pin. They represent one method for communicating with other devices. One application of *serout* is to interface a liquid-crystal display (LCD) to the PICAXE.
- *shiftin* and *shiftout*. These commands are used in two- or three-wire synchronous serial communications. These commands are useful when communicating with a variety of external hardware, including serial-to-parallel shift registers and serial analog-to-digital converters.
- *parallel tasking*. The 18M2 and other M2 series chips are capable of running up to four separate programs—called *tasks*—at once. Internally, the microcontroller divides its attention among the tasks, spending small slices of time with each one. Parallel tasking is especially handy in robotics where you want to monitor several different things at the same time, such as a set of bumper switches plus infrared and ultrasonic sensors.

Example: Controlling an RC Servo with the PICAXE

The PICAXE contains several commands designed directly for use on R/C servo motors. The commands are *servo*, which sets up the PICAXE to operate a servo, and *servopos*, which moves the servo depending on the position value you give it.

Figure 38-6 shows a typical connection diagram between PICAXE and servo, with an optional 330 Ω resistor in series on the signal line. This resistor is recommended in the PICAXE documentation and limits the current draw from the servo, should it go haywire. Note the separate battery supply for the servo and the connected ground wires shared between the PICAXE and the servo. Both are required for proper operation.

Code is straightforward:

```
init:
    servo 4,150           ' Set up servo
```

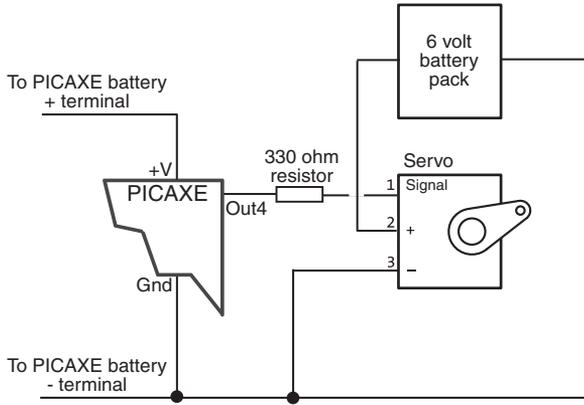


Figure 38-6 Connection diagram for attaching a standard R/C servo to a PICAXE. You must power the servo from its own 6-volt battery. Be sure to connect together the ground (Gnd) leads of the PICAXE battery and the servo battery.

```

main:
  servopos 4,100           ' Move servo to one end
  pause 2500              ' Pause 2.5 seconds
  servopos 4,200         ' Move servo to other end
  pause 2500              ' Pause again
  goto main              ' Repeat
    
```

The servo is connected to the Out4 pin of the PICAXE 08M. The servo is initialized with a value of 150, which equates to 1500 microseconds, or 1.5 milliseconds. The main body of the program then repeats a loop where the servo is positioned from one end to the end, pausing between each transit for 2-1/2 seconds.

Example: Reading Buttons and Controlling Outputs

A common robotics application is reading an input, such as a switch, and controlling an output, such as an LED or motor. The following example shows a simple method of reading the

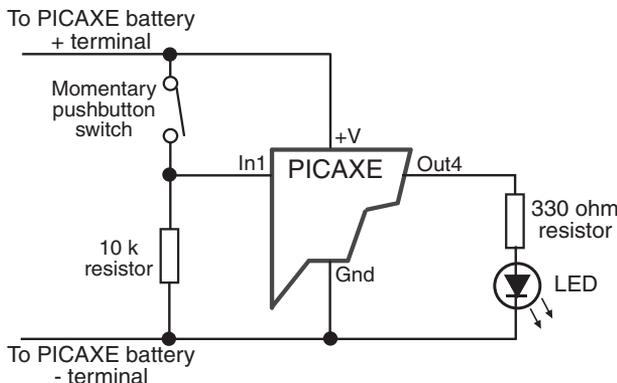


Figure 38-7 Connection diagram for demonstrating a push-button input (with debounce) and LED output. The push-button switch is a normally open momentary type.

state of a switch, and then displaying the result on an LED. The switch is connected to pin In1; the LED to pin Out4. Figure 38-7 shows a connection diagram.

The code notes when the switch closes and produces a 1 (HIGH) on pin In1. The LED connected to Out4 lights up, and the program pauses for 1 second before continuing.

```
main:
  if pin1 = 1 then flash          ' 1 = switch closed
  low 4                          ' Turn off Out4
  pause 10
  goto main                      ' Repeat

flash:
  high 4                         ' Turn off Out4
  pause 1000                     ' Wait 1 second
  goto main
```

Using the BASIC Stamp

Since its inception, the Parallax BASIC Stamp has provided the onboard brains for countless robot projects. This thumbprint-sized microcontroller uses BASIC-language commands for instructions and is popular among robot enthusiasts, electronics and computer science instructors, and even design engineers looking for an inexpensive alternative to microprocessor-based systems. The original BASIC Stamp has been greatly enhanced, and new models sport faster speeds, more memory capacity, easier software programming, and additional data lines for interfacing with motors, switches, and other robot parts.

In this chapter, you'll learn the fundamentals of the BASIC Stamp and how to use it in your robot building. You will also want to read Chapters 37 and 38, which provide full coverage of the very popular Arduino and PICAXE microcontrollers. Like the BASIC Stamp, the PICAXE also has BASIC embedded in it, while the Arduino uses a variation of the C programming language.



This chapter concentrates on the BASIC Stamp 2. There are other versions of the BASIC Stamp, including the older BASIC Stamp 1, BS-SX, and others. Features and functionality vary by model. Be sure to check the Parallax Web site at www.parallax.com for more information.

Inside the BASIC Stamp

The BASIC Stamp starts life as an off-the-shelf PIC from Microchip Technologies. “PIC” stands for “programmable integrated circuit,” though other definitions are also commonly cited, including “peripheral interface controller” and “programmable interface controller.” Embedded in this PIC is a proprietary BASIC-like language interpreter called *PBasic*.

FROM PC TO BASIC STAMP

The chip stores commands downloaded from a PC or other development environment. When you run the program, the language interpreter built inside the Stamp converts the instructions into code the chip can use. Common instructions involve such things as assigning a given data line as an input or output or toggling an output line from high to low in typical computer-control fashion.

The net result is that the BASIC Stamp acts like a programmable electronic circuit, with the added benefit of intelligent control—but *without* the complexity and circuitry overhead of a dedicated microprocessor. Instead of building a logic circuit out of numerous inverters, AND gates, flip-flops, and other hardware, you can use just the BASIC Stamp module to provide the same functionality and do everything in software. (To be truthful, the Stamp often requires that at least some external components interface with real-world devices.) Nor do you need to construct a microprocessor-based board for your robot followed by the contortions of programming the thing in some arcane machine language.

Because the Stamp accepts input from the outside world, you can write programs that interact with that input. For instance, it's a slam dunk to activate an output line—say, one connected to a motor—when some other input (like a switch) changes logic states. You could use this scheme, for instance, to program your robot to reverse its motors if a bumper switch is activated. Since this is done under program control and not as hardwired circuitry, it's easier to change and enhance your robot as you experiment with it.

STAMP MEMORY

The microcontroller of the BASIC Stamp uses two kinds of memory: PROM (programmable read-only memory) and RAM. The PROM memory is used to store the PBasic interpreter; the RAM is used to store data while a PBasic program is running. Memory for the programs that you download from your computer is housed in a separate chip (but is still part of the BASIC Stamp itself; see the description of the BS2 module in the next section). This memory is EEPROM, for “electrically erasable programmable read-only memory” (the “read-only” part is a misnomer, since it can be written to as well).

In operation, your PBasic program is written on a PC, then downloaded—via a serial connection—to the BASIC Stamp, where it is stored in EEPROM. The program in the EEPROM is in the form of “tokens”; special instructions that are read, one at a time, by the PBasic interpreter stored in the BASIC Stamp's PROM memory. During program execution, temporary data are kept in RAM. Note that the EEPROM memory of the BASIC Stamp is nonvolatile—remove the power, and its contents remain. The same is not true of the RAM. Remove the power from the BASIC Stamp, and any data stored in the RAM are gone. Also note that the PBasic interpreter, which is stored in the PROM memory of the microcontroller, is not replaceable.

As a modern microcontroller, the BASIC Stamp 2 is a little tight when it comes to available memory space. The chip sports only 2K of EEPROM and just 32 bytes of RAM. Of those 32 bytes, 6 are reserved for storing the settings information of the input/output pins of the BASIC Stamp, leaving only 26 bytes for data. For many robotics applications, the 2K EEPROM (program storage) and 26-byte RAM (for data storage) are sufficient. However, for complex designs, you may need to use a second BASIC Stamp or select a microcontroller—such as the BASIC Stamp 2-SX or the Parallax Propeller—that provides more memory.

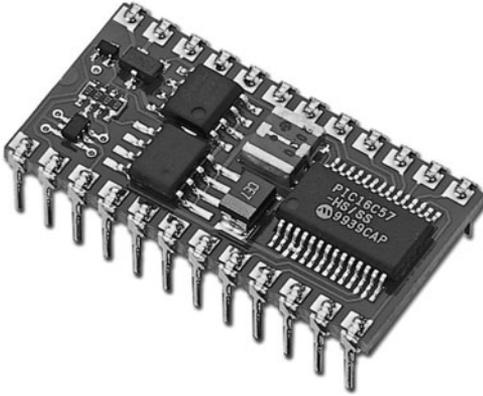


Figure 39-1 The BASIC Stamp 2 is a self-contained microcontroller, complete with the controller chip itself, plus voltage regulator and other parts. It comes on a small circuit board that has the same size as a 24-pin-wide integrated circuit. (Photo Courtesy Parallax Inc.)

Stamp Alone or Developer's Kit

The BASIC Stamp is available directly from its manufacturer or from a variety of dealers the world over. The prices from most sources are about the same. In addition to the BS1, BS2, and BS2-SX variations mentioned earlier, you'll find that the BASIC Stamp is available in several different premade kits as well as a stand-alone product.

- *BS2 Module.* The BASIC Stamp module (see Figure 39-1) contains the actual microcontroller chip as well as other support circuitry. All are mounted on a small printed circuit board that is the same general shape as a 24-pin IC. In fact, the BS2 is designed to plug into a 24-pin IC socket. The BS2 module contains the microcontroller that holds the PBasic interpreter, a 5-volt regulator, a resonator (required for the microcontroller), and a serial EEPROM chip.
- *BASIC Stamp Board of Education.* Typically sold without a BS2 module, the Board of Education, known also as BOE, offers you a convenient way to experiment with the controller. It contains connectors for four R/C servo motors, and a mini-size solderless breadboard. The BOE provides connectors for quick hookup to a PC to program the Stamp chip. It's available in serial and USB versions.
- *BS2 Discovery Kit.* The discovery kit is ideal for those just starting out. It includes a BS2 module, a Board of Education board, a programming cable, a power adapter, software on CD-ROM, and an assortment of electronics components used in a series of tutorial lessons.
- *Boe-Bot.* The Boe-Bot is a small metal-fabricated mobile robot kit built around the BS2 chip and includes the BASIC Stamp Board of Education. If you already have a BS2 and BOE, you can purchase just the Bot chassis, which comes with motors, wheels, and all hardware.

Physical Layout of the BS2

The BASIC Stamp 2 is a 24-pin device; 16 of the pins are input/output (I/O) lines that you can use to connect with your robot. For example, you can use I/O pins to operate a

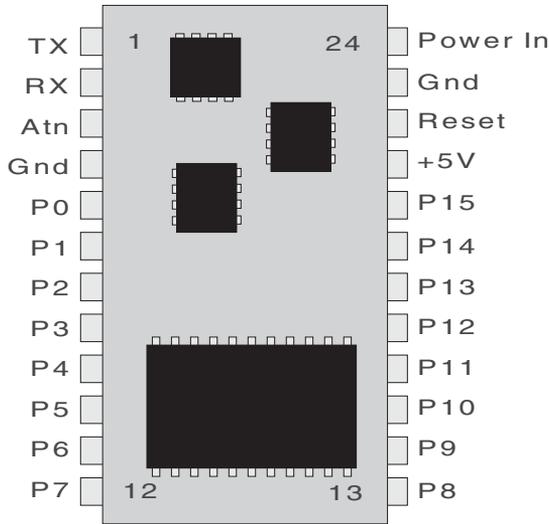


Figure 39-2 Pinout diagram of the BS2 chip. As usual, pin numbering starts at the upper left and goes counterclockwise.

radio-controlled (R/C) servo. Or you can use a stepper motor or a regular DC motor, when you use them with the appropriate power interface circuitry. As outputs, each pin can:

- Source 20 mA of current (that is, supply current to the load).
- Sink about 25 mA.

However, the entire BS2 should not source or sink more than about 80 to 100 mA for all pins in combination. You can readily operate a series of LEDs, without needing external buffer circuitry to increase the power-handling capability.

Or you can connect the BS2 to an ultrasonic ranging module (see Chapter 43, “Proximity and Distance Sensing”), various bumper switches, and other sensors. The *direction* of each I/O pin can be individually set—some pins can be used for outputs and others for inputs. You can dynamically configure the direction of I/O pins during program execution. This allows you to use one pin as both an input and an output.

Figure 39-2 shows the pin layout of the BS2. In addition to power and communications pins, the BS2 offers 16 I/O lines, customarily identified as P0 through P15 (sometimes the P is omitted). Through PBasic commands, you can control all 8 bits of the port together or each pin individually.

Hooking Up: Connecting the BASIC Stamp to a PC

The BASIC Stamp was engineered to make it easy to connect to a personal computer, using only a serial port or a USB port. These days many computers don’t have a serial port; they use USB instead. If your computer lacks a serial port, you’re best off getting a USB-to-serial

adapter, or opt for a BASIC Stamp experimenter's board that already has a USB connector on it. You shouldn't rely on power provided by the USB connection to operate your BASIC Stamp. Provide the BASIC Stamp with its own power source. If using the BASIC Stamp by itself, you can connect a 9-volt battery to its power terminals; the built-in voltage regulator will provide the proper volts for the chip.

Once connected to the PC (USB or serial), you need only install the software from the CD or downloaded from the Parallax site, and you're set to go.

Understanding and Using PBasic

As you've read earlier in this chapter, at the heart of the BASIC Stamp is PBasic, which is the language used to program the BASIC Stamp. PBasic has undergone many changes during the life of the BASIC Stamp products, and the syntax and the commands are different between PBasic for the BASIC Stamp 1 (known as PBasic1) and PBasic for the BS2 (PBasic 2.0 and 2.5).

Unless otherwise noted, what follows is PBasic 2.0 for the BASIC Stamp 2. Though there is a later version of the language, 2.0 programs still work in all current BS2 chips. Examples of programming code in PBasic 2.0 are more common on the Web; as this book doesn't live in a vacuum, I decided to stick with version 2.0-style code, to avoid causing unnecessary confusion.

Version 2.5 adds some additional (and worthwhile) software statements to make your programs more streamlined, and these are noted separately later on in the chapter. You're free to adopt whatever version you wish, and mix and match them. Just be sure that, if you wish to use PBasic 2.5 features, you have the correct version of the software from the Parallax Web site.

PBasic programs for the BASIC Stamp are developed in the BASIC Stamp Editor, an application that comes with the Starter Kit—it's also available for free download at the Parallax Web site. The Editor lets you write, edit, save, and open BASIC Stamp programs. It also allows you to compile and download your finished programs to a BASIC Stamp. The download step requires that your BASIC Stamp be connected to a carrier board or other circuit board attached to a download cable. The download cable is connected to your PC via a serial or USB serial port. Figure 39-3 shows the BASIC Stamp Editor.

Like any language, PBasic is composed of a series of *statements* that are strung together in a logical syntax. A statement forms an instruction that the BS2 is to carry out. For example, one statement may tell the chip to fetch a value on one of its I/O pins, while another may tell it to wait a certain period of time. The majority of PBasic statements can be categorized into three broadly defined groups: variable and pin or port definitions, flow control, and special function. We'll cover each of these next.

VARIABLE AND PIN/PORT DEFINITIONS

As with any programming language, PBasic uses variables to store bits and pieces of information during program execution. Variables can be of several different sizes; you should always strive to choose the smallest variable size that will accommodate the data you wish to store. In

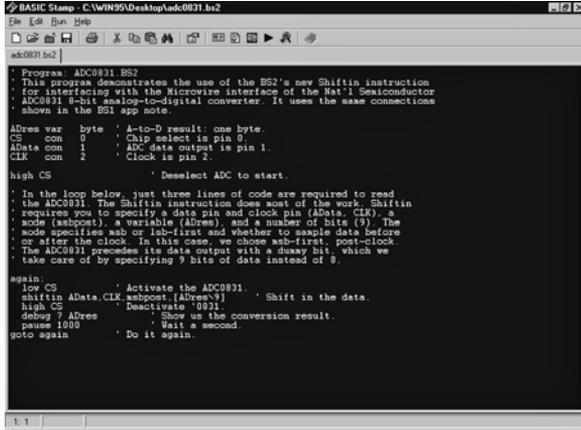


Figure 39-3 The BASIC Stamp 2 integrated programming environment, which runs on your personal computer. Here, you write your programs and download them to the BS2 chip via a serial or USB connection.

this way you will conserve precious RAM space (remember, you have only 26 bytes of RAM to work with!).

PBasic provides the following four variable types:

- Bit—1 bit (one-eighth of a byte)
- Nibble—4 bits (4 bytes)
- Byte—1 byte (8 bits)
- Word—2 bytes (16 bits)

Variables must be declared in a PBasic program before they can be used. This is done using the *var* statement, like so:

VarName	var	VarType
---------	-----	---------

where *VarName* is the name (or *symbol*) of the variable, and *VarType* is one of the variable types just listed. Here's an example:

```

Red var bit
Blue var byte
    
```

Red is a bit, and *Blue* is a byte. Note that capitalization does not matter in a PBasic program.

Once declared, variables can be used throughout a program. The most rudimentary use for variables is with the = (equals) assignment operator, as in

```

Red = 1
Blue = 12
    
```

Variables can also be assigned as the result of a math expression (2 + 2) or as the value of an input pin. For example, suppose an input pin is connected to a mechanical switch. Ordinarily, the switch is open, and the value at the pin is LOW (0). Suppose a variable, called *Switch*, stores the current value of the pin. The *Switch* variable would contain 0 as long as the switch is opened. If the switch is closed, the *Switch* variable then stores 1 (or logical HIGH). More about I/O pins in a bit.

Variables store values that are expected to change as the program runs. PBasic also supports constants, which are used as a convenience for you. Constants are declared much as variables are, using the *con* statement:

```
MyConstant      con      5
```

MyConstant is the name of the constant, and its value is 5.

The BASIC Stamp treats its 16 I/O pins like additional memory. The instantaneous value of an I/O pin functions exactly like a 1-bit variable: the value is either 0 or 1. If the I/O pin is an input, then the value of that input will be 0 or 1, depending on the condition of the circuit on the outside of the BASIC Stamp. The mechanical switch is a good example of this: depending on whether the switch is opened or closed, the value of the input pin is 0 (open) or 1 (closed).

When I/O pins are used as outputs, their logical state is changed using the *high*, *low*, and *toggle* statements. In each case, the number of the pin (0 through 15) is given to tell the BASIC Stamp which I/O you want to change:

- *High* brings the I/O pin HIGH (1).
- *Low* brings the I/O pin LOW (0).
- *Toggle* changes the state of the I/O pin from 0 to 1, or vice versa, depending on its previous value.

Here's an example (using the traditional apostrophe character for inline comments):

```
high 1          ' put I/O pin 1 high
low 12         ' put I/O pin 12 low
toggle 5       ' change I/O pin 5 opposite
               ' to its previous value
```



As with nearly all microcontrollers, *I/O pins* are distinct from the *physical pins* on the chip. When it says *low 12*, this means the pin labeled P12 in the BASIC Stamp documentation and on carrier boards such as the Board of Education. It is *not* physical pin #12, which is actually I/O pin 7 (P7).

There are many ways to determine the current value of an I/O pin that is used as an input. Most are used with special functions, which are outlined later in this chapter. You can also directly reference the value of an input by using the *Inx* statement, where *x* is a number from 0 to 15. For instance, to read the value of pin 3 and put it into a variable, you'd use the following:

```
SomeVar  Var  Byte
Main:
  SomeVar = In3
  . . . rest of program
```

FLOW CONTROL

Flow control statements tell your program what to do next. A commonly used flow control statement is *if*, which is used in conditional expressions that execute one part of the program if condition A exists and another part of the program if condition B exists. Two other flow control statements are *goto* and *gosub*, which are used to unilaterally jump from one part of

a program, and the *for* statement, which is used to repeat a block of code a specific number of times. Let's look first at the *if* statement.

Using the *if* Statement

The *if* statement, which is always used in conjunction with the *then* statement, conditionally branches execution depending on the outcome of an expression. The syntax is as follows:

```
if Expression then Label
```

Expression is the condition that must resolve to a True or False statement, and *Label* is an identified label elsewhere in the program that the execution is to jump to. An example of a typical *Expression* is checking the value of a variable or input pin against an expected value:

```
if MyVar=1 then Flash
```

If the content of *MyVar* is equal to 1 (the expression is True), then the program is expected to jump to the *Flash* label. This label is identified by the label name, followed by a colon, as in:

```
Flash:
. . . rest of the code goes here
```

The *if* expression can use a number of logical operators:

=	equal to
<>	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

For PBasic 2.0, the *if* statement is a little funky compared to other modern programming languages, in that the result of the expression branches execution to a label. Note that there is no explicit *else* or *endif* keyword in PBasic 2.0, that is, an action to be taken if the expression is False. As cited in the BASIC Stamp manual, you must write *if* statements that have a True and False component along these lines:

```
if aNumber < 100 then isLess
debug "greater than or equal to 100"
stop

isLess:
debug "less than 100"
stop
```

FYI

If you prefer the *if . . . else . . . endif* structure, you can use the enhanced flow control statements found in PBasic 2.5 and described more fully later in this chapter.

Notice how this bit of programming works: should *aNumber* be less than 100, then the program jumps to the *isLess* label, and the *debug* statement (which prints text in the debug

window of the BASIC Stamp programming environment on your PC) prints “less than 100.” However, if *aNumber* is 100 or higher, the jump to *isLess* is ignored, and the program simply executes the next line, which is yet another debug statement (“greater than or equal to 100”). Note the introduction of another flow control statement: *stop*. The *stop* statement stops program execution.

Using the *goto* and *gosub* Statements

The *goto* and *gosub* statements are used with labels to divert execution to another part of the program. *Goto* is most often used to create endless loops, as shown here:

```
high 1
RepeatCode:
  pause 100
  toggle 1
  goto RepeatCode
```

In this program, I/O pin 1 is set to 1 (HIGH). The program then pauses for 100 milliseconds (one-tenth of a second) and then toggles I/O pin 1 to its opposite state. The *goto* statement makes the program jump back to the *RepeatCode* label. With each trip through the code, I/O pin 1 is toggled HIGH or LOW. If the pin is connected to an LED, for example, it would flash on and off rapidly 10 times each second.

Gosub is similar to *goto*, except that when the code at the label is done, the program returns to the statement immediately after *gosub*. Here’s an example:

```
high 1
low 2
gosub FlashLED
' . . . some other code here
stop

FlashLED:
  toggle 1
  toggle 2
  pause 100
  return
```

The program begins by setting I/O pin 1 to HIGH and I/O pin 2 to LOW. It then “calls” the *FlashLED* routine, using the *gosub* statement. The code in the *FlashLED* routine toggles I/O pins 1 and 2 from their previous state, waits one-tenth of a second (100 milliseconds), and then returns with the *return* flow control statement. Note the *stop* statement used before the *FlashLED* label. It prevents the code from reexecuting the *FlashLED* routine when it is not intended.

Using the *for* Statement

The *for* statement is used with the *to* and *next* statements. All form a controlled counter that is used to repeat the code a set number of times. The syntax for the *for* statement is:

```
for Variable = StartValue to EndValue [more statements] next
```

Variable is a variable that is used to contain the current count of the *for* loop. *StartValue* is the initial value applied to *Variable*. Conversely, *EndValue* marks the maximum value that will

be applied to *Variable*. The loop breaks out—and the rest of the program continues to execute—when the *Variable* exceeds *EndValue*. For example, if you use the following,

```
for VarName = 1 to 10
```

the loop starts with 1 in *VarName* and counts to 10. The *for* loop is repeated 10 times. You don't have to start with 1, and you can use an optional *step* keyword to tell the *for* loop that you want to count by 2s, 3s, or some other value:

```
for VarName = 5 to 7           ' counts from 5 to 7
for VarName = 1 to 100 step 10 ' counts from 1 to 100,
                               ' but steps by 10
```

For loops are used to execute whatever programming lies between the *for* and *next* statements. Here's a simple example:

```
VarName  Var  Byte
high 1
for VarName = 1 to 10
  toggle 1
  pause 500
next
```

This program repeats the *for* loop a total of 10 times. At each iteration through the loop, I/O pin 1 is toggled (HIGH to LOW, and back again).

The BASIC Stamp supports additional flow control statements, all of which are detailed in the BASIC Stamp manual. These include *Branch* and *End*.

SPECIAL FUNCTIONS

The PBasic language supports several dozen special functions that are used to control some activity of the chip, including ones to sound tones through an I/O pin or to wait for a change of state on an input. I'll briefly review here the special functions most useful for robotics. You'll want to study these statements more fully in the BASIC Stamp manual (available for free download from Parallax and also included in the Starter Kit as a printed book).

- *button*. The *button* statement momentarily checks the value of an input and then branches to another part of the program if the button is in a LOW (0) or HIGH (1) state.
- *debug*. The BASIC Stamp Editor has a built-in terminal that displays the result of bytes sent from the BASIC Stamp back to the PC. The *debug* statement “echoes” numbers or text to the screen and is highly useful during testing.
- *freqout*. The *freqout* statement is used to generate tones primarily intended for audio reproduction. You can set the I/O pin, duration, and frequency (in Hertz) using the *freqout* statement.
- *input*. The *input* statement makes the specified I/O pin an input. As an input, the value of the pin can be read in the program.
- *pause*. The *pause* statement is used to delay execution by a set amount of time. To use *pause*, you specify the number of milliseconds (thousandths of a second) to wait.
- *pulsin*. The *pulsin* statement measures the width of a single pulse with a resolution of 2 microseconds (2 μ s). *Pulsin* is handy for measuring time delays in circuits, such as the return “ping” of an ultrasonic sonar.

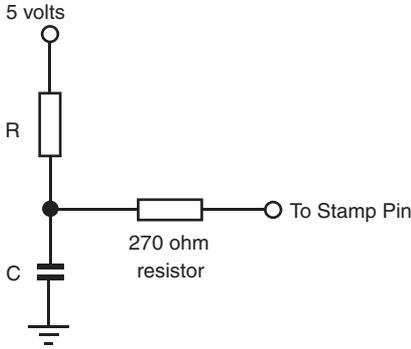


Figure 39-4 Basic RC (resistor-capacitor) setup for testing a voltage at an input pin, using a resistor and a capacitor. The BASIC Stamp manual provides more information on selecting the value of the resistor and capacitor.

- *pulsout*. *Pulsout* is the inverse of *pulsin*: with *pulsout* you can create a finely measured pulse with a duration of between 2 μ s and 131 milliseconds (ms).
- *rctime*. The *rctime* statement measures the time it takes for an RC (resistor/capacitor) network to discharge to an opposite logical state. The *rctime* statement is often used as a simplified analog-to-digital circuit. Figure 39-4 shows a sample circuit.
- *serin* and *serout*. *Serin* and *serout* are used to send and receive asynchronous serial communications. Both commands require that you set the particulars of the serial communications, such as data (baud) rate, and the number of data bits for each received word. One application of *serout* is to interface a liquid-crystal display (LCD) to the BASIC Stamp.
- *shiftin* and *shiftout*. The *serin* and *serout* statements are used in one-wire asynchronous serial communications. The *shiftin* and *shiftout* statements are used in two- or three-wire synchronous serial communications. The main difference is that with *shiftin/shiftout* a separate pin is used for clocking the data between its source and destination.

Interfacing Switches and Other Digital Inputs

You can easily connect switches, either for control or for “bump” or other contact sensors, to the BASIC Stamp using either of the approaches shown in Figure 39-5. The simplest way to detect a switch closure is with the *Inx* statement (*x* is a number from 0 to 15 that denotes a pin). For example:

```
if In3 = 1 then
```

checks if input pin 3 is 1 (HIGH).

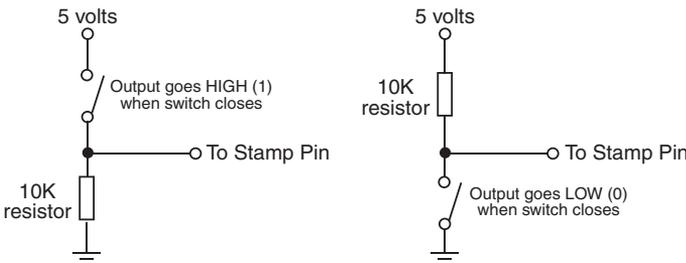


Figure 39-5 When using switch inputs, use a 10 k Ω resistor connected to ground or 5 volts, depending on whether you want the input pin to be LOW or HIGH when the switch is open.

You can also use the *button* statement, described briefly earlier in the chapter, to determine the current value of the switch. The *button* statement also includes a built-in debounce feature, so the BASIC Stamp will ignore the typical “noise” that occurs when a mechanical switch closes. Consult the BASIC Stamp manual for how to use *button*. It’s not very intuitive, but it gets the job done.

Interfacing DC Motors to the BASIC Stamp

The BASIC Stamp is ideal for controlling a DC motor that is connected to an H-bridge circuit (see Chapter 22, “Using DC Motors”). In the typical H-bridge for a single motor, the BASIC Stamp controls the on/off operation of the motor using one pin and the direction using another pin. By using the *high* and *low* statements, you can control the motor easily, turning it on and off and reversing its direction.

The BASIC Stamp code for controlling a DC motor is relatively straightforward: use the *high* or *low* statements and indicate which I/O pins you wish to use. For example, suppose your motor H-bridge is connected to pins 0 and 1, with pin 0 used for on/off control and pin 1 used for direction. Note that when pin 0 is low (0), the motor is off and therefore the setting of pin 1 doesn’t matter.

Pin	LOW	HIGH
0 (on/off)	Motor off	Motor on
1 (direction)	Forward	Reverse

Here is an example:

```
Main:
  low 1           ' set direction to forward
  high 0          ' turn on motor
  pause 2000      ' wait two seconds
  low 0           ' turn off motor
  pause 100       ' wait 1/10 second
  high 1          ' set direction to reverse
  high 0          ' turn on motor
  pause 2000      ' wait two seconds
  low 0           ' turn off motor
  goto Main
```

By using labeled routines and the *gosub* statement, you can define common actions and develop more compact programs:

```
Main:
  gosub motorOnFwd
  gosub waitLong
  gosub motorOff
  pause 1
  gosub motorOnRev
  gosub waitShort
  gosub motorOff
  goto Main
```

```

motorOnFwd:
  low 1
  high 0
  return

motorOnRev:
  high 1
  high 0
  return

motorOff:
  low 0
  return

waitLong:
  pause 5000
  return

waitShort:
  pause 1000
  return

```

Interfacing RC Servo Motors to the BASIC Stamp

Servo motors for radio-controlled (R/C) cars and airplanes can be easily connected to and controlled with a BASIC Stamp. In fact, the code required for operating a servo motor is very simple.

You may connect any I/O pin of the BASIC Stamp directly to the signal input of the servo (see Chapter 23, “Using Servo Motors,” for more information on servo motors, how they work, and their electronic connections). Keep in mind that the BASIC Stamp cannot provide operating power to the servo motor; you must use a separate battery or power supply for it.

Figure 39-6 shows a good approach for connecting a common RC servo to the BASIC Stamp by using a separate battery supply for the servo. Note that the ground connections of the power supplies for both the BASIC Stamp and the servo are connected. If electrical noise

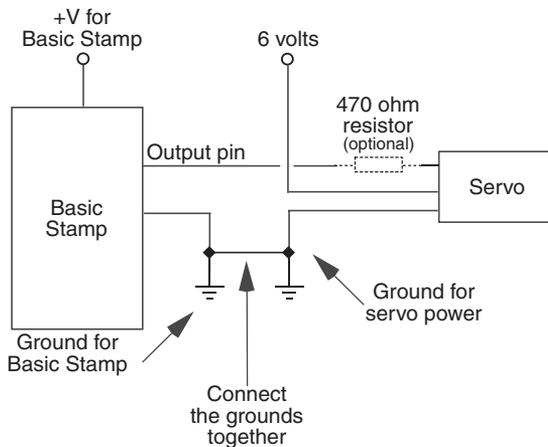


Figure 39-6 Basic connection for attaching a BS2 chip to an RC servo.

poses a problem for you, you can try adding a 1 μ F tantalum capacitor. between the +V and ground connections for the BASIC Stamp.

To operate the servo, use the *pulsout* statement, which sends a pulse of a specific duration to an I/O pin. Servos need to be “refreshed” with a pulse about 50 times each second to maintain their position. By adding a delay (using the *pause* statement) and a loop, your BASIC Stamp can move and maintain any position of the servo. The following examples show the basic program, using pin 0 as the control signal line to the servo. It sets the servo in its approximate center position:

```
low 0
Loop:
  pulsout 0,750
  pause 20
  goto Loop
```

Here’s how the program works: pin 0 is set as an output and set to low, with the *low 0* statement. The repeating loop is defined as the code between the *Loop:* label and the *goto Loop* statement. The *pulsout* statement sends a 1500-microsecond pulse (LOW-HIGH-LOW) to pin 0. The value of 750 is used because *pulsout* has a minimum resolution of 2 μ s; 750 times 2 is 1500 μ s. A pulse of 1500 μ s will position the output shaft of the servo in its approximate center position.

The *pause* statement pauses execution for 20 milliseconds (ms). When the loop is run, it will repeat about 50 times each second (20 ms * 50 = 1000 ms, or 1 second). Note that you can insert additional *pulsout* statements if you need to control other servos. The BASIC Stamp can adequately control seven or eight servos, but in doing so, it won’t have much processing time left over for anything else.

To change the angular position of the servo motor, merely alter the timing of the *pulsout* statement:

```
pulsout 0,1000      ' 2000 usec pulse, approx. 180
                   ' degrees position
pulsout 0,500       ' 1000 usec pulse, approx. 0
                   ' degrees position
```

These values assume a strict 1000- to 2000- μ s operating range for a full 0 to 180° rotation. This is actually *not* typical. You will likely find that your servo will have a full 180° rotation with higher and lower values than the nominal 1000 to 2000 μ s. You can determine this only through experimentation. See Chapter 23, “Using Servo Motors,” for more details on this topic.

Additions in PBasic 2.5

With the introduction of PBasic 2.5 you can apply more structure to your programs, making them easier to write, fix, and read months after you’ve written them. Here are several of the more prominent changes made in PBasic 2.5.

IF-ELSE-ENDIF STRUCTURE

Rather than write *if* statements using *goto*’s and *gosub*’s, which can create so-called spaghetti code (that is, it goes all over the place and the strands are hard to track), in PBasic 2.5 you

can define module *if-else-endif* structures to keep everything contained in one spot. For example, the code

```
' {$PBASIC 2.5} 'Tells compiler you're using v2.5
if in0 = 1 then
  high 1
else
  low 1
endif
```

looks at I/O pin 0, and if HIGH (1), I/O pin 1 is switched HIGH; otherwise, it's switched LOW.

SELECT-CASE-ENDSELECT

Sometimes you want to test a value against a variety of conditions. The value of I/O pins will always be either one of two states, 0 or 1, but the value of a byte-wide variable, or four I/O pins reading a 4-bit value, needs something more. You can construct convoluted *if* statements to test for all of the variations (16 for a nibble; 256 for a byte), but the easier approach is to use the *select-case-endselect* structure.

An example:

```
' {$PBASIC 2.5}
MyVar  Var  Byte
' . . . some additional code here
select MyVar
  case 1
    low 10: high 12
  case 2
    high 10: low 12
  case 3
    high 10: high 12
  case else
    low 10: low 12
endselect
```

MyVar is assigned as a Byte variable. Somewhere along the line in the program the value of *MyVar* is set to some value. The example code looks for three explicit values, 1, 2, and 3. Depending on the value, I/O pins 10 and 12 are set LOW or HIGH, in some combination.

Note the *case else*. This is the “catch-all,” what happens if some other value (if any) matches what's in *MyVar*. Here, for any value other than 1, 2, or 3, I/O pins 10 and 12 are both set LOW.

DO/WHILE/UNTIL-LOOP

Structured loops allow your program to continue until some condition is met—or not met. Statements within the loop structure are repeated. PBasic 2.5 supports three variations of conditional loops:

- *do-loop* repeats commands infinitely. There is no actual condition here; the do-loop is engineered to repeat code over and over again, as is common in the main loop of a robot control program.

- *do-while-loop* repeats commands until a condition is false. The “do while” is evaluated like an *if* statement.
- *do-until-loop* repeats commands until a condition is true.

do-loop Example

```
do
' . . . repeat these statements
loop
```

The statements between *do* and *loop* repeat over and over again. As noted, you'd typically use this for your main program code.

do-while loop Example

```
do while in0 = 1
' . . . repeat these statements
loop
```

The statements between *do while* and *loop* are repeated as long as I/O input 0 is 1 (HIGH).

do-until loop Example

```
do until in0 = 1
' . . . repeat these statements
loop
```

The example shows how *do-while* and *do-until* are the inverse of one another. The statements between *do until* and *loop* are repeated as long as I/O input 0 is *not* 1 (HIGH).

Exiting do-loops Early

From time to time you may wish to exit a *do-loop* before the conditions (in the case of *while* and *until*) are met. This can be accomplished with the *exit* statement, which is usually a part of some *if* expression or other structure. For example:

```
do while in1 = 0
' . . . repeat these statements
  if in0 = 1 then exit
loop
' . . . program resumes here
```

Here the *do-while* loop continues as long as I/O input 1 is 0 (LOW). However, another test can break out of the loop: if I/O input 0 is 1 (HIGH), the loop is also broken.

Interfacing Hardware with Your Microcontroller or Computer

The brains of a robot don't operate in a vacuum, even if you've built a vacuum-cleaning robot. They need to be attached to motors to make the robot move and to sensors to make the robot perceive its surroundings. In most cases, these outside devices cannot be directly connected to the computer or microcontroller of a robot. Instead, it is usually necessary to condition these inputs and outputs so they can be used by the robot's brain.

In this chapter you'll learn about the most common and practical methods for interfacing real-world devices to computers and microcontrollers. For your convenience, some of the material presented in this chapter is replicated, in context, in other chapters of the book. It's also here to bring everything into focus.

Sensors as Inputs

By far the most common use for inputs in robotics is sensors. There are a variety of sensors, from the supersimple to the amazingly advanced. All share a single goal: providing the robot with data it can use to make intelligent decisions. For example, using a low-cost temperature sensor, an "energy watch" robot might record the temperature as it strolls throughout the house, looking for locations where the temperature varies widely, indicating a possible energy leak.

TYPES OF SENSORS

Broadly speaking, there are two types of sensors: analog and digital, so called because of the output signal they produce:

- Basic *digital sensors* provide on/off results. A switch is a good example of a digital sensor: either the switch is closed or it's open. Closed and open are analogous to on and off, True

and False, HIGH and LOW (these are called *logic levels*). More complex digital sensors can provide a stream of HIGH/LOW pulses that represents a value other than on or off—for example, it can represent a temperature or an angle. These sensors must connect to your robot's brain in very specific ways.

- *Analog sensors* provide a range of values, usually a voltage. In many cases, the sensor itself provides a varying resistance or current, which is then converted into a voltage by an external circuit. For example, when exposed to light, the resistance of a CdS (cadmium sulfide) cell changes dramatically. In a simple circuit with a second fixed resistor, the resistance of the CdS cell is used to provide a voltage.

As shown in Figure 40-1, in both digital and analog sensors, the result is a voltage level that can be fed to a computer, microcontroller, or other electronic device. In the case of a simple digital sensor like a switch, the robot electronics are only interested in whether the voltage is a logical LOW (usually 0 volts) or a logical HIGH (usually 5 volts). As such, these simple digital sensors can often be directly connected to a robot control computer without any additional interfacing electronics.

In the case of an analog sensor, you need additional robot electronics to convert the varying voltage levels into a form that a control computer can use. This typically involves an analog-to-digital converter, which is discussed later in this chapter.

And in the case of a digital sensor that provides a stream of HIGH/LOW pulses to represent more complex data, this stream must be captured and analyzed by the robot's control computer. Exactly how this is done depends on the type of digital signal the sensor produces. In many cases, the value provided by the sensor can be interpreted using software running on the control computer. Sophisticated interfacing electronics are not required.

There are many kinds of analog signals. Shown in Figure 40-1 are three common types: analog sine wave, analog square wave, and analog variable voltage. The actual voltages involved in these signals can vary depending on the circuitry, whereas with digital signals it's usually 0 and 5 volts. Note that analog square waves may look like a digital signal, but it's the frequency (number of waves per second) that is of most interest.

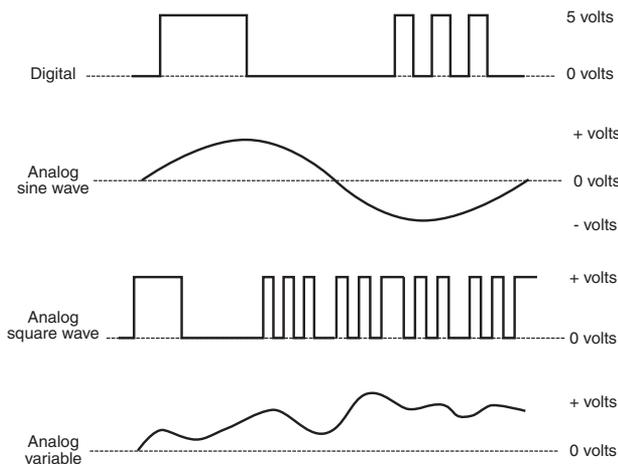


Figure 40-1 Common types of signals used in sensors and other circuits in robots. Digital signals contain information in the sequence of LOW and HIGH pulses. Analog signals contain information either as the instantaneous voltage level or the number of changes in the voltage over a certain period of time, such as 1 second.

EXAMPLES OF SENSORS

One of the joys of building robots is figuring out new ways of having them react to things. This is readily done with the wide variety of affordable sensors now available. New sensors are constantly being introduced, and it pays to stay abreast of the latest developments. Not all new sensors are affordable for the hobby robot builder, of course—you'll just have to dream about getting that \$5000 vision system. But there are plenty of other sensors that cost much less. Many are just a few dollars.

Part 8 of this book discusses many different types of sensors commonly available today that are suitable for amateur robotic work. Here is just a short laundry list to whet your appetite:

- *Contact switches.* Used as “touch sensors,” when activated these switches indicate that the robot has made contact with some object.
- *Sonar range finder.* Reflected sound waves are used to judge distances.
- *Infrared range finder or proximity.* Reflected infrared light is used to determine distance and proximity.
- *Pyroelectric infrared.* An infrared element detects changes in heat patterns and is often used in motion detectors.
- *Speech input or recognition.* Your own voice and speech patterns can be used to command the robot.
- *Sound.* Sound sources can be detected by the robot. You can tune the robot to listen to only sounds above a certain volume level.
- *Accelerometer.* Used to detect changes in speed and/or the pull of the earth's gravity, accelerometers can be used to determine how fast a robot is moving, whether it's tilted dangerously from center, or even when it's abruptly stopped.
- *Gas or smoke.* Gas and smoke sensors detect dangerous levels of noxious or toxic fumes and smoke.
- *Temperature.* A temperature sensor can detect ambient or applied heat. Ambient heat is the heat of the room or air; applied heat is some heat (or cold) source directly applied to the sensor.
- *Resistive.* A resistive sensor detects touch, force, pressure, or strain. One inexpensive form of resistive sensor is the round or square touch pad. It provides a varying resistance depending on the amount of pressure against it.
- *Light sensors.* Various light sensors detect the presence or absence of light. Light sensors can detect patterns when used in groups (called “arrays”). Though not a camera by any means, the greater the number of sensors, the more detail the robot can discern.
- *Vision.* A sensor with an array of thousands of light-sensitive elements is essentially a video camera, which can also be used to construct the eyes of a robot.

Motors and Other Outputs

A robot uses outputs to take some physical action. Most often, one or more motors are attached to the outputs of a robot's brain to allow the machine to move. On the typical mobile robot, the motors serve to drive wheels, which scoot the bot around the floor. On a stationary robot, the motors are attached to arm and gripper mechanisms, allowing the robot to grasp and manipulate objects.

Motors aren't the only ways to provide motility to a robot. Your robot may use solenoids to "hop" around a table, or pumps and valves to power pneumatic or hydraulic pressure systems. No matter what system the robot uses, the basic concepts are the same: the robot's control circuitry (that is, a computer or microcontroller) provides a voltage to the output, which turns on the motor, solenoid, or pump. When voltage is removed, the motor (or whatever) stops.

OTHER COMMON TYPES OF OUTPUTS

Other types of outputs are used for things like:

- *Sound.* The robot may use sound to warn of some impending danger ("There'll be no escape for the princess this time!") or to scare away intruders. If you've built an R2-D2-like robot (from *Star Wars* fame), your robot might use chirps and bleeps to communicate with you. Hopefully, you'll know what "bebop, pureeep!" means.
- *Voice.* Either synthesized or recorded, a voice lets your robot communicate in more human terms.
- *Visual indication.* Using light-emitting diodes (LEDs), numeric displays, or liquid-crystal displays (LCDs), visual indicators help the robot communicate with you in direct ways.

CONSIDERING POWER-HANDLING REQUIREMENTS

Outputs typically drive heavy loads: motors, solenoids, pumps, and even high-volume sound demand lots of current. The typical robotic control computer cannot provide more than 15 to 40 mA (milliamperes) of current on any output. That's enough to power one or two LEDs, but not much else.

To use an output to drive a load, you need to add a power element that provides adequate current. This can be as simple as one transistor or it can be a ready-made power driver circuit capable of running large, multihorsepower motors. One common power driver is the H-bridge, so called because the transistors used inside it are in an "H" pattern around the motor; see Chapter 22, "Using DC Motors," for more information on H-bridges. The H-bridge can connect directly to the control computer of the robot and provides adequate voltage and current to the load.

CURRENT SOURCE OR CURRENT SINK

An electric circuit is said to "source" or "sink" current. The terms are relative to the load, that is, the part of the circuit that is drawing and consuming the current. A good example of a load is a lightbulb, shown in the very simplified circuit in Figure 40-2. The lightbulb is turned on and off using a transistor.

- When *sourcing* current, the circuit supplies current to the load. In the typical wiring, current is sourced when the output goes HIGH in order to turn the load on.
- When *sinking* current, the load draws current from the circuit. The wiring is the reverse of that above: current is sunk when the output goes LOW in order to turn the load on.

The difference between sourcing and sinking is not irrelevant, though it may seem so at first glance. It's important because most circuits can sink more current than they can source.

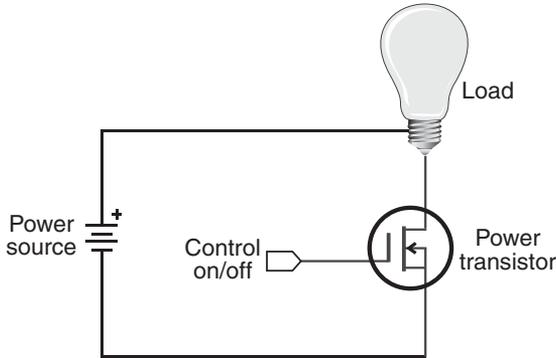


Figure 40-2 Basic electronic control of a “load,” in this case a lightbulb. Depending on how the load is wired, it can draw current either into or out of the circuit.

This is why on many chips you’ll see different current-handling capacities for source and sink. The source is often less—for example, source 40 milliamps (mA) current and sink 50 mA.

Depending on how they are wired, certain types of semiconductors, such as NPN transistors, cannot source current. To provide proper operation of the transistor, a *pull-up resistor* is usually added between the collector pin of the transistor and the +V power connection. (For more on NPN and other types of transistors, see Chapters 30 and 31.) A pull-up resistor is just a normal resistor. It’s used to literally “pull up” the output of the transistor to the level of +V when the transistor is not conducting current.



The output of the LM339 comparator IC is what’s known as *open collector*—it’s just the collector pin of an NPN transistor, without anything else added. This makes the chip more flexible in different kinds of circuits, but it also means you need to add that pull-up resistor to the comparator output, or else the chip won’t work properly.

The value of the pull-up resistor depends on the application and the characteristics of the transistor. Without getting into the nuances of circuit design, so-called *weak pull-up* resistors have values of 20 k Ω and above. They’re often used to reduce power usage of the transistor, chip, or other circuit that uses them. Many microcontrollers have weak pull-up resistors built inside them; a software setting can disable the resistors.

Strong pull-up resistors have values of 2 k Ω to 10 k Ω . They’re preferred in applications where very fast signal changes are required or when noise in the circuit may cause problems for other components.

Input and Output Architectures

As we’ve already seen, the basic input and output of a computer or microcontroller are a two-state binary voltage level (off and on), usually 0 and 5 volts. Two basic types of interfaces are used to transfer these HIGH/LOW digital signals to the robot’s control computer. They are parallel and serial.

PARALLEL INTERFACING

In a *parallel interface*, multiple bits of data are transferred at one time using separate wires. Parallel interfaces enjoy high speed because multiple data bits are transferred at the same

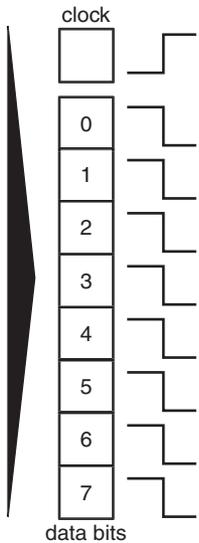


Figure 40-3 In a parallel interface, individual wires are used to carry all the data bits of the communication. A separate clock wire is used to synchronize when the data bits should be read.

time. In a typical parallel interface with an 8-bit-wide *port*, there are eight data lines, and 8 bits are transferred simultaneously. Circuits connected to the parallel interface know the data is read when a clock (or strobe) bit is toggled from one state to another (see Figure 40-3).

One of the most common parallel interfaces you'll encounter in robotics work is when using the HD7740 controller with a liquid-crystal display (LCD). The HD7740 is the de facto standard controller for all-text LCDs, and it supports a 4-bit-wide port. You communicate with it using four data lines, plus three additional data lines for control. See a working example in Chapter 47, "Interacting with Your Creation."

SERIAL INTERFACING

The downside to parallel interfaces is that they consume many input and output (*I/O*) lines on the robot computer or microcontroller. There are only a limited number of *I/O* lines on the control computer—typically 12 to 16, sometimes fewer. If the robot uses even one 8-bit parallel port, that leaves precious few *I/O* lines for anything else.

Serial interfaces, on the other hand, conserve *I/O* lines because they send data on just one or two wires. They do this by separating a byte of information into its constituent 8 bits, then sending each bit down the wire at a time, in single-file fashion (Figure 40-4). Most serial communications schemes use two *I/O* lines, but there are some that use just one, and others that use three or four (see the next sections). The additional *I/O* lines are used for such things as timing and coordinating between the data sender and the data recipient.

A number of the sensors you may use with your robot have serial interfaces, and on the surface it may appear they are a tad harder to interface than parallel connections. But, in fact, this is not the case if you use the right combination of hardware and software.

Before you can use the serial data from the sensor, you have to "clock out" all of the bits and assemble them into 8-bit data, which is used to represent some meaningful value—such as distance between the sensor and some object. The task of reconstructing serial data is made easier when you use a microcontroller with built-in serial communications commands. The BASIC Stamp, PICAXE, and Arduino all provide such commands.

Serial communications can be broadly classified into two groups: asynchronous and synchronous.

In *asynchronous serial*, a single wire is used to convey the bits of data, and to provide necessary timing signals so the listener can follow the conversion. In Figure 40-4, note the idle, start, and stop signals that are part of the 8 bits of data. When *idle*, the listener knows the talker isn't sending any data. But the moment a *start* bit is encountered, the listener knows data is to follow.

In *synchronous serial*, at least two wires are used: one contains just the data bits, and the other contains a control clock; at each pulse of the clock the listener accepts 1 bit of data. The I2C method, detailed next, is an example of one kind of synchronous serial communication.

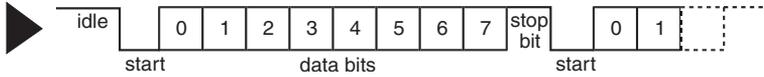


Figure 40-4 Asynchronous serial communication uses one wire to carry data bits. Synchronizing the communications—to know when data bits can be sent and received—is handled by so-called start and stop bits embedded in the data.

INTERFACING WITH I2C AND SPI

Because most microcontrollers readily support serial communications via simple-to-use commands, a rising trend in robotics is using serial-based hardware. LCD modules with serial, rather than the typical parallel, interface have long been popular. The same concept is now found on a growing number of other kinds of hardware, such as motor control modules and ultrasonic sensors.

As noted, synchronous serial communication uses two I/O lines: one for data and another for a clock. With each pulse of the clock line, another bit of data is transferred from sender to receiver. One problem with this approach is that you need two I/O lines for every serial link in your system. If you have four serial links, you need eight I/O lines. You need even more I/O lines if you want bidirectional (two-way) communications between the devices.

Several alternative *serial protocols* are commonly used in microcontrollers that make serial communications more resistant to data errors and reduce the overall pin count when working with multiple serial devices. The two most common are I2C and SPI.

I2C

I2C uses two bidirectional data lines (hence, it's often referred to as a two-wire interface, shown in Figure 40-5) to connect one or more *slave nodes* to a single *master node*. Communication is bidirectional: master and slave can talk to one another.

The two lines of an I2C connection are referred to as SDA and SCL. *SDA* stands for Serial Data Line, and it contains the data bits themselves; *SCL* stands for Serial Clock Line, and it contains the clock pulses used to marshal the data.



Technically speaking, I2C is I²C, but the superscript isn't always shown in discussions or documentation, and is a source of confusion for some. For simplicity, we show it as I2C, and, in fact, many people now refer to it as “eye two see” rather than “eye squared see.” Use whatever form you're comfortable with.

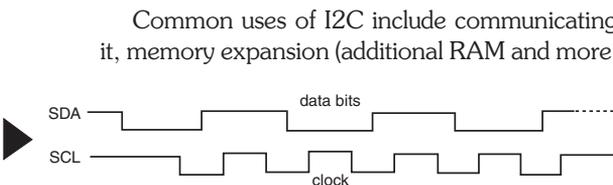


Figure 40-5 Synchronous serial communication uses two (sometimes more) wires; one wire carries the data bits themselves, and the other wire, a clock signal to provide synchronization.

Common uses of I2C include communicating with sensors and other devices that support it, memory expansion (additional RAM and more EEPROM), and allowing two (or more) microcontrollers to talk to one another. This latter example has numerous applications when using so-called subcontrollers: a main controller that does most of the work, but additional microcontrollers for specific tasks, such as operating motors.

Implementing I2C is fairly simple, when given the right supporting software.

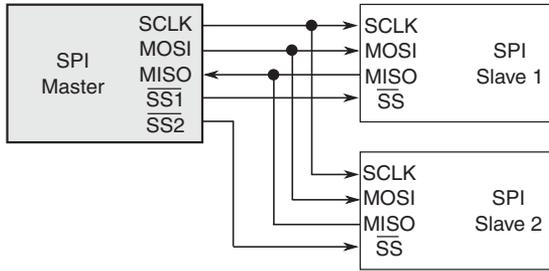


Figure 40-6 Serial peripheral interface (SPI) uses a minimum of four wires for bidirectional communications. But you can set up additional communications links with other devices by adding only one wire for each one.

For instance, two of the I/O lines on the Arduino microcontroller are designed with I2C in mind. The controller comes with a function library that makes setting up and using I2C relatively painless. In Part 8 there are several examples of using I2C with sensors and the Arduino.

SPI

SPI, or serial peripheral interface, uses three I/O lines to communicate between master and slave devices, with a fourth I/O line reserved as a kind of “raise your hand” signal to keep everyone from talking at the same time. This fourth I/O is referred to as the *Slave Select* (SS) line. The functions of all the I/O lines in SPI are aptly named and indicate their roles:

- *SCLK*. Serial Clock (provided by the master).
- *MOSI*. Master Output, Slave Input.
- *MISO*. Master Input, Slave Output.

To connect two slaves to one master, for example, you need the three main I/O lines wired to each slave. And then you need two additional lines, each one separately wired from the master to the slave. See Figure 40-6 for an example.

Interfacing Outputs

As mentioned previously, most output circuits require more voltage or current than the control electronics (computer, microprocessor, microcontroller) of your robot can provide. You need some way to boost the current needed for proper operation. Techniques include:

- Direct connection:** Some types of output devices can be driven directly by your robot electronics because their current consumption is low. These typically include LEDs and small piezo buzzers.
- Bipolar and MOSFET transistors:** Transistors are used to amplify current. With a transistor on the output of a microcontroller it’s the transistor that provides the current to the load, and not the microcontroller.
- Motor bridge module:** A motor bridge module is a special type of integrated circuit that’s intended to be used to interface to a DC motor. While motor drive is the most common application of these ICs, in fact they can be used to operate just about any high-current application.

Relay: A low-tech but still usable output interface is the mechanical relay. For very small relays you can sometimes connect it directly to your robot's microcontroller or other electronics, but in most cases you need a resistor and transistor to boost the current.

Interfacing Digital Inputs

The following sections describe common ways to connect digital inputs to the control electronics (microprocessor, computer, or microcontroller) of your robot.

BASIC INTERFACE CONCEPTS

Switches and other strictly digital (on/off) sensors can be readily connected to your robot. The most common methods are:

Direct connection: The most basic interface is simply a wire between devices. This is an acceptable approach when there are no voltage or current issues involved. This is the most common method when connecting switches as inputs to a microcontroller, for use as bump detectors (Figure 40-7), or when connecting a piezo element to produce sound.

Switch debounce: Mechanical switches give dozens to hundreds of false triggers each time they change from opened to closed state. Call these "glitches" or "transients" or *bounces*, the net effect is the same: your microcontroller may react to each of these false triggers, rerunning the same code multiple times, and causing potential behavioral problems in your robot (if there's such a thing as attention deficit disorder in a robot, this is it!). A (very) basic switch debounce circuit like that in Figure 40-8 provides a clean transition when the switch changes state. The value of the capacitor can be selected empirically, a fancy word for trial and error—I like to think of it as *learning through experimentation!* On a switch with lots of glitches you'll need to select a higher value, but the higher the value, the more sluggish the switch reaction gets. Note the polarity of the capacitor if using a tantalum or aluminum electrolytic components.

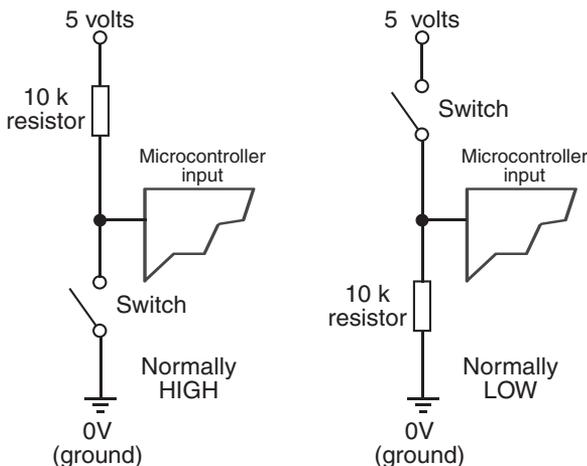


Figure 40-7 Direct connection of an input, in this case a simple switch, to a microcontroller or other circuit.

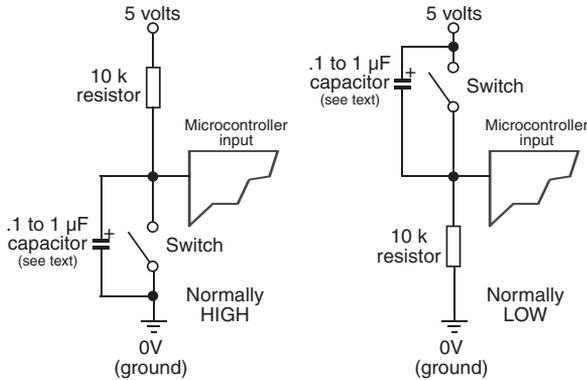


Figure 40-8 Mechanical switches create electrical noise—or “bounces”—when opened or closed. A circuit like this one removes the extra noise, helping the control electronics better determine exactly when the switch is opened or closed.

Limit current: Too much current can kill a microcontroller, and some hardware interfacing may require ways to limit current going into, or out of, the controller. An ordinary resistor (see Figure 40-9) is typically used to limit current. One common use: a resistor between microcontroller and LED. The resistor prevents the LED from drawing too much current, which will definitely destroy the LED and could also damage the microcontroller. Other instances of current limiting include inserting a resistor in line with servo motors and from switches that could carry an electrostatic discharge into the microcontroller. For 5-volt sources, the typical resistor value is between 330 Ω and 680 Ω .

Buffer input: A buffer is any of a number of active electronics between input/output and the controller. There are lots of kinds and uses of buffers; a common form is a transistor between a microcontroller and a relay that operates a motor. The transistor not only helps to isolate the I/O line of the controller, but also boosts current needed to drive the relay. Other forms of buffers include an op-amp and specialty integrated circuits that contain several independent buffered inputs and outputs (Figure 40-10).

USING OPTO-ISOLATORS

Sometimes you might wish to keep the power supplies of the inputs and control electronics totally separate, in order to provide the maximum of protection between input and output. This is most easily done using opto-isolators, which are readily available in IC-like packages. Figure 40-11 shows the basic concept of the opto-isolator: the source controls a light-emitting diode. The input of the control electronics is connected to a photodetector of the opto-isolator.

Note that since each “side” of the opto-isolator is governed by its own power supply, you can also use these devices for simple level shifting, for example, changing a +5 vdc signal to +12 vdc, or vice versa.

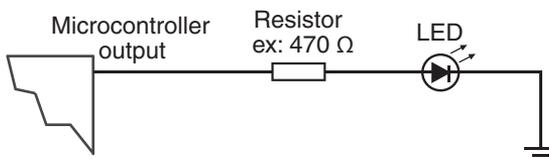


Figure 40-9 Some electronic devices, such as light-emitting diodes, require current limiting, or else damage to the circuit could result. The resistor reduces current to the LED.

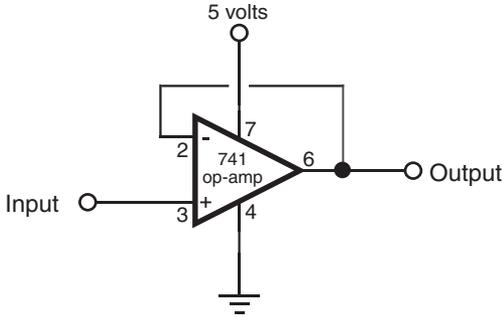


Figure 40-10 A buffer is used to isolate one circuit from another, but it may also be used to provide additional drive current, to invert the digital signal from LOW to HIGH (or vice versa), and for many other applications.

ZENER DIODE INPUT PROTECTION

If a signal source may exceed the operating voltage level of the control electronics, you can use a zener diode to “clamp” the voltage to the input. Zener diodes act like valves that turn on only when a certain voltage level is applied to them. By putting a zener diode across the +V and ground of an input, you can basically divert any excess voltage and prevent it from reaching the control electronics.

Zener diodes are available in different voltages; 4.7- or 5.1-volt zeners are ideal for interfacing to inputs in most robot electronics. You need to use a resistor to limit the current through the zener. Using zeners, and selecting the proper resistor value, requires some simple math, which is detailed in Chapter 19, “Robot Power Systems.”

Interfacing Analog Input

In most cases, the varying nature of analog inputs means they can’t be directly connected to the control circuitry of your robot. If you want to *quantify* the values from the input, you need to use some form of analog-to-digital conversion, detailed later in this chapter.

Additionally, you may need to condition the analog input so its value can be reliably measured. This may include amplifying, as detailed in this section.

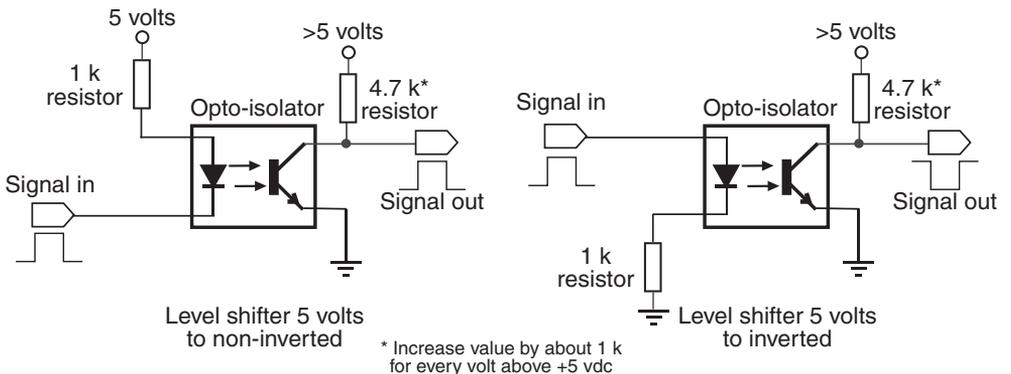


Figure 40-11 An opto-isolator keeps two circuits separate from one another, yet provides a way for one circuit to influence another.

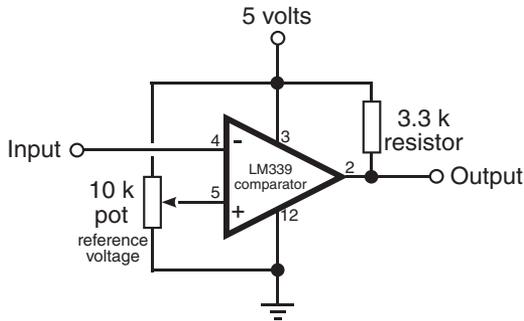


Figure 40-12 In a voltage comparator, an input voltage is compared against a reference voltage. A common scheme is shown here, where the reference voltage is provided by a potentiometer.

VOLTAGE COMPARATOR

The voltage comparator takes an analog voltage and outputs a simple off/on (LOW/HIGH) signal to the control electronics of your robot. The comparator is handy when you're not interested in knowing the many possible levels of the input, but you want to know when the level goes above or below a certain threshold.

Figure 40-12 shows a common voltage comparator circuit. The potentiometer is used to determine the “trip point,” or threshold, of the comparator. To set the potentiometer, apply the voltage level you want to use as the trip point to the input of the comparator. Adjust the potentiometer so the output of the comparator just changes state.

Note that a pull-up resistor is used on the output of the comparator chip (LM339) used in the circuit. The LM339 uses an open collector output, which means that it can pull the output LOW, but it cannot make it HIGH. The pull-up resistor allows the output of the LM339 to go HIGH when it needs to.



The LM339 integrated circuit actually has four independent voltage comparators in it. This is handy in many robotics experiments, where you want to provide multiple inputs for sensors for the left and right of the robot, or for the front and back.

You might have noticed that the voltage comparator has two inputs, one marked + and the other marked -. These are more commonly referred to as the *noninverting* and *inverting* inputs, respectively. You can alter the operation of the comparator by switching the roles of the inputs. That is, instead of having the output of the comparator switch *off* when the threshold voltage is reached, by flipping the inputs the comparator will turn *on*.

SIGNAL AMPLIFICATION

Some types of analog sensors don't provide a signal that is strong enough to be directly used by the rest of your robot's circuitry. In these instances you must amplify the signal, which can be done by using a transistor or an operational amplifier.

The op-amp method is the easiest in most cases. While the LM741 op-amp is perhaps the most famous, it's not always the best choice, depending on the application. So I've specified an OPA344, which is a low-cost op-amp available from a number of online sources. It provides two benefits over the LM741: its output is rail-to-rail, meaning that assuming a 5-volt supply, the full voltage swing is 0 to 5 volts (or very nearly so). Second, it is made to operate from a single-ended power supply. No need for a split (+ and - power) supply.

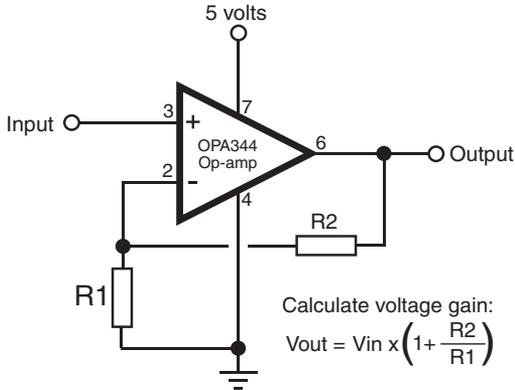


Figure 40-13 The op-amp is a highly versatile circuit, able to amplify and condition signals in hundreds of ways. This is the basic connection scheme for an op-amp designed to amplify a signal.



If you can't locate an OPA344 chip, you can substitute most any other op-amp that is both rail-to-rail and single-supply.

Figure 40-13 shows the basic op-amp as an amplifier. The resistors marked $R1$ and $R2$ set the *gain*, or amplification, of the circuit. The basic formula to use to calculate approximate voltage gain is provided in the illustration.

OTHER SIGNAL TECHNIQUES FOR OP-AMPS

There are many other ways to use op-amps for input signal conditioning, and they are too numerous to mention here. A good source for simple, understandable circuits is the *Forrest Mims Engineer's Notebook*, by Forrest M. Mims III, available at most online bookstores. No robotics lab should be without Forrest's books.

COMMON ANALOG INPUT INTERFACES

Many types of analog devices can be connected to robot electronics through simple interfaces. Most are engineered to provide a varying voltage, which can then be applied to the analog-to-digital converter of a microcontroller (see later in this chapter), or a voltage comparator, to determine if the voltage exceeds a certain threshold. Among the most common interfaces for robotics:

- *CdS* (cadmium-sulfide) cells are, in essence, variable resistors that are sensitive to light. By putting a CdS cell in series with another resistor between the +V and ground of the circuit (Figure 40-14), a varying voltage is provided that can be read directly into an analog-to-digital converter or voltage comparator. No amplification is typically necessary.
- A *potentiometer* forms a *voltage divider* when connected as shown in Figure 40-15. The voltage varies from 0 volts (ground) to +V. No amplification is necessary.
- The output of a *phototransistor*, or light-sensitive transistor, is a varying current that can be converted to a voltage by using a resistor (see Figure 40-16). The higher the resistance, the higher the sensitivity of the device. The output of a phototransistor is typically from 0 volts (ground) to close to +V, and therefore no further amplification is necessary.

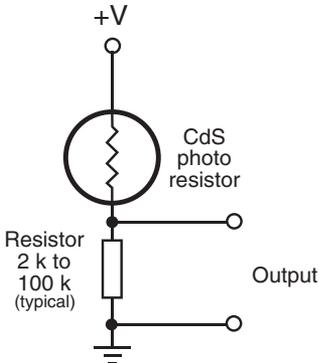


Figure 40-14 A cadmium-sulfide photo cell is a variable resistor. Its resistance changes depending on the amount of light falling on the sensor. When connected with another resistor, the output of the cell can be read as a varying voltage.

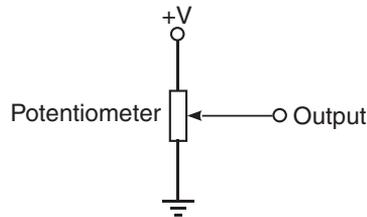
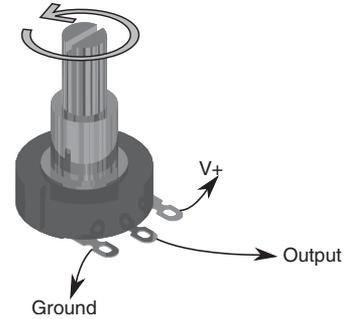


Figure 40-15 A potentiometer provides a convenient way to detect a varying voltage. The shaft of the potentiometer can be connected to a moving part of your robot (like an arm), and the voltage may be used to indicate position.



- Like a phototransistor, the output of a *photodiode* is a varying current. This output can also be converted into a voltage by using a resistor. The output of a photodiode tends to be fairly low. That means amplification is usually required.

Connecting with USB

USB stands for universal serial bus, and it's now the most common way for a computer to communicate with devices connected to it. You may frequently use a USB to download programs that you develop on your computer to your microcontroller. In order for your PC and microcontroller to talk via USB, the controller must have a USB connector on it. Microcon-

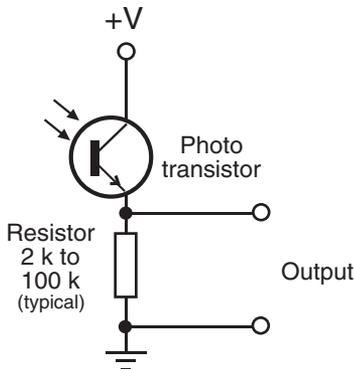


Figure 40-16 A photo transistor is a transistor that is sensitive to light. As light falls on the device, it turns on (conducts current). The more light, the more the transistor turns on. When used with a resistor, the output of the transistor is a variable voltage.

trollers that are built on a larger circuit board, such as the Arduino, have integrated USB; when not built-in, USB is available as an option.



Other methods of communicating between PC and microcontroller include using the older-style serial ports and even the now-almost-forgotten parallel port. Parallel port connections are not as common these days, but many single-chip controllers (PICMicro, AVR, PICAXE) are programmed via the PC serial port. If your computer lacks a serial port, you can use a USB-to-serial converter. These are available from most any local or online store that specializes in personal computers.



Some robo-experimenters prefer to use self-powered USB hubs. Not only do the hubs provide isolation between your project and your PC (protecting your PC in case of a bad short), but because they use their own power supply, they are better able to deliver the requisite current to each port (500 mA for USB 2.0, 900 mA for USB 3.0).

Using Analog-to-Digital Conversion

Computers are binary devices: their *digital* data is composed of 0s and 1s, strung together to construct meaningful information. But the real world is analog, where data can be most any value, with literally millions of values between “none” and “lots”!

Analog-to-digital conversion is a system that takes analog information and translates it into a digital, or, more precisely, *binary*, format suitable for your robot. Many of the sensors you will connect to the robot are analog in nature. These include temperature sensors, microphones and other audio transducers, variable output tactile feedback (touch) sensors, position potentiometers (the angle of an elbow joint, for example), light detectors, and more. With analog-to-digital conversion you can connect any of these to your robot.

HOW ANALOG-TO-DIGITAL CONVERSION WORKS

Analog-to-digital conversion (ADC) works by converting analog values into their binary equivalents. Low analog values (like a weak light striking a photodetector) might have a low binary equivalent, such as “1” or “2.” But a high analog value might have a high binary equivalent, such as “255.”

The *smaller* the change in the analog signal required to produce a different binary number, the *higher* the “resolution” of the ADC circuit. The resolution of the conversion depends on both the voltage span (0 to 5 volts is most common) and the number of bits used for the binary value to represent the analog voltage.

Suppose the signal spans 10 volts, and 8 bits (or a byte) are used to represent the levels of this voltage. There are 256 possible combinations of 8 bits, which means the span of 10 volts will be represented by 256 different values—from 0 to 255. Figure 40-17 shows a changing analog signal of 1 to 10 volts, with equivalent digital values showing at 0, 5, and 10 volts.

- 0 volts is binary 0.
- 5 volts is binary 127.
- 10 volts is binary 255.

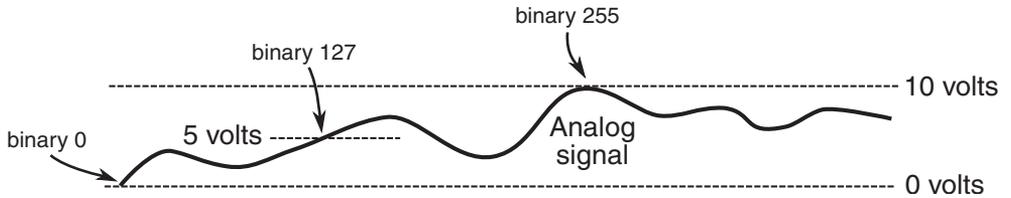


Figure 40-17 How an analog voltage (in this case 0 to 10 volts) is converted to digital values 0 to 255. The full 10 volts is equivalent to binary 255; 0 volts is binary 0. Five volts—halfway between 0 and 10 volts—is binary 127.

Given 10 volts and 8 bits of conversion, the ADC system will have a resolution of 0.039 volts (39 millivolts) per step. Obviously, the resolution of the conversion will be finer the smaller the span or the higher the number of bits. With a 10-bit conversion, for instance, there are 1024 possible combinations of bits, or roughly 0.009 volts (9 millivolts) per step.

ANALOG-TO-DIGITAL CONVERSION ICs

You can construct analog-to-digital converter circuits using discrete logic chips—basically a series of voltage comparators strung together. One *much* easier approach is a special-purpose ADC integrated circuit. While somewhat “old skool” these days, ADC chips are still widely available, are fairly cheap, and come in a variety of forms. Here are some:

- *Single or multiplexed input.* Single-input ADC chips, such as the ADC0804, can accept only one analog input. Multiplexed-input ADC chips, like the ADC0809 or the ADC0817, can accept more than one analog input (usually 4, 8, or 16). The control circuitry on the ADC chip allows you to select the input you wish to convert.
- *Bit resolution.* The basic ADC chip has an 8-bit resolution. Finer resolution can be achieved with 10- and 12-bit chips. For example, the LTC1298 is a 12-bit ADC chip that can transform an input voltage (usually 0 to 5 volts) into 4096 steps.
- *Parallel or serial output.* ADCs with parallel outputs provide separate data lines for each bit. Serial ADCs have a single output, and the data is sent 1 bit at a time. The LTC1298 is an example of an ADC that uses a serial interface.

INTEGRATED MICROCONTROLLER ADCs

Many microcontrollers and single-board computers come equipped with one or more analog-to-digital converters built in. This saves you the time, trouble, and expense of connecting a stand-alone ADC chip to your robot. You just tell the system to fetch an analog input, and it tells you the resulting digital value.

Using Digital-to-Analog Conversion

Digital-to-analog conversion (DAC) is the inverse of analog-to-digital conversion. With a DAC, a digital signal is converted to a varying analog voltage. DACs are common in some types of products, such as audio compact discs.

In the field of robotics, digital-to-analog conversion is typically performed indirectly using an approach referred to as pulse width modulation (*PWM*). It's most common in controlling the speed of motors. In operation, a circuit applies a continuous train of pulses to the motor. The longer the pulses are "on," the faster the motor will go. This works because motors tend to "integrate" the pulses to an average voltage level; no separate digital-to-analog conversion is required. See Chapter 22 for additional information on PWM with DC motors.

When needed, you can accomplish digital-to-analog conversion using integrated circuits specially designed for the task. The DAC08, for example, is a time-honored 8-bit digital-to-analog converter IC. It's inexpensive (a couple of dollars) and is easy to use.

Expanding Available I/O Lines

A bane of the microcontroller- and computer-controlled robot is the shortage of input and output pins. It always seems that your robot needs one more I/O pin than what's available. One way to tackle the problem is to drop a feature or two from the robot or even add a second computer or microcontroller. Fortunately, there are other alternatives.

FROM FEW TO MANY

The *data demultiplexer*, or *demux*, lets you turn a few I/O lines into many. You'd typically use this in a microcontroller application to expand the number of available outputs—like control eight LEDs with just a few output lines.

Demultiplexers are available in a variety of types; a common component uses three input lines and eight output lines. You can individually activate any one of the eight output lines by applying a binary control signal on the three inputs.

The following table shows the binary control signals using 3 bits to operate eight outputs. The 3 bits are shown in binary format, where 0 means off or LOW, and 1 means on or HIGH. The specific value of the 3 bits corresponds to a selected output. (To operate additional outputs you merely need to set more input control bits. To control 256 outputs, for example, you need 8 bits.)

Input Control	Selected Output	Input Control	Selected Output
000	1	100	5
001	2	101	6
010	3	110	7
011	4	111	8

An example of the demultiplexer is the venerable 74138 chip, which is designed to bring the selected output LOW, while all the others stay HIGH. One caveat regarding demultiplexers is that only one output can be active at any one time. As soon as you change the input control, the old selected output is deselected, and the new one is selected in its place.

One way around this is with a serial-to-parallel shift register, such as the 74595 (this is a family of chips; the actual chip you'll use can be most any member of the family, such as 74HCT595 or 74LS595). The 74595 chip uses three inputs (and optionally a fourth, but for

our purposes it can be ignored) and provides eight outputs. Set the outputs you want to activate by sending the 74595 an 8-bit serial word. For example,

Serial "Word"	Selected Output(s)
00000001	1
00001001	1 and 4
01000110	2, 3, and 7

. . . and so on. Figure 40-18 shows how to set up a 74595 to turn on any of eight LEDs, individually or in groups. In operation, software on your robot's computer or microcontroller sends eight clock pulses to the Clock line. At each clock pulse, the Data line is sent 1 bit of the serial word you want to use. When all eight pulses have been received, the Latch line is activated. The outputs of the 74595 remain active until you change them (or power to the chip is removed, of course).

If this seems like a lot of effort to expend just to turn three I/O lines into eight, many microcontrollers (and some computers) used for robotics include a "Shiftout" command that does all the hard work for you. A key benefit of the 74595 is that you can "cascade" them to expand the I/O options to more than eight outputs.

FROM MANY TO FEW

So now we want to go the other way: transform many inputs into one. This is the role of the *multiplexer* (or *mux*). In the world of microcontrollers, multiplexers are most commonly used

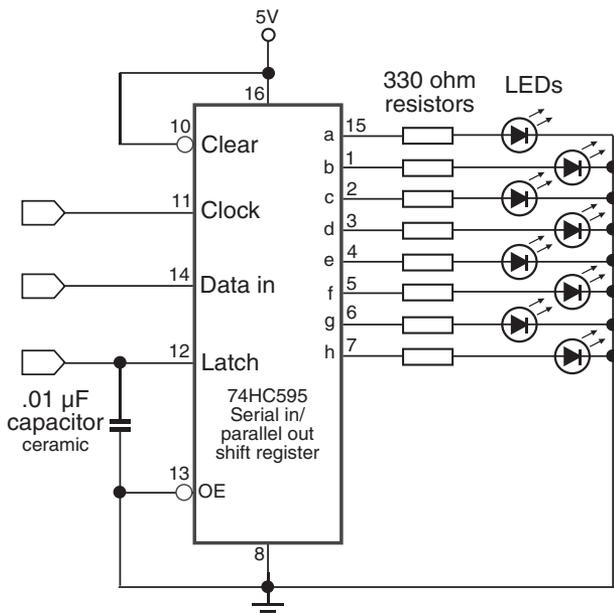


Figure 40-18 How to use a 74595 serial-in, parallel-out (SIPO) IC to exchange serial data to parallel data. The SIPO lets you expand the number of I/O lines from a microcontroller or computer; as shown here, three input lines control eight LEDs.

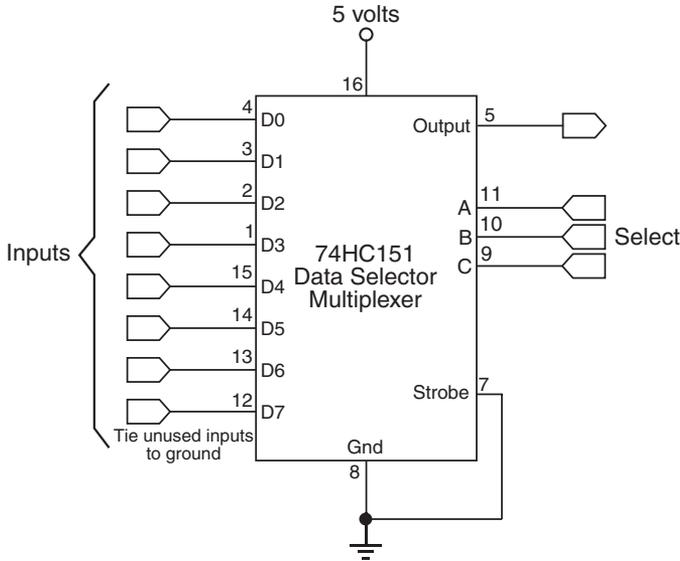


Figure 40-19 Using the 74151 data selector/multiplexer to detect the state of up to 16 inputs. Choose the input to read using the four *Select* pins; the value of that input appears on the *Out* pin of the chip.

to minimize the number of inputs needed to connect to a control circuit. For example, you might use a multiplexer to accept the signals from 12 push-button touch sensors mounted around your robot. All 12 switches are routed through the multiplexer, which in turn connects to just a single input on the microcontroller.

Multiplexers are the inverse of demultiplexers, and the CD4051 integrated circuit is a good example of how they work. This chip *muxes* up to eight signals into one, using three *Select* lines. The bits to specify which input you want are the same as in the preceding table for 8-bit demultiplexing. For instance, to read the value at Input 3, you apply 011 to the three *Select* lines: the binary value 011 is equal to decimal 3.

ANALOG VERSUS DIGITAL MUX AND DEMUX

Traditional (that is, the most commonly used) multiplexers and demultiplexers are designed for use with digital on/off signals. The SN74151 is a good example; it's a digital multiplexer—also referred to as a data selector—commonly used to pick from an array of sources and funnel them to a single input. A basic hookup diagram for the SN74151 is in Figure 40-19. Be sure to tie any unused inputs to ground (don't let them “float”).

The chip accepts up to eight inputs, labeled *Inputs* in the diagram. The *Select* pins are then set LOW or HIGH to route the desired input line to the *Output*. The following table shows which Input pin is active, according to the value of the four *Select* pins.

Select			Input Line Read
C	B	A	
L	L	L	D0
L	L	H	D1

Select			
C	B	A	Input Line Read
L	H	L	D2
L	H	H	D3
H	L	L	D4
H	L	H	D5
H	H	L	D6
H	H	H	D7

L = LOW, H = HIGH

Yet there are analog mux/demux chips available for when working with sensors and other devices that use analog signals. For example, the CD4051 is multipurpose; it functions with digital signals or analog signals. It works the same way as the 75HC151, though its pinouts are different; check the datasheet for the chip for specifics.

- Use an *analog multiplexer* to select from any of a number of analog sources, and route to a single analog input on your microcontroller. Typical use might be reading the value of a series of CdS (a light-dependent resistor) cells. You might arrange a series of 8 or 16 light sensors to form a compound “eye” for your robot, then take individual readings of each one through the multiplexer.
- Use an *analog demultiplexer* to distribute one input multiple ways. You could use this technique, for instance, to build a series of voltage-controlled oscillators (VCOs) using LM555 timer ICs. By varying the voltages you could create unearthly sound effects and sci-fi-style music.



Chips like the CD4051 can serve as both multiplexers or demultiplexers. Signals can travel through the device in either direction. The eight Inputs become outputs, and the Output becomes an input.

Be on the lookout for ingenious variations on the theme. One is the CD4066 quad bilateral switch IC, a curious critter that contains four single-pole, single-throw (SPST) switches in it. You can wire up the switches with a single common point and use it to combine analog signals—for example, from different sound sources to one amplifier and speaker. Another application is to electronically switch out different components, such as resistors, in a circuit, or to select one of four different voltage levels.

Understanding Port Changing

Port changing is going from serial to parallel, or the reverse. The concept is similar to that of using multiplexers and demultiplexers, but port changing involves more programming when interfacing to a microcontroller, while at the same time offering greater flexibility. Because of the way they work, circuits that change parallel-to-serial and serial-to-parallel are often referred to as *shift registers*.

PARALLEL GOES IN, SERIAL COMES OUT

Let's start with changing parallel to serial. This is accomplished using a *parallel-in, serial-out*, or *PISO*, chip. An example is the 74165 (and all the various members of the family, such as the 74HCT165); another is the CD4014. The ICs use eight parallel inputs, plus additional pins for loading in the parallel value and providing the serial data output. To use a PISO:

1. Apply the 8 bits to the parallel input pins. For example, this might be the instantaneous condition of eight mechanical bumper switches placed around your robot.
2. Load the parallel data to lock it in. This is done by activating one of the control pins on the chip.
3. Apply clock pulses to the chip, and then read the serial data that comes out. The serial data is a series of LOW and HIGH signals, timed with the clock you provide. So a sequence of:

0 1 0 0 1 1 0 1

means the parallel input pins in these positions are LOW for a 0 and HIGH for a 1.

FROM SERIAL TO PARALLEL, WITH LOVE

Okay, I thought of that dumb joke because I'm listening to a James Bond soundtrack CD. Anyway, going from serial to parallel is the inverse of the above, with obvious consequence. These types of circuits are known as *serial-in, parallel-out*, or *SIPO*, which you were introduced to previously in the section "Expanding Available I/O Lines." Besides the 74595, a popular IC that handles this task is the 74164 (this is a family of ICs that include the 75HC164, 74LS164, and others; you can use most any of them). A basic connection diagram is shown in Figure 40-20.

As with demultiplexers (see the preceding sections), SIPO circuits are handy for just about any situation where you want to expand just a few I/O ports to many. In robotics a common use of the SIPO is to illuminate multiple light-emitting diodes, without requiring one pin for

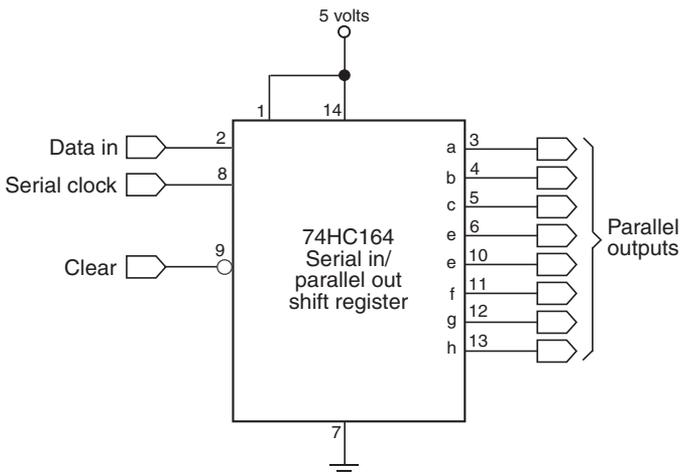


Figure 40-20 Using the 74164 shift register to convert simple serial data to 8 bits of parallel data.

each LED. One SIPO chip, using two pins of a microcontroller, can operate eight LEDs. To get any combination of LEDs to light, you merely apply 8 bits of serial data, clocking in each, and toggling the *Clock* pin of the chip.

The 74164 is an example of a nonlatching SIPO, meaning that as you enter data, the chip starts changing its parallel outputs immediately. This is usually of no concern for tasks like LED displays and when you clock in the data very quickly. The CD4094 IC is an example of a SIPO with latching outputs, meaning that the outputs won't change until (and unless) you change the latch pin.

CASCADING SHIFT REGISTERS

Many (but not all) shift register ICs let you cascade or *daisy-chain* them one after the other, so you can work with more bits. Cascading makes the most sense—at least for robotics—when using a SIPO. A single serial data train can control 8, 16, 24 . . . up to 256 parallel outputs.

SPECIALTY I/O EXPANDER CHIPS

For microcontrollers that are short on available input/output pins there are also specialty I/O expander integrated circuits. Many use the I2C serial interface standard to communicate with a host processor and offer 8, 16, and even more input and outline lines.

Examples of I/O expander chips include the PCF8575 from Texas Instruments and NXP, Microchip MCP23016, NXP PCA9555N, and the Maxim MAX7314. Most expander ICs are in the range of \$3 to \$5, and are available through the larger online electronics distributors, such as Digikey and Mouser. If there's a downside to expander chips it's this: most come only in surface-mount packages. The MCP23016 is one of a few exceptions, and it is available in a DIP-style package. For the others you'll need a generic surface-mount carrier or a custom "breakout board" that allows you to use the chip with standard breadboards.



Check the RBB Online Support site for circuits and programming examples using several popular I/O expander chips. See Appendix A for more details on the support site.

On the Web: Understanding Bitwise Port Programming

Each of the pins on a microcontroller can be controlled independently. Many controllers also let you work with multiple pins at a time. When pins are combined, they form a *port*. I/O pins are most often grouped in sets of eight, to create 8-bit ports.

With *bitwise port programming*, you can manage all the bits of the port as a whole. This makes the programming code easier for certain tasks, like operating a string of LEDs or a bunch of motors. Check out the RBB Online Support site (see Appendix A) for a detailed look at bitwise port programming.

Remote Control Systems

The most basic robot designs—just a step up from motorized toys—use a wired control box for operation. You flip switches to move the robot around the room or activate the motors in the robotic arm and hand.

This chapter details several popular ways to achieve links between you and your robot. You can use the remote controller to activate all of the robot's functions, or with a suitable onboard computer working as an electronic recorder, you can use the controller as a teaching pendant. You manually program the robot through a series of steps and routines, then play it back under the direction of the computer. Some remote control systems even let you connect your personal computer to your robot. You type on the keyboard, or use a joystick for control, and the invisible link does the rest.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, "RBB Online Support," for more details. To save page space, the lengthier programs are not printed here, but you can find them on the support site. The support site also offers parts lists (with sources) for projects, updates, and more examples you can try!

Build a Joystick "Teaching Pendant"

No doubt you've been to Disneyland or other theme parks that use robotic or "animatronic" performers. These on-stage automatons are operated via a sophisticated computerized system that plays back the audio portion of the program and controls every movement of every robot on the stage.

Animatronic shows are most commonly "acted out" by a human director who operates a joystick or other control in real time. As the sound portion of the program is played, the director moves the joystick to operate the various animatronic devices onstage. The movements of the joystick are recorded for later playback. This same concept is used in many kinds of

manufacturing robots, whose actions are programmed not from the keyboard but from a *teaching pendant*, a controller that records the actions of a human operator.

Using an older (but still available) game joystick for the IBM PC, you can create your own inexpensive teaching pendant for your robot (or animatron, if that’s to your liking). For this next project, I’ll use a common, garden-variety IBM PC-style analog joystick, one with a DB-15 (15-pin) gameport connector on it. While this style of joystick is no longer widely manufactured, it’s available cheap in surplus stores and resale shops. You may even have one lying about the house.



As an alternative to an old-fashioned analog PC joystick you may be able to hack a USB joystick to provide the same basic functionality of the old-style joystick. What you’re after is the analog voltage from the wiper of the joystick, and the push-button contacts of two buttons. If you’ve got a USB joystick you’re no longer using, open it up and see how easy it would be to hack.

Figure 41-1 shows the pinout of the DB-15 (15-pin) gameport connector. The pin functions are summarized in the following table. Note that the connector supports up to two separate joysticks, joined with a Y cable. In this project you only use one joystick (Joystick 1) and its two buttons (Button 1 and Button 2).

Pin	Function	Description
1	+5V	+5V
2	B1	Button 1
3	X1	Joystick 1 X-axis
4	GND	Ground for Button 1
5	GND	Ground for Button 2
6	Y1	Joystick 2 Y-axis
7	B2	Button 2
8	+5V	+5V (or no connection)
9	+5V	+5V
10	B4	Button 4
11	X2	Joystick 1 X-axis
12	GND	Ground for Button 3, Button 4

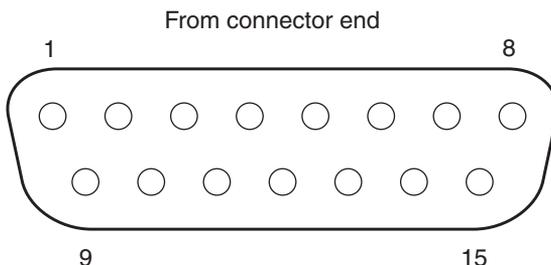


Figure 41-1 The 15-pin gameport connector used on older-style PC joysticks. This is the connector as viewed straight on.

Pin	Function	Description
13	Y2	Joystick 2 Y-axis
14	B3	Button 3
15	+5 V	+5V (or no connection)

Programming allows the joystick teaching pendant to control the motors of a two-wheel robot. You can record and play back up to 30 seconds of commands. You can also use the joystick teaching pendant in “free” (no record or playback) mode, where you control the robot by manually pushing the stick.

For the control electronics, we’ll connect the joystick to an Arduino microcontroller by way of a simple interface. Figure 41-2 shows how the joystick connects to the Arduino board. The Arduino, in turn, connects to whatever motor control electronics you are using.

IBM PC-style joysticks contain analog potentiometers; the resistive value of these pots changes as you move the joystick around. We actually won’t be using the analog nature of the joystick for this project, but you can add this feature on your own if you wish. For example, instead of controlling the power and direction of the motors, you could rig the joystick so that the more you push on the stick, the faster the motor goes.

```
101010
010101
101010
010101
```

Arduino Joystick—joystick.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details.

USING THE JOYSTICK TEACHING PENDANT

Test the program by pushing the joystick. For the purposes of verification and testing, the joystick.pde program uses the Arduino Serial Monitor window (choose Tools, Serial Monitor) to display the binary value of the 4 motor control bits (only the last 4 bits are used). For example, when you push the joystick the value 0000101 is shown in the Serial Monitor window. The last four bits are 0101:

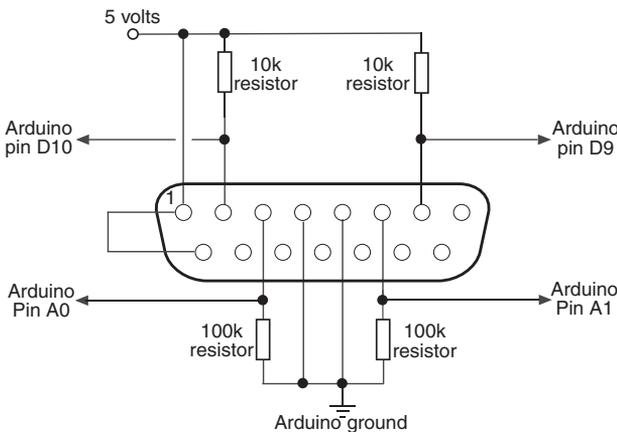


Figure 41-2 Only simple interface electronics are needed between a joystick with a 15-pin gameport plug and an Arduino microcontroller.

0	1	0	1
LeftMotDir	LeftMotCtrl	RightMotDir	RightMotCtrl

A value of 0 for LeftMotDir/RightMotDir means the motor is going forward (conversely, a value of 1 means the motor is going in reverse). A value of LeftMotCtrl/RightMotCtrl means that the motor is activated. With the bits 0101, both motors are operating and are going forward. Note that the program samples the position of the joystick once every half-second.

RECORDING AND PLAYING BACK STEPS

Briefly depress button 1 (usually labeled as the “fire” button). A “Recording On” message is displayed in the Serial Monitor window. The joystick is now in record mode, and the joystick positions are being stored in memory. Recording is simple in the *joystick.pde* program: each half-second, the joystick position is stored in one element of a 60-element array. Since there are 60 elements and a new “snapshot” of the joystick controls is made every half-second, this means there is a maximum of 30 seconds of recording.

You can revise the program to add longer programming time, but note that, like all micro-controllers, the Arduino has only so much memory for data storage. The more elements there are, the more memory is consumed.

When you are done recording the steps you want, *briefly* depress button 1 again. The joystick will be taken out of record mode. You can play back your previously stored steps by *briefly* depressing button 2. While a previously stored set of steps is being played, any joystick motions are ignored. If necessary, you can abort play mode at any time by depressing button 2 again. When playback is complete, the program automatically goes back into “free-run” mode.

Commanding a Robot with Infrared Remote Control

You can use a TV remote control to operate a mobile robot. A computer or microcontroller is used to decipher the signal patterns received from the remote via an infrared receiver. Because infrared receiver units are common finds in electronics and surplus stores, adapting a remote control for robotics use is actually fairly straightforward.

It’s mostly a matter of connecting the pieces together. With your infrared remote control you’ll be able to command your robot in just about any way you wish—to start, stop, turn, whatever.

SYSTEM OVERVIEW

Here are the major components of the robot infrared remote control system:

- *Infrared remote.* Most any modern *universal* infrared remote control, like the one in Figure 41-3, will work. These are priced starting at a few bucks at most any discount store, and I’ve even seen them at dollar stores. Important: You want a universal remote that sup-



Figure 41-3 Example universal infrared remote control. To use a universal remote with the PICAXE microcontroller, you must select the code for a Sony TV, VCR, or other home electronics device.

ports Sony TVs and other Sony brand gear—98 percent of all universal remotes do, but check just to be sure.

- *Infrared receiver/demodulator.* This all-in-one module contains an infrared light detector, along with various electronics to clean up, amplify, and demodulate the signal from the remote control. The remote sends a pattern of on/off flashes of light. These flashes are modulated at about 38 to 40 kHz in order to reduce interference from other light sources. The receiver strips out the modulation and provides just the on/off flashing patterns.
- *Computer or microcontroller.* You need some hardware to decode the light patterns, and a computer or microcontroller, running appropriate software, makes the job straightforward. For this project we'll use the PICAXE microcontroller, because it has a very handy built-in command for directly reading the codes sent by Sony remote controls. The same techniques described here can be used with other microcontrollers, but you'll need to adapt the program.

INTERFACING THE RECEIVER/DEMODULATOR

The first order of business is to interface the receiver/demodulator to the PICAXE. Most any receiver module for 38-kHz infrared operation should work well. I've specified the Vishay TSOP4838 because (at least as of this writing) it's widely available and fairly inexpensive. But you can use just about any other infrared receiver/demodulator that operates at 38 kHz. For experimenting, you can use a solderless breadboard to make the connections.



In developing this project I came across some very old infrared receiver-demodulators in my parts bin that only supported 5-volt operation—the modern ones work over a wider voltage range. If you happen to have one of these ancient modules, you can still use it in your project, but be sure to operate your circuit at 5 volts. It won't function, or will behave erratically, at anything under about 4.6 volts.

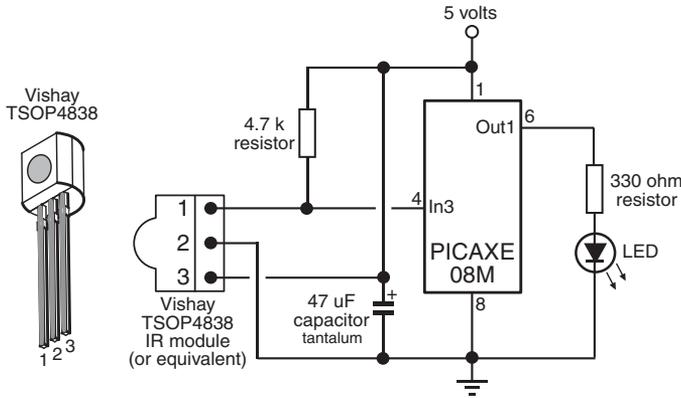


Figure 41-4 Connection diagram between infrared receiver/demodulator and the PICAXE 08M microcontroller. An LED provides visual feedback.

Figure 41-4 shows the simple interface for the infrared receiver-demodulator to a PICAXE 08M microcontroller. You can use any larger PICAXE microcontroller, as long as it supports the *infrain2* command. If you do use another version of the PICAXE chip, remember to change the pins for power, ground, and input, accordingly. Figure 41-4 also shows the pin assignment of the TSOP4838 receiver/demodulator.



The PICAXE supports a number of built-in commands for decoding signals from an infrared remote control. There's also the *infrain* and *irin* commands, supported by various versions of the PICAXE chip. Refer to the PICAXE documentation for the details on these commands.

PROGRAMMING THE PICAXE

Sonyremote.bas is a simple demonstration program for use with the PICAXE, which appears below. Refer to Chapter 38, "Using the PICAXE," for more information on how to use the chip, including how to compile, download, and execute programs. For now, I'll assume you're familiar with all these things and cut right to the chase.

To use the program, be sure to set your universal remote control to output Sony TV infrared codes. The remote control will come with an instruction booklet on how to select the proper code setting. Look under the TV listing for Sony and note which code number(s) to use. You may need to try several of them before you will have positive results.



For these projects I used an RCA-brand model RCU403 universal remote control. These, and their close cousins (like the RCU410, available at RadioShack), are common finds at department stores and online outlets, and they are priced at under \$10 (sometimes a lot less). The exact model you use isn't critical, as long as it supports Sony TVs and VCRs.

I used code 002 for TV, and 099 for VCR/DVD—the VCR/DVD codes provide support for the Play/Rew/FF/etc. shuttle controls at the bottom of the remote. There are other Sony codes supported as well, and using them may result in different behaviors. Experiment.

```
101010
010101
101010
010101
```

sonyremote.bas

```
main:
  infrain2          'wait for new IR signal
  select case infra
```

```

case 1          ' Button 2
  high 1        'switch on output 1
case 4          ' Button 5
  low 1         'switch off output 1
endselect
goto main

```

DETERMINING CONTROL VALUES

In *sonyremote.bas* the values 1 and 4 represent the 2 and 5 buttons. But how did I know that? I used a simple debug program that displayed the values in the PICAXE Debug window:

```

main:
  infrain2
  debug 'open Debug window when run
  goto main

```

Like all programs that use the Debug window, your PICAXE must be connected to your computer via its serial or USB download cable, so that the controller can send back values. The *b13* variable represents the data collected by the *infrain2* command. As you press buttons on the remote, the first column for *b13* in the Debug window shows the numeric equivalent of the *infrain2* value. For example, pressing the On/Off button displays a 21.

Different PICAXE chips support other versions of infrared decoding commands. For example, the PICAXE 18X and 28X chips support *infrain*, which works in a similar manner to *infrain2*, except that it automatically “adjusts” the value of codes returned by the numeric keypad and other keypresses—that is, pressing the “1” key returns a 1. The *infrain* command is now considered obsolete, so use either *infrain2* or, on PICAXE parts that support it, *irin*.

I got the following results for the RCA403, set to VCR/DVD with code 099, testing both *infrain2* and *infrain*:

	infrain2	infrain
Function	Value	Value
Power	21	17
Volume up	18	12
Volume down	19	15
Channel up	16	10
Channel down	17	13
Mute	20	16
1	0	1
2	1	2
3	2	3
4	3	4
5	4	5
6	5	6
7	6	7

Function	infrain2	infrain
	Value	Value
8	7	8
9	8	9
0	9	11
Enter	11	0
Antenna	42	0
Rew	27	15
Play	26	12
FF	28	16
Rec	29*	17*
Pause	25	13
Stop	24	10

* Press twice.

Note: *infrain2* and *irin* share the same code values.

CONTROLLING ROBOT MOTORS

Let's assume you want to drive the traditional two-motor robot, using DC motors. You could use the PICAXE 08M, which has just enough I/O pins for the job. But that doesn't leave any pins for other tasks. Unless you plan on using the 08M just as an infrared signal decoder, you'll want to select a PICAXE chip with more pins.

Figure 41-5 shows a wiring diagram for using the PICAXE 18M2 to steer a robot based on buttons pressed on a universal remote. The 18M2 has more than enough I/O pins for what we want to do, allowing you room for other hardware expansion. The code is *ir-bot.bas*.

By itself, the PICAXE lacks the drive current to power the motors, so a motor driver circuit is used between the chip and the two motors. See Chapter 22, "Using DC Motors," for more information on motor driver circuits. Figure 41-5 shows connecting to the popular L293D motor H-bridge IC, which handles smallish motors up to 600 mA current draw. Note that the circuit specifically uses a separate 6-volt power supply for the motors. This makes things a lot easier all the way around, and the regulated 5-volt supply for the PICAXE can remain small and simple.

Note: Whatever motor drivers you use, make sure that you include 0.1 μ F polyester or ceramic disc capacitors directly across the terminals of the motor. These decoupling capacitors prevent excess electrical noise from appearing on the signal line from the IR receiver-demodulator. As noted in Chapter 21, "Choosing the Right Motor," DC motors—particularly the inexpensive kind—generate copious noise from radio frequency (RF) interference as well as "hash" in the power supply lines of the circuit.

The circuit also shows larger-value capacitors used to condition the power line. These also prevent electrical motor noise from disrupting the received infrared signals. If you notice that you can turn the motors on, but not off, noise is probably the culprit.

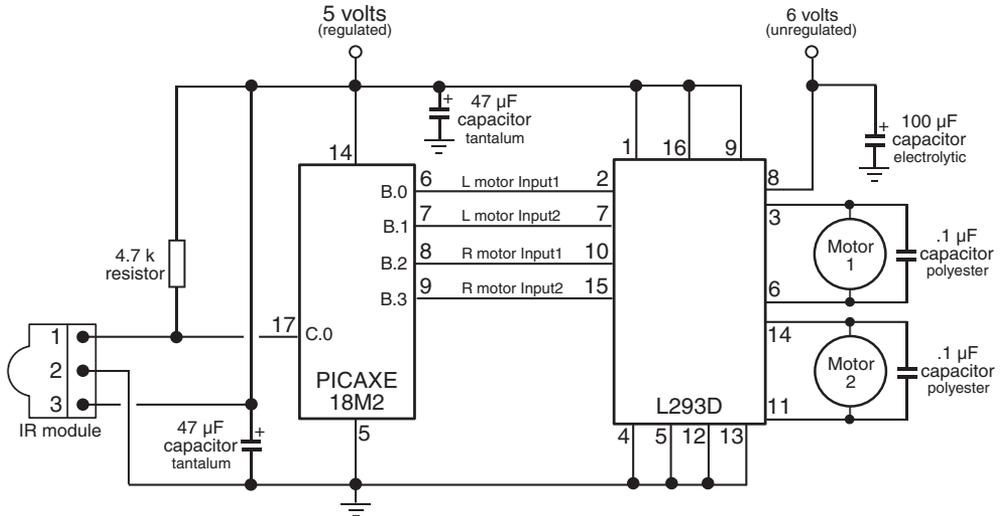


Figure 41-5 Using a PICAXE 18M2 microcontroller to receive signals from a compatible infrared receiver/demodulator and an H-bridge motor driver. Refer to Chapter 22 for details on using motor drivers.

```
101010
010101
101010
010101
```

ir-bot.bas

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A for more details.

OPERATING THE ROBOT WITH THE REMOTE

Now that you have the remote control system working and you're done testing, it's time to play! With *ir-bot.bas* downloaded, disconnect the PICAXE from its programming cable, set your robot on the ground, and apply all power. In the beginning, the robot should not move. Point the remote control at the infrared receiver-demodulator, and press the following buttons to command the robot:

Key	Action	Key	Action
1	Left motor forward	3	Right motor forward
4	Left motor stop	6	Right motor stop
7	Left motor reverse	9	Right motor reverse
2	Left + right motor forward	8	Left + right motor reverse

You may press 5 to stop both motors.

SPECIAL CONSIDERATIONS

Keep the following in mind when experimenting with this project:

- Your robot spins when it should go forward? Reverse the leads to one of the motors to change its polarity. For example, switch the wires to pins 11 and 14 on the L293D.
- The *infrain2* command is sensitive to the operating speed of the PICAXE chip. As noted in the PICAXE documentation, use *infrain2* at the default 4-MHz speed only.
- Press only one button on the remote at a time. If you want to create combinational events, use another button to code those events into your program. For instance, you might use Volume Up/Down or Channel Up/Down to move the robot straight forward and backward.
- Infrared links work best when used indoors, out of direct sunlight and away from bright light sources.
- The better remote controls put out more light power, allowing you to operate your robot from greater distances. Depending on the model of your remote, you may be limited to about 6 to 8 feet away in controlling the bot.
- If your robot doesn't appear to react when you press buttons, or if it behaves erratically, double-check your wiring, or try another Sony product code for the remote. Or try a different remote.

On the Web: Control by Radio Signal

Remotely controlling a robot by way of wires or infrared limits the range to just a few feet. And when operated by wire, the robot's tether might get caught up on things, snagging itself on its own lifeline.

You can extend the range of communication by switching to radio frequency (RF) remote control. Though more expensive to implement, remoting by RF is more flexible; the signals work around corners (infrared control works by line of sight) and, with the right radio units, can even broadcast video pictures back from the robot.

If you're interested in commanding your robot via radio waves, see the RBB Online Support site (check Appendix A for details). There you'll find information on:

- Types of communications: simplex, half-duplex, and full-duplex
- 802.11 WiFi, Bluetooth, and Zigbee communications standards
- Using a data radio with your robot
- Maximizing radio signal range

Broadcasting Video

Telerobotics is the technique of remotely controlling a robot vehicle while it's simultaneously broadcasting video pictures to you. If you've gotten as far as operating your robot via any kind of remote control, adding video to the mix is fairly simple. There are two general approaches:

Feed the video signal back through the data radio. This works if you're using 802.11g (or faster) WiFi, as you need the higher data rates to keep up with the video. This is a



Figure 41-6 Backup camera system, designed to be mounted on the rear license plate holder of a car or van. The camera broadcasts color video to the receiver, which contains its own LCD screen.

doable approach if the video is already in digital format; if you're using an analog camera, you'll need to digitize it before passing it back through the communications link.

You'd use this technique, for example, if your robot were equipped with a laptop or other computer with a USB port; you could capture video using a Web camera that, in turn, sent its signals via radio signal back to you.

Use a separate analog video transmitter. This approach works with any kind of remote control, because it doesn't use the data link for the transmission of video. You use your regular wired, infrared, or radio link, but the video is sent back using its own transmitter.

Wireless cameras are pretty inexpensive, and many are already designed for use with low voltage supplies. Figure 41-6 shows a wireless "car backup camera" that is designed to mount to the license plate holder on the rear of an automobile. The receiver is mounted on the dash and, using a small LCD screen, displays the image broadcast by the camera. Naturally, both are designed to work with a standard 12-volt car system. I bought this particular unit at a closeout sale for about \$20.

When looking for a wireless camera ensemble for the hole, find one powered with external wall transformers. These are the easiest to hack for mobile robotics use. On many low-cost wireless cameras, the receiver is intended to be plugged into a USB port, for viewing on a computer. This is handy if that's how you want to teleoperate your robot.

If it's not, you want a receiver that can plug into any standard video input, so you can view the picture. Some wireless cameras have audio, which is handy. Older wireless surveillance cameras broadcast only in black and white, and the picture is worse than terrible. Stay away from these. Modern wireless cameras provide a reasonably sharp color image with plenty of detail . . . useful when piloting your robot through a maze of living room furniture, curious family pets, and young children bent on destroying anything that moves.

Sensors, Navigation, and Feedback

This page intentionally left blank

Adding the Sense of Touch

A sure way to detect objects is to make physical contact with them. Contact is the most common form of object detection, and it's also the cheapest to implement—often with just an inexpensive switch. Touch sensing can be used on the base of a robot for when it's crisscrossing the living room carpet, or on arms and grippers, to detect when objects have been grasped or even when they're about to be crushed to smithereens.

This chapter deals with touch-sensing systems, whether it's a simple switch on the back of the bot so it knows when it's bumped into something, or artificial "skin" that detects the amount of pressure applied to it.



This chapter deals only with "touch," your robot making actual contact with something. Other principal forms of robotic sensing are proximity detection and distance measurement. These are used to detect objects before your robot bounces into them. Read more about proximity and distance sensing in Chapter 43.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, "RBB Online Support," for more details. To save page space, the lengthier programs are not printed here. The support site also offers source code with added comments, parts lists (with sources) for projects, updates, extended construction plans, and more examples you can try!

Understanding Touch

Touch, also called *tactile feedback*, is a reactive event. The robot determines its environment by making physical contact; this contact is registered through a variety of touch sensors. Most

often, a collision with an object is a cause for alarm, so the reaction of the robot is to stop what it's doing and back away from the condition.

But in other cases, contact can mean the robot has found its home base, or that it's located an enemy bot that is about to pound the living batteries out of it. For the typical robot, touch is reduced to sensing only mechanical pressure.



Because robotic touch is based on pressure, the amount of pressure dictates how sensitive the robot is. A large mechanical switch that requires a great deal of force to actuate will be insensitive to routine contact. In order for the switch to activate, it may require a forceful collision with an object.

Conversely, a lightweight wire that is actuated by a slight sideways pressure may be activated by a gentle nudge. You can tailor your robot's contact sensitivity by altering the type and size of its touch detectors.

Mechanical Switch

The lowly mechanical switch is the most common, and simplest, form of tactile (touch) feedback. Just about any momentary, spring-loaded switch will do (see Figure 42-1). When the robot makes contact, the switch closes, completing a circuit—or in some cases, the switch opens, breaking the circuit. Either way works.

The switch may be directly connected to a motor, or, more commonly, it may be connected to a microcontroller or other electronic circuit. A typical wiring diagram for the switch is shown in Figure 42-1. The pull-down resistor is there to provide a consistent LOW output for the switch when there is no contact. When contact is made, the switch closes, and the output of the switch goes HIGH.

When using a microcontroller, you can determine how the robot reacts to the physical collision by altering its programming. Typically, for a switch used for a touch sensor, the programming instructs the robot to stop, back up, and head in a new direction.

PHYSICAL CONTACT BUMPER SWITCH

You can choose from a wide variety of switch styles when designing contact switches for tactile feedback. A *leaf* or *lever* switch (sometimes referred to as a Microswitch, after a popular

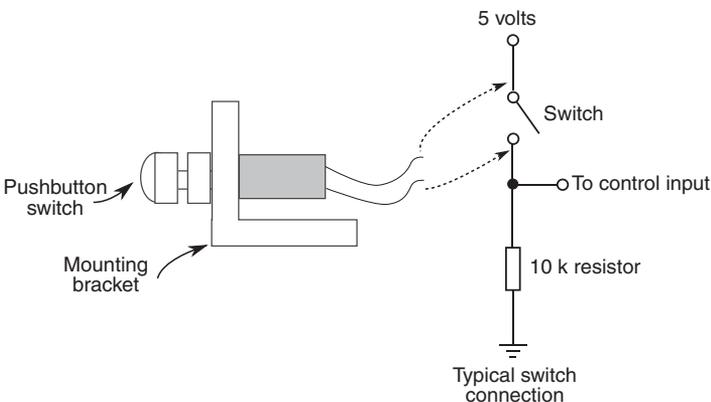


Figure 42-1 Mechanical and electrical connection of a momentary (spring-loaded) push-button switch. The 10 k Ω resistor ensures that the output is LOW until the switch closes.

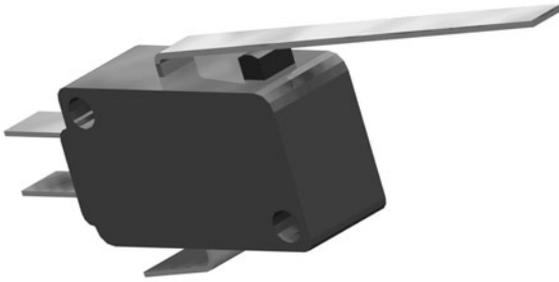
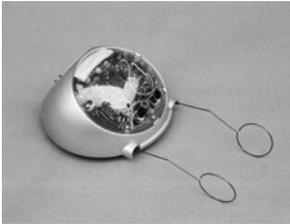


Figure 42-2 Leaf or lever switches make for ideal bumpers for robotics. You can choose from among switches with a long or a short leaf.



brand name) comes with plastic or metal strips of different lengths that enhance the sensitivity of the switch. See Figure 42-2 for an example of a lever switch ideal for use as a contact bumper for a small robot.

Leaf switches require only a small touch before they trigger. The plunger in a leaf switch is extra small and travels only a few fractions of an inch before its contacts close. As the leaf is really a mechanical lever, lengthening it increases sensitivity. But it also increases the distance (called *throw*) that the end of the lever must travel before the switch makes contact.

ENLARGING THE CONTACT AREA OF THE SWITCH

The surface area of most switches is pretty small. You can enlarge the contact area by attaching a metal or plastic plate or a length of wire to the switch plunger. A piece of rigid 1/16"-thick plastic or aluminum is a good choice for bumper plates. Glue the plate onto the plunger.

Low-cost push-button switches are not known for their sensitivity. The robot may have to crash into an object with a fair amount of force before the switch makes positive contact, and for most applications that's obviously not desirable. You're better off spending a little more for higher-quality switches.

For a leaf switch, you can mount a plastic or metal plate to the end of the leaf to increase surface area. If the leaf is wide enough, you can use miniature 4-40 machine screws and nuts to mount the plate in place.

EXTENSION WHISKER

The whiskers of a cat help its brain form a 3D topographical map of the animal's surroundings. At its most simplistic level, the whiskers can be used to measure space. We can apply a similar technique to our robot designs—whether or not kitty whiskers are actually used for this purpose.

You can use thin 20- to 25-gauge piano ("music") wire for the whiskers of the robot. Attach the wires to the ends of switches, or mount them in a receptacle so the wires are supported by a small rubber grommet.

By bending the whiskers, you can extend their usefulness and application. The whiskers in Figure 42-3 make contact with a small (and sensitive) leaf switch. You can cement or tape the wire to the switch, and bend the wire to make a whisker of the size and shape you want. When the switch and whiskers are positioned so they detect vertical motion, they can determine changes in topography to watch for such things as the edge of a table or the corner of a rug.

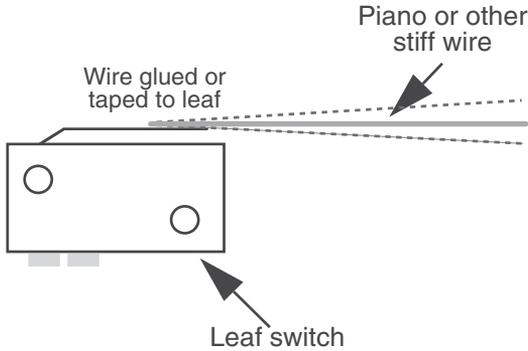


Figure 42-3 Use piano (music) wire or small rigid tubing to extend the length of a leaf switch. Use glue or tape to hold the wire in place.



You want to avoid sharp wires sticking out of your robot. Not only can they snag on objects, they can poke at skin, eyeballs, and other sensitive body parts of humans and animals. Bend the ends of the whiskers or other wires to form a small blunted loop. If the wires are heavy enough, insert a rubber stopper, like that used for knitting needles (also called point protectors). The stoppers come in several sizes.

SPRING WHISKER

You can make your own switches using stiff wire and a spring. One approach is shown in Figure 42-4. Use a long wire and a short spring, both mounted on a perforated circuit board. The two pieces form a switch circuit: when the wire bends, it touches the spring, which signals contact. It's important that the spring is held concentric around the wire. The spring should touch the wire only upon contact. The larger the diameter of the spring, the less sensitive the switch.

You can also reverse the concept: use one narrow spring nested inside a larger spring. For the larger spring you can substitute stiff wire with a wide loop bent at the end. When the spring touches the loop, the switch is closed.

HOMEBREW CONTACT WHISKER

With some stiff music wire, you can build your own contact whiskers and form them into any shape you like. Figure 42-5 shows the basic idea of two wire whiskers for a robot. Both designs are made directly on an experimenter's solderboard (don't use a solderless breadboard).

- In Style A, the wire goes between three male header pins, where the middle pin of the three has been cut off. The whisker makes contact with any side-to-side movement. If the

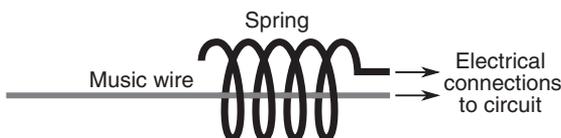


Figure 42-4 Spring-and-wire homemade whisker switch. When the wire makes contact with the spring, the circuit is closed.

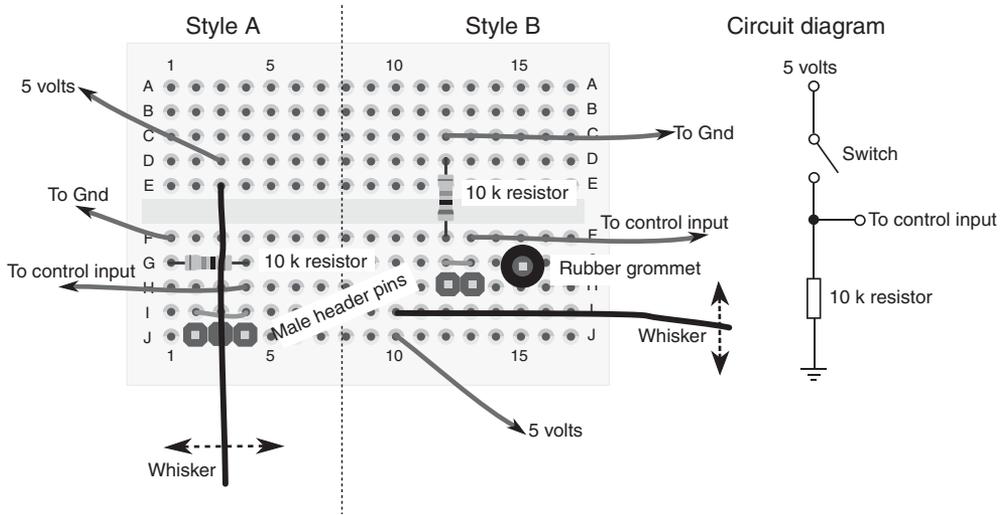


Figure 42-5 Plan view of two ways to implement the electronics of homebrew whisker switches. The one on the left is most sensitive to side-to-side motion; the one on the right, back-and-forth motion. See the text for how to use the male header pins as whisker contacts.

fit between the pins is too small, use instead a set of four male header pins and cut off the middle two.

- In Style B, the wire goes in front of a series of male header pins. I'm also using an additional pin outfitted with a rubber grommet; it acts as a backstop. You can add more pins and connect them all together electrically if you want more contact area.

Both styles show the stiff wire connected directly to the electrical contacts of the solderboard. In order to solder this connection, the wire should be copper or brass; metals like stainless steel are harder to solder. You should apply a soldering iron or pencil with a higher wattage than you'd normally use for electronics. Solder the wire connections first, before adding the other components.

Figure 42-5 shows the typical electrical connection of the whisker to your control electronics. It works the same as any switch, so the working concept is the same. The *control input* is a microcontroller I/O line, a connection to a motor, and so forth.



Avoid damage or injury by adding a small loop at the outside end of the whisker. This prevents the wire from poking into ankles, eyeballs, and antique furniture legs.



MULTIPLE BUMPER SWITCHES

What happens when you have many switches scattered around the periphery of your robot? You could connect the output of each switch to your microcontroller, but that's a waste of I/O pins. A better way to do it is to use a priority encoder, a multiplexer, or a parallel-in, serial-out (PISO) IC.

All three schemes allow you to connect several switches to a common control circuit. The robot's controller gets the switch information from the control circuit instead of from the individual switches.

Using a Priority Encoder

The circuit in Figure 42-6 uses a 74148 family IC (such as the 74HC148) priority encoder IC. Switches are shown at the inputs of the chip. When a switch is closed, its binary equivalent appears at the A-B-C output pins. With a priority encoder, only the switch that represents the most significant bit is indicated at the output (pin 4 is the most significant bit; pin 10 the least significant bit).

In other words, if switches connected to pins 4 and 1 are both closed, the output will reflect only the closure of switch 4, as 4 has a higher priority. Sometimes this is useful information (some bumper switches are more important than others); sometimes it's not. When it's not, use one of the other approaches discussed next.

When using a 74148 chip with fewer than eight switches, be sure to tie any unused inputs HIGH. Connect the switches in descending significant order: 4, 3, 2, 1, 13, 12, 11, 10.



Back up the truck a bit before going on. The 74148 has an interesting feature you might not want to miss. Notice pins 14 and 15. These are "group" outputs, meaning their state changes

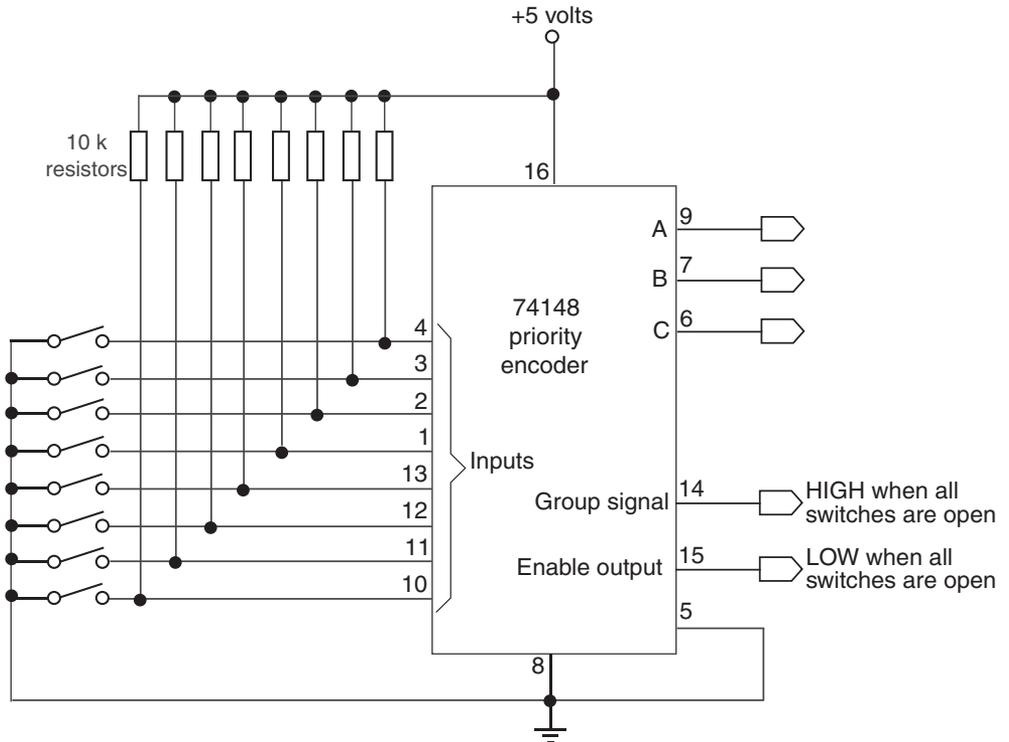


Figure 42-6 A priority encoder IC tells you the value of the most significant bit in a field of 8 bits. The 10 kΩ resistors are used as pull-ups for the switches.

when any of the switches are closed. You can use this feature with a microcontroller that has a hardware interrupt pin (like the Arduino).

If any of the eight switches close, the group output can signal the interrupt. The microcontroller can then check the A-B-C output lines to see which switch is closed (if more than one switch is closed, the most significant switch is indicated). Use this feature when your microcontroller has only a few hardware interrupt pins and your robot has a lot of switches.

There are methods of cascading (daisy-chaining) priority encoders, so you can check 16 or more switches. See the datasheet for the 74148 chip for ideas.

Using a Multiplexer

Another method is shown in Figure 42-7. Here, a 74151 family multiplexer IC (such as the 74HC151) is used as a switch selector. To read whether a switch is set or not, the microcontroller or computer applies a binary weighted number to the three input select pins. The state of the desired input is provided at the Output pin (pin 5). The advantage of the 74151 is that the state of any switch can be read at any time, even if several switches are closed.

Here's the *truth table* for the 74151. The A-B-C input select pins control which of the eight input lines appears at the Output pin.

Input Select			Selected Line
C	B	A	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

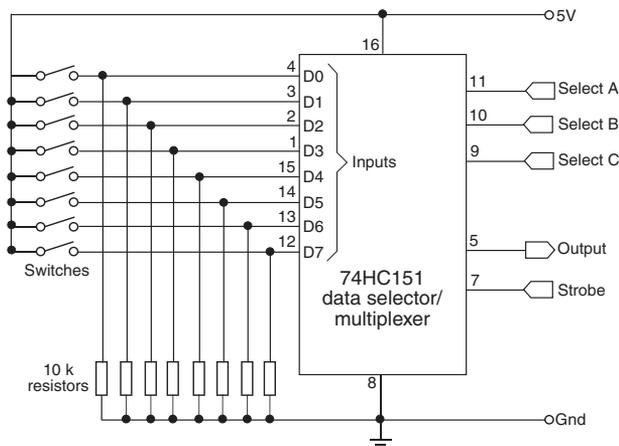


Figure 42-7 The 74151 multiplexer IC will tell you the instantaneous value of any of eight switches. You set the switch to read with the three Input Select pins.

The *switch_mux.pde* sketch, for the Arduino microcontroller, provides a quick example of reading a single 74151 chip, connected to eight bumper switches. Open the Serial Monitor window to see the bit pattern of the switch states: 0 means the switch is open; 1 means it's closed. The 8 bits are ordered left to right from D0 to D7.

```
101010
010101
101010
010101
```

switch_mux.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, "RBB Online Support," for more details.



Okay, so you're probably wondering: why the big deal over saving a measly three I/O pins? It takes five pins to work the 74151 chip, which reads just eight switches.

The secret is that you can use the simple polling technique with multiple '151 ICs. All of the chips can share the same Select and Strobe lines. For each 74151, you connect a separate Output to the microcontroller

Suppose you want to track 24 switches. That takes three 74151 ICs, and a total of seven I/O lines: four for the A-B-C Select and Strobe pins, and three for the Output line of each of the chips. All three '151 multiplexers send back their output at the same time. Each output represents a group of eight switches.

Using a PISO IC

PISO stands for parallel-in, serial-out: you apply a set of values in parallel and the circuit converts it to a simple serial data train. A PISO (also called shift register) chip, like the 74HC165, gives you eight switch inputs, as shown in Figure 42-8. The instantaneous value of any switch is reflected in the serial output.

PISO (and its inverse, the SIPO) shift registers are covered in more detail in Chapter 40, "Interfacing Hardware with Your Microcontroller or Computer." Basically, you connect the *Clock*, *Data*, and *Latch* pins of the PISO to your microcontroller. Program *switch_piso.pde*, for the Arduino, shows how to read the parallel bits on a 74HC165 parallel-in, serial-out shift register IC and display the current values in the Serial Monitor window. Note that while I have specified the HC type for this chip, you can use any in the 74165 family, such as the HCT or LS.

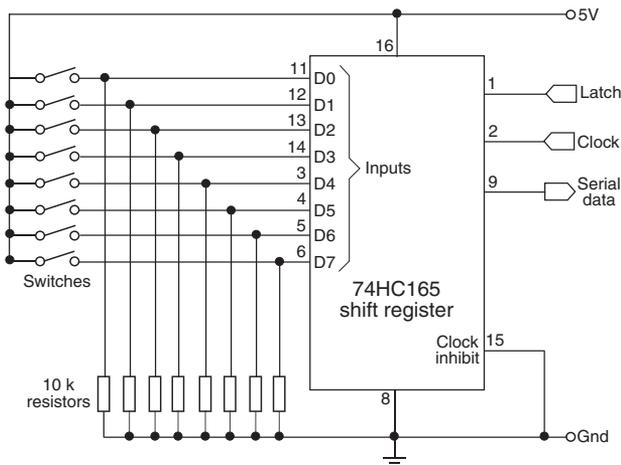


Figure 42-8 The 74HC165 shift register takes the value of eight switches, and provides a single stream of serial data.

- Momentarily set the *latch* pin HIGH to read in the eight switches.
- Set the *latch* pin LOW to begin reading the serial data.
- Pulse the *clock* pin eight times (for eight switches), while reading the LOW and HIGH values coming in over the *data* pin.

The switch settings are shown in binary form in the Serial Monitor window, where 0 means the switch is open and 1 means it's closed. For example,

```
01110001
```

means the number 2, 3, 4, and 8 switches are closed. The bits are in D0 to D7 order.

```
101010
010101
101010
010101
```

switch_piso.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details. You'll also find a version combining two shift register chips, in order to read up to 16 switches at a time.

In the *switch_piso.pde* program, the switch states are stored as 8-bit values in the *dataStore* variable. Use this variable for any additional processing. If *dataStore* > 0, then you know at least one switch is closed. You can use the Arduino *getBit* statement to determine which switch (or switches) is closed, and act upon it accordingly.



The 74165 reads up to eight switches, but you don't always have to use that many. Wire up the switches you want, starting with D0 (pin 11). Don't leave an unused input pin dangling; connect it directly to ground. That keeps the pin from *floating* (giving possibly bogus results), which appears as nonsense data.

Using a Button Debounce Circuit

Bounce is what happens when the contacts in a switch open or close. The contacts don't just immediately open or close once when the switch is pushed. There may be dozens of “tentative” opens and closes each time the switch changes state. The bounces are a kind of electrical noise that can influence the operation of your circuits and programming.

There are numerous ways to remove the extra bounces when a switch opens or closes. They all operate on the principle of stretching the duration of the first switch event that occurs. Most bounces are less than 10 or 20 milliseconds (often much shorter than that, depending on the switch); by stretching out the switch change, all the other bounces that come after are simply missed.

Figure 42-9 shows a common approach to hardware button debouncing. It uses one-sixth of a 74HC14 Schmitt trigger inverter IC, along with a resistor and capacitor to form an *RC timing* network. Note that the 74HC14 contains inverting buffers, meaning that the polarity of the input signal is reversed on the output—LOW becomes HIGH, and HIGH becomes LOW. Remember this when you connect your circuit to your microcontroller. (If you really, really hate this aspect, and you have inverters to spare, connect two in daisy-chain fashion. This method inverts that which was inverted, and you're back to life as usual.)

Schmitt triggers do their magic because their output is either LOW or HIGH. There are never any in-between voltages, which can occur because of the introduction of the capacitor in the switch network. The capacitor changes the switch transitions from sharp cliffs to grad-

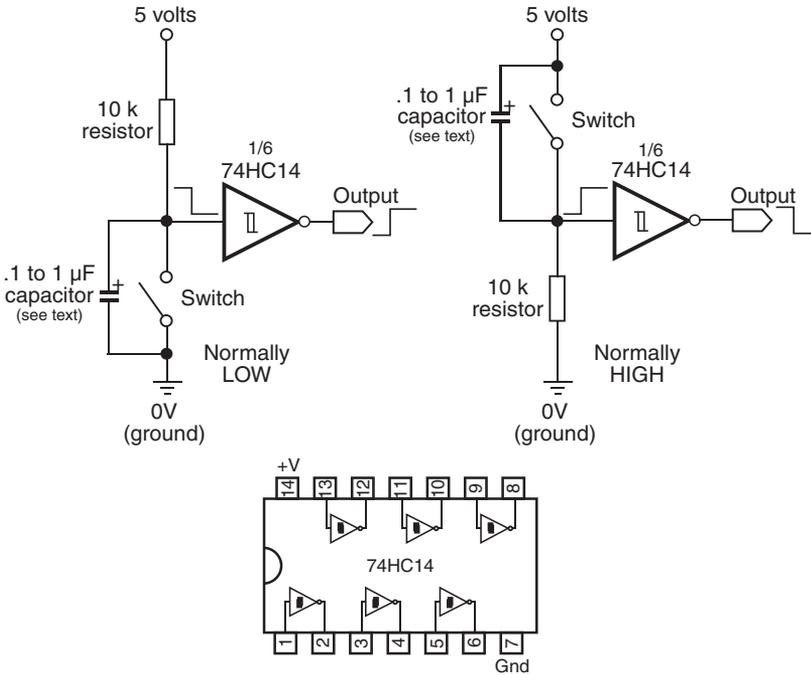


Figure 42-9 Switch debouncer built around a Schmitt inverter buffer. The 7414 IC has six Schmitt inverters in it. Vary the value of the capacitor (from about .1 to 1 μF) for a longer or shorter pulse output. Higher values produce longer delays and are useful with really dirty (electrically speaking) switches.

ual slopes (doing so masks the bounces), and the gradual slopes can cause your microcontroller to confuse what is and isn't a proper LOW or HIGH. The Schmitt trigger eliminates this problem.

You can implement switch debouncing in many different ways. Check out the RBB Online Support site for several additional methods, including using an LM555 timer IC to literally “stretch out” the pulse. This circuit is useful for debouncing, and to make even the most fleeting switch contact last longer. That can be handy when programming, to keep your microcontroller or other circuit from missing the switch action.



Debouncing Switches in Software

Most or all of the functionality of a debouncer circuit can be built in software—good when you're already using a microcontroller or computer for your robot's brain. Like the capacitor, the software technique uses delays to momentarily slow down the program when the first switch transition occurs.

Most microcontrollers have programming statements that provide debounce delays, saving you the hassle of adding the code yourself. The Arduino has available a *Bounce* library that you can download and add to your sketches. The PICAXE and the BASIC Stamp support the *button* statement which includes a parameter for specifying a delay period. Use these whenever you find it necessary to debounce a switch.

Programming for Bumper Contacts

Bumper switches and other forms of touch produce what's known as transitory events: they may not occur for long periods of time, and when they do, they may not last long. When using a microcontroller, you need to program it to watch for these events, so that your robot can take the appropriate action when contact is made.

There are three general approaches for programming a microcontroller for bumper contacts and other events of short duration: preemptive wait, polled, and interrupt.

- *Preemptive waiting* involves using a command of your microcontroller to continuously check the state of a single switch or other transitory input. Because of how preemptive waiting works, the microcontroller is not able to perform other parts of its regular program.
- *Polling* involves periodically checking the state of any switches or other transient event sensors in your robot, while allowing the rest of your program to run. Because most microcontrollers are quite fast compared to the various functions of your robots, this is often a perfectly acceptable approach. As part of the main program code, your robot checks each switch in turn, testing whether the switch has been activated.
- *Interrupts* are handled internally by the microcontroller, and trigger by themselves. Your software need only read the interrupt, telling the microcontroller what you want to have happen when it's triggered. When an event occurs, the interrupt momentarily stops the regular program and performs whatever special task you've set up.

Program *button-press.pde* shows how polling and interrupts are handled on the Arduino. To simplify things, only two switches are used; one is polled, and the other is set up for an interrupt. Two of the digital input pins on the Arduino may be used as hardware interrupts. Depending on the exact model of the Arduino hardware you are using, this is usually pins D2 and D3. To try this program, connect one switch to pin D12 and another to D2.

The program sets up pin D13, which has an integrated LED already on it, as an output. It also attaches an interrupt to watch for any change on pin D2 (known as interrupt 0). Ordinarily, the LED shows the value of the switch on D12. It's off if the switch is open, on if the switch is closed. This is handled by the *poll* routine.

When the switch connected to D2 opens or closes, the microcontroller immediately branches to the *handle_interrupt* routine, which blinks the LED on and off for 1 second. After the *handle_interrupt* routine is finished, the regular program resumes.

This example also demonstrates that when the Arduino is "servicing" the interrupt, the other parts of the code aren't executed. This is by design. Only when the interrupt is finished does program execution pick up where it left off. Normally you wouldn't have such long delays in your interrupt routines.

Also notice I used the *delayMicroseconds* statement, rather than *delay*. Why? The *delay* statement uses some internal interrupts of the Arduino, so it's disabled while in an interrupt handler. If you want a delay, you need to use *delayMicroseconds*. The *for* loop is used to extend the delayed time in 1-millisecond (1000-microsecond) increments.



```
101010
010101
101010
010101
```

button_press.pde

```
const int led = 13;          // Built-in LED
int bumperA = 12;          // Digital pin D12
int bumperB = 0;          // Interrupt 0 (digital pin D2)
```

```
void setup()
{
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  attachInterrupt(bumperB, handle_interrupt, RISING);
}

void loop() {
  poll();
}

void poll() {
  digitalWrite(led, digitalRead(bumperA));
}

void handle_interrupt() {
  digitalWrite(led, HIGH);
  for (int i=0; i <= 1000; i++)
    delayMicroseconds(1000);
  digitalWrite(led, LOW);
  for (int i=0; i <= 1000; i++)
    delayMicroseconds(1000);
}
```



See the RBB Online Support site (refer to Appendix A) for an extended version of this code, with enhanced comments. The extended version is coupled with controlling two motors, to demonstrate a touch-reactive robot.

Mechanical Pressure Sensors

A switch is a go/no-go device that can detect only the presence of an object, not the amount of pressure on it. A pressure-sensitive detector senses the force exerted by the object onto the robot, or vice versa—how hard the robot has crashed into something.

There are a number of pressure-sensitive detectors you can use in your robots. Some you can make, some you can steal from old parts, and yet others are available as specialty sensors, available from numerous online sources.

ANTISTATIC CONDUCTIVE FOAM

You can make your own pressure-sensitive detector (or *transducer*, if you like fancy terms) out of a piece of discarded conductive foam—the stuff used to package static-sensitive integrated circuits. The foam is like a resistor. Attach two pieces of wire to either end of a 1" square hunk and you get a resistance reading on your volt-ohm meter. Press down on the foam and the resistance changes.

The foam comes in many thicknesses and densities. I've had the best luck with the semistiff foam that bounces back to shape quickly after it's squeezed. Very dense foams have a cellular structure that is not as useful because it doesn't quickly spring back to shape. Save the foam from the various ICs you buy and test other types until you find the right stuff for you.

Experimenting with a Basic Pressure-Resistive Pad

Here's how to make a "down-and-dirty" pressure sensor for basic testing purposes. Cut a piece of foam 1/4" wide by 1" long. Attach leads to it using 30-gauge wire-wrapping wire.

Wrap the wire through the foam in several places to ensure a good connection, then apply a dab of solder to the wires to keep everything in place.

Connect the leads of the pad to a multimeter. If the meter is not autoranging, select a midway value of about 200 k Ω . Place the pad on a hard surface like your worktable, and then slowly press down on it with your finger. Watch the resistance value drop. Release your finger and watch it climb back up.

Enhancing the Pressure Pad

Once you've seen how conductive foam behaves, you're ready to make a better sensor by sandwiching several pieces of material together, as depicted in Figure 42-10. The conductive foam is placed between two *very* thin sheets of copper foil, which you can get at most any craft or hobby store. A short piece of 30-AWG wire-wrapping wire is lightly soldered onto the foil; or you can use conductive glue, which you can get from many sources online (just do a Web search).

Mylar plastic, like the kind used to make heavy-duty garbage bags, is glued on the outside of the sensor to provide electrical insulation, though this part is optional. Use a glue stick for the adhesive.

Variations in Readings

The output of the foam transducer changes abruptly when it is pressed in. The output may not return to its original resistance value (see Figure 42-11). So in your control software, you should always reset the transducer just prior to grasping an object.

For example, suppose you're using conductive foam in a robotic gripper. Without any pressure on the foam, the transducer may first register an output of 30 k Ω —the exact value depends on the foam, the dimensions of the piece, and the distance between wire terminals. The software reads this value and uses it as the set point for normal (nongrasping) level to 30 k Ω .

When an object is grasped, the output might drop to 5 k Ω . The difference—25 k Ω —is the amount of pressure. Keep in mind that the resistance value is *relative*, and you must experiment to find out how much pressure is represented by each 1 k Ω of resistance change.

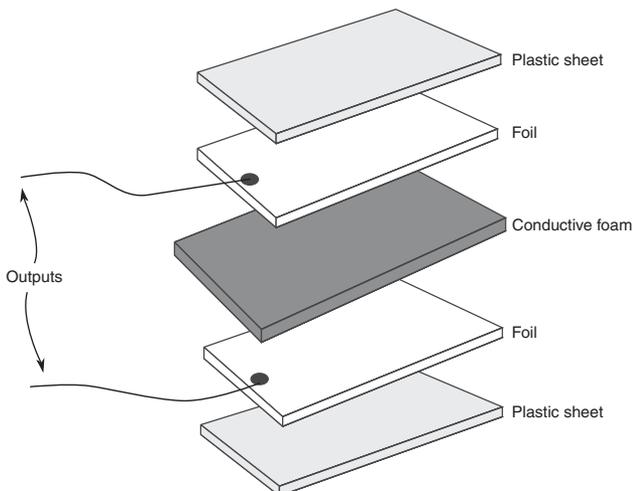


Figure 42-10 Construction of a pressure sensor using an antistatic foam pad. For the foil, use a very thin sheet of brass or copper (available at craft and hobby stores), cut to size. The sheets are relatively easy to solder wires to.

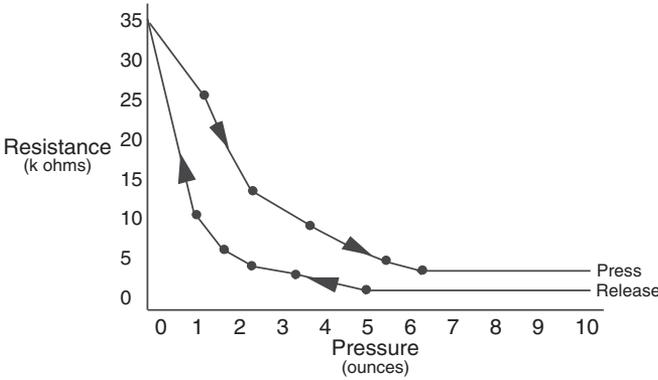


Figure 42-11 The readings from foam pad pressure sensors are not linear, and the values may be different depending on whether you are pressing down or releasing.

Now here’s the kicker: the transducer may not go back to 30 kΩ immediately when the object is released. It may spring up to 40 kΩ or go only as far as 25 kΩ. What’s important is that the software uses this new value as the new set point for the next occasion when the gripper grasps an object.

Connecting to a Microcontroller

Taking resistance readings is handy when you’re using a multimeter. Most microcontrollers (and computers, for that matter) don’t have an input that directly reads resistance. But many do have an analog input that converts a voltage to a digital value. From the digital value, your control program can do something useful with the information provided by the pressure transducer.

As detailed in Chapter 40, “Interfacing Hardware with Your Microcontroller or Computer,” you can readily convert resistance to voltage by adding a series resistor to the circuit. This creates a voltage divider. Figure 42-12 shows how to wire the pressure pad with a 10 kΩ series resistor.

In operation, a voltage will appear at the point where the transducer and resistor meet. Connect that to an analog-to-digital (ADC) microcontroller input, and you’re in business.

 As all pieces of conductive foam are different, you’ll need to experiment with the value of the resistor. Start with the 10 kΩ value, and test for sensitivity. Adjust the value of the resistor up or down to improve the sensitivity, in order to make reading the pressure value easier. Try not to go much below 2.2 kΩ.

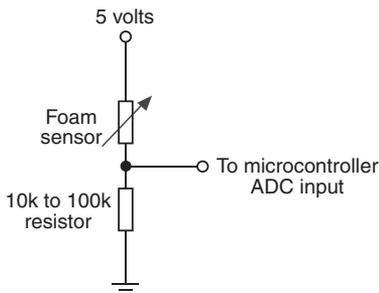


Figure 42-12 Electrical connection of a pressure pad to a microcontroller or other circuit that can read a voltage. You can increase or decrease the sensitivity of the pad by altering the value of the fixed resistor.

ON/OFF PRESSURE PAD

A variation on the theme is the on/off pressure pad. Rather than connect the conductive foam directly to an ADC input, you can route it to an analog comparator circuit—useful if your microcontroller doesn't have an ADC or you don't care about the exact pressure being exerted. The venerable LM339 IC contains four of these comparators in one very inexpensive package. Figure 42-13 shows a wiring diagram.

Experiment with the value of the dividing resistor connected to the foam. For the foam I used, I found good overall response with a 56 k Ω resistor, but you should feel free to try other values. What you want is a good voltage swing (foam pressed and not pressed) without many false readings. The 1- μ F tantalum capacitor in parallel with the dividing resistor is optional. It smooths the voltage output of the foam.

In operation, the conductive foam and series resistor produce a voltage, which is fed into the noninverting input of the comparator. A 10 k Ω potentiometer is used to vary the reference voltage for the comparator. While applying fingertip pressure to the foam, set the potentiometer so that the output of the LM339 just goes from LOW to HIGH. Release the pressure, and the output should return to LOW again. In my experiments, I found that a reference voltage of 2.5 volts was just about right to register a modest push on the foam, with an almost immediate return to LOW when the pressure was released.

From time to time you may need to tweak the potentiometer, in order to set a new reference point. Conductive foam can age (dry out, go brittle), and its resistance can vary over days, months, or years.

FORCE-SENSITIVE RESISTORS

The homebuilt pressure sensors described so far leave a lot to be desired in terms of accuracy and repeatability. If you need greater accuracy, you should consider commercially available force-sensitive resistors, or FSRs. These cost more, but provide a more accurate resistance when a given pressure is exerted on them.

An FSR appears to a circuit like an ordinary resistor, so it can be used in the same kind of voltage divider circuits detailed previously. All the ways of interfacing a voltage divider to a circuit are available to you. See the previous section, and substitute the pressure pad with your force-sensitive resistor. Depending on the FSR, you may want to experiment with a higher or lower value for the bottom (fixed) resistor.

Force-sensitive resistors consist of a main pad that forms the sensing element. The pad comes in various shapes and sizes. For use as a robot fingertip, you can opt for a small, 5- to

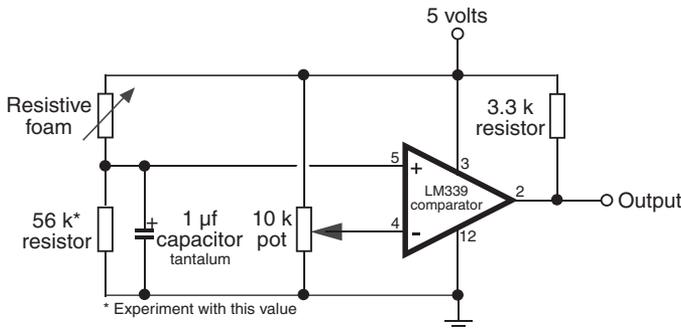


Figure 42-13 An LM339 voltage comparator IC can be used to construct an on/off switch using a pressure pad. Adjust the potentiometer for the threshold between on and off. This circuit works with any kind of resistive sensor.

10mm pad. For use as a robot bumper to detect collisions, you may want to go with a somewhat larger pad.



You can also enlarge the contact area of the pad by using rubber balls cut in half. A 2" round ball has a cross section of 2 inches, yet at the opposite pole it has a contact area of mere millimeters—perfect for a smallish FSR. Let your mind wander and you'll think of plenty of clever homemade designs for enlarging the contact area of a force-sensitive resistor.

FLEX RESISTORS

A variation on the force-sensitive resistor is the flex resistor, similar to a resistive pad but greatly elongated (see Figure 42-14). This makes it more sensitive to the effects of bending. It's ideal as a bumper detector on the front of your robot.

Example: Mount the flex resistor on a very thin piece of plastic (such as 1mm polystyrene from the hobby store), and mount this strip on the front of the bot so that it creates a flexible bow. Push in anywhere along the strip, and the value of the flex resistor changes.

As flex resistors are just variable resistors, use them in circuits like pressure-sensitive anti-static foam, previously discussed.

HOMEBREW SENSORS WITH CONDUCTIVE COATINGS

Conductive inks, paints, and adhesives may be used with rigid and flexible materials to make all manner of pressure, flex, and bend sensors. These coatings are impregnated with conductive materials (usually nickel, copper, or silver, or a combination of these). They may be combined with carbon-based rubber of the kind used in consumer remote control units or electrically conductive fabrics and threads.

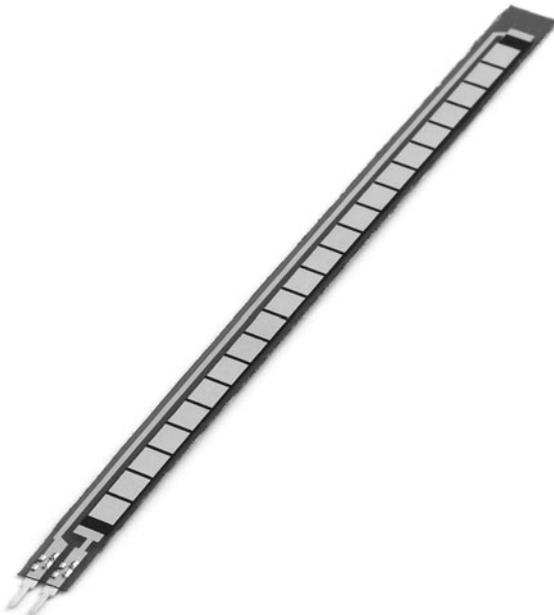


Figure 42-14 This resistive sensor detects when the plastic strip is flexed. Its resistance changes as the strip is twisted and deformed. Use the same kind of interface circuit as in Figure 42-12 to connect this to your microcontroller. (Photo courtesy SparkFun Electronics.)

An example conductive coating is CaiKote 44, a copper/silver which may be applied to glass, plastic, rubber, and many other materials. Use with conductive rubber bands to make stretch sensors (resistance changes as they stretch) or pieces of pressure-sensitive rubber sheet. See the RBB Online Support site (Appendix A for details) on how to find links to these and other conductive materials products.

Experimenting with Piezoelectric Touch Sensors

A new form of electricity was demonstrated more than a century ago when two scientists, Pierre and Jacques Curie, placed a weight on a certain crystal. The strain on the crystal produced an odd form of electricity—significant amounts of it, in fact. This new form of electricity was coined “piezoelectricity”; *piezo* is derived from the Greek word meaning “press.”

Later discoveries demonstrated that piezoelectric crystals undergo a physical transformation when voltage is applied to them. The piezoelectric phenomenon is a two-way street. Press the crystals and out comes a voltage; apply a voltage to the crystals and they respond by flexing and contracting.

HOW PIEZO MATERIALS PRODUCE VOLTAGE

All piezoelectric materials share a common molecular structure, in which all the movable electric dipoles (positive and negative ions) are oriented in one specific direction. Piezoelectricity occurs naturally in crystals that are highly symmetrical—things like quartz, Rochelle salt crystals, and tourmaline. The alignment of electric dipoles in a crystal structure is similar to the alignment of magnetic dipoles in a magnet.

When the piezoelectric material is placed under an electric current, the physical distance between the dipoles changes. This causes the material to contract in one dimension (or *axis*) and expand in the other. This is how piezoelectric buzzers work.

Conversely, placing the piezoelectric material under pressure (in a vise, for example) compresses the dipoles. This causes the material to release an electric charge.

While natural crystals were the first piezoelectric materials used, synthetic materials have been developed that greatly demonstrate the piezo effect. With these, even a small electric charge, applied through wires bonded to a piezoelectric element, causes the piezo material to vibrate at high frequencies.

Piezo activity is not confined to brittle ceramics like those used in piezo buzzers. It can also be made by pressing piezo material onto thin, clear sheets. These sheets are commonly referred to as *PVDF* piezo film—PVDF stands for polyvinylidene fluoride, a kind of plastic.

PVDF film is used in many commercial products, including noninductive guitar pickups, microphones, even solid-state fans for computers and other electrical equipment. For your robot, you can use premade PVDF elements as touch and force sensors, among other components.

EXPERIMENTING WITH PIEZO SENSORS

The ubiquitous ceramic piezo disc is perhaps the easiest form of piezoelectric transducer to experiment with. A sample disc is shown in Figure 42-15. The disc is made of nonferrous (no



Figure 42-15 A homemade touch or “knock” sensor made with a piezoelectric disc, available as stand-alone components from many online electronics stores or in piezo speakers.

iron) metal. A ceramic-based piezo material is applied to one side. Most discs available for purchase are made for use as small speakers or buzzers. They have two leads already attached. The black lead is the “ground” of the disc and is usually directly attached to the metal rim.

When the piezo material of the disc is under pressure—even a slight amount—the disc outputs a voltage proportional to the amount of pressure. This voltage is short-lived: shortly after the initial change in pressure, the voltage output of the disc will return to 0. A negative voltage is created when the pressure is released.

THINGS TO KEEP IN MIND

Whether you are experimenting with ceramic or flexible PVDF film, it’s important to understand a few basic concepts about piezoelectric materials:

- *Piezoelectric materials are voltage sensitive.* What this really means is that the more force you exert on a piezo element, the higher the voltage it will produce. That’s nice, as you can use it to determine relative force of impact. But it also means that you need to protect your electronic circuit by limiting the volts it gets from a piezo element.
- *Piezoelectric materials act as capacitors.* This means they can develop and retain an electrical charge just by sitting there. Ordinarily, not good for a sensor. The easiest way to counter this is to “bleed” the voltage off using a resistor.
- *Piezoelectric materials are bipolar.* Press down, and the material produces (for example) a positive voltage. Release, and the material produces a negative voltage. Remember that negative voltages can harm some kinds of electronic inputs.

BUILDING AN INTERFACE CIRCUIT

A piezoelectric element is both a *consumer* and a *producer* of electricity. When the disc is connected to an input, any physical tap or pressure on the disc will produce a voltage. The exact voltage is proportional to the amount of force exerted on the disc: apply a little pressure or tap, and you get a little voltage. Apply a heavier pressure or tap, and you get a bigger voltage.

The piezoelectric material on ceramic piezo discs is so efficient that even a moderately strong force on the disc will produce in excess of 5 or 10 volts.

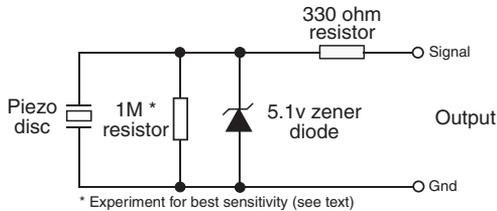


Figure 42-16 Protective interface circuit for use with piezo discs. The zener diode limits the voltage; the resistor limits the current. Your microcontroller or other circuit may already have input diodes, but this provides added protection.

- That's good: it makes it easy to interface the discs to a circuit, since there is usually no need to amplify the signal.
- But it's also bad; the voltage and current from the disc could exceed the maximum inputs of the other electronic device you're interfacing with. (In fact, pound on a piezo disc with a hammer, and, though it might be broken when you're done, it will also produce a thousand volts or more.)

Note the 5.1-volt zener diode shown in Figure 42-16. The diode, and the 330 Ω resistor are included to not only *clamp* the output of the disc to +5.1 volts (a safe level for most interface circuitry), but also limit the current produced by the disc.



In practice, most modern microcontrollers, including the Arduino, already have equivalent protection circuitry built into them, so the zener and resistor may not be required. These components don't alter the operation of the circuit, so add them if you're not sure.

Recall that piezoelectric discs exhibit a capacitance. This means that over time the disc will take a charge, and the charge will show up as a constantly changing voltage at the output of the disc. To prevent this, be sure to insert a resistor across the output of the disc and ground. This resistor bleeds off the excess capacitance.

In my experiments with the specific discs I used, I found that a resistor of about 1 megohm eliminated the charge buildup without excessively diminishing the sensitivity of the disc.

- A higher value will increase sensitivity, but it could cause an extra charge buildup. You probably don't want, or need, to exceed about 5.6 megohms.
- A lower value will reduce the buildup but also reduce the sensitivity of the disc.

Experimenting with Piezo Film

Piezo film, such as *Kynar* and other brands, is available in a variety of shapes and sizes. The wafers, which are about the same thickness as the paper in this book, have two connection points, as illustrated in Figure 42-17. Like ceramic piezo discs, these two connection points are used to activate the film with an electrical signal or to relay pressure on the film as an electrical impulse.

You can use piezo film to sense vibration, shock, touch, and pressure—everything you can do with a piezo disc you can do with the film.

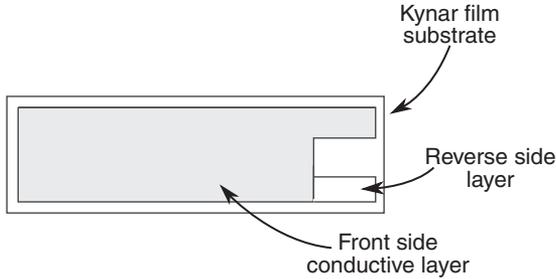


Figure 42-17 Piezo film is a special plastic material coated on either side with conductive material. The film comes in all shapes and sizes.

ATTACHING LEADS TO PIEZO FILM



Unlike piezoelectric ceramic discs, piezo film doesn't always come with preattached leads. Soldering won't work, as that'll just melt the film plastic. For just a little bit more you can get sensors with leads already bonded to the film, and I highly recommend these. Otherwise, you can try one of the following alternative methods:

- *Conductive ink, paint, or glue.* Conductive ink, such as GC Electronics Nickel-Print paint, bonds thin wire leads directly to the contact points on piezo film. Apply a small globule of paint to the contact point, and then slide the end of the wire in place. Depending on drying time, you may need to wait several hours for the ink or glue to set up. Apply a strip of electrical tape to provide physical strength.
- *Self-adhesive copper-foil tape.* You can use copper-foil tape designed for repairing circuit boards to attach wires to piezo film. The tape uses a conductive adhesive and can be applied quickly and simply. As with conductive inks and paints, apply a strip of electrical tape to the joint.
- *Metal hardware.* Use small 2-26 machine screws, washers, and nuts (available at hobby stores) to mechanically attach leads to the film. Poke a small hole in the film, slip the bolt through, add the washer, and wrap the end of a wire around the bolt. Tighten with the nut.

USING PIEZO FILM AS A MECHANICAL TRANSDUCER

Figure 42-18 shows a simple circuit you can build that indicates when a piece of piezo film is struck or bent. Bending the film produces a voltage output, which is provided at the output. Connect this output to a microcontroller ADC (analog-to-digital converter) pin, an LM339 or similar-voltage comparator, or other circuit of your choosing. Note the extra components besides the piezo film:

- The small disc capacitor helps eliminate spurious signals. You can select a different value to better match the piezo film you're connecting to. It really helps to have an oscilloscope to view the actual output of the circuit, but if you don't, try different values to see what works best.
- The 1-megohm (or thereabouts) resistor prevents a charge from developing across the piezo film, causing an uneven voltage output. Experiment with this value—a higher value makes the circuit more sensitive, but also more prone to voltage fluctuations. A lower value does the reverse.

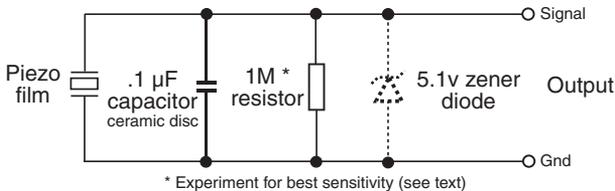


Figure 42-18 Simple circuit setup for experimenting with piezo film.



Figure 42-19 Prototype piezo film bend sensor. Two short pieces of piezo film are taped to a thin plastic sheet (a school report cover); the sheet acts as a backing and is formed into a half-circle to detect when the robot bumps into something.

- The 5.1 zener diode helps to prevent overvolting the circuit you're connected to. It's added as a precaution; many modern microcontrollers and other ICs already have this kind of diode protection on their inputs, and the zener may not be necessary. YMMV (your mileage may vary).



See the RBB Online Support site (refer to Appendix A) for hands-on examples of interfacing piezo discs and film to the Arduino and other microcontrollers. You'll find commented programming code and alternative connection techniques.

CONSTRUCTING A PIEZO FILM BEND SENSOR

You can easily create a workable touch sensor by attaching one or two small piezo film transducers to a thick piece of plastic. The finished prototype sensor is depicted in Figure 42-19. The plastic membrane could be mounted on the front of a robot, to detect touch contact, or even in the palm of the robot's hand. Any flexing of the membrane causes a voltage change at the output of one or both piezo film pieces.

For the plastic backing, I cut up one of those report covers available at office supply stores. And then I told the teacher my robot ate my homework.

A demonstration sketch for the Arduino is simple. This one displays (via the Serial Monitor window) a value from 0 to 1023, depending on how much flex there is in the bend sensor. In practice, and depending on the film you use, the value only goes from 0 to about 200, when using the standard 0-to-5-volt analog-to-digital converter reference (the Arduino lets you set a different reference, but this demo works fine for our purposes). In a working program, you might ignore any values under some minimal threshold (say, 5 or 10) and have your robot react to anything above that as a collision with some object.

```
// Piezo film connected to analog pin A0
const int filmSensor = A0;

void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
  Serial.println(analogRead(filmSensor), DEC);
  delay(500);
}
```



Remember that piezo film produces both positive- and negative-going signals. But the ADC on an Arduino only registers positive voltage change. Depending on how the film is oriented on the plastic, you may get a higher span of readings by reversing the connections of the film to the interface circuit in Figure 42-18.

On the Web: Build a Piezo Bumper Bar

Previous editions of *Robot Builder's Bonanza* included full project plans for a “bumper bar” made with two piezo discs. The completed bumper bar is shown in Figure 42-20. In order to make room in this edition for new projects, I've moved the bumper bar to the RBB Online Support site; see Appendix A for information about the site. Plans for the bumper bar include construction, interfacing to microcontrollers, and sample code.

Also on the RBB Online Support site is a unique project that uses fiber-optic “whiskers” and the magic of laser light, plus other compliant (“soft-touch”) collision detection ideas you might like to try.

Other Types of “Touch” Sensors

The human body has many kinds of “touch receptors” embedded within the skin. Some receptors are sensitive to physical pressure, while others are sensitive to heat. You may wish to endow your robot with some additional touchlike sensors. You can find projects for many of these ideas in the remaining chapters of this book.

- *Heat sensors* can detect changes in the heat of objects within grasp.
- *Air pressure sensors* can be used to detect physical contact. The sensor is connected to a flexible tube or bladder (like a balloon); pressure on the tube causes air to push into or out of the sensor, thereby triggering it.



Figure 42-20 The completed piezo disc bumper bar. See full construction details on the RBB Online Support site.

- *Microphones and other sound transducers* make effective touch sensors. You can use microphones, either standard or ultrasonic, to detect sounds that occur when objects touch.
- *Accelerometers* measure shock and vibration. If your robot hits something, it will cause a shock or vibration that'll be picked up by an accelerometer. Should it stop moving (the object is solid enough), the accelerometer will detect that as well.
- *Touch fabric* is fabric and threads that are electrically conductive. Originally designed for use in making garments, carpets, and other textiles resistant to static electricity buildup, you can use it in any robotic application where you need a flexible sensing area.

Proximity and Distance Sensing

You've spent hundreds of hours designing and building your latest robot creation. It's filled with complex little doodads and precision instrumentation. You bring it into your living room, fire it up, and step back. Promptly, the beautiful new robot smashes into the fireplace and scatters itself over the living room rug. You remembered things like motor speed controls, electronic eyes and ears, even a synthetic voice, but you forgot to provide your robot with the ability to look before it leaps.

Proximity and distance sensing is all about collision avoidance. These systems take many forms, and the most basic are easy to build and use. In this chapter, we present a number of passive and active detection systems you can use in your robots. You'll read about several affordable and easily reproducible sensors you can use for collision avoidance.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, "RBB Online Support," for more details. To save page space, the lengthier programs are not printed here. The support site also offers source code with added comments, parts lists (with sources) for projects, updates, extended construction plans, and more examples you can try!

Design Overview

For a robot to be self-sufficient in the human world, it must be able to determine its environment. It does this by sensing objects, obstacles, and terrain around it. This can include you, the cat, an old sock, the wall, the little hump on the ground between the carpet and the kitchen floor, and a million other things.

Robots sense objects using either contact or noncontact means. With contact sensing, the robot detects a collision after it's happened. With noncontact sensing, the robot is able to

avoid a collision before it happens. Collision detection and collision avoidance are two similar but separate aspects of robot design.

- In Chapter 42, “Adding the Sense of Touch,” you learned about techniques in *collision detection*, which concerns what happens when the robot has already gone too far and contact has been made with whatever foreign object was unlucky enough to be in the machine’s path.
- With *collision avoidance*, the subject of this chapter, the robot uses noncontact techniques to determine the proximity and/or distance of objects around it. It then avoids any objects it detects.

Collision avoidance can be further broken down into two subtypes: near-object detection and far-object detection.

FYI

Robot builders commonly use certain object detection methods to navigate a robot from one spot to the next. Many of these techniques are introduced here because they are relevant to object detection, but for navigation they are developed more fully in Chapter 45, “Navigating Your Robot.”

NEAR-OBJECT DETECTION

Near-object detection does just what its name implies: it senses objects that are close by, from perhaps just a breath away to as much as 8 or 10 feet. These are objects that a robot can consider to be in its immediate environment, objects it may have to deal with, and soon. These objects may be people, animals, furniture, or other robots. By detecting them, your robot can take appropriate action, which is defined by the program you give it.

There are two ways to effect near-object detection: proximity and distance (see Figure 43-1):

- *Proximity* sensors care only that some object is within a zone of relevance. That is, if an object is near enough in the physical scene the robot is looking at, the sensor detects it and triggers the appropriate circuit in the robot. Objects beyond the proximal range of a sensor are effectively ignored because they cannot be detected.

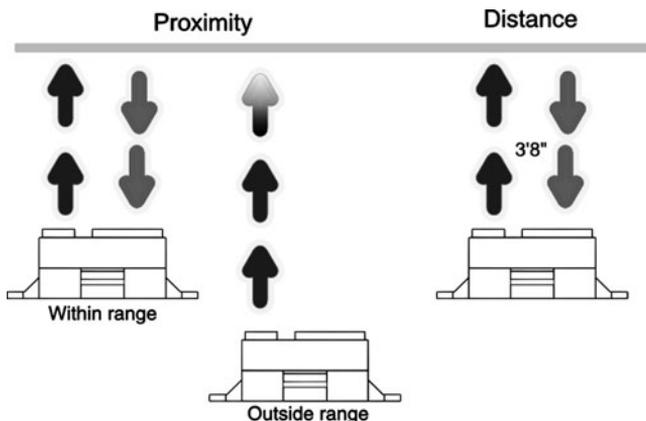


Figure 43-1 Proximity versus distance detection. Proximity provides a go/no-go result, while distance gives the actual closeness of an object.

- *Distance measurement* sensors determine the distance between the sensor and whatever object is within range. Distance measurement techniques vary; almost all have notable minimum and maximum ranges. Few yield accurate data if an object is smack-dab next to the robot. Likewise, objects just outside range can yield inaccurate results. Large objects far away may appear closer than they really are; very close small objects may appear abnormally larger.

Both detection schemes use similar technologies. The most common proximity detection schemes use infrared light or ultrasonic sound. If enough light (or sound) is reflected off the object, and received back to the robot, then an object is within proximity.

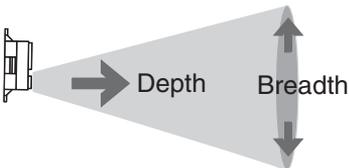
FAR-OBJECT DETECTION

Far-object detection focuses on objects that are outside the robot's primary area of interest, but still within a detection range. A wall 50 feet away is not of critical importance to a robot, but a wall 1 foot away is *very* important.

The difference between near- and far-object detection is relative. As the designer, builder, and master of your robot, you get to decide the threshold between near and far objects. Perhaps your robot is small and travels fairly slowly. In that case, far objects are those 4 to 5 feet away; anything closer is considered "near." With such a robot, you can employ ordinary sonar distance systems for far-object detection, including area mapping.

SENSOR DEPTH AND BREADTH

Sensors have depth and breadth limitations:



Depth is the maximum distance an object can be from the robot and still be detected by the sensor. Except for some ultrasonic sensors, most proximity and distance-measuring detectors for amateur robotics are limited to less than about 6 feet.

Breadth is the height and width of the sensor detection area at any given distance. Some sensors have a very wide "beam" width, covering a large area at a time. Others are much more narrow and focused. There's a place for both.

Sensors that use light can use lenses to focus the beam into a smaller spot. In this way the sensor detects only what's directly in front of it. Ultrasonic sensors use various means to control their beam pattern. Some are very narrow, while others are wide. Makers of ultrasonic sensors for use in robotics provide a picture of the beam spread, so you can match the detector to your planned use.

Simple Infrared Light Proximity Sensor

Avoiding a collision is better than detecting it once it has happened. Short of building some elaborate radar distance measurement system, the ways for providing proximity detection to avoid collisions fall into two categories: light and sound. Let's start with a simple sensor using light.

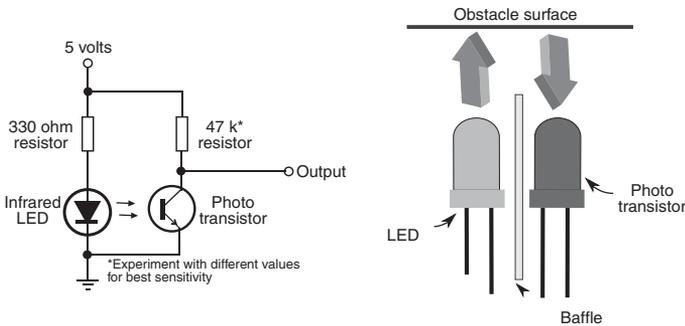


Figure 43-2 This simple proximity detector works by sensing infrared light reflected off an object. The output is a voltage proportional to the brightness of light falling on the phototransistor.

Light may always travel in a straight line, but it bounces off nearly everything. You can use this to your advantage to build an infrared collision detection system. You can mount several infrared “bumper” sensors around the periphery of your robot. They can be linked together to tell the robot that “something is out there,” or they can provide specific details about the outside environment to a computer or control circuit.

A basic (and very simplistic) infrared detector is shown in Figure 43-2, along with a suitable interface circuit. It uses an infrared LED and infrared phototransistor. The output of the transistor can be connected to any number of control circuits, including a microcontroller or voltage comparator. Experiment with the value of the resistor above the phototransistor. Lower resistance values decrease sensitivity; higher values increase sensitivity.

ADJUSTING SENSITIVITY

Sensitivity of the detector can be adjusted by changing the value of the resistor above the phototransistor; reduce the value to increase sensitivity. An increase in sensitivity means that the robot will be able to detect objects farther away. A decrease in sensitivity means that the robot must be fairly close to the object before it is detected.



Objects reflect light in different ways. You’ll probably want to adjust the sensitivity so the robot behaves itself best in a room with white walls. But that sensitivity may not be as great when the robot comes to a dark brown couch or the coal gray suit of your boss.

The infrared phototransistor should be baffled—blocked—from both ambient room light as well as direct light from the LED. Try placing a short piece of black heat-shrink tubing (unshrunk) over either or both the infrared LED or the phototransistor. The positioning of the LED and phototransistor is very important, and you must take care to ensure that the two are properly aligned.

You may wish to mount the LED-phototransistor pair in a small block of wood. Drill holes for the LED and phototransistor. Or, if you prefer, you can buy the detector pair already made up and installed on a circuit board. The one in Figure 43-3 is made by Parallax (see Appendix B, “Internet Parts Sources”).

USING INFRARED DETECTORS ON GRIPPERS

You can mount an infrared detector pair on the tips of the fingers or pinchers of a robot gripper, like that in Figure 43-4. This arrangement is useful for telling the robot that there’s some-

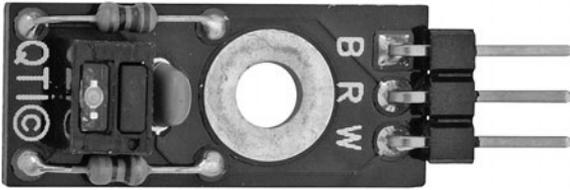


Figure 43-3 Integrated IR emitter and detector modules provide a quick way to add light sensors to your robot. Standard .100" connector pins provide easy hookup. (Photo courtesy Parallax Inc.)

thing nearby to grasp. In a typical gripper design, two or more LEDs and phototransistors are placed along the length of the grippers or fingers to provide more positive control. Alternatively, you may wish to detect when an object is closest to the palm of the gripper. You'd mount the LED and phototransistor accordingly.

Modulated Infrared Proximity Detector

The basic infrared emitter/detector system has a distinct disadvantage—namely, the phototransistor is susceptible to ambient lighting. This kind of sensor tends to work best in a darkened room, where there is little chance of light from a lamp or the sun striking the phototransistor.

Another method is to use a simple technique that's grown much in popularity over the past years: the *modulated infrared proximity detector*, or *IRPD*. These also use infrared emitters and detectors. But to avoid the problems of ambient light spoilage, the system uses a beam of rapidly pulsating, or *modulated*, light. The detector looks only for light that is modulated at the proper frequency, that is flashing on and off at the correct speed. Everything else, it ignores.

Sounds complicated, but in practice it's fairly easy to do, thanks to the wide availability of remote control receiving modules, which form the heart of the IRPD. These are the same modules used in TVs, DVD players, and other devices to receive commands from an infrared remote control.

THE BASIC IRPD CIRCUIT

Figure 43-5 shows a typical IRPD circuit. It has two halves, working in concert:

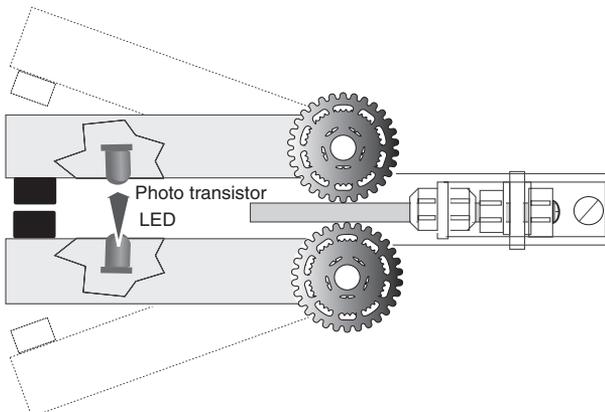


Figure 43-4 Light sensors can be used for more than avoiding contact with obstacles. They can also be used within the robot itself as position or grip indicators. This IR LED and detector can sense when the fingers of a gripper are very close together.

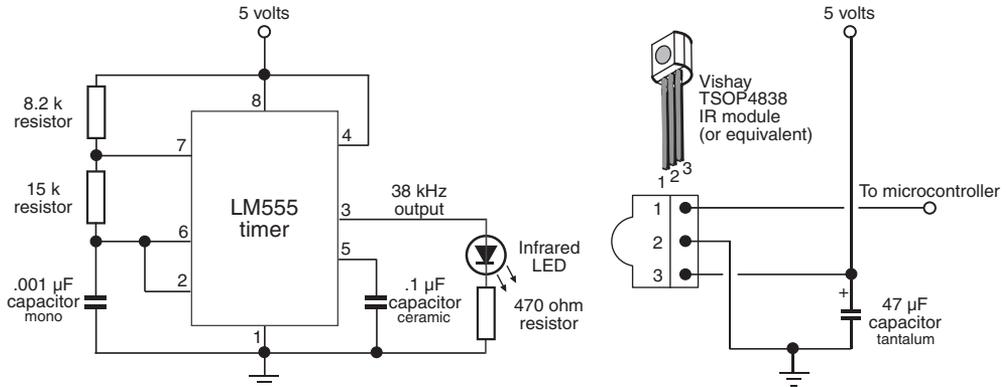


Figure 43-5 Modulated light overcomes many of the problems of using simpler forms of infrared detectors. The sensor is less likely to be influenced by ambient light. This circuit comprises a 555 timer IC providing a steady stream of pulses, that are then picked up by an infrared receiver/demodulator that is turned to the same pulse rate.

- A timing circuit, in this case a low-power LM555 IC, pulses an infrared LED to about 38 kHz (38,000 cycles per second).
- The infrared receiving module is tuned to the same 38-kHz frequency. If there's no 38-kHz light beam around, the output of the module stays HIGH. The moment it detects pulses of 38-kHz light, the output of module goes LOW.

Many IRPDs use two LEDs, which are pulsed alternately. By checking which LED is pulsing at the time a signal is received through the module, the robot can detect if the object is to the left, to the right, or straight ahead. We'll look at this technique in a bit.

The frequency doesn't have to be *exactly* 38 kHz, though the closer it is to this value, the more sensitive the detector will be. There is some latitude in the accepted frequency range of the infrared receiver. With the components shown, and assuming exact values, the frequency of the timer is 37.8 kHz, which is certainly close enough.

In my prototype the measured frequency was actually 39.2 kHz, and the circuit still worked. The variance in measured frequency is due to normal component tolerance. I used resistors with 5 percent tolerance, and the .001 µF monolithic capacitor was rated at 10 percent.

If you wish to experiment with the frequency setting, replace the 15 kΩ resistor with a combination 12 kΩ fixed resistor and a 5 kΩ potentiometer. Wire the pot so that one of its legs is connected to the 8.2 kΩ resistor, and the wiper is connected to pin 6 of the LM555 IC. From one extreme of the pot to the other, the frequency range should span from about 34.2 kHz to 44.8 kHz.

Detection range can be from inches to well over a foot, depending on the exact frequency of the timer and the power output of the infrared LED. You probably don't want a supersensitive detector. Alter the value of the 470 Ω dropping resistor connected to the infrared LED. Higher values weaken the strength of the light, which reduces detection range; lower values increases range. Don't go below about 200 Ω, or you risk smoking the LED by passing it too much current.

For best results:

Point the LED and emitter away from your solderless breadboard or circuit board. Otherwise you'll get false readings due to reflected light.

Increase directionality (and decrease false readings) by inserting a 3/4" length of black heat-shrink tubing over the emitter LED. The longer the tube, the more discriminating the sensor will be.

Add a baffle between the IR emitter LED and the receiver module. Try a small piece of that black conductive foam they use to hold static-sensitive ICs and other components.

Try different infrared LEDs to experiment with range. Small IR emitters with 2- to 10-mW output are useful for "close in" detection of just a few inches. Check the data-sheet for the LED to determine its power output.

Keep lead lengths trimmed (especially the capacitors) or else you could get erratic results. Once you get the circuit tested and working, transfer it from your solderless breadboard to a soldered board.

"Detune" the frequency to alter the effective range of the detector. Try a frequency that's off by 1 to 4 kHz from 38 kHz to make your detector a little less sensitive.

CONNECTING TO A MICROCONTROLLER

Interfacing the IRPD to a microcontroller is straightforward. You can generate the pulses for the infrared detector using a separate circuit, such as an LM555 timer as shown in Figure 43-5, do it in your microcontroller, or use a separate microcontroller.

Using an LM555 timer IC or a separate microcontroller to generate the modulation allows your main controller more freedom to do other, more important things.

LM555-Based Timer

Refer again to Figure 43-5. The LM555 timer is left free-running, generating a steady stream of (approximately) 38-kHz pulses. If the detector module is close enough to receive the pulses, its output goes LOW, which is registered in software running in the microcontroller.

The sketch *irpd.pde* for the Arduino demonstrates reading the IR sensor and lighting the built-in LED whenever the receiver detects a stream of 38-kHz pulses. The LED stays lit for a quarter of a second and turns back off if the pulse stream is no longer detected.

In my tests, using the component values shown in Figure 43-5 and a high-output IR LED rated at 16 mW (RadioShack, catalog #276-143), I got a detection distance of well over a foot.

```

101010  irpd.pde
010101  const int led = 13;                               // Built-in LED pin
101010  const int receiver = 12;                       // Connect receiver to pin D12
010101
void setup() {
  pinMode(led, OUTPUT);
  pinMode(receiver, INPUT);
}

void loop(){
  // read the receiver, LED on if detection
  if(digitalRead(receiver) == LOW) {
    digitalWrite(led, HIGH);
  }
}

```

```

    delay(250);
  }
  digitalWrite(led, LOW);
}

```

Auxiliary Microcontroller

Most infrared modules prefer that that modulation not be continuous. What it likes best is that the pulses turn on and off once every millisecond or so. This improves operating range. The on/off sequence works with the detector's internal autogain circuitry; it's how the detector adjusts to the ambient light of the room.

With the low cost of many microcontrollers, you can implement a full timing circuit that mimics the LM555 and also toggles the modulation on for 50 milliseconds and off for 20 milliseconds. The controller need not have lots of pins, features, or memory, as producing timed signals is a simple task. One good candidate for this task is the PICAXE 08M, costing just a few dollars. It's both easy to program and simple to interface with other electronics.

Figure 43-6 shows the basic hookup. The PICAXE 08M is connected to an infrared emitter by way of a current-limiting resistor. Program code running in the PICAXE controls the modulation frequency and the on/off cycling. The sample program is shown in *irpd-picaxe-simple.bas*.



This example shows the PICAXE 08M generating the light modulation, with some other microcontroller or other circuit solution to detect proximity. With just a little bit of extra code you can develop a self-contained IRPD system where the PICAXE provides the modulation and monitors the detector. See the next section, "Enhanced IRPD Circuit."

```

101010
010101
101010
010101

```

irpd-picaxe-simple.bas

```

main:
  high IR           ' Toggle output high for 5 ms
  pause 5
  low IR
  pause 1
  pwmout IR, 25, 52 ' Approx.38.4 kHz for 50 ms
  pause 50
  pwmout IR, 0, 0   ' Turn off PWM
  pause 20          ' 20 ms "quiet" time
  goto main        ' loop

```

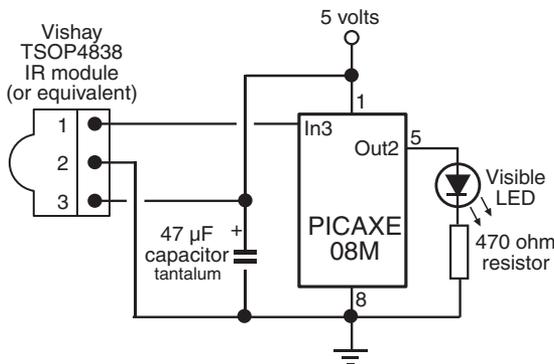


Figure 43-6 A microcontroller may be used to generate the modulated light pulses for use with an infrared receiver/demodulator.

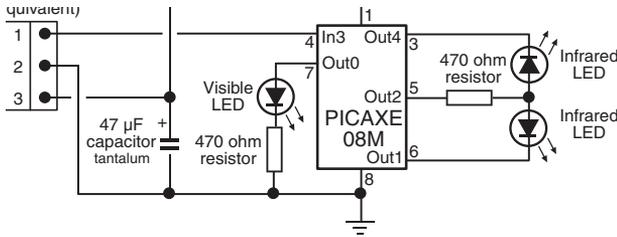


Figure 43-7 Enhanced version of the light modulator based on the PICAXE 08M microcontroller. This one alternatively pulses two infrared LEDs; the receiver/demodulator is placed between the LEDs. Another microcontroller (your design) checks to see which LED was active if/when the receiver was triggered.

ENHANCED IRPD CIRCUIT

As noted, by using a pair of infrared emitters, each producing a short burst of modulated light, your robot can determine whether an object is to the right, to the left, or straight ahead. While you can produce such a sequence using a couple of LM555 timers (or its LM556 dual-timer cousin), it's easier and perhaps even more cost effective to use a microcontroller. Once again, a good job for the PICAXE 08M.

The wiring diagram for the enhanced IRPD is shown in Figure 43-7. Two infrared emitters are used (you still use one detector positioned between them). The distance between the emitters and the detector is variable; a few inches is typical, but you can try 6" to 8", and angle the front of the emitters slightly inward to help pinpoint the infrared light so that the emitter adequately sees it. Feel free to play!



By placing the detector slightly ahead of the emitters, you can reduce problems of "crosstalk," where light from the emitters goes straight into the infrared detector. You can also try adding a heavy black paper or plastic baffle along the sides of the emitter, or outfitting the front of it with a small length of black heat-shrink tubing to keep stray light from the emitters.

Program code for the enhanced IRPD is found in *irpd-picaxe-enhanced.bas*. Two infrared LED emitters are connected to *pwm2* (physical pin 5 of the PICAXE 08M) through one current-limiting resistor—this is okay because only one LED will ever be turned on at any time. Each LED is then toggled on and off via program code: the program statement *low* turns it on, and *high* turns it off.

```

101010 irpd-picaxe-enhanced.bas
010101 symbol IR_L = 1           ' Left
101010 symbol IR_R = 4       ' Right
010101 symbol LED = 0        ' Results LED
                                ' Approx. 38.4 kHz
pwmout 2, 25, 52

Main:
  gosub ToggleL
  gosub ToggleR
  goto Main

ToggleL:                          ' Toggle right on/off
  low IR_L
  pause 50
  if pin3=0 then gosub f_flash
  high IR_L

```

```

    pause 20
    return

ToggleR:                                ' Toggle right on/off
    low IR_R
    pause 50
    if pin3=0 then gosub s_flash
    high IR_R
    pause 20
    return

' Routines for displaying which side (left or right
' is currently being triggered

f_flash:                                  ' Fast flash display LED
    for b0 = 1 to 4
        high LED
        pause 30
        low LED
        pause 30
    next b0
    return

s_flash:                                  ' Slow flash display LED
    for b0 = 1 to 4
        high LED
        pause 75
        low LED
        pause 75
    next b0
    return

```

The visible LED connected to pin 0 is a visual indicator that shows which side—if any—is currently triggering the sensor.

Visual Indicator LED	Meaning
Off	Nothing detected left or right
Fast flashing	Object detected to the left
Slow flashing	Object detected to the right
Fast and slow flashing	Object detected ahead

The LED flashing code is for your own amusement. In a working robot you'd probably want to connect the output of the PICAXE-as-proximity-detector to the main controller, and indicate left/right/center by some other means. One simple way might be:

- Output LOW: no detection
- Output HIGH: left detection
- Output rapid flash: right detection

Or you can use a serial communications link to another controller. The PICAXE 08M supports several serial commands, including *sertxd*, which just happens to use pin0, the same pin the LED is connected to (so you don't have to do any rewiring). Set up a link to your other

processor and send simple codes to indicate proximity detection. Purely as a demonstration, the following sends explanatory text, plus ASCII codes to insert a new line.

```
ToggleL:
  ' activate left IR LED
  if pin3=0 then
    sertextd("Something is to the left", 13, 10)
  end if
  ' deactivate left IR LED . . .
```

Instead of text, you can use code values that only you and your robot's microcontrollers know—maybe 0 for no detection, 1 for left, 2 for right, and 3 for both.

If you're using a PICAXE chip with additional I/O to spare, you can use two pins to denote four possible states. In the following table the pins are marked *Pin A* and *Pin B*; the actual pins are the two outputs you have selected to use for your interface.

Pin A	Pin B	Meaning
LOW	LOW	No detection
LOW	HIGH	Left detection
HIGH	LOW	Right detection
HIGH	HIGH	Right + left detection

Infrared Distance Measurement

Not only can infrared light be used to detect if something is nearby, it can measure the distance between your bot and some object. Infrared distance measurement detectors use the displacement of reflected light across a linear sensor (see Figure 43-8).

Here's how the technique works: A beam of infrared light from the sensor illuminates some object. The beam reflects off the object and bounces back into the sensor. The reflected beam is focused onto what's known as a *position-sensitive device*, or PSD. The PSD has a surface whose resistance changes depending on where light strikes it. As the distance between sensor and object changes, so does the linear position of the light falling on the PSD. Circuitry in the sensor monitors the resistance of the PSD element and calculates the distance based on this resistance.

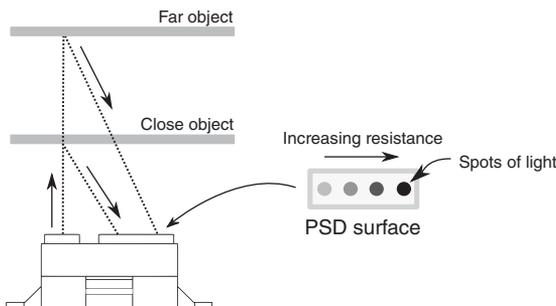


Figure 43-8 The inner workings of the Sharp infrared distance and proximity sensors. Light from an emitter bounces off an object and reenters the sensor at some angle, as shown. The reflected beam strikes against a position-sensitive detector. This sensor detects when light falling on it is off center.



Figure 43-9 Example Sharp IR sensor. This variation of the sensor has convenient screw eyelets for easy mounting.

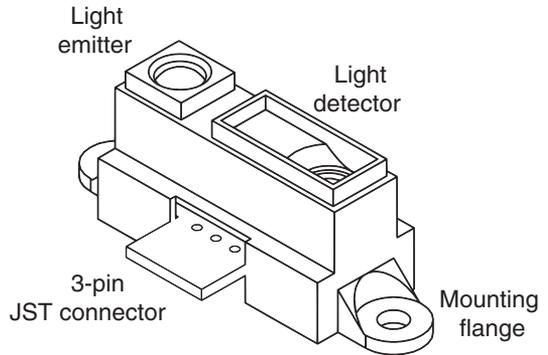


Figure 43-10 The important parts of a Sharp IR sensor. Note the three-pin JST connector. This is smaller than the typical .100" connector you may be used to, and it requires a special cable.

Nearly all of the infrared distance-measuring modules you'll find are made by Sharp. Figure 43-9 shows a typical Sharp infrared proximity sensor, and Figure 43-10 an outline view of the important parts on it. All share better-than-average immunity to ambient light levels, so you can use them under a variety of lighting conditions (except very, very bright light outdoors).

The sensors use a modulated—as opposed to a continuous—infrared beam that helps reject false triggering. It also makes the system accurate even if the detected object absorbs or scatters infrared light, such as heavy curtains or dark-colored fabrics.



Most of the Sharp IR modules use a miniature JST-type three-prong connector. At a distance it looks like the typical 0.100" connector, but it's smaller. When buying an infrared module, be sure to get the right connector with it.

The circuitry of the sensor can provide either a digital or an analog output. Both are routinely used in robotics, so we'll cover them both.

EXPLORING THE DIFFERENT TYPES OF SENSORS

Infrared distance sensors aren't made with robotics in mind—it's just that they're perfect for the job. Rather, they are intended for use in industrial control. Because of this, there are several models to choose from, each with a unique set of features.

- *Working distance* is the effective minimum and maximum distance of the sensor. "Effective" means the distance at which you get accurate detection. Distances are almost always noted in centimeters. The minimum working distance of the most popular Sharp sensor model is 10 cm, or roughly 4". Maximum working distance is 80 cm (31.5"). Outside this range, the sensor may still pick up an object, but measurement accuracy may be affected.
- *Beam width* (or spread) relates to how much of an area the sensor "sees." Most IR modules tend to have a fairly narrow beam width, often just 5° to 10°. This means that the object needs to be almost straight on to the sensor to be detected. Some versions are designed to have a wider field, and some a narrow field of view. Pick the one you want based on your application.

- *Digital go/no-go output* sensors are the simplest to use. Their output is either LOW or HIGH, depending on whether an object is within a set proximity range. These are called *distance judgment* sensors, because they merely judge that the distance is within a certain range.
- *Analog measurement output* provides a varying voltage representing distance. The distance to voltage is not a linear (straight path) relationship, which makes very accurate readings a challenge. These are called distance measurement sensors, as they tell you the distance between the module and the object.

I'll talk about two of the more popular Sharp IR modules here, but recognize that there are others available from your favorite online robotics specialty retailer (model numbers can come and go, so always look for the newest stuff). They work pretty much along the same lines, except they have different working distances and beam widths.

- *GP2D15*—LOW/HIGH digital output indicates whether an object is within 24 cm, or about 9.5". It's very simple to use
- *GP2D12*—Analog output indicates range as a voltage level.



Sharp's datasheets for their infrared modules have been known to contain errors, such as saying that a module has a digital output when it's really analog. These errors eventually get corrected in updates. Be sure to get the latest version of the datasheets, ideally from the Sharp Web site itself.

BASIC ELECTRICAL HOOKUP

Many of the Sharp IR modules follow a standard connection scheme, shown in Figure 43-11.

- Pin 1—Signal output. This terminal may provide a digital or an analog signal, as described in the previous section.
- Pin 2—Power supply ground.
- Pin 3—Power supply voltage, which is 4.5 to 5.5 volts (most units).



Several sources, such as Lynxmotion, offer adapter cables that go from the specially JST connector to a standard 0.100" connector. On many of these adapter cables *the pin order is altered*, so be mindful of this. Follow the color coding of the wires to keep track of what goes where.

Why do they flip the pins? They're following good practice by placing the +V connection in the center. That way, if the cable is plugged in backward, there is less chance of wrecking the sensor. It also standardizes on the same connection scheme used by radio control servo motors.

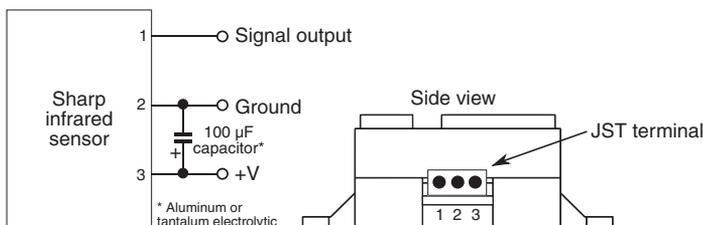


Figure 43-11 Typical connection diagram for the Sharp IR sensors. Not all follow this hookup, but most do. The output may be a digital (on/off) signal or an analog voltage.

Note that some earlier versions of the Sharp sensors, such as the GP2D02, used a different pin configuration. Several other compact models, not covered here, use a six-pin connection and require external components for proper operation.

USING THE GP2D15 INFRARED DISTANCE JUDGMENT SENSOR

The GP2D15 is a “distance judgment” sensor rather than a ranging sensor. It has a 1-bit output that is either HIGH or LOW depending on whether an object has been detected within a threshold range, which is factory-set at 25 cm. This simple connection makes the GP2D15 the easiest to use, as long as you only need to know there’s an object closer than 9-1/2”. Use it as you would a mechanical bumper switch.

The GP2D15 replaces an earlier version, the GP2D05, where you could adjust the threshold range. I’ve seen some instructions on the Web for hacking the range of the GP2D15, but unless you’re real good at it, I’d advise against this. It’s easy to ruin one of these modules by taking it apart.

You’re better off using an analog output sensor, like the GP2D12 (see next section), and connecting it to an LM339 comparator, like that shown in Figure 43-12. Don’t forget the pull-up resistor on the output of the comparator. Dialing the potentiometer alters the reference voltage to the comparator.

USING THE GP2D12 ANALOG OUTPUT INFRARED RANGING SENSOR

The GP2D12 analog output infrared sensor is one of the more commonly used of the Sharp units (along with its close sibling, the GP2D120, which has different optics for a shorter range). Its output is an analog voltage that follows the distance between module and object. Starting from the minimum working distance, the voltage goes down as the distance to the object increases; the lowest voltage is present at the maximum working distance, and beyond.

The voltage span of the GP2D12 is approximately 0.4 to 2.4 volts; this can vary minutely from unit to unit. The ideal way to measure this voltage is with a microcontroller equipped with at least one ADC input. A sample program using the Arduino is shown in *gp2d12.pde*.

Remember that the voltage output of the GP2D12 is not linear (see Figure 43-13), which means that you can’t expect a 1:1 ratio between the value you get and the distance separating the sensor and the detected object. While you could write a math function that attempts to linearize the curved analog response of the GP2D12, many people use a short lookup table or *switch* statement that correlates voltage to approximate distance. Conduct tests with objects placed at set distances from the sensor (use a tape measure for accuracy), and use those as benchmarks.

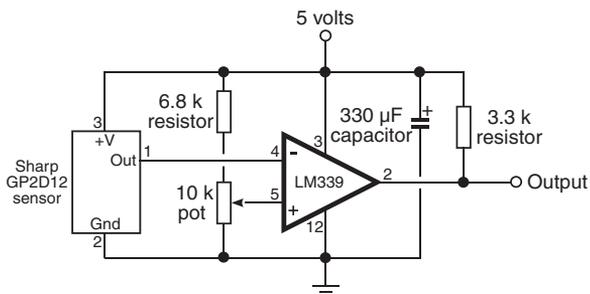


Figure 43-12 How to use an analog output sensor with a voltage comparator. The comparator triggers when the voltage from the sensor reaches a certain point, as set by the 10 k Ω potentiometer.

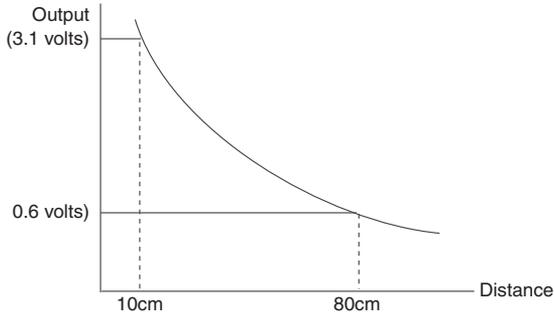


Figure 43-13 The analog voltage output of the Sharp IR sensors is not linear.

The accuracy of the readings will depend greatly on the width of the target. You may wish to experiment by placing the sensor in front of a smooth, white wall. Vary the distance between wall and sensor, and note your results.

Personally, I think it's better to use an ultrasonic sensor when accurate distance measurement is needed, and use IR sensors for ballpark approximations—instead of worrying about exact fractions of an inch, you're more interested in knowing whether an object is far, near, or really close.

Note that when an object is very close—less than the minimum detection range—the value returned by the sensor is meaningless. Depending on the exact sensor you use, it'll show the same value as no detection at all. So be sure to back up an IR sensor with, at the very least, a mechanical contact switch to detect physical collision.



```
101010
010101
101010
010101
```

gp2d12.pde

```
int distance = 0;
int averaging = 0;

void setup() {
  Serial.begin(9600);           // Use Serial Monitor window
}

void loop() {
  // Get a sampling of 10 readings from sensor
  for (int i=0; i <= 5; i++) {
    distance = analogRead(A0);
    averaging = averaging + distance;
    delay(55);
  }
  // Average out the 5 readings
  distance = averaging / 5;
  averaging = 0;
  Serial.println(distance, DEC); // Display result
  delay(250);                   // Short delay
}
```

SHARP IR MODULE GOOD CODING PRACTICE

Over the years, robot experimenters have discovered many useful tips and tidbits about how to best use the Sharp infrared distance measurement and distance judgment sensors. Here are several of the most important issues to keep in mind:

- The Sharp modules tend to draw a lot of current when operating—up to 50 mA, depending on model and the instantaneous measured distance. Keep this in mind when planning the power and battery requirements for your robot.
- Though not an absolute requirement, adding a 100 μF bypass capacitor between +V and ground of the sensor helps to reduce errant readings due to induced spikes in the power supply. You may also want to add a secondary .1 μF monolithic capacitor in the same way.
- After turning on a module, wait at least 100 ms (milliseconds) before taking a reading. This allows the device to settle; otherwise, the reading may be inaccurate.
- For greater accuracy, take several successive readings from the sensor, at no less than 50-ms intervals, and average them out.
- In addition to averaging, you may wish to disregard any readings within a group that are wildly different from the rest. For example, if you get readings of 100, 105, 345, 97, and 101, all within 55 ms of one another, you can be fairly sure the 345 result is spurious and should not be considered.
- Accuracy is somewhat diminished when using an infrared sensor on a fast-moving robot, or when attached to a rotating sensor turret. Movement affects the reading. During travel or motion, you can use the measurements for general proximity detection, but you may wish to slow down or stop the robot (or turret) to get a more accurate distance reading.
- The width of the infrared beam increases with distance. Use this to your advantage when placing multiple sensors on your robot. To maximize beam coverage of two sensors, point them so they cross. To minimize beam coverage (make the sensors more selective), point them away from one another.

On the Web: Passive Infrared Detection

Passive infrared sensors detect the proximity of humans and animals. These systems, popular in both indoor and outdoor security systems, work by detecting the change in infrared thermal heat patterns in front of a sensor. This sensor uses a pair of *pyroelectric* elements that react to changes in temperature. Instantaneous differences in the output of the two elements are detected as movement, especially movement by a heat-bearing object, such as a human.

Check out more about passive infrared detection on the RBB Online Support site (see Appendix A), where you'll find several projects using pyroelectric sensors to detect movement and objects.

Ultrasonic Distance Measurement

A police radar system works by sending out a high-frequency radio beam that is reflected off nearby objects—maybe your car as you are speeding down the road. The difference between the time the transmit pulse is sent and when the echo is received denotes distance.

Radar systems are complex and expensive, and most require certification by a government authority, such as the Federal Communications Commission for devices used in the United States. Fortunately, there's another approach you can use with robots: high-frequency sound.

Ultrasonic distance measurement—also called *ultrasonic ranging*—is now an old science. Polaroid used it for years as an automatic focusing aid on their instant cameras. To measure distance, a short burst of ultrasonic sound is sent out through a *transducer*; in this case, the transducer is a specially built ultrasonic speaker. The sound bounces off an object,



Figure 43-14 Ultrasonic distance sensor, using one transducer (both sends and receives sound). (Photo courtesy Maxbotics Inc.)

and the echo is received by another transducer (this one a specially built ultrasonic microphone). A circuit then computes the time interval between the transmit pulse and the echo and comes up with distance.

(Note that some ultrasonic distance measurement sensors use one transducer for both send and receive. The one transducer is engineered as both a high-frequency speaker and a microphone.)

Not long ago, if you wanted to use an ultrasonic ranger in your robot you had to either build one yourself out of separate parts, hack an old Polaroid camera, or purchase a rather expensive ultrasonic distance-measuring experimenter's kit. These days, complete and ready-to-go ultrasonic distance-measuring sensors, like the one in Figure 43-14, are commonly available from a variety of sources. Depending on features, prices start at about \$25 to \$30. These ready-made modules are the ones we're learning about in this chapter.

FACTS AND FIGURES

First, some statistics. At sea level, sound travels at a speed of about 1130 feet per second (about 344 meters per second) or 13,560 inches per second. This time varies depending on atmospheric conditions, including air pressure (which varies by altitude), temperature, and humidity.

The time it takes for the echo to be received is in microseconds if the object is within a few inches or even a few feet of the robot. The overall time between transmit pulse and echo is divided by two to compensate for the round-trip travel time between the robot and the object.

Given a travel time of 13,560 inches per second for sound, it takes 73.7 μ s (microseconds, or 0.0000737 seconds) for sound to travel 1 inch. If an object is 10 inches away from the ultrasonic sensor, it takes 737 μ s to travel there, plus an additional 737 μ s to travel back, for a total time of 1474 μ s. The formula is:

$$(1474 / 2) / 73.7 = 10$$

First, divide the total transit time by 2, then divide by 73.7 (use 74 to avoid floating-point math), the time it takes sound to travel 1 inch at sea level.

USING AN ULTRASONIC DISTANCE SENSOR

Ultrasonic distance sensors come in several varieties. The least expensive require you to do all the calculation math yourself, though as the previous section showed, it's not that difficult.

More advanced (and more expensive) sensors perform the math calculations themselves and provide you with either a digital or an analog output. All you have to do is tell the sensor you want a "ping," and it sends out a brief pulse of ultrasonic sound to see if there's anything nearby.



An advantage of using sound is that, unlike infrared sensors, it's not sensitive to objects of different color and light-reflective properties. Still, some materials reflect sound better than others; some may even absorb sound completely.

For this reason, it's often a good idea to combine both an ultrasonic sensor and an infrared proximity or distance sensor, and retrieve values from both. This method is called sensor fusion, and it's extremely useful when one sensing technology performs better than the other for any given situation.

PROGRAMMING THE BASIC ULTRASONIC SENSOR

United Kingdom-based Devantech manufactures the widely used SRF05. This sensor replaces the company's original SRF04 module (available as of this writing), which was among the first of its type, marketed directly at robot builders and other homebrew experimenters.

The SRF05 is a basic ultrasonic sensor, intended to be connected to a microcontroller or other circuit that contains its own timer. To use, your circuit initiates a sound pulse, then measures the time until the receiver picks up the echo. Your control program, such as *srf05.pde* for the Arduino, is required to perform the math to calculate time of flight. See Figure 43-15 for the very simple diagram for hookup to the Arduino.

```
101010
010101
101010
010101
```

srf05.pde

```
int duration;           // Stores duration of pulse
int distance;         // Stores distance
int srfPin = 2;       // SRF05 connected to digital pin D2

void setup() {
  Serial.begin(9600);
}

void loop() {
  pinMode(srfPin, OUTPUT);           // Set pin to OUTPUT
  digitalWrite(srfPin, LOW);        // Ensure pin is low
  delayMicroseconds(2);
  digitalWrite(srfPin, HIGH);       // Start ranging
  delayMicroseconds(10);            // with 10 microsecond burst
  digitalWrite(srfPin, LOW);        // End ranging
  pinMode(srfPin, INPUT);           // Set pin to INPUT
  duration = pulseIn(srfPin, HIGH); // Read echo pulse
  distance = duration / 74 / 2;     // Convert to inches
  Serial.println(distance);         // Show distance in Serial Monitor
  delay(100);
}
```

To use, compile and upload the sketch, then open the Serial Monitor window to view the results.

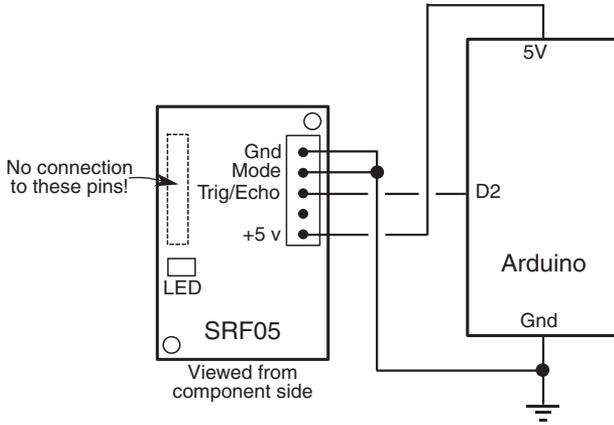


Figure 43-15 Basic connection diagram for the Devantech SRF05 ultrasonic sensor to the Arduino microcontroller.



Be careful when connecting the SRF05 to avoid swapping the 5V and Gnd connections, or you'll reset your Arduino to failsafe mode (its internal fuse "blows" and won't reset itself until you remove power). Figure 43-15 shows connection to the SRF05 from the component side, that is, the side with the LED, ICs, and other surface-mount parts. The two ultrasonic elements are on the other side. Carefully observe the 5V and Gnd pins.

PROGRAMMING THE ENHANCED ULTRASONIC SENSOR

If you don't care to do the time-of-flight arithmetic in your microcontroller, or you're using a microcontroller that lacks the ability to time events with the accuracy needed for ultrasonics, you can instead use a distance-measuring sensor that performs all the math for you. The output of the sensor is digital data (usually as a serial data stream) of the measured distance.

There are a variety of choices. Staying with the Devantech line, there's the SRF02, which has a built-in microcontroller that does the math for you. The sensor requires that your microcontroller have I²C I/O lines; many do these days. (You can also use the ranger in serial mode; see the manufacturer's spec sheet for details.)

Connection is fairly simple; Figure 43-16 shows the SRF02 hooked up to an Arduino. As with the previous warning for the SRF05, be very careful not to flip the 5V and Gnd connections.

The control sketch is shown in *srf02.pde*. To use, compile and upload the sketch, then open the Serial Monitor window to view the results.

```
101010
010101
101010
010101
```

srf02.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, "RBB Online Support," for more details.

ULTRASONIC SENSOR SPECS

Not all ultrasonic sensors offer the same characteristics. Read the specifications to better match the sensor to your task. Things to consider:

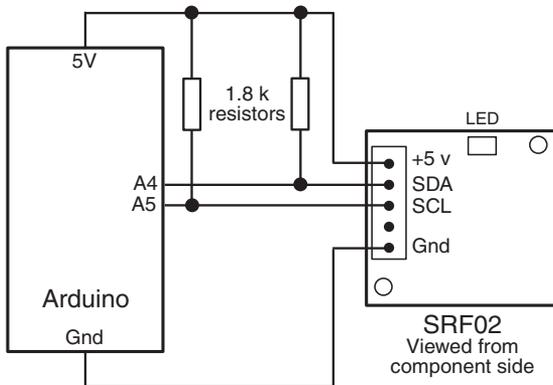


Figure 43-16 Connection diagram for the Devantech SRF02 ultrasonic sensor to the Arduino microcontroller, using I2C serial communications.

Working distance: Ultrasonic sensors have a minimum and maximum effective distance. For sensors like the SRF08, the minimum range is about 2 inches; the maximum about 20 feet. Some sensor ranges are more, some less. A very small minimum range is handy when using ultrasonic sensors in arms and grippers.

Sensitivity: The term “sensitivity” can refer not only to the resolving power of the sensor (see “Resolution,” next), but also to the ability of the sensor to accurately detect objects at a distance. In addition to the design of the receiver electronics, sensitivity is affected by things you can’t easily control: temperature, humidity, even wind, if you’re using the sensor outdoors.

Resolution: The frequency of the sound dictates its ability to detect small or thin objects. The 40-kHz sound waves used in low-cost ultrasonic sensors are fairly coarse. That equates to roughly 8.5mm from peak to peak of each wave. This means the smallest object an ultrasonic sensor operating at 40 kHz can fully resolve is less than half an inch. (In practice, most 40-kHz sensors can detect objects smaller than 8.5mm, but the readings are not always dependable.) Resolution is decreased when there’s a lot of air movement, or when the sensor is used near sources of heat, like hot pavement outdoors or an indoor air duct.

Beam spread and pattern: The typical ultrasonic sensor has a fairly wide cone-shaped echo pattern. The maximum distance of the sensor changes for objects that are off center (not aligned at 0°). On some, the far end of the cone is very wide; these are useful when you want to take in a large area. On others, the cone is very narrow. These so-called pencil-thin sound beams are useful when you want your robot to measure objects some distance away from itself and ignore closer objects that aren’t straight ahead.

Robotic Eyes

For a robot to be truly useful it needs senses; the more senses, the better. It's easy to endow even the most basic robot with a sense of touch—all it takes is a couple of small switches, or maybe an infrared or ultrasonic detector here and there.

But things get a little tougher when it comes to the sense of sight. Producing images for a robot to see is no problem; you need nothing more than an inexpensive video camera. The trouble comes when trying to make sense of those images . . . is that the family cat pretending to be asleep in the middle of the living room, or is it a pair of old, discarded socks?

Despite the challenges in endowing a robot with eyes, there are a number of affordable and relatively easy methods of creating robotic vision, including rudimentary “Cyclops” vision systems that are used to detect the presence or absence of light, as well as more elaborate arrays of sensors to decode relative intensities of light. Let's get started.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details. To save page space, the lengthier programs are not printed here. The support site also offers source code with added comments, parts lists (with sources) for projects, updates, extended construction plans, and more examples you can try!

Simple Sensors for Robotic Eyes

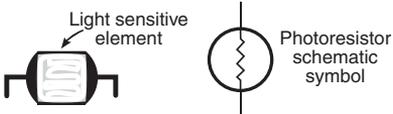
Throughout this chapter I'm going to limit the discussion to the low end of the spectrum: with \$10,000 you can purchase a robust machine vision system and do most anything with it. Alas, this is a tad bit higher than most of us have to spend! Everything here can be done for \$0 to \$100, much of it at the low end of this scale.

So to start, most people think about “robot vision” as some full video-like snapshot, complete with auxiliary text explaining what’s going on, a la *Terminator*. It’s not always like that. A number of very simple electronic devices can be used as very effective eyes for your robot. These include:

- *Photoresistors*, which are also known as light-dependent resistors and photocells
- *Phototransistors*, which are like regular transistors, except they are activated when light strikes them
- *Photodiodes*, which are photo-sensitive diodes that begin to conduct current when exposed to light

Photoresistors, photodiodes, and phototransistors are connected to other electronics in the same general way: you place a resistor between the device and either +V or ground. This forms a voltage divider. The point between the device and the resistor is the output, as shown in Figure 44-1. With this arrangement, the device outputs a varying voltage.

PHOTORESISTORS



Photoresistors are typically made with cadmium sulfide, so they are often referred to as *CdS cells* or *photocells*. A photoresistor functions as a *light-dependent resistor* (also known as an LDR): the resistance of the cell varies depending on the intensity of the light striking it.

When no light strikes the cell, the device exhibits very high resistance, typically in the high 100 kilohms, or even megohms. Light reduces the resistance, usually significantly—a few hundreds or thousands of ohms.

Photoresistors are easy to interface to other electronics. Figure 44-2 shows two ways to connect photoresistors to a circuit. The resistor is selected to match the light and dark resistance of the particular photocell you’re using. You’ll need to experiment a bit here to find the ideal resistor value. Start with 100 k Ω and work your way down. A 100 k Ω potentiometer works well for testing. If you use a pot, add 1 k Ω to a 5 k Ω resistor in series with it to prevent a near short circuit when the photocell is exposed to bright light.

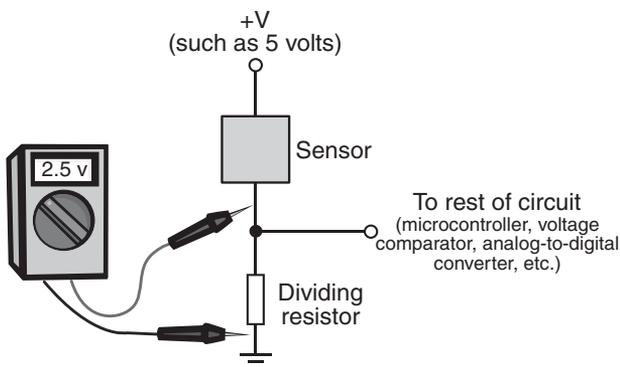


Figure 44-1 The output of photoresistors, phototransistors, and photodiodes can be converted to a varying voltage by using a divider resistor. You may measure the voltage with a multimeter.

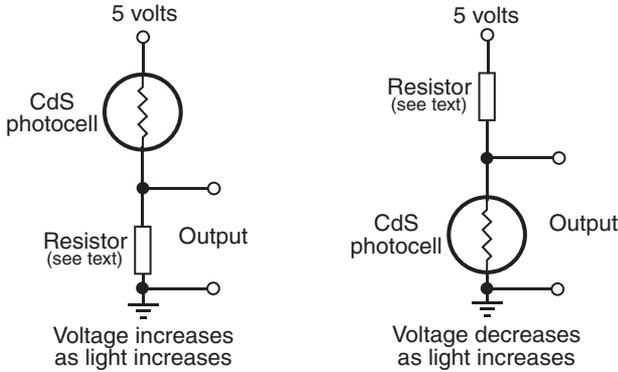
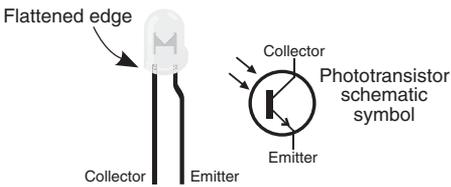


Figure 44-2 A CdS photocell, the most common type of photoresistor, may be connected so that the output voltage increases or decreases with more light. Experiment with the value of the fixed resistor for best sensitivity.

Note that photocells are fairly slow reacting, and are unable to discern when light flashes more than 20 or 30 times per second. This trait actually comes in handy because it means photoresistor cells basically ignore the on/off flashes of AC-operated lamps. The cell still sees the light, but isn't fast enough to track it.

PHOTOTRANSISTORS



All semiconductors are sensitive to light. If you were to take the top off of a regular transistor, it would act as a phototransistor. Only in a real phototransistor, the light sensitivity of the device is much enhanced. A glass or plastic cover protects the delicate semiconductor material inside. Many phototransistors come in a package that looks a lot like an LED. And like an LED, one side of the

plastic case is flattened. Unless otherwise indicated in the datasheet for the phototransistor, the flat end denotes the collector (C) lead. The other lead is the emitter (E).

Unlike photoresistors, phototransistors are very quick acting and able to sense tens of thousands of flashes of light per second. Because of this, they can be used for optical data communications. It also means that when used under AC-operated or fluorescent lamps, the output of the sensor can rapidly vary as it registers the fluctuation in light intensity.

The output of a phototransistor is not "linear"; that is, there is a disproportionate change in the output of a phototransistor as more and more light strikes it. A phototransistor can become easily "swamped" with too much light. Even as more light shines on the device, the phototransistor is not able to detect any more change.

See Figure 44-3 for ways to connect a phototransistor to the rest of a control circuit. Like the photoresistor, experiment with the value of the fixed resistor to determine the optimum light sensitivity. Values of 4.7 k Ω to 250 k Ω are typical, but it depends on the phototransistor and the amount of light that you want to cause a trigger. Start with a lower-value resistor and work your way up. Higher resistances make the circuit more sensitive.



You may instead wish to add a small, 250 kohm potentiometer in series with a 1 k Ω to 5 k Ω fixed resistor. Dial the potentiometer for the sensitivity you want. Connect your multimeter as shown in Figure 44-5 to test the *voltage swing* you get under light and no-light situations. I like to select a resistor so that the sensor outputs about half of the supply voltage (that is, 2.5 volts) under ambient, or normal, light levels.

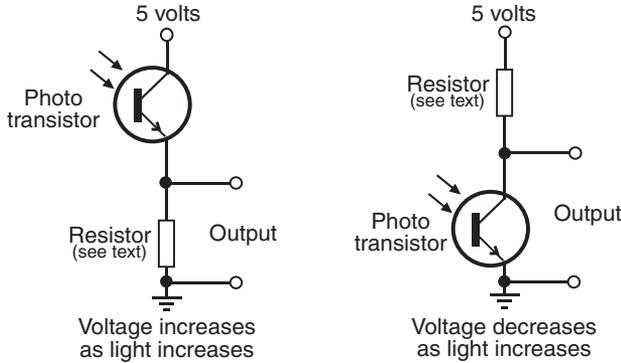


Figure 44-3 Variations in connecting a phototransistor to the rest of the circuit. The output is a varying voltage. Experiment with the value of the fixed resistor for best sensitivity.

PHOTODIODES

These work much like phototransistors but are simpler devices. Like phototransistors, they are made with glass or plastic to cover the semiconductor material inside. And like phototransistors, photodiodes are very fast acting. They can also become “swamped” when exposed to extrabright light—after a certain point, the device passes all the current it’s going to, even as the light intensity increases.

Photodiodes are essentially LEDs in reverse; in fact, you can even use some LEDs as photodiodes, though as sensors they aren’t very sensitive.

If you’d like to experiment with the LED-as-photodiode trick, try to find one that’s in a “water-clear” casing. The light should enter straight into the top of the LED; off to the sides doesn’t work. You might need a simple focusing lens to improve the light-gathering capability. The LED is most sensitive to light at the same wavelength as the color it produces. To register green light, use a green LED, for example.

One common characteristic of most photodiodes is that their output is rather low, even when fully exposed to bright light. This means that, to be effective, the output of the photodiode must usually be connected to a small operational amplifier or transistor amplifier.

SPECTRAL RESPONSE OF SIMPLE SENSORS

Light-sensitive devices differ in their spectral response, which is the span of the visible and near-infrared light region of the electromagnetic spectrum that they are most sensitive to. As depicted in Figure 44-4, CdS photoresistors exhibit a spectral response close to that of the human eye, with the greatest degree of sensitivity in the green or yellow-green region.

Both phototransistors and photodiodes have peak spectral responses in the infrared and near-infrared regions. In addition, some phototransistors and photodiodes incorporate optical filtration to decrease their sensitivity to a particular part of the light spectrum. In most cases, the filtration intentionally makes the sensors more sensitive to infrared and near-infrared light, and less to visible light.

A few special-purpose photodiodes are engineered with enhanced sensitivity to shorter-wavelength light. This allows them to be used with ultraviolet emitters that cause paint and ink pigments to fluoresce in the visible light spectrum. These are used, for example, in currency verification systems. For robotics you might use such a sensor to follow a line on the ground painted with a fluorescent dye.

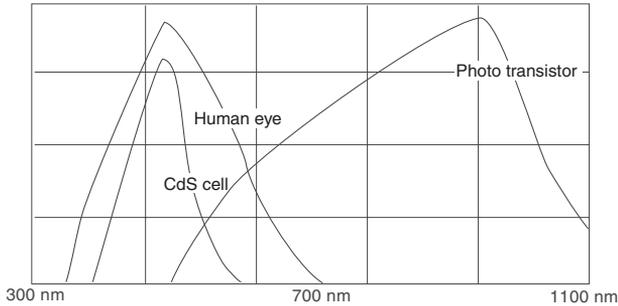


Figure 44-4 Comparison of the light spectrum sensitivity of the human eye, the CdS photoresistor, and the most common form of phototransistor. Visible light is between about 400 nanometers (nm) and 750 nm.

Building a One-Cell Cyclops Eye

A single light-sensitive photoresistor is all your robot needs to sense the presence of light. As noted, the photoresistor is a variable resistor that works much like a potentiometer but has no control shaft. You vary its resistance by increasing or decreasing the light shining on it.

Connect the photocell as shown in the “voltage increases as light increases” version of Figure 44-2. As explained in the previous section, a resistor is placed in series with the photocell and the “output tap” is between the cell and resistor. This converts the output of the photocell from resistance to voltage. Voltages are a lot easier to measure in a practical circuit.

Typical resistor values are 1 to 10 k Ω , but this is open to experimentation. You can vary the sensitivity of the cell by substituting a higher or lower value. Test the cell output by connecting a multimeter to the ground and output terminals, as shown in Figure 44-5. For experimenting, you can connect a 2 k Ω resistor in series with a 50 k Ω potentiometer—these replace the single resistor that’s shown. Try testing the cell at various settings of the pot.

So far, you have a nice light-to-voltage sensor, and when you think about it, there are numerous ways to interface this ultrasimple circuit to a robot. One way is to connect the output of the sensor to the input of a comparator. The LM339 quad comparator IC is a good choice. The output of the comparator changes state when the voltage at its input goes beyond or below a reference voltage or “trip point.”

In the circuit shown in Figure 44-6, the comparator is hooked up so the noninverting input (marked +) serves as a voltage reference. Adjust the potentiometer to set the trip point higher or lower than what you want to trigger at. To begin, set it midway, then adjust the trip point

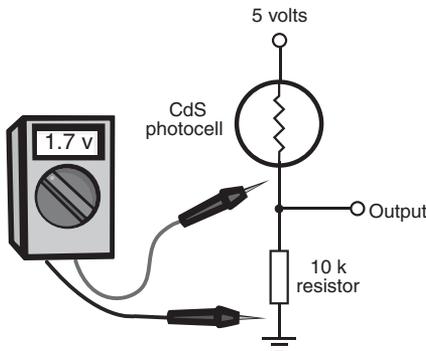


Figure 44-5 How to test a CdS photoresistor using a multimeter. Dial the meter to read DC voltage, then take the measurement as shown.

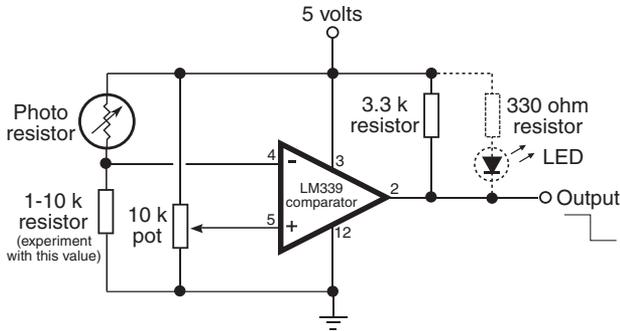


Figure 44-6 Using an LM339 voltage comparator to provide a basic on/off output from a photoresistor (this also works with a phototransistor). Adjust the 10 k Ω potentiometer so that the comparator triggers at just the desired amount of light.

higher or lower as you experiment with different light levels. The output of the photoresistor circuit is connected to the inverting input (marked $-$) of the comparator. When the voltage at this pin goes above or below the point with the potentiometer, the output of the comparator changes state.

With the wiring shown, the output goes from HIGH to LOW as the light increases. One application is to adjust the potentiometer so that under normal room light the output of the '339 just goes HIGH. Shine a light directly into the sensor, and the output switches LOW.



If you want the opposite effect—the output goes from LOW to HIGH under increasing light—simply switch the connections to pins 4 and 5 of the comparator. This makes the logic go in reverse.

The circuit also shows an optional LED and resistor, useful as a visual indicator, for testing purposes only. The LED will wink on when the output of the comparator goes HIGH and wink off when it goes LOW. Remove the LED and resistor if you connect the comparator to another digital circuit, such as a microcontroller.

CREATING A LIGHT-RECEPTIVE ROBOT

One practical application of this circuit is to detect light levels that are higher than the ambient light in the room. Doing so enables your robot to ignore the background light level and respond only to the higher-intensity light, like a flashlight. You don't even need a microcontroller or other brain for your robot. It can be done using only simple components and a relay. Figure 44-7 shows one example of an old-skool bot that reacts to light falling on a phototransistor.

To begin, set the 10 k Ω potentiometer so that in a darkened room the circuit just switches HIGH (this deenergizes the relay). Use a flashlight with fresh batteries to focus a beam directly onto the phototransistor. Watch the output of the comparator change state from HIGH to LOW. When LOW, the relay is energized through the 2N3906 PNP transistor.

You can use this so-simple-it's-dumb circuit to activate your robot so it will move toward you when you turn the flashlight on it. Or the inverse: It runs away from you. Just reverse the connections to the motor.

You can also use photoresistors instead of phototransistors. Experiment with the value of the series resistor. You'll need to pick a value that best matches the actual photocells you are using.

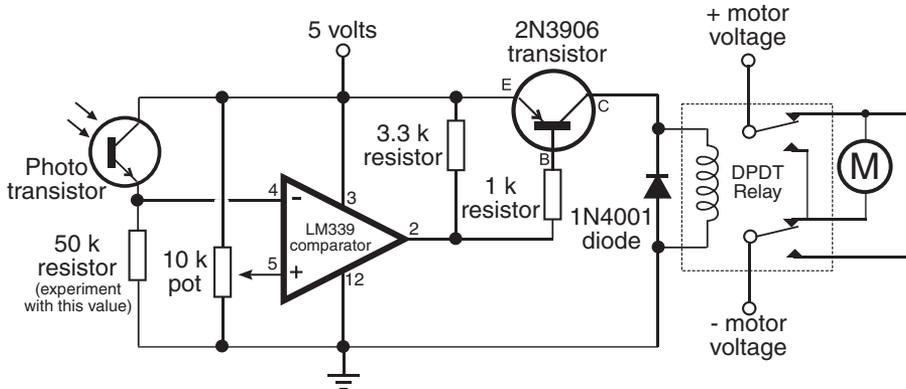


Figure 44-7 All-in-one control circuit for operating a motor based on the amount of light falling on a phototransistor. When the relay triggers, the motor reverses. Adjust the light sensitivity of the circuit by changing the value of the fixed resistor below the phototransistor.

When your robot advances toward the light, it's said to be *photophilic*—it “likes” light. When your robot shies away from the light, it's said to be *photophobic*—it “hates” light. When combined with other actions, these *behaviors* allow your robot to exhibit what appears to be artificial intelligence.



Look for more light-dependent robot examples on the RBB Online Support site (see Appendix A), including microcontroller code for a completely programmed light-follower/light-avoider bot.

Building a Multiple-Cell Robotic Eye

The human eye has millions of tiny light receptacles. Combined, these receptacles allow us to discern shapes, to actually “see” rather than just detect light levels. A surprisingly useful implementation of human sight is given in Figure 44-8. In this design by robotics guru Russell

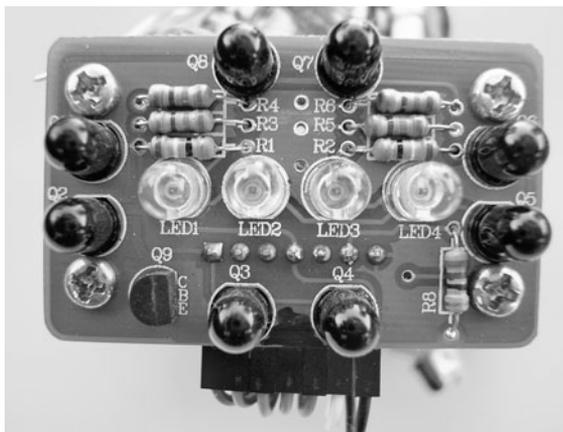


Figure 44-8 “Compound eye” created with four infrared LEDs and eight phototransistors. The phototransistors are in four groups of two, placed on the top, bottom, left, and right, allowing the sensor to detect basic orientation of obstacles in front of it.

Cameron, eight phototransistors are connected in four sets of two; the four sets are then routed to four separate inputs of a microcontroller. A benefit of this design is that it's also available as a low-cost commercial product from DAGU Electronics, provided by several online specialty robotics stores, including RobotShop.

Under program control, the Arduino selects one of the four inputs while illuminating the scene immediately ahead of the phototransistors by turning on a set of infrared LEDs. Each of the four sets of phototransistors is read in turn. The sensor can detect objects 6" to 8" away, and in the case of a hand, ankle, ball, or kitten, can determine whether the offending object is directly in front of it or to the sides.



This project shows a compound robotic eye with eight individual light sensors, but you can build it using more or fewer sensors if you wish. While you could conceivably build a vision component with greater than 32 phototransistors, it would end up being a pretty big and ungainly eye.

PHYSICAL CONSTRUCTION OF THE COMPOUND EYE

Figure 44-9 shows the layout of the eye, and Figure 44-10 shows the wiring diagram. The infrared LEDs are located in the center of the eye sensor board, the four pairs of phototransistors on each of the four sides.

All the LEDs and all the phototransistors should be identical, and, for the best results, they should be brand-new prime components. Surplus and odds-and-ends components may vary too much in their output to be useful. If you can, select 4.7 k Ω resistors with a 1 percent or 2 percent tolerance. For best results, select phototransistors with a built-in infrared filter. These will look dark red, dark blue, or brown. The filtration will help block unwanted room light. Operate the eye indoors, under modest light. Bright lights or sunlight will reduce its effectiveness.

TESTING PROGRAM

Program *flyeye.pde* contains the Arduino code for testing the compound eye. Use the circuit in Figure 44-11 to visually see the sensors in action. Slowly wave your hand side to side and up and down. The four indicator LEDs should light in response to your movements. The pro-

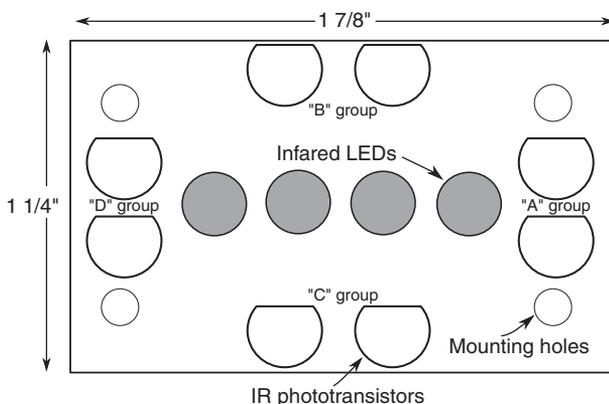


Figure 44-9 LED and phototransistor layout of the compound eye.

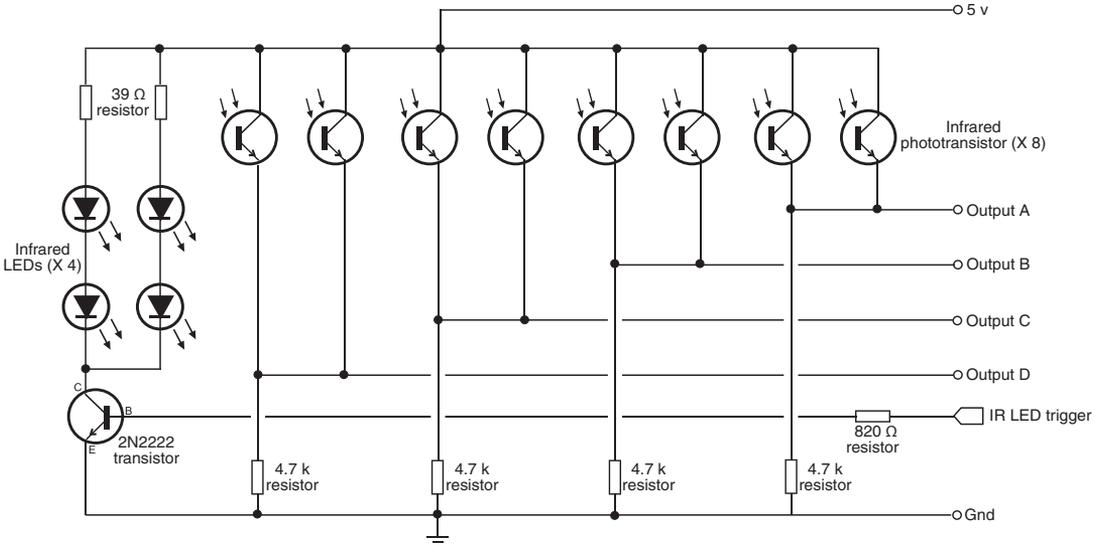


Figure 44-10 Hookup diagram for the compound eye. The LEDs may be turned on and off (when not needed), or pulsed to provide basic modulation. (Circuit courtesy Russell Cameron.)

gram assumes the eyes are connected top, bottom, left, and right, as shown in Figure 44-9. If the eyes are switched around on your prototype, then simply alter the wiring to the A0 to A3 analog inputs of the Arduino.



Note the *thresh* variable; it sets the threshold, or sensitivity, of the eyes. Decrease the value of *thresh* to increase the sensitivity. The practical minimum value depends on many factors. In my tests the minimum threshold for all eyes was about 200, but that caused false reads now and then. I set it to 250 for an extra margin of safety. (The *thresh* value was set with the four IR LEDs

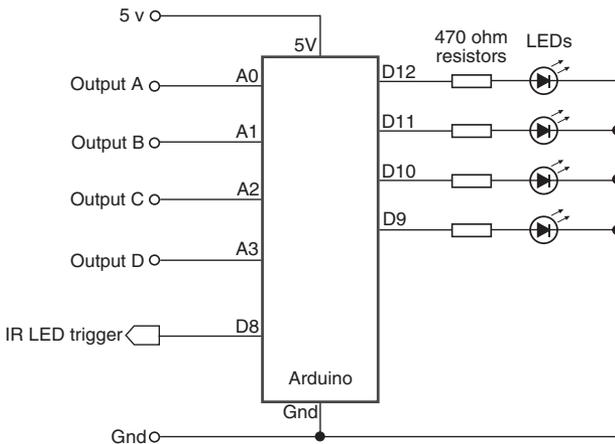


Figure 44-11 Hookup scheme for the compound eye and the Arduino microcontroller. The four outputs of the eye connect to four of the Arduino's ADC inputs. A set of LEDs visually indicate when a set of eyes has detected an object.

activated; the number is a lot lower when the LEDs are off and the eyes are responding only to the natural light of the room.)

You can test the numeric values from each pair of the eyes with this code:

```
const int eye = A0;
const int LED = 8;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  digitalWrite(LED, HIGH);
}

void loop() {
  Serial.println(analogRead(eye), DEC);
  delay(500);
}
```

Open the Serial Monitor window and look at the number returned from the eye connected to analog line A0. Change the first line to A1, A2, or A3 to read the other pairs of eyes.

You may find that under the exact same lighting conditions some of the eyes return higher or lower values. This is due to differences in the phototransistors, their mounting, and the tolerance of the 4.7 k Ω resistors you use. As needed, you can set a different threshold for each eye in order to “balance them out,” but you’ll probably find that isn’t necessary as long as the eyes are within about 100 points of one another.

```
101010
010101
101010
010101
```

flyeye.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details. Additional examples are provided that demonstrate using the compound eye to directly control a two-wheeled robot.

DEALING WITH LIGHT SPOILAGE

The bane of any light-sensitive detector is *spoilage* from stray light sources. Examples include:

- *Infrared light coming from outdoors, a desk lamp, or other source, and not from the infrared LEDs you have so carefully placed on your robot.* You can help mitigate this by using tubes and baffles to block unwanted light. A simple light tube can be constructed using a small piece of black heat shrink tubing (not shrunk) cut to length and placed around the sensor.
- *Ambient (natural room) light striking the sensor from the sides, or even from behind, rather than straight down its gullet.* CdS photocells are sensitive to light coming from behind them (through their backing). To avoid this spoilage, always place your photoresistors against a black or opaque backstop that will block light.
- *Light spilling from the side of the LED and directly into the sensor.* Use the heat shrink tubing trick, or add a heavy felt baffle between the two. It also helps to position the LED slightly forward of the sensor.
- *Visible-light LEDs and LCD display panels* that are located too close to the sensor. Their light can accidentally influence the sensor. Be sure to always locate your light sensors away from any potential light sources.

Using Lenses and Filters with Light-Sensitive Sensors

Simple lenses and filters can be used to greatly enhance the sensitivity, directionality, and effectiveness of both single- and multicell vision systems. By placing a lens over a small cluster of light cells, you can concentrate room light to make the cells more sensitive to the movement of humans and other animate objects. And you can also use optical filters to enhance the operation of light cells.

See The RBB Online Support site (refer to Appendix A) for an extended discussion of selecting and using lenses and filters with robotic optical systems.

Video Vision Systems: An Introduction

The one sense that offers the most potential to the science of robotics is also the most elusive: vision. Robotic sight is something of a paradox. The sensor for providing a video image is actually quite mundane, you can now purchase black-and-white video cameras for under \$20, complete with lens.

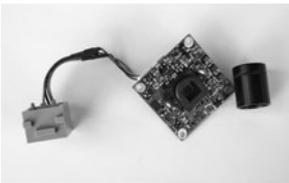
But there's a second part to the vision equation: what to do with the image data once it's been acquired. There is a burgeoning science of *machine vision* that seeks to provide answers.

The single- and multicell vision systems described previously are useful for detecting the absence or presence of light, but they cannot make out anything except very crude shapes. This greatly limits the environment into which such a robot can be placed. By detecting the shape of an object, a robot might be able to make intelligent assumptions about its surroundings and perhaps navigate those surroundings, recognize its "master," and more.

FYI

The machine vision described here is different from video beamed from a robot back to its human operator. The latter technique is used in teleoperated robots, where you get to see what the robot sees. Video for teleoperation is covered in Chapter 41, "Remote Control Systems."

CAMERAS AND EQUIPMENT



A video system for robot vision need not be overly sophisticated. The resolution of the image can be as low as about 300 by 200 pixels. A color camera is not mandatory, though in some kinds of vision analysis techniques, the use of color is how the system tracks objects and movement.

Video systems that provide a digital output are generally easier to work with than those that provide only an analog video output. You can connect digital video systems directly to a PC, such as through a serial, parallel, or

USB port. Analog video systems require the PC to have a video capture card, fast analog-to-digital converter, or similar device attached to it.

PC-BASED VISION

If your robot is based on a laptop or desktop personal computer, you're in luck! There's an almost unlimited array of inexpensive digital video cameras that you can attach to a PC. The proliferation of Webcams for use with personal computers has brought down the cost of these devices to under \$50, and often less for bare-bones models. You can use a variety of operating

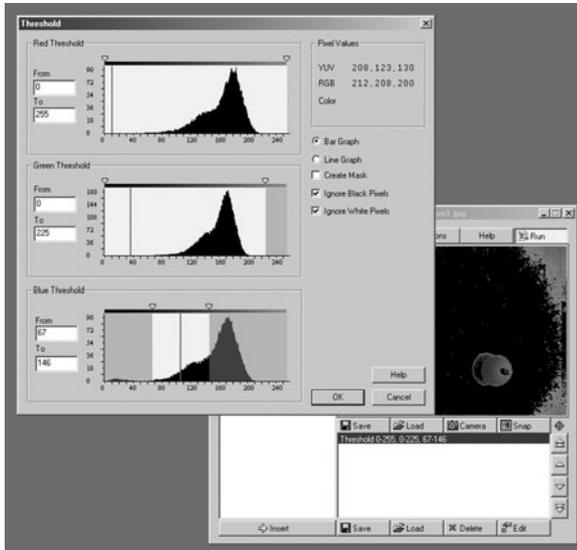


Figure 44-12 A screenshot of a color threshold window in the RoboRealm computer vision software. (Courtesy RoboRealm.)

systems—Windows, Linux, or Macintosh—though be forewarned that not every Webcam has software drivers for every operating system. Be sure to check before buying.

Connecting the camera to the PC is the easy part; using its output is far more difficult. You need software to interpret the video scene that the camera is capturing. An example is RoboRealm, at www.roborealm.com. RoboRealm (see a screenshot in Figure 44-12) is a set of low-cost vision analysis tools that allows your *tobor* to recognize shapes and objects, and even track them in real time.



You'll also want to track down past issues of *Nuts and Volts* and *SERVO* magazines, both of which have published excellent articles on machine vision. If you don't have them already, back issues are available on CD-ROM. Look for the series written by robot vision expert Robin Hewitt.

If you're a user of the .NET programming platform under Windows and are fairly familiar with C# or VB programming, be sure to investigate the DirectShow.Net Sourceforge open-source project at directshounet.sourceforge.net. DirectShow.Net is a managed .NET wrapper that allows you to tap into the incredibly powerful DirectShow architecture of Windows, without the need to use C++.

The project authors provide samples of capturing video, and several samples demonstrate how to retrieve the bitmap of each video frame as it goes through the system. With this bitmap you can write your own image analysis routines, such as looking for pixels of a specific color. Or you can use fairly simple scene-averaging techniques to determine if there's movement in the frame.

OTHER EXAMPLES OF MICROCONTROLLER-BASED VISION

A PC makes it easy to integrate a Webcam, but it's not the only way to provide vision to your robot. Several microcontroller-based vision systems are available that work with most any robot brain, including inexpensive MCUs like the Arduino or BASIC Stamp.

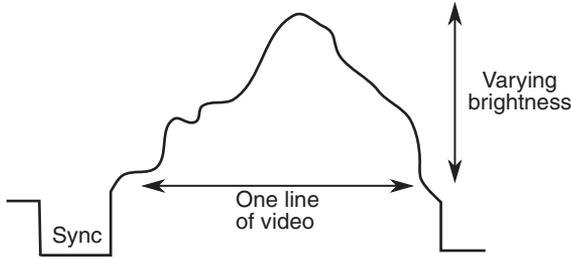


Figure 44-13 Composite video is composed of many separate lines that together make a whole picture. This graph shows just one line of video, starting with a sync signal.

A popular microcontroller-based vision solution is the CMUcam, available commercially from a number of specialty robotics stores; see Appendix B, “Internet Parts Source” for a list of some of these. This device incorporates a color imager, lens, and image analysis circuitry. The CMUcam tracks objects by color and motion. It can connect to a computer or microcontroller via a serial data link.

USING COMPOSITE VIDEO FOR VISION

In the modern digital world, it’s sometimes easy to forget that analog isn’t dead—not by a long shot. Most low-cost black-and-white and color security cameras provide an analog composite video signal, for direct connection to a time-lapse VCR or television monitor.

There are several standards used for video signals—NTSC and PAL being the two most common around the world. Select a camera that supports the standard used in your location. NTSC is used in North America, Japan, and Canada, and PAL is used in most of Europe, Australia, and China. (A third system, called SECAM, is used in Russia and France, among others.)



Technically speaking, NTSC defines a standard for providing color. The RS-170 video standard defines the basic video signal format used with NTSC. Even for a black-and-white camera, manufacturers will still often state that their product is “NTSC compatible.”

In order to process video from a composite video camera, you need a means to separate out the different parts of the signal. Monochrome video is composed of two major components: sync and video (see Figure 44-13). Low-cost integrated circuits like the LM1881, (\$3 from Digikey, Newark, and other chip distributors; use www.findchips.com to locate a source) allow you to apply a composite video signal and get back separate outputs for vertical and composite sync.

With a microcontroller, you can use these signals as a means to synchronize with the video information that is also being provided by the camera. You can capture a full frame of video or analyze one line of video at a time.

In order to effectively use a sync separator IC like the LM1881, you should have a fairly good understanding of how analog video works. As this is an old technology, you can check the local library for books on the subject. Even one a few decades old will give you the framework for developing custom vision projects using analog video. Refer to Appendix A, “RBB Online Support,” for links to learn more about processing analog video using the LM1881 and other techniques.

Navigating Your Robot

Robots suddenly become useful once they can master their surroundings. Being able to wend their way through their surroundings is the first step toward that mastery.

The projects and discussion in this chapter focus on navigating your robot through space—not the outer-space kind, but the space between two chairs in your living room, between your bedroom and the hall bathroom, or outside your home by the pool.

The techniques used to provide such navigation are varied: line followers, wall followers, compass bearings, odometry, and more.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details. To save page space, the lengthier programs are not printed here. The support site also offers source code with added comments, parts lists (with sources) for projects, updates, extended construction plans, and more examples you can try!

Tracing a Predefined Path: Line Following

Perhaps the simplest navigation system for mobile robots involves following some predefined path that’s marked on the ground. The path can be a black or white line painted on a hard-surfaced floor, a wire buried beneath a carpet, a physical track, or any of several other methods. This type of robot navigation is used in some factories. The reflective tape method is preferred because the track can easily be changed without ripping up the floor.

You can readily incorporate a tape-track navigation system into your robot. The line-following feature can be the robot’s only means of semi-intelligent action, or it can be just one part of a more sophisticated machine. You could, for example, use the tape to help guide a robot back to its battery charger nest.

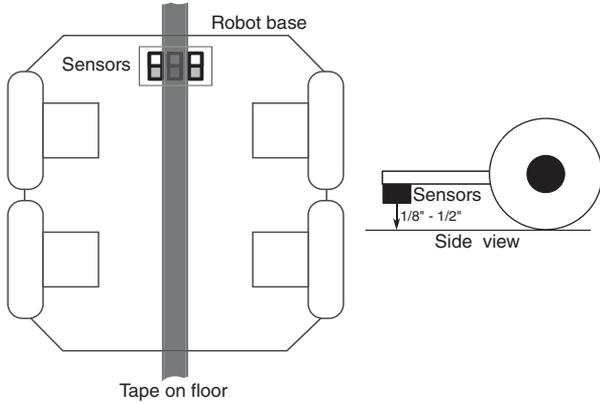


Figure 45-1 Placement of line-following sensors on the bottom of the robot. For best results, the sensors should be no more than 1/8" to 1/2" from the surface of the floor.

SETTING UP FOR FOLLOWING A LINE

With a line-following robot, you place a piece of white, black, or reflective tape on the floor—the color of the tape selected to contrast with the floor.

For the best results, the floor should be hard, like wood or linoleum, and not carpeted. One or more optical sensors are placed under the robot. These sensors incorporate an infrared LED and an infrared phototransistor. When the transistor turns on, it sees the light from the LED reflected off the tape. Obviously, the darker the floor, the better, because the tape shows up against the background.

You can build your own line following sensors, or use a commercially available LED-phototransistor pair. Mount the detectors on the bottom of the robot, as shown in Figure 45-1, in which detectors have been strategically placed so they straddle the width of the tape.

Figure 45-2 shows several variations, using 1/4"-, 1/2"-, and 3/4"-wide art tape. You might try the spacings as shown, but feel free to experiment with other variations. Figure 45-3 shows the basic sensor circuit and how the LED and phototransistor are wired. Most line-following robots use three pairs of these LED/phototransistors to straddle the track:

- The pairs on either side tell the robot it's veered off too far left or right. To counter the mistracking, the robot steers in the opposite direction of the sensor pair that was activated. For example, if the right sensor pair sees the line, the robot corrects by steering to the left.
- The sensor pair in the center is used to indicate correct tracking. As long as this sensor sees the line, then the robot is headed in the correct direction.

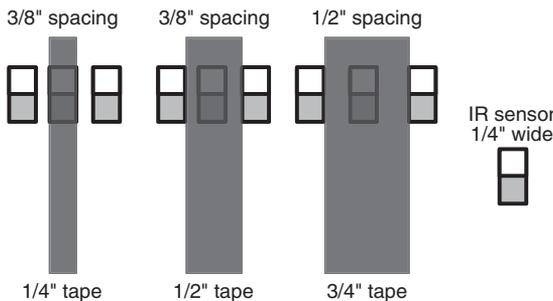
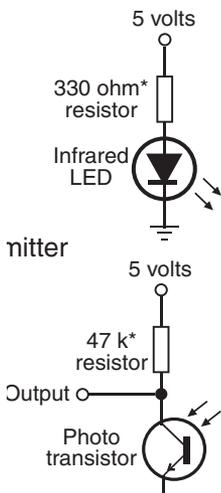


Figure 45-2 There is some flexibility in the spacing of the sensors in relation to the width of the tape, but on most line-following bots the sensors are spaced at about the same width as the line being tracked.



Tape on
floor

Figure 45-3 Circuit diagram for an LED emitter and phototransistor pair, used for line following (and other robotic tasks). The resistor above the infrared emitter should be selected to provide a high current to the LED, but without destroying it, of course. If the light from the emitter is too weak, the phototransistor may not pick it up. See Chapter 31 on how to calculate the value of the resistor used for limiting current through an LED.

Some robot builders prefer using a photoresistor (CdS cell) rather than a phototransistor. Photoresistors are often desired because their sensor sees a larger area, and they are naturally slower to react to changes in light intensity.

However, since photoresistors are most sensitive to the middle of the visible light spectrum (see Chapter 44 for more details), it's best to use green or yellow LEDs for illumination, rather than infrared emitters. The output of photoresistors is also not as consistent as that of phototransistors. Even among components of the same brand and model, one cell may have a higher output value than another. This may require you to selectively adjust the sensitivity of the photoresistor by altering the value of its series resistor.

SELECTING PROPER RESISTOR VALUES

Specifications for infrared LEDs and phototransistors vary widely. You may need to play with the values for the two resistors shown in Figure 45-3 to determine what works best for the parts you have.

- The resistor above the infrared emitter LED controls the brightness of the LED.
- The resistor above the phototransistor controls the sensitivity of the phototransistor.

Both work hand-in-hand to provide the proper amount of light and detection.

Some LEDs don't emit as much light as others, so you may need to increase their current so that they glow more brightly. I've specified a 330 Ω resistor to start; *decrease* the value of the resistor (to *no less than* 200 Ω) to increase the output of the LED. Conversely, if the LED is too bright—it spills light everywhere and causes the phototransistor to falsely trigger—you'll need to *increase* the value of the resistor to perhaps 470 Ω to 1 k Ω .

Similarly, I've specified a 47 k Ω resistor for the phototransistor. You can experiment with

values as low as 3.3 k Ω and as high as 470 k Ω and higher. The lower the resistance, the less sensitive the phototransistor is to light. If you don't seem to be getting much voltage swing when alternately hiding and exposing the phototransistor to a bright light source, try a higher value for this resistor. For my prototype tests, I used a RadioShack infrared transistor, #276-0148. The 47 k Ω resistor value provided good response, but the phototransistor continued to work even with much lower and much higher values.

EMITTER/SENSOR LAYOUT

Though many line-following robots use three sets of emitters/detectors as previously described, there are plenty of variations you might want to experiment with, as shown in Figure 45-4. The three-pair arrangement is simply a convenience because you can purchase emitter/detectors as a prepackaged set.

One emitting source and two detectors: A single LED (infrared or visible) is placed in the center, flanked by two photo-sensitive sensors (phototransistors in the case of an infrared source; photoresistors for a visible source). This has the benefit that both sensors receive the same intensity of light when over the same color. This saves you from having to calibrate the sensors or light sources to make them even.

Four (or more) emitting sources/detectors: There's no rule that says you just have to use three pairs of emitters and detectors. With more sensor pairs you can write more elaborate software for controlling the robot. Sensor pairs in the middle can be used for modest course correction, while those toward the outside indicate serious mis-steering.



Not sure if your infrared LEDs are putting out any light? Try taking a picture of them using a camcorder or a cell-phone camera. These devices tend to be sensitive to infrared, so if the LEDs are working, you'll see it in the picture as a white light.

USING READY-MADE EMITTER/DETECTOR PAIRS

You may opt for a ready-made solution whereby the IR LED and phototransistor are in a single module, perhaps mounted on a small circuit board that you can attach to your robot. Figure 45-5 shows an array of eight light emitter/detector pairs on a single board (two of the pairs can be snapped off, making for two or six pairs, if that's more convenient).

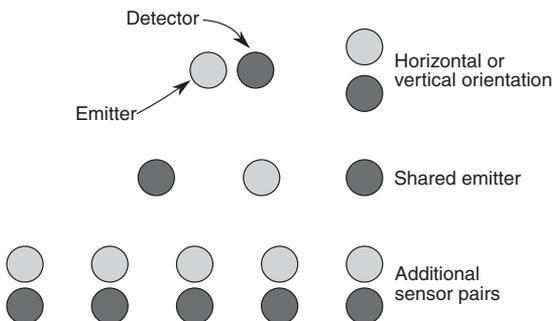


Figure 45-4 Alternative arrangements for emitter/detector pairs used for line following. You can orient the pairs horizontally and vertically, share one emitter with several detectors, or use more than three emitter/detector pairs.

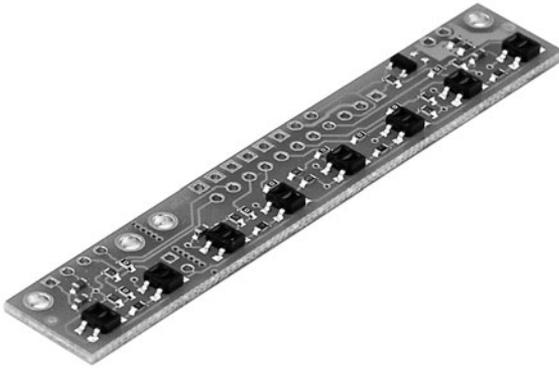


Figure 45-5 Infrared emitter/detector sensor array. Each pair is separated by 0.375" (3/8"). Two of the pairs can be snapped off on their own. (Photo courtesy Pololu.)

AVOID OVERLY TIGHT TURNS

Your robot won't be able to make the turn if it's too tight. When drawing a line for your bot to follow, try more gradual turns rather than abrupt intersections. Otherwise, the robot might skip the line and go off course. The actual *turn radius* will depend entirely on the robot—its size, wheel base dimensions, and speed.

At the same time, you don't want to shy away from the challenge of making your robot handle increasingly tight turns. Line following is a learned art; start out simple, and progress from there. With the right design and programming, your bot may be able to literally turn on a dime. It takes experience to get to that point in your robot-building prowess.

LINE-FOLLOWING SOFTWARE

Any microcontroller is up to the task of navigating a robot over a line. On a three-sensor line follower, three input lines are required. Your program monitors the status of the three sensors and adjusts the motors accordingly.

Figure 45-6 shows a wiring diagram for connecting a trio of infrared light emitters and detectors to an Arduino. Refer to the section "Selecting Proper Resistor Values," previously in this chapter, for more information on experimenting with the values of the resistors used in the circuit. The program *line_follow.pde* controls the typical two-wheeled robot using these three sensors.

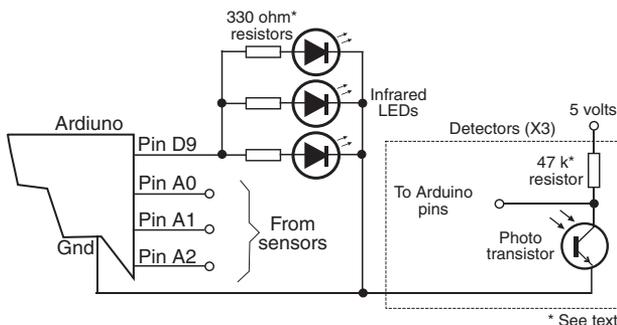


Figure 45-6 Arduino connection diagram for a three-pair emitter/detector line follower. The LEDs are driven through a digital output (so they can be turned on and off, or even pulsed). The output of each phototransistor is connected to a separate analog I/O pin of the Arduino.

In operation, when the sensor pair to the left sees the line, the left motor is very briefly stopped. This has the effect of veering the robot to the left. The reverse happens when the sensor pair to the right sees the line.



The “color” of the line is important, and ideally your line-following software should be selectable between black-on-white and white-on-black. This entails reversing the logic on the input pins: LOW is treated as HIGH, and vice versa.

```
101010
010101
101010
010101
```

line_follow.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, “RBB Online Support.”

OTHER USES FOR LINE-FOLLOWING SENSORS

You can adapt line-following sensors for other robotic applications. For example, use the sensors to detect when the robot is about to go off the edge of a table or stairs. As long as the sensor(s) see light, the robot is over even ground. But when the reflected light is absent, it could indicate a steep falloff to the ground below.

Obviously, this technique works best for robots on which the line sensors are ahead of the drive wheels. It’s less helpful if the wheels are in the front with the sensors; the robot won’t know it’s falling over a cliff until it’s already heading down!

Wall Following

Robots that can follow walls are similar to those that can trace a line. Like the line, the wall is used to provide the robot with navigation orientation. One benefit of wall-following robots is that you can use them without having to paint any lines or lay down tape. Depending on the robot’s design, the machine can even maneuver around small obstacles.

VARIATIONS OF WALL FOLLOWING

Wall following can be accomplished with any of four methods. All are shown in Figure 45-7.

- *Contact.* The robot uses a mechanical switch, or a stiff wire that is connected to a switch, to sense contact with the wall. This is by far the simplest method, but the switch is prone to mechanical damage over time.
- *Noncontact, active sensor.* The robot uses active proximity sensors, such as infrared or ultrasonic, to determine its distance from the wall. No physical contact with the wall is needed.
- *Noncontact, passive sensor.* The robot uses passive sensors, such as linear Hall effect switches, to judge distance from a specially prepared wall that is equipped with a wire car-

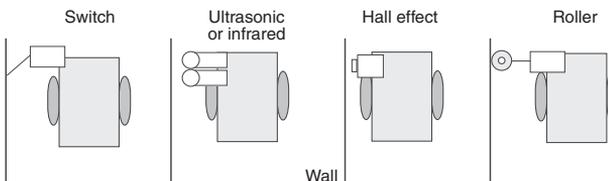


Figure 45-7 Several methods for wall following, including noncontact (ultrasonic, infrared, Hall effect) or contact (roller, whisker switch).

rying an alternating current. When the robot is in the proximity of the wire, the sensor will pick up the induced magnetic field provided by the alternating current.

- “*Soft-contact.*” The robot uses mechanical means to detect contact with the wall, but the contact is “softened” by using pliable materials. For example, you can use a lightweight foam wheel as a “wall roller.”

In all cases, upon encountering a wall, the robot goes into a controlled program phase to follow the wall in order to get to its destination. In a simple contact system, the robot may back up a short moment after touching the wall, then swing in a long arc toward the wall again. This process is repeated, and the net effect is that the robot “follows the wall.”

With the other methods, the preferred approach is for the robot to maintain proper distance from the wall. Only when proximity to the wall is lost does the robot go into a “find wall” mode. This entails arcing the robot toward the anticipated direction of the wall. When contact is made, the robot alters course slightly and starts a new arc.

Odometry: Calculating Your Robot's Distance of Travel

Odometry is the technique of counting the revolution of a robot's wheels to determine how far it's gone. Odometry is perhaps the most common method to determine where a robot is at any given time (it also can provide other information, such as speed). It's cheap and easy to implement, and it is fairly accurate over short distances.

Odometers are often referred to as *encoders* or *shaft encoders* because they are mounted on the shafts of motors and wheels to count revolutions. The terms “odometer,” “encoder,” and “shaft encoder” are often used interchangeably.

Encoders allow you to measure not only the distance of travel of a robot, but its velocity. By counting the number of transitions provided by the encoder, the robot's control circuits can keep track of the revolutions of the drive wheels.

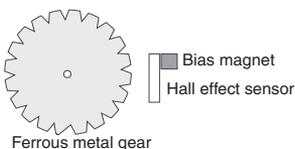
OPTICAL ENCODERS

Perhaps the most common method of adding odometry to robots is to attach a small disc (also called a *codewheel*) to the hub of a drive wheel. The disc works as part of an optical system, either reflective or transmissive. With either method, a pulse is generated each time the photodetector senses the light.

- With a *reflective* disc, infrared light strikes the disc and is reflected back to a photodetector. Both the infrared LED and the photodetector are mounted on the same side of the disc.
- With a *transmissive* disc, infrared light is alternatively blocked and passed by slots or small cutouts. The light is detected by a sensor mounted on the opposite side of the LED.

MAGNETIC ENCODERS

You can construct a magnetic encoder using a Hall effect switch (a semiconductor sensitive to magnetic fields) and one or more magnets. A pulse is generated each time a magnet passes by the Hall effect switch. A variation on the theme uses a metal gear and a special Hall effect sensor that is sensitive to the variations in the magnetic influence produced by the gear.



A bias magnet is placed behind the Hall effect sensor. A pulse is generated each time a tooth of the gear passes in front of the sensor. The technique provides more pulses on each revolution of the wheel or motor shaft, and without having to use separate magnets on the rim of the wheel or wheel shaft.

MECHANICAL ENCODERS

A low-cost form of encoder uses mechanical contacts rather than light or magnets. While technically encoders, these are used more as digital potentiometers; in fact, many mechanical encoders are made with a push-on/push-off switch feature for use as such things as power and volume controls on car radios. Many have a “detent,” tactile feedback when turning the dial.

While you can use a ready-made mechanical encoder with your robot, you’ll probably find it doesn’t last nearly as long as the other types. After a period of time spinning back and forth in your robot, the mechanical contacts of the encoder will simply give out.

A CLOSER LOOK AT ODOMETRY

Your robot uses odometry to judge distance of travel (and speed, if needed) by counting electrical pulses.

Let’s say the codewheel disc has 50 slots around its circumference. That represents a minimum sensing angle of 7.2° . As the wheel rotates, it provides a signal pulse to the counting circuit every 7.2° . By counting those pulses and calculating how many pulses there are each second, and knowing the diameter of the wheels on your robot, you can determine distance and speed.

Suppose the robot is outfitted with a 7” wheel (circumference = 21.98”; you calculate circumference by multiplying the diameter of the wheel by π , or approximately 3.14). Given a pulse every 7.2° of the wheel’s rotation, that produces a resolution that is approximately 0.44 linear inch per pulse. This figure was calculated by taking the circumference of the wheel and dividing it by the number of slots in the codewheel disc. If your robot counts 100 pulses, it knows it’s moved $0.44" \times 100$, or 44”. If the robot uses the traditional two-wheel-drive approach, optical encoders are attached to both wheels. This is necessary because the drive wheels of a robot are bound to turn at slightly different speeds over time. By integrating the results of both optical encoders, it’s possible to determine where the robot really is, as opposed to where it should be. As well, if one wheel rolls over a cord or other small lump, its rotation will be hindered; this can cause the robot to veer off course.

ANATOMY OF A REFLECTIVE ENCODER

Perhaps the easiest type of odometer to build is the reflective encoder, like that in Figure 45-8. You can use the inside surface of a wheel to paste on a circular pattern of white/black stripes. This forms the encoder disc (codewheel) itself. You then mount an infrared emitter and detector close to the codewheel. As the wheel turns, the detector senses the light/dark pattern on the codewheel, providing a signal for use with your control electronics.



See the RBB Online Support site (see Appendix A) for a variety of free codewheel designs that you can download and print.

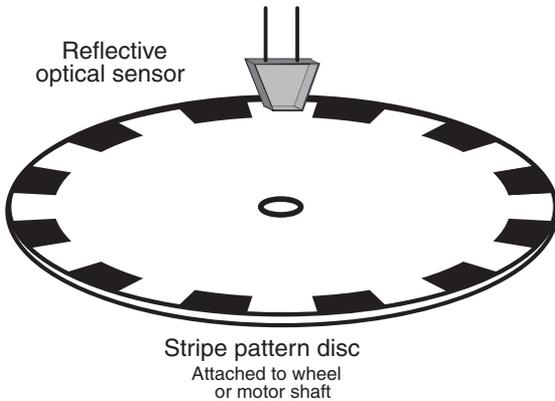


Figure 45-8 Single-stripe reflectance disc for wheel odometry. An emitter/detector pair is positioned over the stripe and senses the white/black pattern as the disc rotates.

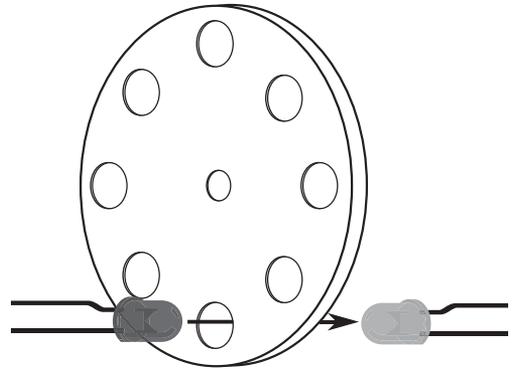


Figure 45-9 A transmissive disc uses a break-beam approach to sensing the stripe pattern. The disc may be optically opaque, with holes, or it may be a clear film, glass, or plastic, with solid black stripes coated over it.

When printing a pattern for a reflective codewheel, the paper should be thick (24# or more) gloss or semigloss. Photo-quality paper is a good choice to start. Be sure your printer has a relatively new ink or toner cartridge. Set the print quality to Extra Super Super High Fine (or whatever the highest setting is).

As an alternative, you can take your pattern to a professional copy center and have them make a print using a wax-based or other dense-medium toner/ink.

ANATOMY OF A TRANSMISSIVE ENCODER

A transmissive encoder is a disc that has numerous holes or slots near its outside edge. Rather than reflect light off a printed pattern, transmissive encoders pass light through the disc. An infrared emitter is placed on one side of the disc, so that its light shines through the holes or slots. An infrared-sensitive detector is positioned directly opposite (see Figure 45-9) so that when the disc turns, the holes pass the light intermittently. The result, as seen by the detector, is a series of flashing lights.

Transmissive encoders are harder to build because you need to cut or drill the holes/slots into a solid material. That material must be opaque to infrared light; many types of dark or black plastics actually let through infrared, making them unsuitable. A thin sheet of metal is the best overall choice, followed by an ABS or PVC plastic. Black-tinted polycarbonate or acrylic likely won't do.

You may be able to find already machined parts that closely fit the bill, such as the encoder wheels in a discarded mouse (the computer kind, not the live rodent kind). The mouse contains two encoders, one for each wheel of the robot.

ENCODER RESOLUTION

With both reflective and transmissive odometers, the number of stripes/holes/slots in the disc determines the resolution of the encoders. This in turn affects the accuracy of the distance measurement. The more stripes, holes, or slots, the better the accuracy.

For reflective encoders, increasing the number of stripes is straightforward. Depending on the type and quality of your printer, and the size of the pattern, you can easily make discs with 100+ stripes.

MEASURING DISTANCE

Odometry measurements can be made using a microcontroller that is outfitted with a *pulse accumulator* or *counter* input. These kinds of inputs independently count the number of pulses received since the last time they were reset. To take an odometry reading, you clear the accumulator or counter, then start the motors. Your software need not monitor the accumulator or counter. Stop the motors, then read the value in the accumulator or counter. Multiply the number of pulses by the known distance of travel for each pulse (this will vary depending on the construction of your robot: consider the diameter of the wheels and the number of pulses of the encoder per revolution).

If the number of pulses from both encoders is the same, you can assume the robot traveled in a straight line, and you have only to multiply the number of pulses by the distance per pulse.

Another method of odometry uses the timing between the pulses. In some ways this technique is easier and doesn't require your robot's controller to tally up the pulses it has received. Programming commands for timing pulses include *pulseIn* for the Arduino, and *pulsin* for the PICAXE. Downsides to this method:

- The pattern used in the codewheel should be exact. No hand-drawn art here. Use a commercially made codewheel, or print one out from a pattern, such as those on the RBB Online Support site.
- Pulse timing commands have a minimum resolution and depend greatly on the operating speed of the microcontroller. A PICAXE running at 4 MHz has a minimum pulse measuring resolution of 10 μ s, for example; at 16 MHz it's 2.5 μ s. For best results, use a code-wheel with no more than 6 to 16 stripes (or holes). This helps to maximize the duration of the pulses, because there aren't as many transitions during one revolution of the wheel.

MOUNTING THE CODEWHEEL AND OPTICS

With the codewheels made, you need to mount them to the wheel (or motor shaft) and secure the infrared emitter/detector optics.

Mounting the Codewheel Discs

- For reflective encoders, cut out the disc and paste it to the inside of the wheel. If the wheel isn't smooth enough, find or make a thin plastic disc that can act as a backstop. Unless the paper for the disc is absolutely opaque, best use a white-colored backstop.
- For transmissive encoders, mount the disc 1/4" or more away from the inside of the wheel. Use a spacer at the hub of the wheel to maintain separation.

Mounting the Sensor Optics

Using brackets, attach the infrared emitter and detector so they fit snugly near the codewheel. For reflective encoders, you'll want to place the sensor within about 1/8" to no more than 1/4" away from the disc. For transmissive encoders, be sure to orient the emitter and detector so that they face each other directly. You can bend the lead of the emitter and/or detector a bit to line it up with the holes.

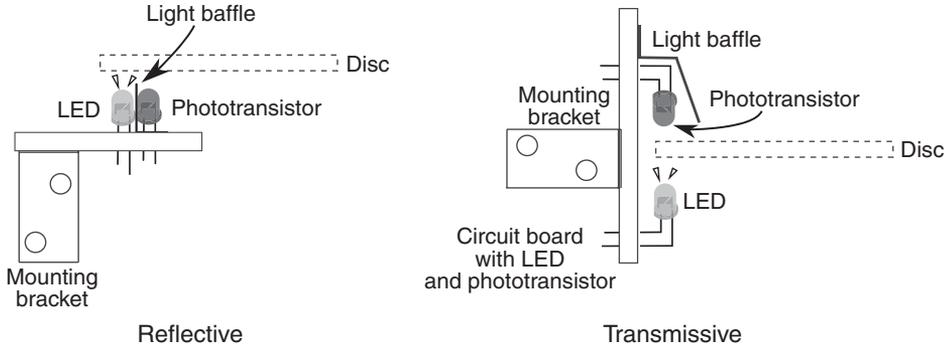


Figure 45-10 Mounting for both reflective and transmissive codewheels. For both, you may wish to use light baffles or tubes to prevent stray light from striking the detector.

For either encoder type, you'll want to mask the phototransistor detector with a light baffle (thick piece of black felt or something similar) so it doesn't pick up stray light or reflected light from the emitter. See Figure 45-10 for details. If the phototransistor you're using doesn't have a built-in infrared filter (and it should), you can increase the effectiveness of the sensor by adding an infrared filter over it.

ELECTRICAL INTERFACE AND CONDITIONING

Figure 45-11 shows the hookup diagram for the emitter and detector of an optical encoder. The same circuit works for both reflective and transmissive, and from now on, for simplicity, I'll assume you're using the reflective type.

The outputs of the phototransistors might need to be conditioned before they can be connected to a microcontroller. Otherwise, you could get all kinds of false signals and inaccurate readings. The circuit uses a *Schmitt trigger*, a kind of buffer circuit that smoothes out the wave shape of the light pulses. The output of the buffer is limited to only on and off. This helps prevent spurious triggers and provides a cleaner output.

The output of the Schmitt trigger is applied to the control circuitry of the robot. You can use either an inverting or a noninverting Schmitt trigger, as the circuit is interested only in on and off transitions.

The exact values of the resistors to use depend on the specifications of the infrared LED and phototransistor you use in your encoder. Try the values shown, and adjust as needed.

- The resistor above the LED controls the brightness of the LED: use a lower value to make the LED brighter; a higher value if the LED is too bright, and the phototransistor always

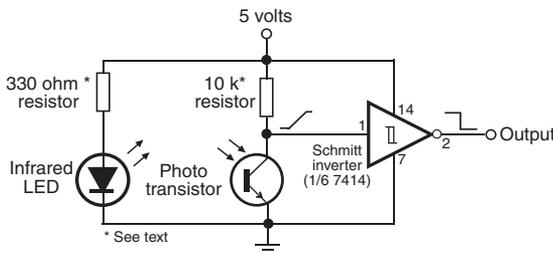


Figure 45-11 Electrical connection of infrared LED emitter and phototransistor used for wheel odometry, including a Schmitt trigger interface.

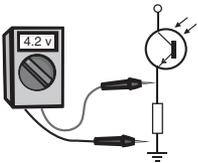
sees light. Be sure not to choose too low of a value for the resistor (anything below about 200 Ω) or the LED could be damaged from too much current.

- The resistor above the phototransistor determines the sensitivity of the phototransistor. A lower value makes the transistor less sensitive; a higher value, the inverse.



Note that the Schmitt trigger inverts the signal: as the output of the phototransistor *increases*, the Schmitt trigger changes from HIGH to LOW. If you'd rather it work the other way, flip the order of the transistor and its resistor (the resistor goes on the bottom). Or route the output of the trigger to another trigger—that inverts the inversion.

TESTING YOUR ENCODERS



To test your encoder, connect the phototransistor to a multimeter. Slowly rotate the wheel or motor shaft. If the system is working properly, you should see a definite voltage swing as the detector goes past the light and dark stripes. If you don't see much difference, double-check your work, and be sure the detector isn't being spoiled by room light (turn the work lamp on your desk away from the wheel).

- If the voltage stays the same and is *too high*, the emitter may be overdriven. Try a higher value for the resistor above the emitter LED. This reduces the output of the emitter.
- If the voltage stays the same and is *too low*, the emitter may be underdriven. Try a lower value for the resistor above the emitter LED—but never go below about 200 Ω , or the emitter may be damaged.
- If the voltage changes only slightly, then the stripe pattern may not be dark enough. The black stripes might be reflecting too much infrared light. Reprint the discs, being sure to use fresh toner or ink cartridges. If available, set the printing options to a darker level.



Not sure if the infrared emitters are emitting infrared? A quick check is to point a digital camera or camcorder at them. You should see a white glow from the emitter. Though invisible to the human eye, digital imaging sensors are sensitive to the light from infrared emitters.

PROGRAM FOR COUNTING PULSES

Your wheel encoders are now ready to rumble and need only be connected to a microcontroller or other circuit. The program listing in *single_encoder.pde* shows how to implement a basic pulse-counting odometer using an Arduino microcontroller.



`single_encoder.pde`

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details.

USING QUADRATURE ENCODING

So far, we've investigated encoders that have just one output. This output pulses as the code-wheel spins. By using two emitters and two detectors, positioned 90° out of phase from one another (see Figure 45-12), you can construct a system that tells you not only the amount of travel, but the direction as well.

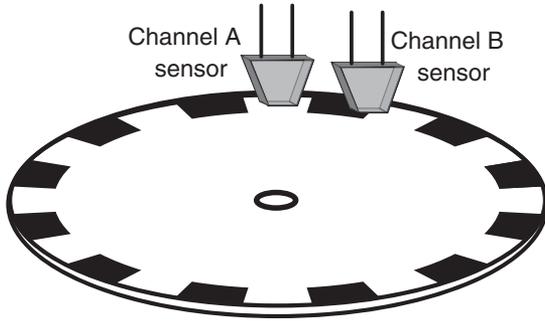


Figure 45-12 Single-stripe reflectance codewheel discussed for quadrature encoding. The emitter/detector pairs are placed so that one is 90° out of phase with the other.

Being able to determine the direction of travel is useful if the wheels of your robot slip. You can determine if the wheels are moving when they aren't supposed to be, and you can determine the direction of travel. This so-called two-channel system uses *quadrature encoding*—the channels are out of phase by 90° (one-quarter of a circle).

Rather than just counting on/off transitions, quadrature encoding counts all the variations between the two sensors:

```
off/off
on/off
on/on
off/on
( . . . and repeat)
```



It might not be immediately obvious, but because quadrature encoders trigger on four events instead of just one, the effective resolution of the odometers is increased by four. That is, if the disc has 10 stripes, rather than a resolution of 10 transitions per revolution, you get 40.

Figure 45-13 shows the on/off waveform of a quad encoder. The two sets of stripes on the disc define the channels of the encoder, referred to as Channel A and Channel B. Note that in one direction, Channel A “leads” Channel B; when the disc spins in the other direction, Channel B leads Channel A. This is how direction can be determined in a quad encoder setup.

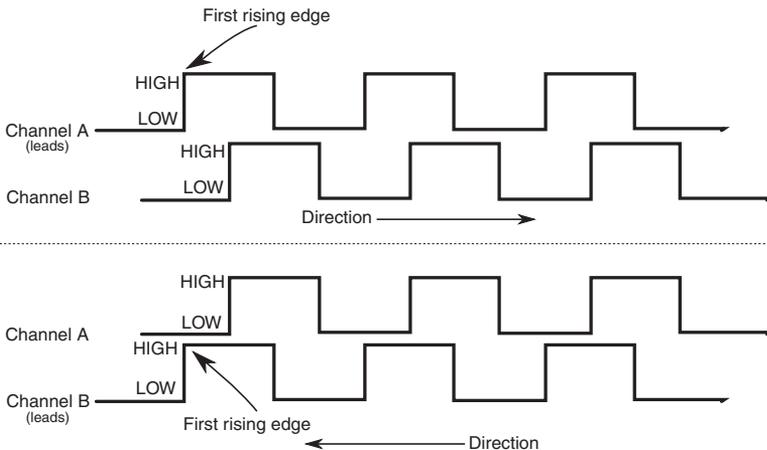


Figure 45-13 How the timing relationship between the Channel A and Channel B pulses indicates the direction of rotation in a quadrature encoder.

The “secret” behind quadrature encoding is that the two channels are set apart by 90°. Starting at the top of the disc, there is a white stripe, followed by a black stripe; think of these two segments as a whole circle. The stripes on the inside are 90° out of phase with the stripes on the outside.

Another depiction of quadrature timing and the “phases” that are created is Figure 45-14— here, the stripes are laid out horizontally to aid in understanding the sequence. The signals from the two channels rise and fall between the white and black stripes.



If you’re using quadrature encoding, you’re best off using a specialty integrated circuit for signal conditioning. These circuits are specially made to condition, equalize, and process the pulses from encoders that use two sensors instead of just one. See the RBB Online Support site (see Appendix A) for a free bonus project that uses an affordable and easily attainable IC specially designed for quadrature encoding.

USING READY-MADE ENCODERS

You’re not limited to using just homebrew encoders. Commercially made optical, magnetic, and mechanical encoders are available, some of them are affordable, and a few of them are readily adaptable to robotics.

Perhaps the premier solution to quadrature encoding for robotics is the Wheel Watcher (see Figure 45-15), developed by Pete Skeggs at Nubotics. The product is sold through Acroname and many other online specialty robotics retailers. The Wheel Watcher consists of a circuit board with all optics and signal-conditioning circuitry already added, plus a professionally printed striped codewheel.

Versions are available for use with standard R/C servo motors, as well as popular DC gear motors such as the GM2 and GM3 imported by Solarbotics. The codewheel is designed to fit several popular 2.5”-diameter injected molded wheels, and a spacer kit is available to secure the disc to most any other wheel that is attached to an R/C servo with a standard servo mounting horn.

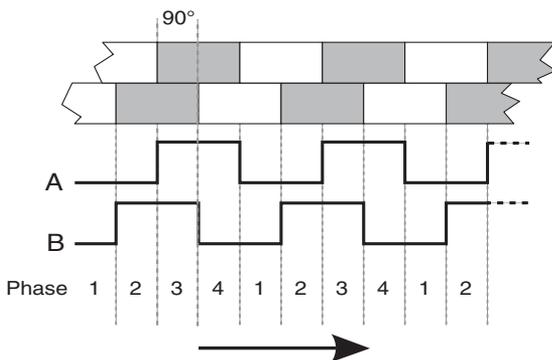


Figure 45-14 In quadrature encoding, the signals go through four phases, noted by the four possible values of the two channels: 0/0, 0/1, 1/1, and 1/0.

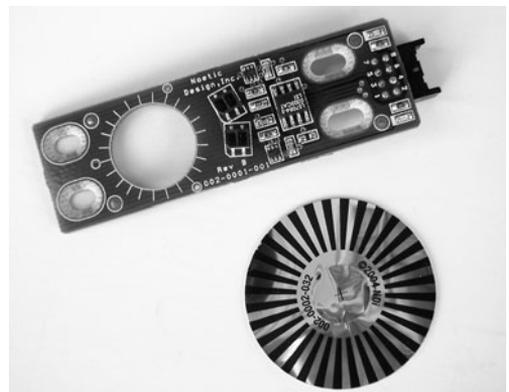


Figure 45-15 The Wheel Watcher module and accompanying reflective codewheel, made of black ink printed onto metalized plastic foil. The module provides an all-in-one quadrature encoding solution.

Other low-cost commercial odometers include kits from US Digital (www.usdigital.com) that work with motor shafts from 1.5mm to 4mm (0.59" to 0.157"). These cost a bit more than other solutions, but they are still relatively inexpensive, considering their high quality and high accuracy. Several online robotics stores, including Lynxmotion and Pololu, offer similar products.



US Digital also sells transmissive optical discs of various diameters, reflective encoder modules with all the optics built in and properly spaced, and many other solutions. Perfect for any intermediate or advanced robotics project.

UNDERSTANDING ERRORS IN ODOMETRY

Odometers aren't perfect. Over a 30- to 50-foot range it's not uncommon for the average odometer to misrepresent the position of the robot by as much as half a foot or more!

Why the discrepancy? First and foremost: Wheels slip. As a wheel turns, it is bound to slip, especially if the surface is hard and smooth, or when there's an obstacle on the floor. Second, certain robot drive designs are more prone to error than others. Robots with tracks are steered using slip, odometry is basically useless on most any tracked vehicle design.

And there are less subtle reasons for odometry error. If you're even a hundredth of an inch off when measuring the diameter of the wheel, the error will be compounded over long distances.

You're best off relying on odometry for short distances only. Assume that if your robot has made a turn, the actual distance traveled may be a bit different from what's being reported.

WHEN YOU DON'T NEED ODOMETRY

Keep in mind that some robotics applications simply don't need odometry, saving you the trouble of incorporating it into your designs. A good example is line-following robots. These are navigated using a predefined line, so their route is known.

Other examples include wall-following bots, though in some cases these can be augmented by simple distance measuring. And for obvious reasons, robots that walk on legs, swim over water, or fly in the air aren't suitable for the type of odometry covered here.

Compass Bearings

Besides the stars, the magnetic compass has served as humankind's principal navigation aid over long distances. You know how it works: a needle points to the magnetic north pole of the earth. Once you know which way is north, you can more easily reorient yourself in your travels.

Robots can use compasses as well, and a number of electronic and electromechanical compasses are available for use in hobby robots. One of the least expensive is the Dinsmore 1490, from Dinsmore Sensors. The 1490 looks like an overfed transistor, with 12 leads protruding from its underside. The leads are in four groups of three; each group represents a major compass heading: north, south, east, and west. The three leads in each group are for power, ground, and signal. A Dinsmore 1490 compass is shown in Figure 45-16.

The 1490 provides eight directions of heading information (N, S, E, W, SE, SW, NE, NW) by measuring the earth's magnetic field. It does this by using miniature Hall effect sensors and

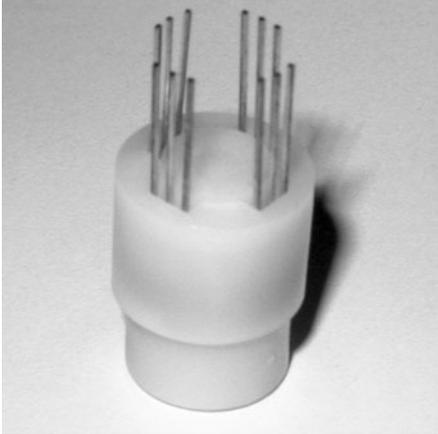


Figure 45-16 Dinsmore 1490 electromechanical compass. It has four outputs and can provide eight different compass bearings: N, E, S, W, and the in-between headings.

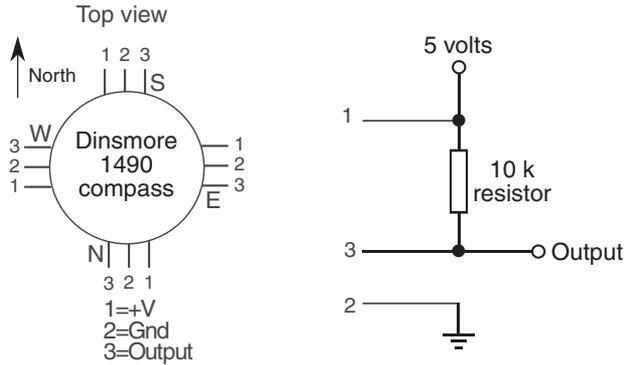


Figure 45-17 Connection diagram for the Dinsmore 1490 compass. Each set of three pins is connected in the same manner.

a rotating compass needle (similar to ordinary compasses). The sensor is said to be internally designed to respond to directional changes much like a liquid-filled compass. It turns to the indicated direction from a 90° displacement in approximately 2.5 seconds. The manufacturer's specification sheet claims that the unit can operate with up to 12° of tilt with acceptable error, but it is important to note that any tilting from center will cause a corresponding loss in accuracy.

Figure 45-17 shows the circuit diagram for the 1490, which uses four inputs to a computer or microcontroller. Note the use of pull-up resistors. With this setup, your robot can determine its orientation with an accuracy of about 45° (less if the 1490 compass is tilted). Dinsmore also makes an analog-output compass that exhibits better accuracy.



Reversing the polarity of the leads of this device will damage it. Be sure to double-check your work before applying power. Figure 45-17 shows the pinout of the 1490 from the top, same as the datasheet provided by the manufacturer.

Electronic compasses with digital or analog outputs are now fairly common. A popular model is the Devantech CMPS09 compass, which is equipped with a three-axis magnetometer, a three-axis accelerometer, and its own microcontroller. The output of the sensor provides a compass bearing, as well as pitch and roll information (useful if robot is not level).

The sensor connects to your microcontroller using I2C or serial communications, or via PWM, where the width of pulses indicates bearing, from 0° to 360° , in 0.1° increments. The sketch in *compass.pde* shows the CMPS09 interfaced to an Arduino using IC2. The connection diagram is shown in Figure 45-18. Be very careful not to flip the 5V and Gnd connections. Note the $1.8\text{ k}\Omega$ pull-up resistors: these are shown in the manufacturer's example demos, but not everyone agrees on the "perfect" value for pull-up resistors used with I2C devices. If $1.8\text{ k}\Omega$ resistors give you problems (and they shouldn't unless your wiring is really careless), try slightly higher values, up to $4.7\text{ k}\Omega$.

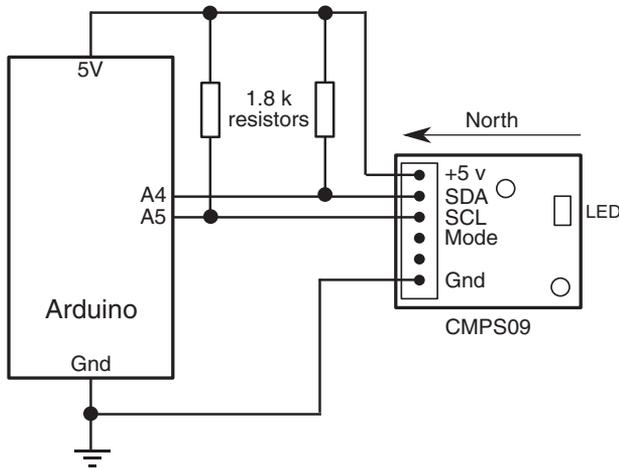


Figure 45-18 Connecting a Devantech CMPS09 compass to an Arduino. The CMPS09 uses I2C serial communications to provide reliable heading information.

```
101010
010101
101010
010101
```

compass.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details.



Compasses require careful placement on the robot to be useful. Because they sense magnetic fields, the compass must not be located near any large metal mass or near the strong magnetic fields of DC motors.

The *compass.pde* sketch also demonstrates retrieving pitch and roll information from the three-axis accelerometer that’s built into the CMPS09 compass. The built-in accelerometer is used to compensate for the tilt of the sensor, as a crooked compass will distort the reading of the compass. But you can also query the CMPS09 to display pitch (forward and back) and roll (side-to-side) position.

Experimenting with Tilt and Gravity Sensors

As every schoolchild learns, the human body has five senses: sight, hearing, touch, smell, and taste. These are primary developed senses; yet the body is endowed with far more senses that are often taken for granted. These more “primitive” human senses are typically termed “sixth senses”—a generic phrase for senses that don’t otherwise fit within the common five.

One of the most important “sixth senses” is the sense of balance. This sense is made possible by a complex network of nerves throughout the body, including those in the inner ear. The sense of balance helps us to stand upright and to sense when we’re falling.

Our sense of balance combines information about both the body’s angle and its motion. At least part of the sense of balance is derived from a sensation of gravity—the pull on our bodies from the earth’s mass.

Similarly, robots can be made to “feel” gravity. The same forces of gravity that help us to stay upright might provide a two-legged robot with the same ability. Or a rolling robot—on wheels or tracks—might avoid tipping over and damaging something by determining if its angle is too steep.

SENSORS TO MEASURE TILT

One of the most common means for providing a robot with a sense of balance is to use a tilt sensor or tilt switch. The sensor or switch measures the relative angle of the robot with respect to the center of the earth. Here are some of the more common methods:

- A *mercury-filled glass ampoule* that forms a simple on/off switch. The liquid mercury makes or breaks a connection to electrical contacts depending on the angle of the ampoule. The major disadvantage of mercury tilt switches is the mercury itself, which is a *highly toxic* metal.
- The *ball-in-cage* is an all-mechanical switch popular in old pinball machines. The switch is a square or round capsule with a metal ball inside. The weight of the ball makes it touch the electrical contacts that are inside the capsule. The capsule may have multiple contacts so it can measure tilt in all directions.
- The *ball-in-optical sensor* is a variation on the ball-in-cage. These use an infrared emitter and an infrared detector in a housing. Put a small ball inside between the emitter and detector, and as the device tilts, the ball either blocks or allows light to pass.
- An *electronic spirit-level sensor* uses the common fluid bubble you see on ordinary levels at the hardware store plus some interfacing electronics. It's basically a glass tube filled with fluid. A bubble forms at the top of the tube, since it isn't completely filled. When you tilt the tube, gravity makes the bubble slosh back and forth, and the position of the bubble indicates the angle.
- An *electrolytic tilt sensor* is like a mercury switch but is more complex and a lot more costly. A glass ampoule is filled with a special electrolyte liquid—that is, a liquid that conducts electricity but in very measured amounts. As the switch tilts, the electrolyte in the ampoule sloshes around, changing the conductivity between metal contacts.

ENTER THE ACCELEROMETER

One of the most accurate, yet surprisingly low-cost, methods for tilt measurement involves an accelerometer. Once the province only of high-tech aviation and automotive testing labs, accelerometers are quickly becoming common staples in consumer electronics.

New techniques for manufacturing accelerometers have made them more sensitive and accurate yet less expensive. A device that might have cost upward of \$500 a few years ago sells in quantity to manufacturers for under \$5 today. Fortunately, the same devices used in cars and other products are available to hobby robot builders, though the cost is a little higher because we're not buying 10,000 at a time!

The basic accelerometer is a device that measures *change* in speed. Many types of accelerometers are also sensitive to the constant pull of the earth's gravity. It is this latter capability that is of interest to us, since it means you can use the accelerometer to measure the tilt, or "attitude," of your robot at any given time. This tilt is represented by a change in the gravitational forces acting on the sensor.

SINGLE-, DUAL-, AND THREE-AXIS SENSING

The basic accelerometer is *single axis*, meaning it can detect a change in acceleration (or gravity) in one axis only. While this is moderately restrictive, you can still use such a device to create a capable and accurate tilt-and-motion sensor for your robot.

A dual-axis accelerometer detects changes in acceleration and gravity in both the X and Y planes. If the sensor is mounted vertically—so that the Y axis points straight up and down—the Y axis detects up-and-down changes, and the X axis will detect side-to-side motion. Conversely, if the sensor is mounted horizontally, the Y axis detects motion forward and backward, and the X axis detects motion from side to side.



A three-axis accelerometer detects changes in acceleration and gravity in all three planes, basically in 3D. These types of accelerometers are frequently used, along with gyro sensors, in certain kinds of self-balancing robots.

EXPERIMENTING WITH AN ACCELEROMETER

Of accelerometers for hobby use, the dual- or two-axis variety is probably the most common, with the ADXL family of components from Analog Devices among the most popular. The ADXL parts come as surface-mount components and require a few external components. Most folks get an ADXL-based accelerometer already mounted on a convenient *breakout* or experimenter's board, like the one in Figure 45-19. The price is not that much more than just the bare chip, and it's a whole lot more convenient. Just about all of the online robotics specialty outlets carry at least one dual-axis accelerometer, and most offer various models of the ADXL.

The ADXL accelerators include two- and three-axis versions, as well as components sensitive to different levels of gravity (or *g*) forces. For example, the ADXL320 is a dual-axis model, sensitive to $\pm 5g$. Generally, a $2g$ to $10g$ accelerometer will provide a good balance between precision and performance limits. Most robotics applications don't need to read gravity forces in excess of $5g$.



The higher the *g*'s, the higher the force, impact, and acceleration you can measure, but the less sensitive the component to small changes. A model rocket could use an $18g$ or $50g$ accelerometer, but most robots not destined for space can make do with a lot less.

One other variation in accelerometers is the output signal. The three most common are analog, serial, and PWM.

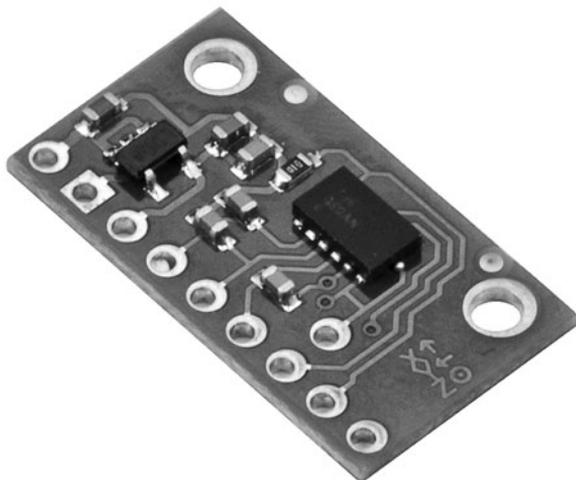


Figure 45-19 A three-axis accelerometer breakout board, ready to be used in a robotics project. This particular model is adjustable between $\pm 1.5g$ and $\pm 6g$, and provides an analog voltage proportional to the *g*-forces on the device. (Photo courtesy Pololu.)

- *Analog output* is a varying voltage, usually something a bit less than the full operating voltage of the accelerometer. Since accelerometers register both positive g-forces and negative g-forces, the output voltage at 0g is halfway between 0 and the operating voltage of the device.
- *Serial output* uses I2C, SPI, or other serial communications to relay the g-force readings to the microcontroller or computer. While a bit more difficult to use than analog output, the serial data is less prone to noise and glitches, and on some accelerometers you can send it commands on the serial link to alter its behavior.
- *PWM output* is a series of pulses whose duration changes depending on the g-forces on the accelerometer. To read the duration of the pulses, you need a microcontroller with an input capture (or similar) pin.

Figure 45-20 shows a typical connection of an ADXL-based analog output breakout board (in this case, that from SparkFun Electronics) to an Arduino microcontroller. (Note that the accelerometer in the figure is a self-contained module or breakout board. It's not just the chip itself.)

The breakout boards have X, Y, and Z outputs, though not all devices provide the third axis. The program *accel2.pde* demonstrates how incredibly easy it is to set up and read the X- and Y-axis values of a dual-axis analog output accelerometer. The values appear in the Serial Monitor window.

The circuit shows attaching power to the 5V pin of the Arduino. This assumes you are using an accelerometer meant for 5 volts. If you're using a 3.3-volt part, be sure to connect its power to the 3.3V pin of the Arduino.



Note also that 3.3-volt accelerometers that use PWM or serial output may not be able to directly connect to a microcontroller powered by 5 volts. You may need to use level-shifting logic to go from 3.3V to 5V. Consult the data sheet that comes with your accelerometer. The RBB Online Support site (see Appendix A) also provides hints and tricks regarding voltage level shifting.

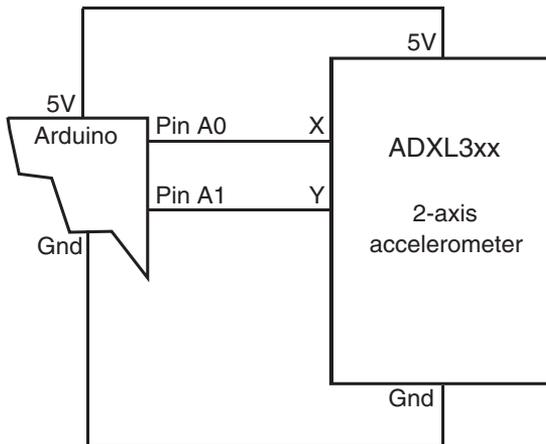


Figure 45-20 Arduino hookup diagram for an ADXL3xx accelerometer breakout module (one that already has the necessary external components on it). The illustration shows a dual-axis model; for the Z axis, simply add another line to an unused analog input.

```
101010
010101
101010
010101
```

accel2.pde

```
const int xpin = A0;           // X-axis to analog pin 1
const int ypin = A1;           // Y-axis to analog pin 2

void setup()
{
  Serial.begin(9600);          // Set up for Serial Monitor
}

void loop()
{
  Serial.print(analogRead(xpin)); // Show X-axis value
  Serial.print("\t");

  Serial.print(analogRead(ypin)); // Show Y-axis value
  Serial.println("");
  delay(100);                  // Delay between reads
}
```

The values shown in the Serial Monitor window represent voltage from 0 (0 volts) to 1023 (5 volts). At $0g$ (the axis arrow sideways; that is, pointing neither up nor down), the output of the axis is 512. This is midway between 0 and 5 volts. At $+1g$ (the axis arrow pointing up), the output rises to approximately 650. And at $-1g$ (axis arrow pointing down), the output lowers to 345.

These values were taken using an ADXL322, which has a $\pm 2g$ response. The exact values you get will be different, depending on the specific model of accelerometer you use. You'll also encounter moderate variations in output levels—even between the X and Y outputs—due to the tolerance of the components. On my prototype, the $+1g$ value was a full 10 points off between the X and Y outputs. This is to be expected, and you can adjust for it in your software.

Given these sample values, one real-world example is knowing when your robot has tipped over on its side, or is about to. Let's assume the X axis of the accelerometer represents side-to-side tilt (you can mount your accelerometer in a number of ways, so you can choose what axis does what). With the accelerometer flat, the X axis will read $0g$, or about 512. Any value above or below that reading indicates at least some tilt in the X axis. You might determine excessive tilt if the value is more than 75 points above or below 512.

ADDITIONAL USES FOR ACCELEROMETERS

Before we go into the details of using an accelerometer for angle or motion measurement, let's review the different robotics-based applications for these devices. Apart from sensing the angle of tilt, a gravity-sensitive accelerometer can also be used for the following tasks:

- *Shock and vibration.* If the robot bumps into something, the output of the accelerometer will “spike” instantaneously. Because the output of the accelerometer is proportional to the power of the impact, the harder the robot bumps into something, the larger the voltage spike. You can use this feature for collision detection, obviously, but in ways that far exceed what is possible with simple bumper switches, since an accelerometer is sensitive to shock from most any direction.
- *Motion detection.* An accelerometer can detect motion even if the robot's wheels aren't moving. This might be useful for robots that must travel over uneven or unpredictable ter-

rain, or if the motors should stall because of an obstruction or obstacle. Should the robot move (or stop moving) when it's not supposed to, this will show up as a change in speed and will therefore be sensed by the accelerometer.

- *Telerobotic control.* You can use accelerometers mounted on your clothes to transmit your movements to a robot. For instance, accelerometers attached to your feet can detect the motion of your legs. This information could be transmitted (via radio or infrared link) to a legged robot, which could replicate those moves. Or you might construct an “air stick” wireless joystick, which would simply be a pipe with an accelerometer at the top or bottom and some kind of transmitter circuit. As you move the joystick, your movements are sent to your robot, which acts in kind.

More Navigational Systems for Robots

For under \$100 your robot can know exactly where on earth it's located. Or precisely what room it's in within your house (or even what wall or furniture it's near). These techniques are described in a series of bonus articles you can find on the RBB Online Support site; see Appendix A. Included:

- *Radio frequency identification*, or RFID, uses small transponders that send signals to non-battery-powered tags you have mounted on nearby walls or objects. Each tag provides a unique code that identifies itself and, therefore, can be used as a kind of beacon system.
- *Global positioning satellite* (GPS) is by now a well-known technique using satellites in space to provide location, altitude, even speed information. You can apply the same technology on your robot.
- *Gyroscopes* sense changes in speed and direction, and provide what's known as inertial navigation. When coupled with other navigation techniques such as odometry and accelerometers, your robot can keep accurate track of where it is. Gyros (the instrument, not the food) are also used to construct self-balancing bots.

Making and Listening to Sound

The robots of science fiction are seldom mute or deaf. They may utter pithy sayings—“Don’t you call me a mindless philosopher, you overweight glob of grease!” Or squeak out blips and beeps in some “advanced” language only other robots can understand.

Voice and sound input and output make a robot more “human-like,” or at least more entertaining. What is a personal robot if not to entertain?

What’s good for robots in novels and in the movies is good enough for us, so this chapter presents a number of useful projects for giving your mechanical creations the ability to make and hear noise. The projects include using recorded sound, generating warning sirens, recognizing and responding to your voice commands, and listening for sound events. Admittedly, this chapter only scratches the surface of what’s possible today, especially with technologies like compressed digitized sound, and you’re encouraged to dive deeper into this interesting topic.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details. To save page space, the lengthier programs are not printed here. The support site also offers source code with added comments, parts lists (with sources) for projects, updates, extended construction plans, and more examples you can try!

Preprogrammed Sound Modules

At the bottom of the sound food chain is the preprogrammed, or “canned,” sound module, typical in such products as greeting cards and musical ornaments. Most are programmed with a song, though a few—like the electronic whoopee cushion—are meant to emit a sound effect. Humor notwithstanding.



Figure 46-1 An assortment of sound modules with prerecorded sounds and songs on them. Most modules can be hacked to operate from electronic control.

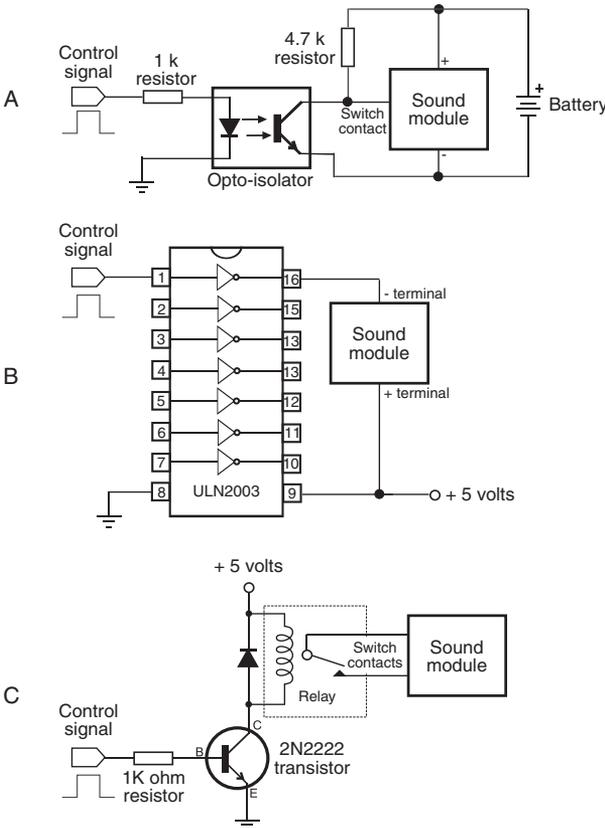


Figure 46-2 Some ways to connect a sound module to a microcontroller or other circuit for activation. Techniques include opto-isolator, power driver (turn battery supply on and off), and relay to replace the mechanical switch.

Most sound modules are completely self-contained, including a small speaker (piezo or dynamic magnet), a button battery, and some means to set it off, usually a small push-button switch. Several are shown in Figure 46-1. You can salvage the sound module from a greeting card or other product and reuse it in your robot. Craft stores are a good source of new sound modules that can be added to home-made ornaments.

Controlling a sound module from a microcontroller involves triggering the module by electrical signal, rather than by using its mechanical push button. Depending on the design of the module, you may be able to trigger it simply by replacing its push-button switch with the circuit shown in Figure 46-2A. An opto-isolator and accompanying parts bypass the switch. You trigger the module by momentarily bringing the microcontroller line that is connected to the opto-isolator to HIGH.

If the sound module is powered by more or less 5 volts, you can try hardwiring the trigger switch to on, and then control the module with operating juice from the microcontroller line itself (Figure 46-2B). If your microcontroller doesn't provide enough current to drive the module, try

adding one of the buffers in a ULN2003 chip. The driver also acts as a buffer to help protect the microcontroller.

And finally, if the module operates at only 1.5 volts (a single button battery), you may need to use something like Figure 46-2C. Yank the mechanical switch from the module, and replace it with a small relay.

Commercial Electronic Sound Effects Kits

Among premade electronic kits, ones for sound effects are always popular. Several companies manufacture and sell sound effects kits that you can use as self-contained modules in your robot projects. For example, the light-sensitive Theremin Kit from Chaney Electronics produces distinctly outworldly sound effects by altering the amount of light falling on two sensors. The company also sells a 10-note sound kit and several others.

Most sound effects kits are designed to be self-contained. That means they come with an amplifier, if they need one, and a speaker. Controlling them on a robot requires interfacing the selector buttons to the outputs of your microcontroller, in much the same way as for the sound modules in the previous section. Most kits come with a schematic, and you can readily determine where to hook things up.



Using a driver IC, as previously described, to power and depower the board is among the best methods to turn the sound on and off, as it reduces overall current draw from the robot's batteries when the sound effect isn't needed. But there are other methods, such as using a CD4051 analog switch to control which sound-producing element is routed to the amplifier and speaker.

Making Sirens and Other Warning Sounds

If you use your robot as a security device or to detect intruders, fire, water, or whatever, then you probably want the machine to warn you of immediate or impending danger. The warbling siren shown in Figure 46-3 will do the trick, assuming it's connected to a strong enough amplifier (some amp circuits are provided later in the chapter).

The circuit is constructed using two 555 timer chips; alternatively, you can combine the functions into the 556 dual-timer chip, but I prefer the separate chips because they provide a bit more room on the breadboard to experiment. The “warble speed” and pitch can be altered by changing the capacitors connected between pin 2 and ground of each chip.

- The timer on the left toggles HIGH and LOW about once a second. It makes the “warble.”
- The timer on the right produces the high-pitched siren sound and alternates between two frequencies. The two frequencies are controlled by the left timer, as it toggles HIGH and LOW.

When using an 8 or 16 Ω dynamic speaker, the sound output is pretty loud—enough for family members to come into your workshop and complain about it (yes, I speak from experience!). But if you need more oomph, connect the output of the second 555 to a high-powered amplifier. You can build an amplifier using an LM386 IC, as described later in the chapter, or

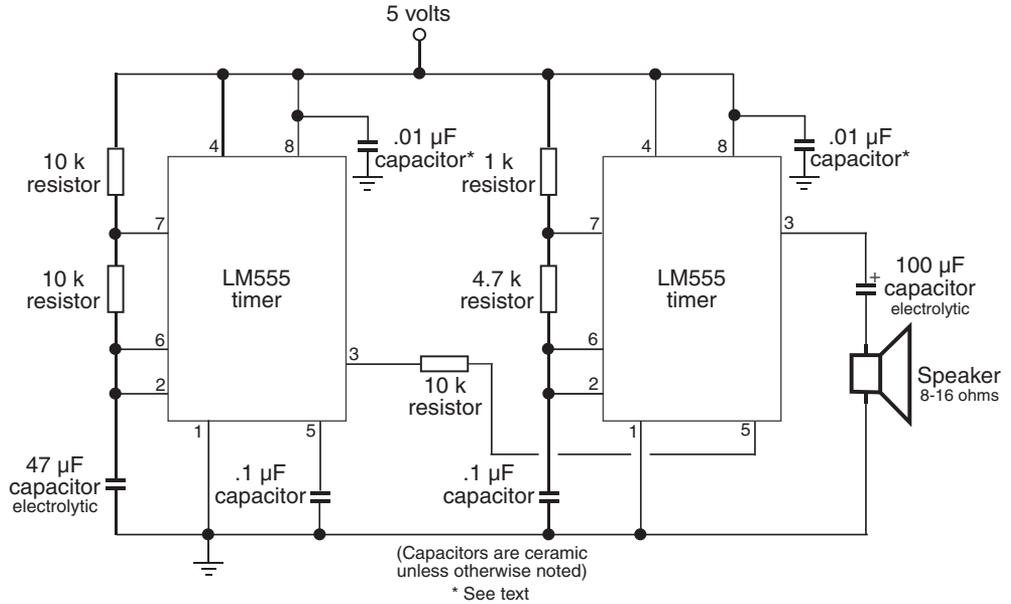


Figure 46-3 Circuit diagram for a warbling siren, made using two LM555 timer ICs.

get an audio amplifier kit with a wattage of 8, 16, or more. Start by looking at sellers of electronic kits in Appendix B, “Internet Parts Sources.”

The circuit needs several small-value ceramic (monolithic or disc) capacitors to prevent the inevitable power supply glitches that are generated by the versatile but electrically “noisy” LM555 chips. I’ve specified 0.1 μF capacitor as a bypass capacitor connected to pin 5 of the “warbler.” More critical are the 0.01 μF capacitors as power supply decoupling, connected between the power pins (pins 8 and 1) of both chips. Try to get these decoupling capacitors as physically close to the IC as you can.



If you breadboard this circuit, keep the lead lengths to a minimum, especially for the capacitors. After you have tested the circuit, you will want to transfer it to a soldered breadboard and clip all the leads as short as possible.

Using a Microcontroller to Produce Sound and Music

Any microcontroller with a pulse width modulation (PWM) feature can be used to produce sound, and even music. When the PWM frequency is within the range of hearing—about 20 Hz to 20 kHz—we hear it as a tone. The sound is heard by passing the I/O line through a speaker or amplifier.

By varying the tones, you make sound. You can produce music with one PWM output (monophonic), or you can combine the outputs of two or more PWM I/O lines to create polyphonic sounds, like a music synthesizer. It’s easy to produce warning sirens, warblers, bio-sounds (a la R2-D2), and other effects.

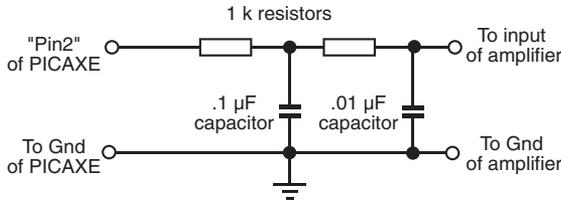


Figure 46-4 Recommended interface for bridging a PICAXE microcontroller with an amplifier circuit. You can use one of the amps detailed later in this chapter.

The PICAXE family of chips includes easy-to-use sound-making statements. Using the low-end PICAXE 08M, you can construct a programmable sound coprocessor for your robot, freeing your main microcontroller to handle the higher-level functions, such as motor control and sensors. The sounds are basic (don't expect Hollywood movie effects) but more than enough for functional effects and feedback.

Figure 46-4 shows an example interface of a single-channel PWM sound output to an amplifier. For the amp you can use the LM368, detailed later in this chapter under "Using Audio Amplifiers." The LM368 connects to a standard speaker with an 8 Ω or 16 Ω impedance.

Use the PICAXE *sound* statement to produce a tone at a certain pitch for a certain duration. The syntax of the *sound* statement is straightforward:

```
sound pin, (tone, duration)
```

- *pin* is the I/O pin number you've connected the amplifier to.
- *tone* is a numeric value of the pitch you wish to make. Valid values are from 0 to 255. See the table that follows for a quick reference of common tone values and the (approximate) musical scale notes they correspond to. (And note that you can pick in-between values for outworldly sounds.)
- *duration* is the duration of the tone, with the value of 50 being approximately 1 second.

Note	Tone Value	Note	Tone Value
A	49	A#	51
B	54	C	57
C#	61	D	65
D#	71	E	78
F	88	G	100

You can combine tones and duration in a single *sound* statement, to play successive notes:

```
sound 2, (57, 25, 71, 25, 78, 12)
```

plays C, D#, and E, with the first two notes lasting a half second, and the last note lasting a quarter second.

You can use loops of different kinds to produce sound effects. From the PICAXE manual, this short bit of code produces the high-low sound of a European police siren:

```
main:
  sound 2, (100, 30)
  pause 100
```

```

sound 2, (85, 30)
pause 100
goto main
    
```

Experiment with other tone and duration values to see what kinds of effects you can come up with.



The *sound* statement uses pulse width modulation to produce noise. The PICAXE 08M and higher chips also have separate *pwm* and *pwmout* statements that provide additional functionality—namely, you can control the duration of the pulses, not just their frequency.

When using an op-amp and basic components like a resistor and capacitor, you can modify the shape of the pulses to alter the *timbre* of the sound; timbre is the tone quality of the sound. For example, by converting the shape of the pulses to a sawtooth, the sound is more like that of a violin (a rough approximation, of course). Sound design and engineering are beyond the scope of this book, but if you’re interested, visit the local library and look up analog music synthesizers.

Using Audio Amplifiers

Some of your sound-generating circuitry won’t need amplification, in which case you can merely connect a speaker directly. But others may need a little help. Figure 46-5 shows a

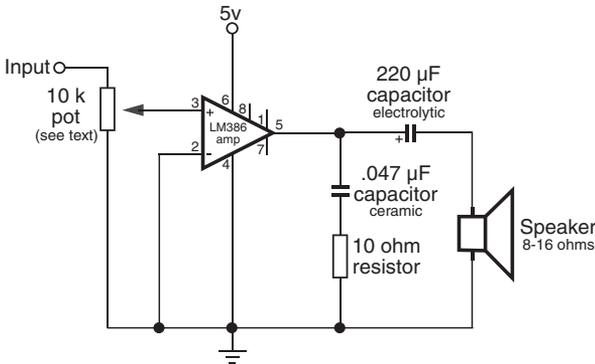


Figure 46-5 LM386 audio amplifier, wired for a gain of about 20.

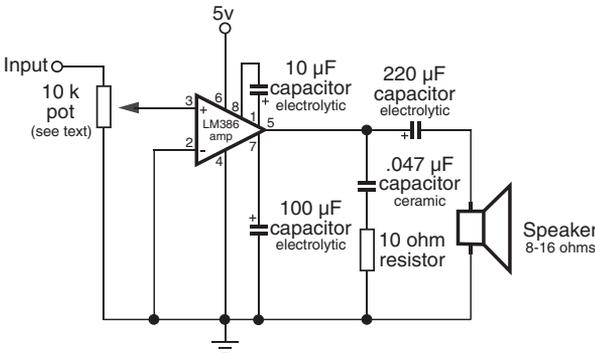


Figure 46-6 LM386 audio amplifier, wired for a gain of about 200.

rather straightforward 0.5-watt sound amplifier that uses the LM386 integrated amp. The sound output won't shatter any windows, but the chip is easy to get, is cheap, and can be wired up quickly. It's perfect for experimenting with sound projects.

The amplifier in Figure 46-5 has an amplification of approximately 20, and is perfectly suited to most robotic sound applications. By adding a few more components, the gain of the amp can be boosted 10 times, to 200 (see Figure 46-6). Either amplifier will drive a small 8- or 16-ohm speaker.



For best results, make sure the 10 k Ω potentiometer is the *logarithmic* (or *audio taper*) kind, not the typical linear taper. When using a linear taper pot, the sound volume will be affected at only one end of the dial. With a logarithmic potentiometer, the volume change will be more evenly spread across the dial.

Sound and Music Playback with a Microcontroller

My first robot used the mechanism from an old cassette recorder for its sound and music playback. It was, shall we say, clunky. Thanks to several ready-made add-on boards for the Arduino and other popular microcontrollers, you can incorporate melodies, effects, and other sounds where you have full control over what's played—from short, quarter-second gunshots to 10-minute-long lullabies.

These add-ons use prerecorded sounds that you create on your computer. The sound files, which can be in WAV, MP3, WMA, OGG, or a proprietary format, are then transferred to a solid-state memory card, typically SD or μ SD (micro SD, the tiny version of standard SD). The memory card is inserted into the playback board, which is controlled via simple commands from a microcontroller.

There are a number of breakout boards and shields for adding sound to the Arduino or other microcontroller. Most sound boards use solid-state memory cards to store sound, music, and effects files. The card is formatted in FAT16 (some support FAT32), meaning you can prepare the card on your computer and copy sound files to it.



The programming libraries to support reading the content of a memory card use up a lot of RAM in the microcontroller. For this reason, it's always a good idea to use these projects with an MCU equipped with at least 2KB of RAM, and 32KB or more of flash memory. When using the Arduino, you want the version with the ATmega328 (or higher) controller chip.

Typically, sound files are stored as 8+3 DOS-style filenames on the memory card. Sound files can be any size, up to the capacity of the memory card, but as you can only place files into the root directory of the card, you are limited to 512 files total. That's generally more than enough for any robotics application.

Some sound boards support the MP3 digital file standard. For example, the MP3 Trigger board from SparkFun (Figure 46-7) is a fully self-contained MP3 playback development module. It includes an onboard MP3 decoder chip (specifically, the VS1053), low-power amplifier (you can use an external amplifier if you need more volume), and a built-in μ SD card reader.

The MP3 Trigger supports memory cards formatted with DOS FAT16, so you can prepare your music and other sound effects on your PC, then transfer the files to the μ SD card for use



Figure 46-7 All-in-one MP3 playback board, which can be connected to a switch or a microcontroller for fully automated sound clip selection. It supports up to 256 sound clips, stored on a μ SD card. You can use your PC to copy sound files to the card. (Photo courtesy SparkFun Electronics.)

with the board. Playback is controlled via a serial or direct triggered input, using 18 trigger pins.

Speech Synthesis: Getting Your Robot to Talk

Not long ago, integrated circuits for the reproduction of human-sounding speech were fairly common. Several companies mass-produced these chips for voice-driven products like the Speak-and-Spell toys. In most cases, these ICs could create unlimited speech because they reproduced the fundamental sounds of speech.

With the proliferation of digitized recorded voice, unlimited vocabulary speech synthesizers have become something of an exception. Using only software and a sound card, it is possible to reproduce a male or female voice. In fact, both Windows and Macintosh OS X come with free speech-making tools for their operating systems.

SPECIALTY SPEECH CHIPS

While most of the big chip makers have long exited the speech chip market, there are several low-cost solutions available to robot builders. These include:

- SpeakJet, from Magnevation
- Soundgin, from Savage Innovations

Both create human voice effects using allophones, which are discrete sounds made by the mouth and vocal cords during speech. The allophones are essentially strung together to create what sounds like someone talking. Though not as clear as a real human voice, the staccato nature of the sound in fact adds to the effect. Your robot actually sounds like it's talking, rather than using prerecorded sound from somewhere.

Both the SpeakJet and Soundgin are also complex sound generators, including “biologic” sound effects, TouchTone phone tones, and alarm sounds. The chips contain independent

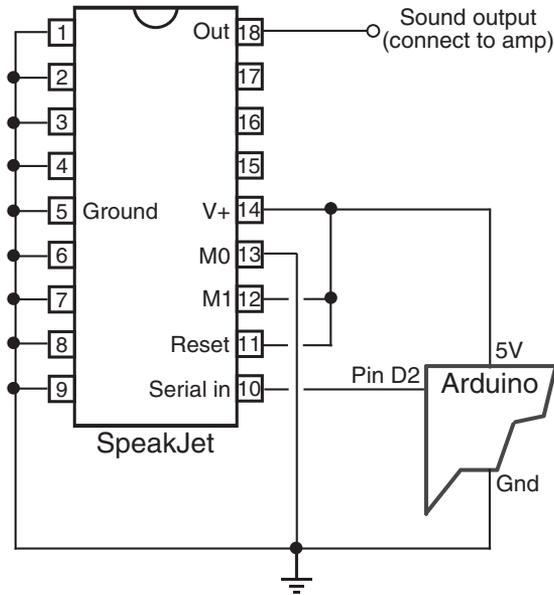


Figure 46-8 Basic connection for connecting a SpeakJet sound-processing IC to an Arduino. Serial data of the voice or sound to play is sent via pin D2 on the Arduino.

oscillators that can re-create the human vocal tract and sound effects, and, in the case of the Soundgin, can synthesize music.

Simple serial communication connects either speech chip to a microcontroller. The SpeakJet can also be commanded to play any of 16 preset phrases through its eight event inputs, which can be directly connected to bumper switches on the robot.

Figure 46-8 shows a minimal circuit for connecting the SpeakJet. I've left out the audio output components, amplifier, and speaker, as these are well documented in the SpeakJet datasheet. The following *speaket.pde* program for the Arduino microcontroller contains a simple phrase that says, "I am a robot." The allophone codes to use for the phrase were obtained with the Phrase-A-Lator program, available from the Magnevation Web site.



When wiring things together, make sure that the ground connections of the SpeakJet, PICAXE, and audio amplifier (if you use one) are all tied together, and that everything operates from the same 5 volt source.

```

101010 speaket.pde
010101 // Use SoftwareSerial library (comes with Arduino IDE)
101010 #include <SoftwareSerial.h>
010101

#define txSerial 2 // Serial out
#define rxSerial 3 // Serial in (not used for SpeakJet, but
// must be defined anyway
SoftwareSerial SpeakJet = SoftwareSerial(rxSerial, txSerial);

void setup() {
  pinMode(txSerial, OUTPUT);
  SpeakJet.begin(9600); // Standard SpeakJet baud rate
}

```

```
void loop() {  
  // Set up speech phrase ("I am a robot")  
  char phrase[] = {20, 96, 21, 114, 22, 88, 23, 5, 157, 132,  
  132, 140, 154, 128, 148, 7, 137, 7, 164, 18, 171, 136, 191};  
  SpeakJet.println(phrase); // Send phrase to SpeakJet  
  delay (4000); // Wait 4 seconds; repeat  
}
```

If your hookup doesn't produce any sound, try putting the SpeakJet into demo mode by connecting pin 13 of the SpeakJet (labeled M0) to 5V instead of to Gnd. The SpeakJet should announce "Ready" and then go through a routine of sound effects. Go back to regular mode by reconnecting pin 13 to 5V.

Be very careful to not place pin 12 of the SpeakJet to Gnd when in demo, or you may put the chip into Baud Configure mode. In this mode the SpeakJet will try to match whatever serial baud rate it detects on its serial input pin (pin 10). The baud rate is stored inside the chip and remains until reconfigured.

If you now get weird random sounds when trying to send the SpeakJet a speech phrase, a likely cause is that the baud rate of the chip is no longer the default 9600 baud and therefore doesn't match the serial data being sent to it by the Arduino.

Refer to the SpeakJet documentation for the details on reconfiguring the baud rate, but here it is in a nutshell: connect pin 13 to 5V, and pin 12 to Gnd. You'll know you're in Baud Configure mode when you hear a series of "sonar pings."

Replace the *char phrase* line in the *speaket.pde* sketch with the following code:

```
char phrase[] = {85};
```

Compile and download the sketch. Let the program run 10 to 15 seconds, then unplug power to the SpeakJet. Rewire pins 13 and 12 to those shown in Figure 46-8, and re-download the original *speaket.pde* sketch to the Arduino. Now, hopefully, the SpeakJet's baud rate will be set back to the proper 9600 baud.



Having to type in codes instead of text can be a real hassle. An alternative is the TTS256 Text to Code IC for SpeakJet and Soundgin, available from www.speechchips.com. This IC accepts words as text, and it in turn creates the allophone sounds for use with the speech synthesizer chips.

It won't pronounce every word correctly (in fact, the designer of the chip guarantees that it won't!), but for those tough-to-decode words, you can enter allophone sounds directly.

Listening for Sound

Next to sight, the most important human sense is hearing. And compared to sight, sound detection is far easier to implement in robots. Simple "ears" you can build in less than an hour let your robot listen to the world around it.

Sound detection allows your robot creation to respond to your commands, whether they take the form of a series of tones, an ultrasonic whistle, or a hand clap. It can also listen for the telltale sounds of intruders or search out the sounds in the room to look for and follow its master. Once detected, the sound can trigger a motor to motivate, a light to go lit, a buzzer to buzz, or a computer to compute.

MICROPHONE

Obviously, your robot needs a microphone (also called a *mic* or *mike*) to pick up the sounds around it. The most sensitive type of microphone is the electret condenser, which is used in most higher-quality hi-fi mikes. The trouble with electret condenser elements, unlike crystal element mikes, is that they need electricity to operate. Supplying electricity to the microphone element really isn't a problem, however, because the voltage level is low—under 4 or 5 volts.

Most all electret condenser microphone elements come with a built-in field-effect transistor (FET) amplifier stage. As a result, the sound is amplified before it is passed on to the main amplifier. Electret condenser elements are available from a number of sources, including RadioShack, for under \$3 or \$4. You should buy the best one you can. A cheap microphone isn't sensitive enough.

The placement of the microphone is important. You should mount the mike element at a location on the robot where vibration from motors is minimal. Otherwise, the robot will do nothing but listen to itself. Depending on the application, such as listening for intruders, you might never be able to place the microphone far enough away from sound sources or make your robot quiet enough. You'll have to program the machine to stop, then listen.

MICROPHONE AMPLIFIER

Use the circuit in Figure 46-9 as an amplifier for the microphone. The circuit is designed around an op-amp; the op-amp used is an OPA344, which you may not be familiar with. Unlike the ubiquitous LM741 op-amp, the OPA344 is intended to run using a *single-ended* power supply—that is, you don't need to provide both + and – power. It's also known as a *rail-to-rail* amp, meaning its output varies from 0 volts to the full supply voltage (or pretty close to it).

Sometimes it's just easier to work with circuits that are ready-made as modules or boards. I especially feel that way about sound amplifier circuits, which tend to work best when they're constructed on printed circuit boards—stray capacitance and all that jazz. Fortunately, circuit boards and modules for amplifying sound are common and relatively inexpensive, sometimes cheaper than if you buy all the parts separately.

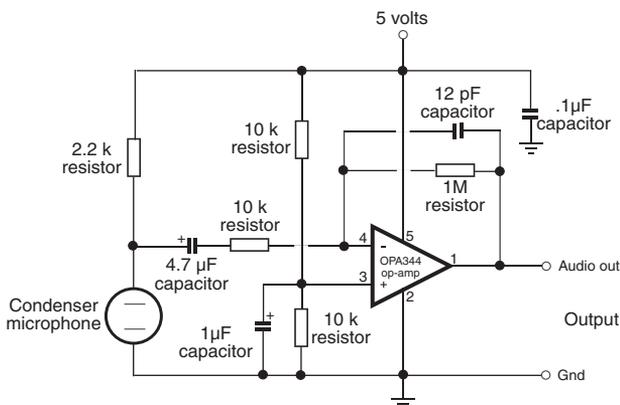


Figure 46-9 Microphone preamplifier circuit. You can build the amp yourself or get it ready-made. (Circuit courtesy SparkFun Electronics.)

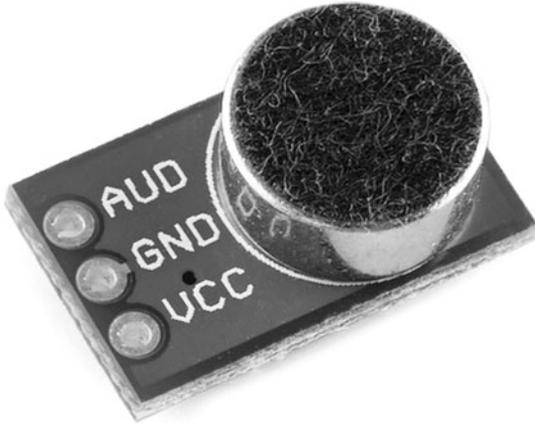


Figure 46-10 Electret condenser microphone and components for a preamplifier mounted on a breakout board. Supply power (see text), and connect the AUD terminal to a circuit or microcontroller to monitor sound. (Photo courtesy SparkFun Electronics.)

The little module in Figure 46-10 is a good example: it's the microphone, op-amp, and other parts shown in Figure 46-9, available as a teeny-tiny *breakout board* (model BOB-09964), ready for immediate use in your project. It comes preassembled, which is good because it uses small surface-mount parts that fly across the room when you sneeze. Just hook up the power (2.7 to 5.5 volts), and connect the AUD amplified audio output to the rest of your circuit.

CONNECTING TO YOUR MICROCONTROLLER

With the sound detector hardware finished, you can connect the electronics to a microcontroller. Figure 46-11 shows an interface that will conform the output signal from the amplifier to a level that is easier to use with a microcontroller analog-to-digital input. The capacitor is included to remove the steady DC voltage present at the amplifier output. The diode makes the output signal positive-going only.



When connecting to an Arduino, you may have better luck using an external power supply, rather than powering the microcontroller via the USB port. If this isn't practical, try powering the microphone module from the 3.3V pin of the Arduino pin, or else using a separate regulated 5-volt supply for the amp.

After downloading the sketch, open the Serial Monitor window to see the actual values registered through the amplifier. Assuming the module is powered by 5 volts, theoretically the

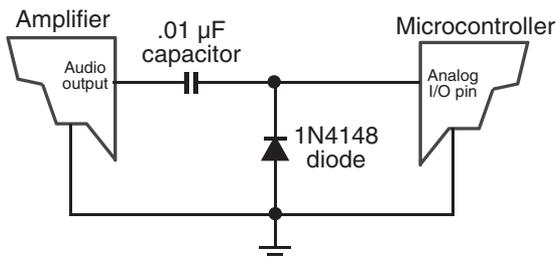


Figure 46-11 How to connect the microphone amplifier to a microcontroller. Be sure the ground connections are shared. See text for power source recommendations.

values will vary from 0 to 1023, representing the full 0- to 5-volt voltage swing of the amplifier output.

Note the *threshold* value set at the top of the sketch. Sounds over this level cause the Arduino's built-in LED to light up. Even with no sound in the room, the amp will deliver some voltage to its output—and more than you might think because of thermal noise, which is the “hiss” you hear when turn up an amplifier. Depending on your wiring, you should expect an ambient (quiet room) level of between 0 and 25 , especially if you experiment near your PC with its cooling fan. Therefore, set the *threshold* to some value above the ambient level, so that louder sounds trigger the LED.

```
101010
010101
101010
010101
```

sounddetect.pde

```
const int ledPin = 13;           // Built-in LED
const int soundSensor = A0;     // Audio output to pin A0
const int threshold = 105;     // Threshold for sound
int sensorReading = 0;

void setup() {
  pinMode(ledPin, OUTPUT);     // Make LED an output
  Serial.begin(9600);         // Use Serial Monitor
}

void loop() {
  sensorReading = analogRead(soundSensor); // Read sound
  // If sound level is over threshold flash LED
  if (sensorReading >= threshold) {
    digitalWrite(ledPin, HIGH);
    delay(300);
    digitalWrite(ledPin, LOW);
  }
  // Display sound level
  Serial.println(sensorReading, DEC);
  delay(50);
}
```



Sound metering is a science in its own right, and the preceding example is a simplistic way to monitor audio levels. One method that you might like to research includes adding a peak detector circuit, useful for catching those fleeting high-output sounds that might otherwise get missed. The peak detector stores the highest volume detected in that period. After checking the volume level, the circuit is cleared.

Peak detector circuits can be constructed using op-amps and a small assortment of standard parts (resistors, capacitors, diodes, and maybe a transistor). Research on the Web to see what you come up with!

On the Web: More Sound Projects

Check out the RBB Online Support site (see Appendix A) for more ideas and projects for using sound with your robots.

- How to use an LM339 voltage comparator to set the “trip point” of the sound level you want to capture. Because the output of the comparator is a digital LOW/HIGH signal, you don't have to use an ADC input on your microcontroller.

- How to literally “stretch out” the fleeting signal from the comparator so that your micro-controller is sure to catch it.
- How to hack inexpensive recording sound modules; put your own voice into your robot.
- How to route multiple sound sources to one amplified output.
- How to produce sound on a PC-based robot.
- How to use PC-based sound generation software to create synthesized speech, music, and robotic voice and special effects.
- How to implement speech recognition in your robot.

Interacting with Your Creation

Robots are like children—occasionally they just like talking back. Using any of several feedback techniques, your robot can communicate back to you, so you know that your programming is working the way it should. Several easy-to-implement feedback techniques are covered in this chapter.

There's another aspect of robot-to-human interaction, bridging that psychological gap between machine and person. You can draw in human spectators by using movement, sound, lights, and color. Previous chapters dealt with providing movement and sound; in these pages you'll learn some simple techniques to add pizzazz and flair (15 pieces or otherwise) to your robot. The more *personal* your bot is, the more others will take an interest in it.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, "RBB Online Support," for more details. To save page space, the lengthier programs are not printed here. The support site also offers source code with added comments, parts lists (with sources) for projects, updates, extended construction plans, and more examples you can try!

Using LEDs and LED Displays for Feedback

One light-emitting diode is all it takes for your robot to communicate with you. The language may not be elegant, and the conversations are strikingly short, but it gets the job done. When you don't need a talkative bot, you can use a single LED, or, for more words in the language, use multiple LEDs or 7-segment LED display panels.

FEEDBACK WITH ONE LED

The basic LED feedback circuit is shown in Figure 47-1: it's one of the I/O pins of your robot's microcontroller connected to a current-limiting resistor and an LED. Code running in your

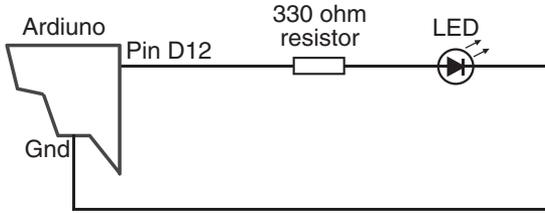


Figure 47-1 Basic connection for illuminating an LED via a microcontroller pin. Bringing the pin HIGH turns the LED on.

controller, like the supersimple program shown in *led.pde* (for an Arduino), turns the LED off and on. Vary the *delay* value to make the LED flash on and off at a faster or slower rate.

```
101010
010101
101010
010101
```

led.pde

```
void setup() {
  pinMode(12, OUTPUT);      // Make D12 an output
}

void loop() {
  digitalWrite(12, HIGH);   // Turn LED on
  delay(500);              // Wait 1/2 second (500 ms)
  digitalWrite(12, LOW);   // Turn LED off
  delay(500);
}
```

Remember that you're not limited to lighting an LED merely to show status—either good or bad. You can flash the LED, using various patterns, to communicate more variations. Use Morse code to relay a variety of conditions or to “speak” phrases.

If you don't know Morse code, you can invent your own system. Here are just a few ideas. Note that in each case, there's an *Off* pause between the three-flash sequence.

Sequence	Meaning
Short-Short-Short	Status A-OK
Long-Long-Long	Unknown trouble
Long-Short-Long	Battery low
Short-Long-Short	Cannot read sensors
Short-Long-Long	Goal reached
. . . and so forth	



Put the LED where it's easy to see, and select a component large and bright enough to make it visible from across the room. I like to use large, 5mm bright red or yellow LEDs, mounted on the top of the robot, that can be seen at any angle.

FEEDBACK WITH MULTIPLE LEDES

Use multiple light-emitting diodes when you want to quickly convey operating or sensor status. For example, you might light an LED each time one of the bump switches or proximity detectors senses an object.

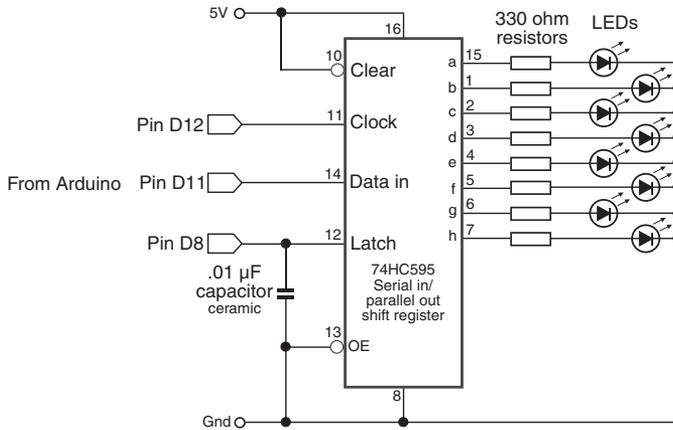


Figure 47-2 A 74595 serial-in, parallel-out (SIPO) shift register converts just a few microcontroller I/O lines to eight outputs.

When only a small number of LEDs are needed, you can wire them directly to the I/O pins of your microcontroller. You simply duplicate the circuit and code in the preceding section, “Feedback with One LED,” and use a different output pin for each light-emitting diode.

If you want to use more than four or five LEDs, then you probably don’t want to dedicate an I/O pin for each one. With a simple serial-in, parallel-out (SIPO) shift register, you can turn three pins into eight.

Refer to Figure 47-2 for a schematic using a 74595 SIPO integrated circuit. This chip is widely available and inexpensive. You can select any member of the ’595 family, such as the 74HC595 or 74HCT595, whatever is available to you.

Sample program code for the Arduino is shown in *multi_led.pde*. The shift register works by first setting the Latch line to LOW and keeping it there for the time being. Then 8 bits of data are sent, bit by bit, to the Data pin of the chip. For each bit, the Clock pin on the ’595 is toggled to tell the chip that new data have been sent.

The Arduino makes sending serial data easy because it packages up the Data and Clock activities in one simple statement, *shiftOut*. To use this statement you specify the number of the Data pin and Clock pin, the order of the data to be sent, and the value—from 0 to 255—that you wish to use.

```
shiftOut(dataPin, clockPin, MSBFIRST, numberValue);
```



MSBFIRST tells the Arduino that you wish to send the data starting with the *most significant bit*, which is the most common. For example, to send the value 127, the Arduino first converts it to binary form, which is 10000000. It then sends the bits left to right, starting with the 1.

The other variation, *LSBFIRST*, sends the *least significant bit* first, or right to left. Some circuits you interface to may require this order.

After all the shifted-out data have been sent, the program returns the Latch pin to HIGH. This sets the output pins of the 74595 chip, illuminating the LEDs as desired.

```
101010
010101
101010
010101
```

multi_led.pde

```
int latchPin = 8;           // Connected to Latch pin
int clockPin = 12;         // Connected to Clock pin
int dataPin = 11;         // Connected to Data pin
```

```

void setup() {
  // Set all pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // Count 0 to 255
  for (int val = 0; val <= 255; val++) {

    // Disable update during count
    digitalWrite(latchPin, LOW);
    // Shift out bits
    shiftOut(dataPin, clockPin, MSBFIRST, val);

    // Activate LEDs
    digitalWrite(latchPin, HIGH);

    // Short delay to next update
    delay(100);
  }
}

```

FEEDBACK WITH 7-SEGMENT LED DISPLAYS

Your robot can also talk to you using a 7-segment numerical LED display. You can light up the segments to produce numerals, in which case your robot can output up to 10 “codes” to indicate its status. For instance, 0 might be okay, 1 might be battery low, and so on.

You can also illuminate the segments to make nonnumeric shapes. You can light up each segment individually or in combination. Figure 47-3 shows some variations, including an E for Error, H for help, and numerous symbols that can mean special things.

Displaying Numerals

The easiest way to show numerals on a 7-segment display is to use a display driver IC, such as the CD4511 or 7447. These chips have four inputs and eight outputs—seven outputs for

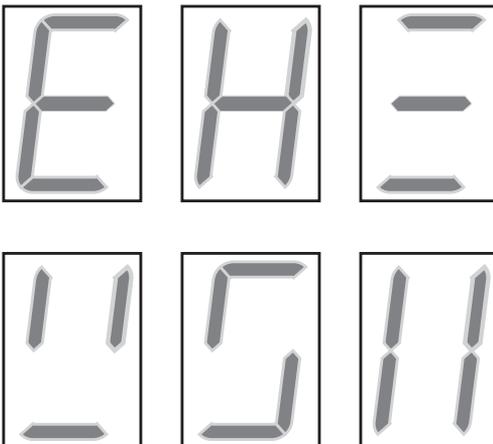


Figure 47-3 Unusual nonnumeric shapes can be created by activating selected segments of a 7-segment LED display. Use this system to display numbers (of course) or codes.

the numeric segments and an eighth for the decimal point, which we won't be using in this example. To display a number, set its binary-coded decimal (BCD) value at the A-D input lines.

4511 Inputs						
BCD	A	B	C	D	4511 Outputs	Numeral
0000	0	0	0	0	1111110	0
0001	1	0	0	0	0110000	1
0010	0	1	0	0	1101101	2
0011	1	1	0	0	1111001	3
0100	0	0	1	0	0110011	4
0101	1	0	1	0	1011011	5
0110	0	1	1	0	0011111	6
0111	1	1	1	0	1110000	7
1000	0	0	0	1	1111111	8
1001	1	0	0	1	1110011	9

Refer to Figure 47-4 for a diagram for hooking up the CD4511 to a common-cathode 7-segment LED display. See *segment.pde* for a simple Arduino sketch that sends different values to the CD4511, lighting up different segments. (You can use the same general concept with individual LEDs.)



The circuit shows a common-cathode 7-segment LED module; that is, all the LED segments in the display share the same cathode connection. Be sure yours is also a common-cathode display and not a common-anode display.

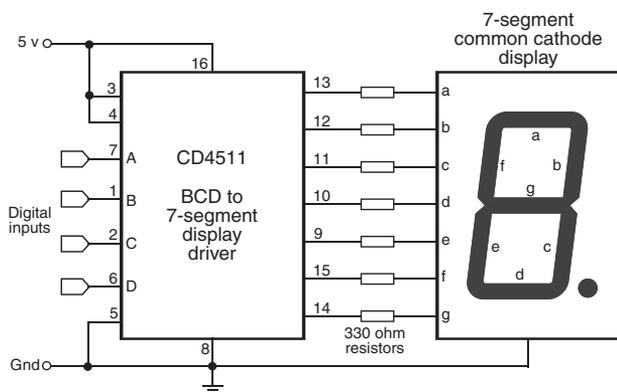


Figure 47-4 Connection diagram for a CD4511 BCD (binary-coded decimal) to a 7-segment driver. Numbers appear in the display based on the binary-coded values on the A to D input pins.

```
101010
010101
101010
010101
```

segment.pde

```
int outA = 8;           // Connected to A pin on 4511
int outB = 9;           // Connected to B pin
int outC = 10;          // Connected to C pin
int outD = 11;          // Connected to D pin

void setup() {
  // Set all pins to output
  pinMode(outA, OUTPUT);
  pinMode(outB, OUTPUT);
  pinMode(outC, OUTPUT);
  pinMode(outD, OUTPUT);
}

void loop() {
  // Display 3
  digitalWrite(outA, HIGH);
  digitalWrite(outB, HIGH);
  digitalWrite(outC, LOW);
  digitalWrite(outD, LOW);
  delay (1000);

  // Display 9
  digitalWrite(outA, HIGH);
  digitalWrite(outB, LOW);
  digitalWrite(outC, LOW);
  digitalWrite(outD, HIGH);
  delay (1000);
}
```

Displaying Arbitrary Shapes

Seven-segment displays are really just multiple LEDs that share a common cathode (or anode) connection. You can light up the segments separately, as in the section “Feedback with Multiple LEDs,” earlier in the chapter.

Refer to Figure 47-5 for how you can connect a 74595 SIPO chip to a 7-segment display. Use the program *segment_shapes.pde* to send out 8 bits to light the seven segments, plus decimal point. With this setup, you’re still able to produce all 10 digits, of course, plus Predator-type alien language symbols that only you (and your robot) understand.

```
101010
010101
101010
010101
```

segment_shapes.pde

```
int latchPin = 8;
int clockPin = 12;
int dataPin = 11;

void setup() {
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // Write a regular 7
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, B11100000);
  digitalWrite(latchPin, HIGH);
  delay(1000);
  // Write some funky character
  digitalWrite(latchPin, LOW);
```

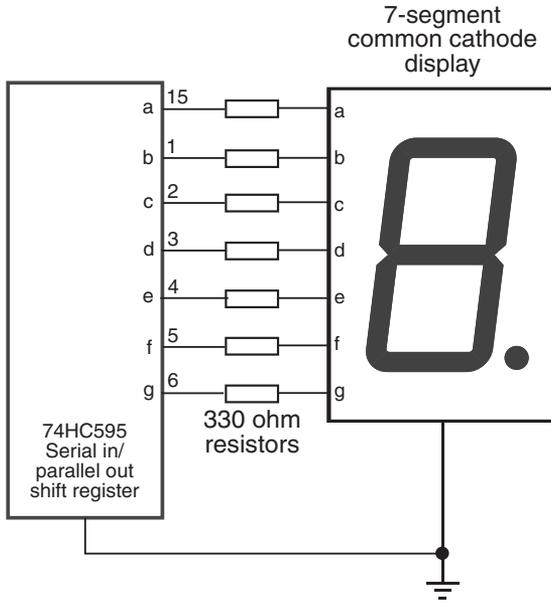


Figure 47-5 The 74595 SIPO (serial-in, parallel-out) shift register driving a 7-segment LED display. Using the shift register allows any combination of segments to light.

```

shiftOut(dataPin, clockPin, LSBFIRST, B01110011);
digitalWrite(latchPin, HIGH);
delay(1000);
}

```



You might find it easier to issue numbers in binary format, least significant bit first, as shown in the example. The first bit is segment A (see Figure 47-4), the second is segment B, and so forth. Here's a handy binary bits guide for producing standard numerals via the 74595 shift register. Even if you don't use the decimal point, be sure to include its bit at the end, or the display won't look right.

Binary Bits	Numeral
11111100	0
01100000	1
11011010	2
11110010	3
01100110	4
10110110	5
00111110	6
11100000	7
11111110	8
11100110	9

Feedback via Simple Sounds

Tones through a small speaker can provide auditory feedback. Connection is simple—you might not even need an amplifier—and so is the programming. You need a microcontroller that can produce pulse width modulated signals. Most can, using built-in commands. For this example I'll use the Arduino, which has sound-producing commands built in.



If you've used IBM or compatible PCs for any length of time, you'll recognize that this concept is just like the power-on self-test ("POST") diagnostics built into these computers. On power-up, but before the operating system was loaded, the PC would emit a series of short tones through its built-in speaker; the sequence of the tones indicated status.

See Figure 47-6 for a hookup diagram showing the Arduino connected to a piezo speaker. You can also connect the Arduino to a small amplifier to boost the sound; see Chapter 46, "Making and Listening to Sound." The sketch in *tones.pde* shows how to produce a couple of simple beeps for the purpose of communicating status. The number and meaning of the tones are completely up to you.

In *tones.pde*, the *tone* statement plays a note through a specified pin—in this case, digital pin D8, which is connected to the piezo speaker. The second parameter is the frequency in hertz (cycles per second). A frequency of 440 Hz is "concert pitch A" or "concert A." This is the A above middle C on a piano. A frequency of 880 Hz is also an A, exactly one octave higher.

The third parameter of the *tone* statement is the duration. The program plays both 440 Hz and 880 Hz A's for half a second, before waiting 1 second, and repeating. Over and over again, and over . . .

```
101010
010101
101010
010101
```

tones.pde

```
void setup() {
}

void loop() {
  tone(8, 440, 500);      // Concert pitch 'A', 1/2 second
  delay(500);
  tone(8, 880, 500);     // Octave higher
  delay(1000);           // Wait 1 second
}
```

Using LCD Panels

Liquid-crystal display (LCD) panels let your robot talk to you in complete words, even sentences. Even the smallest of LCD panels can show up to eight characters—enough for a couple of words. If needed, your robot can display a verbose error code that indicates its state.

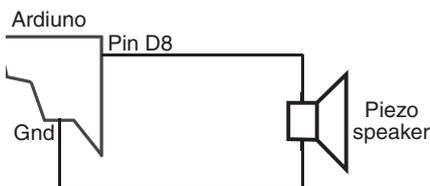


Figure 47-6 Basic connection diagram for driving a piezo speaker from an Arduino microcontroller pin. Sound is produced using the Arduino *tone* programming statement.

LCDs are particularly handy in testing and debugging when your robot is untethered from its programming computer. You can keep the LCD updated with the current program flow, indicating such things as when a sensor is activated and what program subroutine the robot's microcontroller is currently running.

TEXT- OR GRAPHICS-BASED

LCD panels are broadly available in two general forms: text and graphics. The differences are obvious, but let's discuss the two in brief:

- *Text-based* LCDs produce only text and other characters (such as dollar signs or symbols) stored inside the display module. The capacity of display is defined as the number of characters per line and the number of lines. For example, a 16×1 LCD has one line that can display up to 16 characters. A 32×2 LCD has two lines, and each line can display up to 32 characters.
- *Graphics-based* LCDs (GLCDs) are more like computer monitors, where text and other images are produced using a horizontal and vertical array of dots. The graphics LCD, or GLCD, is defined by the number of dots wide and high; 256×128 means the display panel has 256 pixels horizontally and 128 dots vertically.

Some GLCDs come with (or have as an option) a touch screen overlay, allowing you to press directly on the display to provide feedback. Most touch screens are resistive, meaning you can (and probably should) point on them using a rubber stylus.



Many text and graphics panels share standardized interface controllers, making it much easier to work with them. The controller is the electronics built into the LCD that provide the communications gateway.

For text displays, the most common controller is the Hitachi HD44780. For GLCD displays, the de facto standard controller is the Samsung KS0108. Other manufacturers make controllers that are compatible with these standards.

COLOR, MONOCHROME, BACKLIGHTING

Both text and graphics LCD products are available in either monochrome or color versions. Color is more common with GLCD, and color adds to the cost and the complexity in programming. Unless you specifically need it, stick with monochrome displays. (Note that the actual display color can be yellow on green, white on black, or numerous other variations.)

Many of the better LCD panels have their own backlighting, which increases the contrast under many types of indoor and outdoor light conditions. Though backlighting is not strictly required, it's a nice feature to have.

LCD INTERFACE TYPES

Before you can use an LCD panel, you must interface it to a microcontroller. There are two ways, parallel and serial:

- *Parallel interfacing* involves connecting separate I/O pins from the microcontroller to the LCD. Text-based LCDs require seven I/O lines; GLCDs require about 16 I/O lines, which

obviously makes these harder to interface. Your program communicates directly with the controller onboard the LCD panel.

- *Serial interfacing* involves connecting two or three I/O lines from the microcontroller to the LCD. Your program communicates with the LCD via serial commands. Additional electronics on the LCD convert these commands to the parallel interface used on the panel.

As text-based LCDs are by far the most common (and many controllers have built-in functions to support them), I'll concentrate just on these. The example that follows shows how to connect an Arduino microcontroller to an HD44780-based text LCD. To keep it simple, the example uses a 16×2 LCD, which is common and inexpensive.



There are few established standards for the command set used with serial LCDs, so if you're planning on using a serial LCD, refer to the instruction manual that came with it. It'll tell you how to connect the panel to your controller, how to set up communications (such as setting the baud rate), and how to send commands to the LCD.

Figure 47-7 shows the hookup diagram between the Arduino microcontroller and an HD44780-based 4-bit parallel-character LCD module. It also shows how to use a 10 kΩ potentiometer for adjusting the contrast of the display. Dial the pot for the clearest lettering against the display background.



Remember: Your LCD display needs to be compatible with the Hitachi HD44780 driver. Most are, but you'll want to check to make sure. The pinout order shown in Figure 47-7 is the most common you'll encounter, but variations exist in oddball LCD panels. Your panel may have 14 pins (no LED backlight) or 15 or 16 pins (with LED backlight).

Program `lcd16x2.pde` provides a basic programming example that displays “Robot Builder’s Bonanza. 4th Ed.” on the two lines of the display.

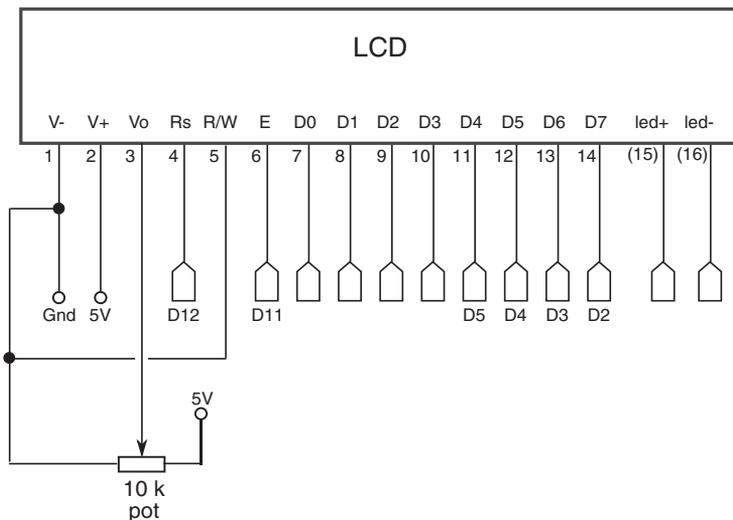


Figure 47-7 Pinout diagram for a standard 4-/8-bit parallel LCD character panel (the LCD must use the Hitachi HD44780 driver). The LCD is used in 4-bit mode to save I/O pins.

```
101010
010101
101010
010101
```

lcd16x2.pde

```
#include <LiquidCrystal.h>

// Initialize interface pins to LCD
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // Configure numbers/rows for LCD
  lcd.begin(16, 2);

  // Print the message

  lcd.setCursor(0, 0);
  lcd.print("Robot Builder's");
  lcd.setCursor(0, 1);
  lcd.print("Bonanza, 4th Ed.");
}

void loop() {
}
```

Once again, the 10 k Ω potentiometer adjusts the contrast of the display. For a quick setup, you can try simply tying pin 3 (Vo) of the LCD to Gnd. Or connect it to pin D9 of the Arduino, and add this line at the start of the `setup()` function:

```
analogWrite(9, 20);
```

This uses the PWM feature of the Arduino to set a very low voltage to pin 3 of the LCD. Play around with different values for the second parameter of the `analogWrite` statement, from 0 to 255. On many LCD panels, the higher the number, the less contrast there is.

Robot-Human Interaction with Lighting Effects

Few onlookers will ever ask you, “What does it do,” if your robot makes funny sounds or blinks lots of lights. Robot interaction includes making it interesting to humans. The more engaging the robot becomes, the more interaction it engenders. That increases the impact the robot has on its human watchers.

There are plenty of ways to attract attention to your robot, such as using musical tones and sound effects, and even making your bot stop every once in a while and do a little booty shake.

I’ll let you figure out the booty shake part (hint: it has something to do with starting and stopping the drive motors), but for music and other auditory effects, be sure to read Chapter 46, “Making and Listening to Sound.”

The premier method of adding light effects to your bot is with light-emitting diodes. LEDs have gone well beyond small, dim, red pinpoint of light. They’re now available in all colors of the rainbow. Brightness has been drastically improved to the point where LEDs are used as flashlights.

FYI

Read more about LED basics in Chapter 31, “Common Electronic Components for Robotics.” Included is how to select the current-limiting resistor used to prevent the LED from burning up.

MULTIPLE LEDs

Don't be content to use just one LED. Use multiple LEDs, of the same or a different color, mounted at various places on your robot. From a calculation standpoint, it's easiest to connect each LED to the robot power supply through its own current-limiting resistor. This also helps ensure an even brightness from each of the LEDs. As long as you stay under the maximum forward current specification of the LED, you can vary the value of the current-limiting resistor to change the brightness of each LED.

SUPERBRIGHT AND ULTRABRIGHT LEDs

The typical LED produces a fairly low amount of light—a few millicandles (a *candle* is a standard unit of light measurement; a millicandle is 1/1000 of a candle). Superbright and ultrabright LEDs produce 500 to 5000 millicandles; some go even higher. A few are so bright that they can cause eye damage if you stare into their beam.

Superbright and ultrabright LEDs are particularly striking on small robots. Turn down the lights and let your bot roam the floor. If you have a camera with an open-shutter (also called *open-bulb*) feature, you can take a long-exposure picture that shows the path of the robot around the room.

When selecting superbright and ultrabright LEDs, pay particular attention to beam pattern. The brightest LEDs have a narrow beam pattern—just 10 or 15 degrees. Select a broader beam pattern if you want the LED to be visible at different viewing angles.

Many very bright LEDs require more current than can be provided by the output pins of some microcontrollers. You'll need to boost the current to the LED using a transistor (Figure 47-8) or a current driver (Figure 47-9). The driver shown is the ULN2003 Darlington transistor array, where each of its seven drivers can provide up to 500 mA (half an amp) of current,

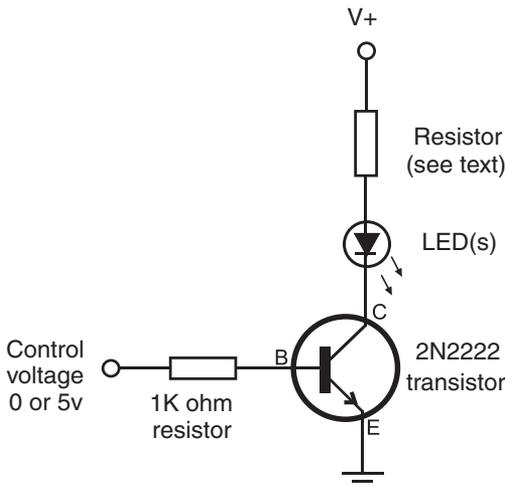


Figure 47-8 Superbright and similar high-output LEDs may require more drive current than a microcontroller I/O pin can supply. The transistor boosts the current driving the LED. Select the resistor to maintain a safe current through the LED.

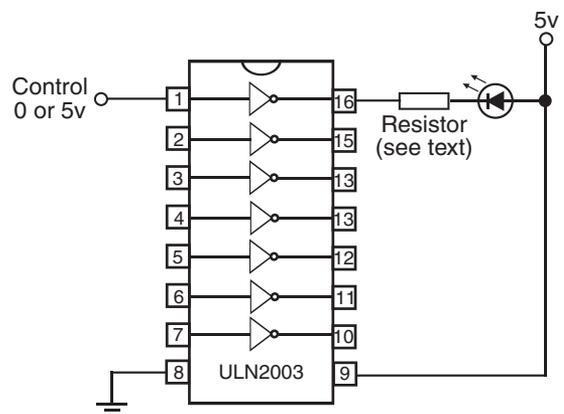


Figure 47-9 A buffer or driver may also be used to drive (one or more) LEDs. The ULN2003 chip contains seven high-output drivers, each capable of supplying several hundred milliamps of current.

depending on the version of the chip. If you need eight drivers, you can use the ULN2803. It's functionally identical to the ULN2003, except for the extra driver. Of course, the chip has more pins.

Again, remember: LEDs need a current-limiting resistor, or else they'll quickly burn out. Refer to Chapter 31 for the formula. In order to calculate the value of the resistor, you need to know the forward voltage through the LED, as well as the maximum current that can be safely passed through the device. Refer to the datasheet that came with the LED you're using.

MULTICOLOR LEDs

Some LEDs are engineered to produce more than one color. Chapter 31, "Common Electronic Components for Robotics," covered this topic, but here it is again in a nutshell. There are several kinds of multicolor LEDs:

- *Bicolor* LEDs contain red and green LED elements (other color combinations are possible, too). You control which color is shown by reversing the voltage to the LED. You can also produce a mix color by quickly alternating the voltage polarity.
- *Tricolor* LEDs are functionally identical to bicolor LEDs, except that they have separate connections for the two color diodes.
- *Multicolor* LEDs contain red, green, and blue LED elements. You control which color to show by individually applying current to separate terminals on the LED.

Figure 47-10 shows how to connect a bicolor LED to two pins of a microcontroller. To turn the LED off, put both pins LOW. To turn on one color or the other, put one of the pins HIGH.

Pin A	Pin B	LED Output*
LOW	LOW	Off
LOW	HIGH	Green
HIGH	LOW	Red
Pulse†	Pulse†	Orange

* The actual color depends on how you connect the LED. And, of course, some bicolor LEDs display colors other than red and green.

† When pulsing, one pin is LOW while the other is HIGH. Pulse at a rate of at least 10 times per second. That way, your eyes will blend flashes into a single combination color.

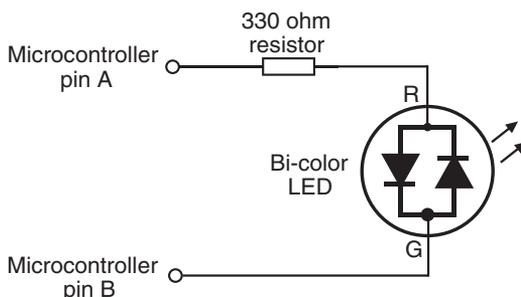


Figure 47-10 Connecting a bicolor LED to two I/O pins of a microcontroller. To display one of the colors, make its pin HIGH. Quickly alternating between the two pins produces a combination hue of the two colors combined.

As an example, the following code for the Arduino toggles between the two colors of a bicolor LED.

```
digitalWrite(led1, HIGH);
digitalWrite(led2, LOW);
delay(1000);
digitalWrite(led1, LOW);
digitalWrite(led2, HIGH);
delay(1000);
```

For the mixed-color effect, simply toggle between the two colors more quickly. Try reducing the delay from 1000 milliseconds (1 second) to 10 milliseconds (1/100th of a second).

Only one diode in a bicolor LED can be on at any time. But recall that in a tricolor LED, each diode has its own connection lead, so you can switch either on or off. You can also have both on at the same time, mixing the colors together. Figure 47-11 shows how to connect a tricolor (three-lead LED) to a microcontroller. Note that *each* diode in the LED gets its own current-limiting resistor. See *tricolor.pde* for a short demonstration program of toggling a tricolor LED from red to green to orange.

Pin A	Pin B	LED Output*
LOW	LOW	Off
LOW	HIGH	Green
HIGH	LOW	Red
HIGH	HIGH	Orange

* Once again, the actual colors depend on the LED you're using.

```
101010
010101
101010
010101
```

tricolor.pde

```
int redPin = 9; // Red diode of LED
int greenPin = 10; // Green diode of LED

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}

void loop() {
  digitalWrite(redPin, HIGH);
  delay(500);
  digitalWrite(redPin, LOW);
  delay(500);

  digitalWrite(greenPin, HIGH);
  delay(500);
```

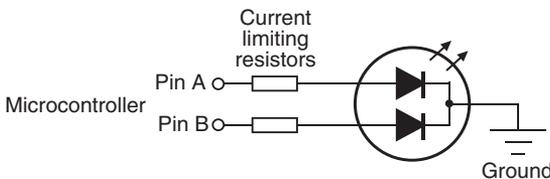


Figure 47-11 Connecting a tricolor LED to two I/O pins of a microcontroller. Either or both of the diodes can be turned on at once.

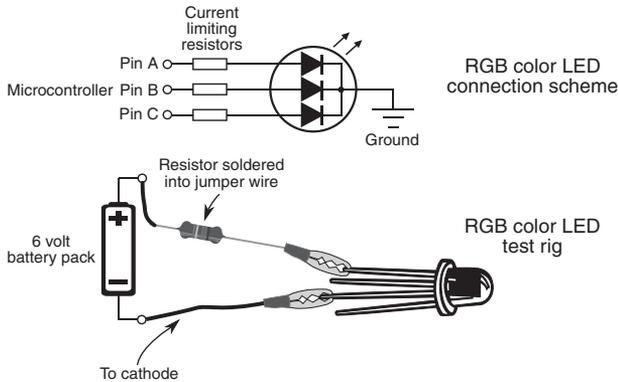


Figure 47-12 Connecting a multicolor (red-green-blue) LED to three I/O pins. Any color in the rainbow can be reproduced by altering the brightness of the three diodes.

```
digitalWrite(greenPin, LOW);
delay(500);

digitalWrite(redPin, HIGH);
digitalWrite(greenPin, HIGH);
delay(500);
digitalWrite(redPin, LOW);
digitalWrite(greenPin, LOW);
delay(500);
}
```

And finally, a multi- or RGB-color LED has red, green, and blue diodes. These three primary colors can be displayed independently or in different combinations to produce many other colors. You use these the same as with a tricolor LED, except that you need a third microcontroller pin to control the additional color.

Figure 47-12 shows the connection scheme for a multicolor LED, plus a testing rig so you can experiment with how it works. Solder a current-limiting resistor (for example, 330 to 470 Ω) inline with the jumper wire. Connect the – terminal of the battery to the cathode lead of the LED, and then touch each of the three anode leads in turn to watch the colors.

Recall that you can mix colors by turning on more than one diode in the LED at a time. With a microcontroller with a PWM (pulse width modulation) output, you can vary the intensity of the light and create thousands of color combinations. The program *rainbow_led.pde*, for the Arduino, shows how to use the *analogWrite* statement, which produces a PWM signal on specific pins of the microcontroller. We'll be using digital pins D9, D10, and D11 for connecting to a multicolor LED (don't forget the current-limiting resistors!).

```
101010
010101
101010
010101
```

rainbow_led.pde

To save space, the program code for this project is found on the RBB Online Support site. See Appendix A, "RBB Online Support," for more details.



The hookup diagrams for the tri- and multicolor LEDs show common-cathode devices—that is, the cathode (negative) ends of all the diodes in the device are tied together. Multiple-color LEDs are also available in common-anode style, where the anode (positive) ends are linked together. The concepts behind using these are the same, though, of course, you must reverse the wiring. The LED is turned on when the cathode connection is brought LOW.

ON THE WEB: MORE STUPID LIGHT TRICKS

And there are more ways you can trick out your robot with light effects. Find these and other ideas on the RBB Online Support site:

- Using electroluminescent (EL) wire, fiber optics, and lasers
- Ornamenting your robot with self-contained body lighting (glow-in-the-dark sticks, rave lights, magnetic LED earrings)
- Outfitting your robot with different colors of cold cathode fluorescent tubes
- Using passive decoration, such as decals, fluorescent paint jobs, and transfer film

Danger, Will Robinson!

Everyone complains that a robot is good for nothin'—except, perhaps, providing its master with a way to tinker with gadgets in the name of “science”! But here’s one useful application you can give your robot in short order: fire and smoke detection. As this chapter will show, you can easily attach sensors to your robot to detect flames, heat, smoke, and poisonous gases, making your robot a kind of mobile safety inspector.



Now, don't go treating your robot sentry as foolproof. The methods described here are for *experimental use only*. For reasons we'll get into, a robot may not be the best at detecting smoke or gas in a room; for true safety, leave these tasks to approved fire, gas, and smoke detector appliances designed for the job.



Source code for all software examples may be found at the RBB Online Support site. See Appendix A, “RBB Online Support,” for more details. To save page space, the lengthier programs are not printed here. The support site also offers source code with added comments, parts lists (with sources) for projects, updates, extended construction plans, and more examples you can try!

Flame Detection

Flame detection requires little more than a sensor that detects infrared light and a circuit to trigger a motor, siren, computer, or other device when the sensor is activated. As it turns out, almost all phototransistors are specifically designed to be sensitive primarily to infrared or near-infrared light. You need only connect a few components to the phototransistor and you've made a complete flame detection circuit.

Interestingly, the detector can “see” flames that we can’t. Many gases, including hydrogen and propane, burn with little visible flame. The detector can spot them before you can, or before the flames light something on fire and smoke fills the room.

DETECTING THE ULTRAVIOLET LIGHT FROM A FIRE

Of all the methods of detecting fire and flame, using ultraviolet light sensors is probably the most popular. UV flame sensors are made for the central heating system industry and used in gas and oil furnaces to detect proper pilot light and flame settings. You can use these same sensors in your robot to look for open flame. Be warned, however, that these sensors are not cheap, and you must always use a UV sensor with an appropriate amplifier.

The Hamamatsu R2868 Flame Detector UVTron sensor, and associated amplifier board, are among the most used flame detectors for amateur robotics. It’s among the least expensive, and available through Acroname and other robotics specialty online retailers. There are others, such as the Honeywell C7027, that perform the same task.

WATCHING FOR THE FLICKER OF FIRE

Fire flickers. You can use this *flame modulation* in a robot fire detection system to help differentiate what is a real fire and what is likely just sunlight streaming through a window or light from a nearby incandescent lamp. By detecting the rate of flicker from a fire and referencing it against known values, it is possible to reduce some of the false alarms.

The technique is beyond the scope of this book, but with a bit of ingenuity, you could create a simple flame-flicker system using just a phototransistor (don’t use a photoresistor), and microcontroller with an analog-to-digital input. The phototransistor will pick up the infrared and near-infrared light emitted by the fire.

Your program reads the value of the photoresistor 15 to 20 times a second, looking for the telltale pattern of flame over sunlight or incandescent lighting. The closer the patterns match, the greater the likelihood that there is a real fire. For example:

- If the intensity of the light doesn’t change, it’s either the sun or light from a steady source like a flashlight. Either way, it’s not a flame. (Well, the sun is one huge ball of flame, but it’s 83 million miles away, and as long as it stays at that distance, we’ll be okay.)
- If the light flickers at regular intervals, at the same intensity, it’s probably an AC-operated incandescent lamp or fluorescent light fixture, and, again, we’re not interested.
- If the intensity of the light constantly changes and flickers at random, there is a greater chance the light is from a flame.

DETECTING INFRARED FROM A FIRE OR FLAME

Fire emits infrared heat, some of which is in the near infrared (just beyond dark red) part of the electromagnetic spectrum, and some in the far infrared spectrum. A simple circuit for sensing near infrared from a fire is shown in Figure 48-1. It uses a phototransistor, which is naturally responsive to infrared light in the near infrared region—about 800 nanometers (nm) to about 1500 nm. For best results, place an IR-pass filter in front of the phototransistor to block extraneous light. An IR-pass filter is one that lets in infrared light, but blocks all others.

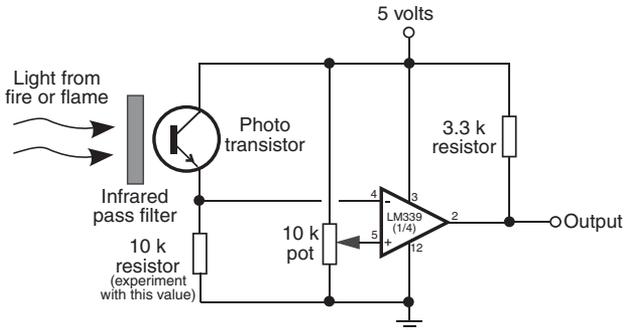


Figure 48-1 Basic test circuit for detecting the light emission of a nearby fire or flame. Adjust the 10 kΩ potentiometer for best sensitivity.

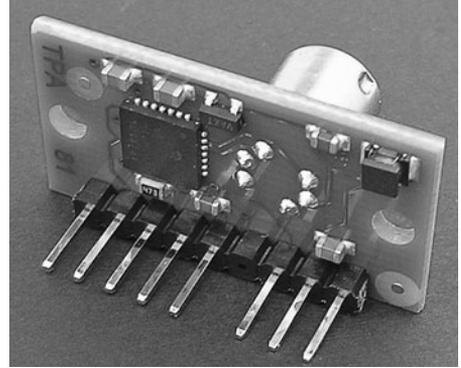


Figure 48-2 This thermopile sensor contains eight infrared detectors in an 8×1 array. It provides a serial output with the instantaneous value of each of the detectors. (Photo courtesy Devantech.)



Some phototransistors have their own built-in IR-pass filters. These are typically set to accept IR light at about 950 nanometers. If the phototransistor has a brown, dark blue, or dark red tint to it, then you know it already has an IR-pass filter built in.

You should be aware that this type of sensing circuit leaves much to be desired, as it will trigger on all kinds of light, including from the sun, an incandescent lamp, and your robot's own infrared navigation sensors. But it's a useful technique for general testing and experimentation when your robot lives in a controlled environment.

A better, but much more expensive, technique is to use a thermopile sensor, like the one shown in Figure 48-2. This particular unit contains eight infrared imaging sensors, aligned in a horizontal array. It's said to be able to detect a candle flame at 6 feet. Because it has more than just one sensor in it, it can provide a kind of simple mosaic of the heat pattern of whatever it's pointed at.

Fire that consumes any organic material burns hydrogen, which emits carbon dioxide, CO₂. This emission puts out a very distinct "spike" of infrared radiation at about 4300 nanometers (see Figure 48-3). By using a very specific IR-pass filter that blocks all radiation except that at 4300 nanometers, the sensor is effectively blind to background radiation, including natural and artificial sources.

Unfortunately, IR-pass filters for 4300 nanometers are not common and are frightfully expensive when purchased new. But you just might luck out and find one at some dank and dusty surplus joint near your hometown or in the storeroom at school.

Smoke Detection

"Where there's smoke, there's fire." For less than \$15, you can add smoke detection to your robot's long list of capabilities and, with a little bit of programming, have it wander through the house checking each room for trouble.

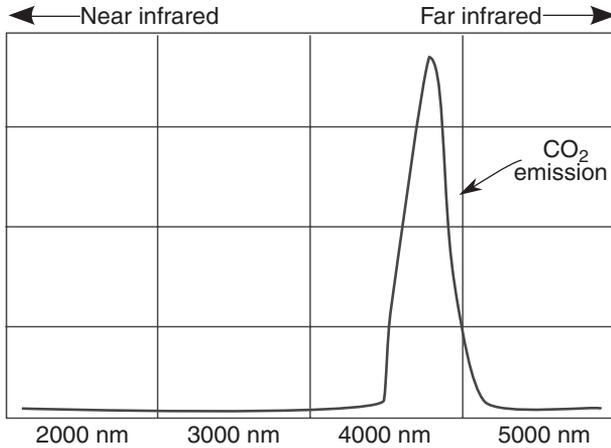


Figure 48-3 Carbon dioxide (CO₂) is emitted during a fire; this emission is accompanied by a “spike” in the 4300-nanometer infrared region of the electromagnetic spectrum.

You can build your own smoke detector using individually purchased components, but some items, such as the smoke detector cell, are hard to find. It’s easier to use a commercially available smoke detector and modify it for use with your robot. In fact, the process is so simple that you can add one to each of your robots. Tear the smoke detector apart and strip it down to the base circuit board.



Many smoke detectors employ an ionization chamber that uses a mildly radioactive substance, Americium 241. This human-made element has a half-life of over 400 years (it keeps going and going and . . .), but the ionization chamber can become ineffective after less than 10 years, due to dust and other contaminants. Using a very old surplus or used alarm will render your “Smokey the Robot” fairly useless at detecting the smoke of fires.

HACKING A SMOKE ALARM

You can either buy a new smoke detector module for your robot or scavenge one from a commercial smoke alarm unit. The latter tends to be considerably cheaper—you can buy quality smoke alarms for as little as \$7 to \$10. In this section, I’ll discuss hacking a commercial smoke alarm, specifically a First Alert model SA300, so it can be directly connected to a robot’s computer port or microcontroller. Of course, smoke alarms are not all designed the same, but on many, the basic construction is similar to that described here. You should have relatively little trouble hacking most any smoke detector you happen to use.

However, you should limit your hacking attempts to those smoke alarms that use traditional 9-volt batteries. Certain smoke alarm models, particularly older ones, require you to use AC power or specialized batteries. These are not suitable for use on your battery-powered bot.

Checking for Proper Operation

Start by checking the alarm for proper operation. If it doesn’t have one already, insert a fresh battery into the battery compartment. *Put plugs in your ears* (or cover up the audio transducer hole on the alarm). Press the “Test” button on the alarm; if it is properly functioning, the alarm should emit a loud, piercing tone.

If everything checks out okay, remove the battery and disassemble the alarm. Less expensive models will not have screws but will likely use a “snap-on” construction. Use a small flat-headed screwdriver to unsnap the snaps.

Getting Inside the Smoke Alarm

Inside the smoke detector is a circuit board that consists of the drive electronics and the smoke detector chamber.

Either mounted on the board or located elsewhere will be the piezo disc used to make the loud tone. Remove the circuit board, being careful you don't damage it. Examine the board for obvious “hack points,” and note the wiring to the piezo disc. More than likely, there will be either two or three wires going to the disc:

- *Two wires to the piezo disc:* The wires will provide ground and +V power. This design is typical when you are using all-in-one piezo disc buzzers, in which the disc itself contains the electronics to produce the signal for audible tones, or the disc is used as a simple speaker.
- *Three wires to the piezo disc:* The wires will provide ground, +V power, and a signal that causes the disc to oscillate with an audible tone.

Find the wire that serves as ground. On the SA300 it's easy because the battery terminals are labelled + and – right on the board. Find the ground (negative or –) terminal, and connect the COM black lead from the multimeter to it. Connect the red test lead to one of the wires or connections to the piezo disc.

Replace the battery in the battery compartment, and depress the “Test” button on the alarm. Watch for a change in voltage. For a two-wire disc, you should see the voltage change as the tone is produced. For a three-wire disc, try each wire to determine which produces the higher voltage; that is the one you should use. If you are using an oscilloscope, find the wire that produces the cleanest on/off pulse.

Once you have determined the functions of the wires to the piezo disc, remove the disc and save it for some other project. Retest the alarm's circuit board to make sure you can still read the voltage changes with your multimeter. Then clip off the wires to the battery compartment, noting their polarity.

On the First Alert SA300, the piezo disc has three leads; the one that produced the largest voltage change was in the upper right corner. So I soldered a long “tap-off” wire to it from the underside of the printed circuit board. On an oscilloscope, this voltage change showed as a series of very fast sawtooth pulses; these pulses are what actually produce the high-pitched tone on the piezo disc. Figure 48-4 shows the SA300 unit, before it was disassembled and gutted, with the piezo disc removed, and tap-off wire soldered to the PCB.



Figure 48-4 The guts of a typical battery-operated smoke detector.

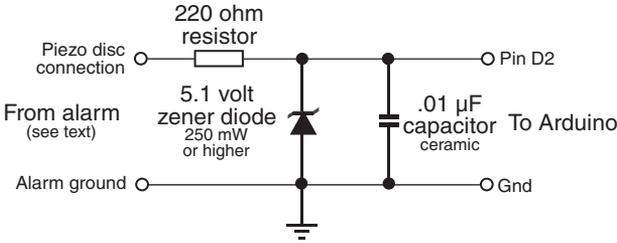


Figure 48-5 A 5.1-volt zener diode clamps the output of the smoke alarm to about 5 volts, helping to protect the input of the microcontroller. You may also wish to add extra protection in the form of an opto-isolator or buffer (see Chapter 41).

INTERFACING THE ALARM TO A MICROCONTROLLER

See Figure 48-5 for interfacing the output of a battery-operated smoke detector to a microcontroller or other circuit that expects 5-volt input. To be on the safe side, the circuit uses a 5.1-volt zener diode and current-limiting resistor to prevent a possible overvoltage condition. Most battery-operated smoke alarms operate at up to 9 volts (a few up to 12 volts), and you don't want more than 5 volts going to your microcontroller.



Remember: There's always a chance of damaging your microcontroller when connecting it to alien devices. Exercise care and *proceed at your own risk!*



The circuit in Figure 48-5 assumes an output voltage to the piezo sounder of the smoke detector of 7 to 12 volts. You may wish to add an opto-isolator (see Chapter 40), which adds protection between the detector and the robot's electronics.

By way of example, let's assume that the microcontroller periodically *polls* the input pin that is connected to the smoke alarm circuit board. The program, shown in *smoke_detector.pde* (for the Arduino), checks the pin several times each second. When the pin goes HIGH, the smoke alarm has been triggered and the Arduino's LED lights up.



smoke_detector.pde

```
const int alarmPin = 2;
const int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(alarmPin, INPUT);
}

void loop(){
  if (digitalRead(alarmPin) == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}
```

TESTING THE ALARM

Once the smoke alarm circuit board is connected to the microcontroller or computer port, test it and your software by triggering the "Test" button on the smoke alarm.

RUNNING ON 5 VOLTS

I found that the First Alert SA300 unit operated at 5 volts, rather than the 9 volts from its regular battery. Or I should say that when operating at 5 volts it triggered and sounded its alarm when pressing the Test button. I have no idea if the unit's smoke detection abilities are diminished at the reduced voltage, but I suspect this may be the case. Anyway, running the board at 5 volts saves you from having to power it from a separate battery, so it's worth looking into.

On the other hand, at the reduced voltage there wasn't enough rise in signal to trigger the Arduino; the voltage at pin D2 wasn't enough to register as a HIGH. An option in this case is to connect a voltage comparator to the hacked output of the smoke detector and adjust its reference voltage to trigger at the reduced signal level (in my tests it was about 2 volts). See Chapter 40 for additional information on using voltage comparator circuits.

LIMITATIONS OF ROBOTS DETECTING SMOKE

You should be aware of certain limitations inherent in robot fire detectors. In the early stages of a fire, smoke tends to cling to the ceilings. That's why manufacturers recommend that you place smoke detectors on the ceiling rather than on the wall. Only when the fire gets going and smoke builds up does it start to fill up the rest of the room.

Your robot is probably a rather short creature, and it might not detect smoke that confines itself only to the ceiling. This is not to say that the smoke detector mounted on even a 1-foot-high robot won't detect the smoke from a small fire; just don't count on it. Always use a regular smoke alarm for actual protection, and treat the robot's smoke alarm as an educational toy.

Detecting Dangerous Gas

Smoke alarms detect the smoke from fires but not noxious fumes. Some fires emit very little smoke but plenty of toxic fumes, and these are left undetected by the traditional smoke alarm. Moreover, potentially deadly fumes can be produced in the absence of a fire. For example, a malfunctioning gas heater can generate poisonous carbon monoxide gas.

Just as there are alarms for detecting smoke, so there are alarms for detecting noxious gases, including carbon monoxide. Such gas alarms tend to be a little more expensive than smoke alarms, but they can be hacked in much the same way as a smoke alarm.

Combination units that include both a smoke and a gas alarm are also available. You should determine if the all-in-one design will be useful for you. In some combination smoke-gas alarm units, there is no simple way to determine which (smoke or gas) has been detected.

BUILDING A NOXIOUS GAS DETECTOR CIRCUIT

There may not be many smoke detector modules available for robotics experimentation, but the same is not true of gas detectors. Parallax, Seeedstudio, and many others offer a wide variety of single-board gas detection modules that you can readily incorporate into your robot designs. No hacking required.

Figure 48-6 shows an example propane LPG sensor and module that is easily connected to a microcontroller. The sensor is a Hanwei MQ-6 gas detector, capable of picking up many kinds of volatile gases, with a special sensitivity to LPG, propane, and butane. A typical hookup



Figure 48-6 Gas sensor detects propane, LPG, and other toxic and flammable gases. (Photo courtesy Parallax Inc.)

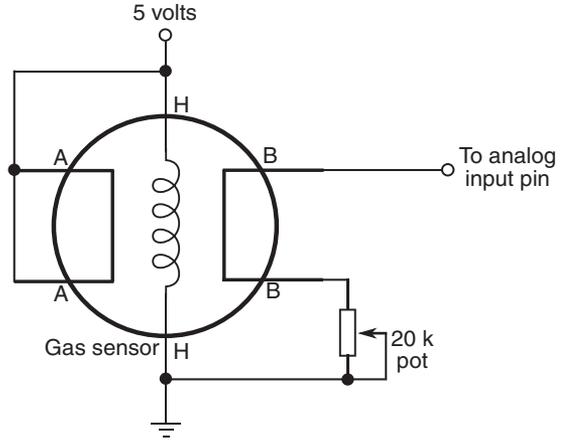


Figure 48-7 Basic connection diagram for a gas sensor that uses an internal heating element. Adjust the potentiometer for best sensitivity and range.

diagram is shown in Figure 48-7. A sample program for the Arduino is provided in *propane.pde*. Connect the output of the sensor to analog pin A0.

Note that the circuit and demo programs work with many of the gas and air quality sensors made by the same company (but not all; some need a much more sophisticated connection and heat/cool cycle). For example, the MQ-3 alcohol sensor could be used to create a Breathalyzer-bot. Check the datasheet that comes with the sensor you're using for details.

Adjust the 20 kΩ pot for best sensitivity. Since you probably don't have a calibrated gas source to work from, you'll have to simply guess at the best setting. Start with the pot at its midpoint, and try adjusting it in small steps higher or lower.



Follow the manufacturer's recommended procedure for testing and using this, or any, noxious gas sensor, or serious injury or death could result. Test only outdoors or in well-ventilated areas, and away from flame, sparks, or heat sources. Use for experimental purposes only!



propane.pde

```
int val = 0;
void setup() {
  Serial.begin(9600);
}

void loop() {
  val = analogRead(0);
  Serial.println(val, DEC);
  delay(250);
}
```

WARM-UP, ADJUSTMENT, AND USE

The MQ-6 works by warming the air inside a chamber. It does this with a built-in heater element (this is why the sensor gets a little warm during operation). You should always allow the

sensor to come to temperature before trying to take a reading. The datasheet for the sensor says to preheat the thing for 24 or more hours, but for general testing and use as a robotic gas sniffer, preheating for a minute or so will do the trick.

Run the program, open the Serial Monitor window, and let the sensor preheat. After preheating, dial the 20 k Ω pot so the reading is right in the middle. On my prototype, turning the pot to the extremes resulted in readings from 0 to 330. So I dialed in 160 as the approximate midpoint.

For the gas test I used an ordinary butane lighter, snuffing out the flame so only the gas escaped. (Of course, I did this in a well-ventilated room, and only for a few seconds.) The reading quickly climbed above 850, indicating a heavy concentration of the gas. It takes several minutes for the reading to return to normal. Don't touch the sensor during operation, as this will affect the accuracy of the reading.

The heater runs full-time and consumes about 750 milliwatts of power at 5 volts. That equates to 150 milliamps. Don't rely on a little 9-volt battery to operate your Arduino and gas sensor. You'll need a beefier battery pack, perhaps a set of six or eight D cells. Remember that when the Arduino is powered from the USB port you are limited to 500 milliamps. Don't use the gas sensor with other current-hogging components, such as servo motors.

HEAVY GAS, LIGHT GAS

As with smoke detection, where the gas detector is located has great bearing on whether it detects anything. The reason: Gases either rise or fall in air, depending on their specific gravity. If a gas is lighter than air, it rises—same thing as when you let go of a helium-filled balloon.

- Gases that are lighter than air, such as methane and natural gas, will rise to the ceiling and may not even be noticed (depending on the concentration) by a gas detector mounted on a robot that's only 6" tall.
- Gases that are heavier than air, or about the same specific gravity as air, will sink to the ground. Sensors for these gases, which include benzene and propane, are ideal candidates for robot experimenting.

Gas	Specific Gravity	What It Does in Air
Acetylene	0.9	Slowly rises
Alcohol vapor	1.6	Sinks
Benzene	2.7	Sinks
Butane	2.0	Sinks
Carbon dioxide*	1.5	Sinks
Carbon monoxide	0.9	Slowly rises
Natural gas	0.6	Rises
Methane	0.5	Rises
Propane	1.5	Sinks

* Carbon dioxide is not considered a noxious gas unless its concentration is very high.

Heat Sensing

In a fire, smoke and flames are most often encountered before heat, which isn't felt until the fire is going strong. But what about before the fire gets started in the first place, such as when a kerosene heater is inadvertently left on or an iron has been tipped over and is melting the nylon clothes underneath?

Realistically, heat sensors provide the least protection against a fire. But heat sensors are easy to build, and, besides, when the robot isn't sniffing out fires it can be wandering through the house giving it an energy check or reporting on the outside temperature or . . . you get the idea.

Figure 48-8 shows a basic but workable circuit centered around the popular LM34 temperature sensor. This device is relatively easy to find and costs just a few dollars. The output of the device, when wired as shown, is a linear voltage. The voltage increases 10 millivolts (mV) for every rise in temperature of 1°F. The *temperature.pde* sketch provides an example for the Arduino microcontroller for reading the sensor.



The LM34 provides output relative to degrees Fahrenheit. Use the LM35 if you want to measure in centigrade.

```
101010
010101
101010
010101
```

temperature.pde

```
int lm34 = A0;
int tempF = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  tempF = (500.0 * analogRead(lm34)) / 1024;
  Serial.print("Current temperature: ");
  Serial.println(tempF, DEC);
  delay(500);
}
```

A variation on the LM34 theme is shown in Figure 48-9. Here, the sensor is connected to an LM339 voltage comparator, for sensing any temperature above some limit you set via the 10 kΩ potentiometer. For example, you might “calibrate” the circuit to trigger at about 100°F. The output of the LM339 comparator will trigger when the temperature exceeds that.

The easiest way to calibrate the circuit is to set the potentiometer when the temperature is what you want. If that's not possible, you can infer the setting by using your multimeter to

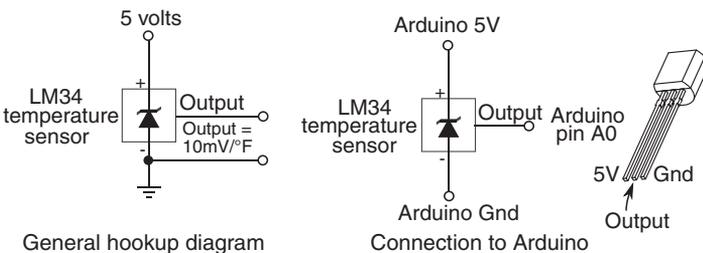


Figure 48-8 Circuit diagram for an LM34 Fahrenheit temperature sensor. For best results, mount the sensor on a small piece of aluminum or other metal, using a thermally conductive adhesive.

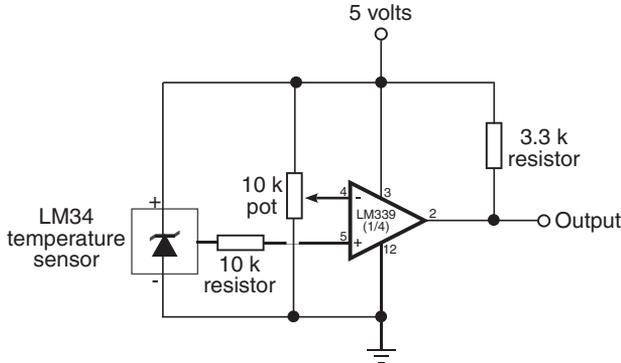


Figure 48-9 Use an LM339 comparator to set a trigger when a certain temperature is reached. The 10 k Ω potentiometer adjusts the temperature at which to trigger the comparator.

measure the output voltage of the LM34 at some known temperature. Calculate the difference between the known temperature and the desired temperature, multiply this result by 0.01 (remember, the LM34 changes 1 mV per degree F). Add or subtract the difference from the known temperature.

For instance, if the current temperature is 75°F, and you want to trigger at 100°F, the difference is +25°F. If the voltage from the LM34 is 2.4 volts at 75°F, add 0.25 (25×0.01). Set the potentiometer so that the reference voltage to the comparator is 2.65 volts ($2.4 + 0.25$).

Robotic Firefighting Contests

If the firefighting robot bug bites you hard, consider entering your machine in the annual Trinity College Fire Fighting Home Robot Contest (see www.trincoll.edu/events/robot/ for additional information, including rules and a description of the event).

This contest involves timing a robot as it goes from room to room in a house-like test field (the “house” and all its rooms are at a reduced scale). The object is to find the fire from a candle and snuff it out in the least amount of time. Separate competitions involving a junior division (high school and younger) and a senior division (everyone else) help to provide an even playing field for the contestants.

Several local robotics groups host their own firefighting contests. Do a Web search for a group near you, and see what kind of competitions they offer.

Finally, Go Out and Do!

Few other moments in life compare to the instant when you solder that last piece of wire, file down that last piece of metal, tighten that last bolt, and switch on your robot. Something you created comes to life, obeying your commands and following your preprogrammed instructions. This is the robot hobbyist’s finest hour. It proves that the countless evenings and weekends in the workshop were worth it after all.

I started this book with a promise of adventure—to provide you with a treasure map of plans, diagrams, schematics, and projects for making your own robots. I hope you’ve followed along and built a few of the mechanisms and circuits that I described. As you finish reading,

you can make me a promise: improve on these ideas. Make them better. Use them in creative ways that no one has ever thought possible. Create that ultimate robot that everyone has dreamed about.

Your ideas, suggestions, and other comments are welcome. If you see a mistake in a circuit or mechanism, I'll make sure the next edition of this book is corrected. Visit the support site for this book; see Appendix A. If you have a unique robot design, why not share it with others? Send me details of your robots—finished or in process.

Now, stop reading—and do. Impress us all!

RBB Online Support

Books have only so many pages, and a topic like robotics is ever evolving. To augment your reading experience, be sure to visit the RBB Online Support site, at:

www.robotoid.com

You'll Find . . .

My First Robot lessons—fully illustrated A–Z guide on constructing your first motorized robot, the expandable and easy-to-build *RBB-Bot*.

Expanded parts source for all the projects—including online sources, with part numbers and direct links to the products (when available). Just click and you're there.

Program listings for all the projects—just copy and paste, or better yet, download the program files for immediate use.

Book updates—changes, corrections, enhancements, alternative approaches.

ArdBot construction and programming project—complete plans for building the Ard-Bot Arduino-based robot. For intermediate robot builders.

Free bonus chapters—stuff that just couldn't fit in this new edition (at least not without chopping down a lot of extra trees).

Plan templates drawn to scale—of over a dozen robot projects and parts. Just download the PDF and print.

Resizable and printable encoder disc templates—pick the one you like and make copies with your ink-jet or laser printer.

Video tutorials—on everything from using shop tools to programming your bot.

Animated teaching tools—learn electronics and robotics with colorful animated guides.

Plus, free articles, expanded projects, news, reviews, and more.

Backup Support Site

Basic support material including all source code for *Robot Builder's Bonanza* may also be found at the following Web page, should the main *robotoid.com* site be unavailable:

www.mhprofessional.com/rbb4

Sources for Special Parts, Web Sites

Mentioned now and then in this book are special parts and Web sites for improving your robot-building experiences. You can find some of these sites in Appendix B, "Internet Parts Sources," but you'll find many more on the RBB Online Support site. They're categorized, with a brief summary for the most important sites on the list.

Among the source listings:

- Plastic and metal brackets
- Free and low-cost CAD programs
- Motor upgrades for Tamiya gearbox motor kits
- Hard-to-find electronic parts (with sources and catalog numbers)
- Web sites for experimenting with sound, music, and voice
- Resources for machine and robotic vision
- Where to buy surplus parts at a bargain price
- The Internet's best self-study and teaching sites on electronics and related topics
- The Internet's best blogs on robots
- Robotics user groups, forums, and competitions

Internet Parts Sources

On these pages you'll find a selected list of online sources for robots and robot-building parts. I haven't listed everyone, just a handful of the best and those that have been around for a while.

FYI

Web sites come and go. New and updated sources may be found at the RBB Online Support site. See Appendix A for details.

Robotics

Robot kits, sensors, servos, wheels, parts.

Acroname—www.acroname.com
Active Robots—www.active-robots.com
Arrick Robotics—www.robotics.com
Budget Robotics—www.budgetrobotics.com
CrustCrawler—www.crustcrawler.com
Fingertech Robotics—www.fingertechrobotics.com
Lynxmotion—www.lynxmotion.com
Machine Science—www.machinescience.com
Mark III Robot Store—www.junun.org/MarkIII/
Mr. Robot—www.mrrobot.com
OWI Robots—www.robotkitsdirect.com
Parallax—www.parallax.com
Pitsco—www.pitsco.com
Pololu—www.pololu.com
RB Robotics—www.rbrobotics.com

Robot MarketPlace—www.robotmarketplace.com
Robotis—www.robotis.com
RobotShop—www.robotshop.com
Robot Store (HK)—www.robotstorehk.com
Solarbotics—www.solarbotics.com
TrossenRobotics—www.trossenrobotics.com
Vex Robotics—www.vexrobotics.com
Zagros Robotics—www.zagrosrobotics.com

Electronics

Components, modules, microcontrollers. May include both new and surplus.

Allied Electronics—www.alliedelec.com
Arrow Electronics—www.arrow.com
Avnet (Avnet Electronics)—www.avnet.com
BG Micro—www.bgmicro.com
Circuit Specialists—www.web-tronics.com
Devantech Ltd.—www.robot-electronics.co.uk
Dick Smith Electronics—dicksmith.com.au
Digi-Key—www.digikey.com
Electronix Express—www.elexp.com
Farnell—www.farnell.com
Future Electronics—www.futureelectronics.com
Hobby Engineering—www.hobbyengineering.com
HVW Tech—www.hvwtech.com
Images SI—www.imagesco.com
Jameco Electronics—www.jameco.com
JDR Microdevices—www.jdr.com
Maplin Electronics—www.maplin.co.uk
Marlin P. Jones & Assoc—www.mpja.com
MCM Electronics—www.mcmelectronics.com
Mouser Electronics—www.mouser.com
Newark Electronics—www.newark.com
Nu Horizons Electronics Corp.—www.nuhorizons.com
Parts Express—www.partsexpress.com
RadioShack—www.radioshack.com
SparkFun Electronics—www.sparkfun.com

Hobby

Servo motors and accessories, radio control sets, parts for model airplanes and cars. Many of the sources listed under Robotics and Electronics also carry these parts.

BP Hobbies—www.bphobbies.com
Central Hobbies—www.centralhobbies.com

Hobby Lobby—www.hobbylobby.com
Hobby People—www.hobbypeople.net
Horizon Hobby—www.horizonhobby.com
Servo City—www.servocity.com
Tower Hobbies—www.towerhobbies.com

Forums and Blogs

Arduino Forum—arduino.cc/forum
Google Groups—groups.google.com
Let's Make Robots—www.letsmakerobots.com
Lugnet—news.lugnet.com/robotics
Parallax Discussion Forums—forums.parallax.com
PICAXE Forum—www.picaxeforum.co.uk
Robots.net—www.robots.net
Society of Robots—www.societyofrobots.com



More on the Web!

Find hundreds of links for specialty sources on the RBB Online Support site (refer to Appendix A for the details). There are several dozen categories, and the site is updated regularly.

Mechanical Reference

Decimal Fractions

Fraction	Decimal	Fraction	Decimal
1/64	.015625	33/64	.515625
1/32	.03125	17/32	.53125
3/64	.046875	35/64	.546875
1/16	.0625	9/16	.5625
5/64	.078125	37/64	.578125
3/32	.09375	19/32	.59375
7/64	.109375	39/64	.609375
1/8	.125	5/8	.625
9/64	.140625	41/64	.640625
5/32	.15625	21/32	.65625
11/64	.171875	43/64	.671875
3/16	.1875	11/16	.6875
13/64	.203125	45/64	.703125
7/32	.21875	23/32	.71875
15/64	.234375	47/64	.734375
1/4	.25	3/4	.75

Fraction	Decimal	Fraction	Decimal
17/64	.265625	49/64	.765625
9/32	.28125	25/32	.78125
19/64	.296875	51/64	.796875
5/16	.3125	13/16	.8125
21/64	.328125	53/64	.828125
11/32	.34375	27/32	.84375
23/64	.359375	55/64	.859375
3/8	.375	7/8	.875
25/64	.390625	57/64	.890625
13/32	.40625	29/32	.90625
27/64	.421875	59/64	.921875
7/16	.4375	15/16	.9375
29/64	.453125	61/64	.953125
15/32	.46875	31/32	.96875
31/64	.484375	63/64	.984375
1/2	.50	1	1.00

Drill Bit and Tap Sizes—Imperial

Tap	Fractional Drill Bit	Numbered Drill Bit	Lettered Drill Bit
0-80	3/64	—	—
1-64	—	53	—
2-56	—	50	—
3-48	—	47	—
4-40	3/32	43	—
5-40	—	38	—
6-32	7/64	36	—
8-32	—	29	—
10-24	5/32	25	—
10-32	5/32	21	—
12-24	11/64	16	—
1/4-20	13/64	7	—

Tap	Fractional Drill Bit	Numbered Drill Bit	Lettered Drill Bit
1/4-28	7/32	3	—
5/16-18	17/64	—	F
5/16-24	—	—	I
3/8-16	5/16	—	—
3/8-24	21/64	—	Q
7/16-14	23/64	—	U
7/16-20	25/64	—	—
1/2-13	27/64	—	—
1/2-20	29/64	—	—
9/16-12	31/64	—	—
9/16-18	33/64	—	—
5/8-11	17/32	—	—
5/8-18	37/64	—	—
3/4-10	21/32	—	—
3/4-16	11/16	—	—

Drill Bit and Tap Sizes—Metric

Tap	Metric Drill Bit	Nearest Fractional Drill Bit
3 mm × 0.5	2.5 mm	—
4 mm × 0.7	3.3 mm	—
5 mm × 0.8	4.2 mm	—
6 mm × 1.0	5.0 mm	—
7 mm × 1.0	6.0 mm	15/64
8 mm × 1.25	6.8 mm	17/64
8 mm × 1.0	7.1 mm	—
10 mm × 1.5	8.7 mm	—
10 mm × 1.25	8.8 mm	11/32
10 mm × 1.0	9.0 mm	—
12 mm × 1.75	10.25 mm	—
12 mm × 1.5	10.7 mm	27/64
14 mm × 2.0	12.0 mm	—

Tap	Metric Drill Bit	Nearest Fractional Drill Bit
14 mm × 1.5	12.5 mm	1/2
16 mm × 2.0	14.0 mm	35/64
16 mm × 1.5	14.5 mm	—

Numbered and Fractional Inch Drill Bit Comparison

Numbered Drill Bit	Nearest Fractional Drill Bit
53	1/16
47	5/64
43	3/32
35	7/64
30	1/8
29	9/64
25	5/32
16	11/64
14	3/16
7	13/64
3	7/32

Fasteners: Standard (Imperial) Threads at a Glance

UNC (Coarse)	UNF (Fine)
4-40	4-48
5-40	5-48
6-32	6-40
8-32	8-36
10-24	10-32
1/4 × 20	1/4 × 28

UNC (Coarse)	UNF (Fine)
5/16 × 18	5/16 × 24
3/8 × 16	3/8 × 24
7/16 × 14	7/16 × 20
1/2 × 13	1/2 × 20
5/8 × 11	5/8 × 18
3/4 × 10	3/4 × 16
7/8 × 9	7/8 × 14
1 × 8	1 × 14

Comparison of Decimal Inch, Fractional Inch, Mil, and Gauge

(Gauge is for sheet aluminum.)

Decimal Inch	Fractional Inch	Mil	Gauge
0.010"	1/96"	10	30
0.016"	1/64"	16	26
0.020"	1/48"	20	24
0.032"	3/96"	32	20
0.040"	3/72"	40	18
0.050"	3/64"	50	16
0.064"	1/16"	64	14
0.080"	5/64"	80	12



More on the Web!

Find many more handy reference charts and lookup tables on the RBB Online Support site (see Appendix A for the details).

Electronic Reference

Formulas

Here are some of the more common electronic formulas you will encounter in your electronics work. Many of these formulas have been with us for a long time.

OHM'S LAW

Ohm's law calculates the relationship between power, voltage, current, and resistance. The basic formulas are:

Unknown Value	Calculating for	Formula
Voltage	Volts	$V = IR$
Current	Amps	$I = V/R$
Power	Watts	$P = VI$
Resistance	Ohms	$R = V/I$

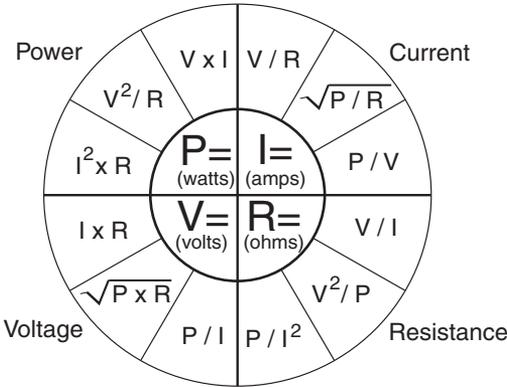
where:

V = voltage (in volts)

I = current (in amps)

P = power (in watts)

R = resistance (in ohms)



Example:

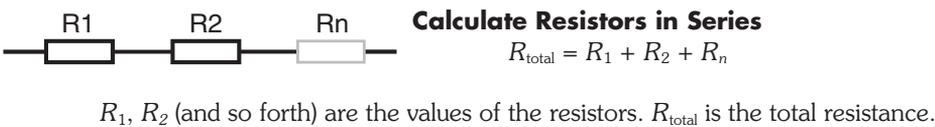
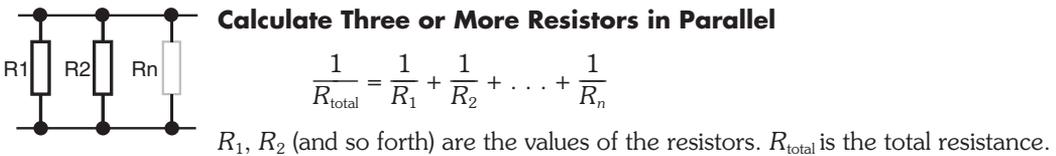
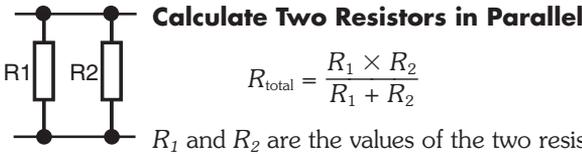
To find the power in a circuit consuming 100 volts at 10 amps, multiply volts times amps ($100 \times 10 = 1000$). Answer: 1000 watts.

The Ohm's law wheel can be used to help you remember the formula for calculating one value when you know the other two. Start with any of the four unknown values in the center of the circle, then apply the appropriate formula, given the two values that you do know. For instance, to calculate volts when you know power and current, you'd use:

$$V = P / I$$

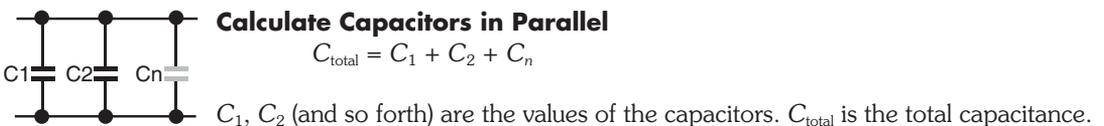
CALCULATING RESISTANCE

The resistance of a single resistor in a circuit is easy to surmise. But resistance changes when you add resistors in parallel or in series.

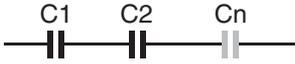


CALCULATING CAPACITANCE

These formulas can be used to calculate total capacitance in a circuit. Note that the formulas are basically the inverse of those for resistors.



Calculate Two or More Capacitors in Series



$$\frac{1}{C_{total}} = \frac{1}{C_1} + \frac{1}{C_2} + \dots + \frac{1}{C_n}$$

C₁, C₂ (and so forth) are the values of the capacitors. C_{total} is the total capacitance.

Abbreviations

Abbreviations may be used for descriptions and circuit diagrams used with electronics. Many of the most common abbreviations are listed here. Note that the use of abbreviations may not follow any kind of standard, and the accepted use of some abbreviations may come and go over time.

Abbreviation	Meaning	Abbreviation	Meaning
AC	alternating current	PCB	printed circuit board
AM	amplitude modulation	PCM	pulse code modulation
amp	ampere, amplifier	PF	power factor
ASCII	American Standard Code for Information Interchange	pos	positive
assy	assembly	pot	potentiometer
aud	audio	preamp	preamplifier
aux	auxiliary	PS	power supply
avg	average	pri	primary
AWG	American Wire Gauge	pwr	power
cap	capacitor	Q	merit of capacitor or coil
C	capacitance, capacitor, collector, Celsius	RC	resistor-capacitor
DC	direct current	R/C	radio control
DMM	digital multimeter	rcvr	receiver
DPDT	double-pole, double-throw (switch)	ref	reference
DPST	double-pole, single-throw (switch)	res	resistor
EMF	electromotive force	RF	radio frequency
EMT	electrical metallic tubing	RFI	radio frequency interference
E	emitter, voltage	rms	root mean square
F	Fahrenheit	RPM	revolutions per minute
f	frequency	RPS	revolutions per second

Abbreviation	Meaning	Abbreviation	Meaning
FM	frequency modulation	R	resistance, resistor
gnd	ground	RC	resistance/capacitance
ID	inside diameter	RL	resistance/inductance
IF	intermediate frequency	sec	second, secondary
inp	input	sig	signal
I/O	input/output	SNR	signal-to-noise ratio
IR	infrared	SPDT	single-pole, double-throw (switch)
I	current	spkr	speaker
IC	integrated circuit	SPST	single-pole, single-throw (switch)
K	Kelvin, dielectric constant	sq	square
l or L	length, inductor, inductance	sw	switch
LP	low-pass (filter)	t or	time
LSB	least significant bit	UF	ultrasonic frequency
LSI	large-scale integration	v or V	voltage
mic	microphone	V_{CC}, V_{DD}	+ power pin on semiconductor
mom	momentary (switch)	V_{EE}, V_{SS}	– power pin on semiconductor
MSB	most significant bit	vid	video
NC	normally closed, no connection	VOM	volt-ohm meter
neg	negative	VTVM	vacuum-tube volt meter
neut	neutral	X	reactance
NO	normally open	Z	impedance
nom	nominal (means normal, representative)	+	plus, positive
norm	normal	–	minus, negative
osc	oscillator, oscillator, oscilloscope	°	degrees
out	output	Ω	ohms
PC	printed circuit		

Letter Symbols Used in Electronics

These symbols are commonly used in descriptions of electronic circuits.

Symbol	Meaning	Symbol	Meaning
A	ampere; amp	MeV	megaelectron volt
Ah	ampere-hour	MΩ	megohm (× 1,000,000 ohms)
c/s, cps	cycles per second (hertz)	mA	milliampere
dB	decibel	mH	millihenry
eV	electron volt	ms	millisecond
<i>f</i>	frequency	mV	millivolt
F	farad	mW	milliwatt
GHz	gigahertz	nF	nanofarad
H	henry	ns	nanosecond
hp	horsepower	pF	picofarad
Hz	hertz	S	siemens
J	joule	V	volt
k	kilo (thousand)	V/A	voltampere
kΩ	kilo-ohm (× 1000 ohms)	W	watt
kHz	kilohertz	μ	micro (millionth)
kV	kilovolt	μA	microampere
kW	kilowatt	μF	microfarad
kWh	kilowatt-hour	μH	microhenry
M	mega (million)	μs	microsecond

Numbering Units in Electronics

The science of electronics uses what's known as *SI*, or *International System of Units*, to denote very large and very small numbers. The most common unit names used in electronics are as follows:

Unit Name	Symbol	Decimal	Scale Name
giga	G	1000000000	billion
mega	M	1000000	million
kilo	k	1000	thousand
	—	1	one

Unit Name	Symbol	Decimal	Scale Name
milli	m	0.001	thousandth
micro	μ	0.000001	millionth
nano	n	0.000000001	billionth
pico	p	0.000000000001	trillionth

You may convert between numbering units by moving the decimal point. SI units smaller than 1 each differ by three decimal spaces between them. Move the decimal point three spaces right or left to accomplish the conversion.

pico	nano	micro	milli
100	.1	.0001	.0000001
1000	1	.001	.000001
10000	10	.01	.00001
100000	100	.1	.0001
1000000	1000	1	.001
10000000	10000	10	.01
100000000	100000	100	.1
1000000000	1000000	1000	1

Examples:

Capacitance: $10,000 \text{ pF} = 10 \text{ nF} = 0.01 \text{ }\mu\text{F}$

Time: $1000 \text{ }\mu\text{s} = 1 \text{ ms}$

The Six Most Common Units of Measure in Electronics

Electronics is a science based on math and relationships of numbers. Different electronic components have different units of measure, and these units are used in making calculations. A large percentage of the study of electronics boils down to the following six most common units of measure.

Unit	Measured As	Symbol
Capacitance	Farad	F
Current	Ampere (or amp)	A

Unit	Measured As	Symbol
Inductance	Henry	H
Power	Watts	W
Resistance	Ohm	Ω
Voltage	Volt	V

Notes:

- A farad is a very large unit. Capacitance is typically measured in millionths of a farad (microfarad or μF) or billionths of a farad (picofarad or pF). See the section “Numbering Units in Electronics” on converting from one numbering unit to another.
- Similarly, a henry is a large unit, and inductors are routinely measured in millionths of a henry (milrohenry or μH).

Resistor Color Coding

Because of their small size, resistors are usually coded for identification. The most common coding scheme for resistors is a series of colored bands.

RESISTOR COLOR CODE CHART

Color	1st Digit	2nd Digit	Multiplier	Tolerance
Black	0	0	1	—
Brown	1	1	10	$\pm 1\%$
Red	2	2	100	$\pm 2\%$
Orange	3	3	1,000	$\pm 3\%$
Yellow	4	4	10,000	$\pm 4\%$
Green	5	5	100,000	$\pm 0.5\%$
Blue	6	6	1,000,000	$\pm 0.25\%$
Violet	7	7	10,000,000	$\pm 0.1\%$
Gray	8	8	100,000,000	$\pm 0.05\%$
White	9	9	—	—
Gold			0.1	$\pm 5\%$
Silver			0.01	$\pm 10\%$
None				$\pm 20\%$

Wire Gauge

Gauge defines the diameter of wire. The larger the wire, the smaller the gauge. Only the diameter of the electrical conductor(s) is considered; the size of insulation or other nonconductive material on the outside of the wire is ignored. This chart shows the cross-section diameter of wire from 12 to 30 gauge, listed in both inch and metric sizes. Wire gauge is sometimes referred to as *AWG wire size*; AWG stands for American Wire Gauge.

Gauge	Diameter		Current Max.*
	Inch	Metric	
12	0.0808	2.053	41
14	0.0641	1.628	32
16	0.0508	1.291	22
18	0.0403	1.024	16
20	0.0320	0.812	11
22	0.0253	0.644	7
24	0.0201	0.511	3.5
26	0.0159	0.405	2.2
28	0.0126	0.321	1.4
30	0.0100	0.255	.86

* Current maximum, in amps at low voltage, assumes stranded conductors. Single-conductor wire has a lower current maximum.

Index

- 3D CAD
 - file formats, 158
 - visualizing design, 156
- 3D printers, 159
- 3-DOF wrist, 344
- 3M
 - adhesive transfer tape, 151
 - Dual Lock, 148
 - Dual Lock, for battery pack, 194
 - Photo Mount, 151
 - VHB self-adhesive tape, 150
- 4WD (four-wheel-drive) and 6WD locomotion, 211, 306
 - linked-drive, construction, 308
 - separate motor, construction, 307
- 12-Servo Hexapod, bonus project, 338
- 754410 motor control IC, 244
- abbreviations, in electronics, 679
- ABS (plastic), defined, 88
- accelerometer
 - dual-axis, 621
 - accelerometer (*Cont.*):
 - experimenting with, 621
 - measuring tilt, 620
 - as touch sensor, 569
 - use in robotics, 619
 - acetal resin, characteristics, 88
 - Ackermann steering, 212
 - acrylic, characteristics, 88
 - acrylonitrile butadiene styrene, characteristics, 88
 - additive manufacturing, 159
 - adhesives and glue, introduction to, 136
 - air pressure sensor, 568
 - air tools, using, 68
 - Airtronics, connector style, 256
 - alkaline battery, 180
 - allophone, defined, 632
 - alloy, defined, 107
 - aluminum, 107
 - bending, 119
 - bracket, DC motor mounting, 268
 - filing, 120
 - painting, 120
 - in quick-turn manufacturing, 159
 - aluminum, extruded, 111
 - framing, 115
 - for motor mounting, 269
 - aluminum electrolytic capacitor, 394
 - amplifier, audio, 631
 - amplifying signals, 523
 - analog
 - comparator, 442
 - demultiplexer, 531
 - interfacing, 522
 - multiplexer, 531
 - sensors, 513
 - analog and digital R/C servos, 257, 325
 - Analog Devices, ADXL dual-axis accelerometer, 621
 - analog-to-digital conversion (ADC), 442, 436, 526
 - analog video
 - security camera, 602
 - sync separator for, 602
 - vision sensor, 602
 - analogWrite, Arduino programming statement, 478
 - And and Or operators, 458
 - android robot, 19, 320
 - And truth table, 458

- animatronic devices, 534
- annealing, metal treatment for, 110
- anode (diode), 395
- antistatic conductive foam, 558
- ArdBot, bonus project, 475
- Arduino
 - circuit boards for use with, 420
 - custom functions in programming for, 474
 - flow control, programming with, 476
 - joystick, 536
 - microcontroller, introduction to, 463
 - operating voltage, 466
 - pin mapping, 467
 - pins, using, 470
 - programming, general, 467
 - programming for switches, 479
 - serial monitor window, using, 477
 - serial servo control, 325
 - shield, printed circuit board layouts, 420
 - USB connection to, 466
 - USB fuse protection, 466
- arm, robotic
 - cartesian coordinate, 342
 - construction sets, built from, 350
 - cylindrical coordinate, 342
 - degrees of freedom (DOF), 339
 - electrical actuation, 343
 - hydraulic actuation, 343
 - mobile robots, 19
 - pneumatic actuation, 343
 - polar coordinate, 341
 - revolute coordinate, 341
 - specialty kits, built from, 350
 - stationary robots, 19
- armatures, dolls and teddy bears, 32
- articulated joints in nature, 12
- art tape, in line following, 604
- Atmel
 - ATmega microcontroller chip, 463
 - ATmega328 microcontroller, 441
- audio amplifier, 631
- AutoCad, 156
- aviation snips, cutting metal with, 115
- AVR, serial port, programming through, 526
- back EMF, 238
- backing board, use when cutting wood, 78
- backsaw, cutting with, 64
- balance
 - legged robots, 319
 - static, 328
 - static and dynamic, 319
 - tilt sensors, 620
- ball bearing construction in R/C servo motor, 255
- bases, plastic, from straight cuts, 94
- BasicBot, 304
- BASIC language
 - BASIC Stamp programming, 500
 - PICAXE programming, 487
 - as programming language, 439
- BASIC Stamp
 - integrated editor, 505
 - introduction to, 496
 - microcontroller, 430
 - serial servo control, 325
 - special programming functions, 505
- batteries
 - amp-hour rating, 184
 - brownouts, 184, 207, 228
 - capacity ratings, 184, 186
 - characteristics and applications table, 187
 - common sizes, 187
 - composition and types, 180
 - current rating, 184
 - discharge, 185
 - environmental concerns, 183
 - batteries (*Cont.*):
 - fuse protection, 198
 - increasing ratings, 189
 - internal resistance, 186
 - multimeter, testing with, 366
 - multiple voltages, 199
 - recharging, 187
 - reverse polarity, preventing, 197
 - safe use of, 44
 - schematic circuit symbols for, 190
 - selection, 182
 - SMA wire, with, 290
 - solder terminals for, 192
 - split power supply, 228
 - table of nominal voltages, 183
 - using polarized plugs with, 196
 - voltage monitors, 208
 - voltage rating, 183
 - voltage regulation, 184, 201
 - wiring to robot, 196
 - batteries, polarity, 197
 - connectors, 197
 - electronic protection, 198
 - mechanical protection, 197
 - using diodes, 198
 - batteries, ratings
 - capacity, 184, 186
 - voltage, 183
 - batteries, recharging
 - fast-charge, 187
 - lithium-ion cells (Li-ion), 181
 - monitoring voltage of, 208
 - recharger, universal, 181
 - batteries, sizes, 192
 - batteries, types
 - alkaline, high-capacity, 180
 - nickel cadmium, nickel metal hydride, 181
 - lithium-ion cells (Li-ion), 181
 - sealed lead-acid, 182
 - battery holder, 193
 - 9-volt, polarized snap, 194
 - for in-between voltages, 195
 - mounting, 194, 196
 - placement, 195

- battery pack
 - from consumer electronics device, 191
 - custom-made, 192
 - layouts, 189, 192
 - premade, 191
 - for R/C models, 191
 - rechargeable, 191
 - recharger table, 192
 - shrink-wrap, 192
- battery test with multimeter, 366
- BEAM robotics, 17, 24
- bearings, 278
- belt drive
 - aligning motors, 308
 - cogged, 308
 - plastic molded sprockets, 309
 - segmented chain for, 308
- bendable posing wire, 32
- bending metal, 119
- bidirectional data communication I2C, 518
- Biologically Inspired Robotics Laboratory, 217
- Bioid walking robot set, 11, 323
- BioMetal shape memory alloy, 287
- bipolar junction transistor (*See* BJT)
- bipolar transistor, 239
- bird's-nest circuit construction, 417
- bit, 438, 448, 451
- bitmap graphics, 154
- bitwise port programming, 533
- BJT (bipolar junction transistor), 239
- Boarduino, Arduino-compatible board, 465
- BOE (Board of Education), 498
- bounce, switch, 555
- bracket
 - DC motor mounting blocks, wood and plastic, 269
 - DC motor mounting, metal, 268
 - double-sided, 324
 - plastic, 136
- bracket (*Cont.*):
 - R/C servo, custom mount, 271
 - R/C servo, tab mount, 271
 - R/C servo, for walking robot, 323
 - using, 135
 - zinc-plated steel, 135
- brains
 - hardwired, discrete components, 428
 - manually operated, 428
 - microcontroller based, 429
 - mini-ITX motherboard, 432
 - personal computer, 431
 - single-board computer, 430
 - Smartphones, tablets, PDAs, 434
 - USB port, connecting through, 433
 - variations in, 430
 - wireless connection, 433
- branch, in programming, 449
- brass, and robot construction, 108
- brass bracket, DC motor mounting, 268
- breakdown voltage, zener diodes, 203
- bridge module
 - intelligent, 247
 - interfacing, bonus projects, 248
 - serial motor control, 246
- broadcasting video, 543
- brownout, 184, 207, 228
- brushed versus brushless motors, 231
- BuggyBot (online content), 175
- bumper bar, piezoelectric touch sensor, 568
- bumper switch, 549
 - multiple, 551
 - multiplexer, 553
 - PISO IC, 554
 - priority encoder, 552
 - programming for, 557
- bumpers, foam rubber, 32
- bump switch, placement in multidirectional robot, 298
- bushing, 278
- bushings in R/C servo motor, 255
- cable clamp, as fastener, 150
- CAD (computer-aided design), 156
 - benefits of, 157
 - etchable PDF, circuit board design, 421
 - introduction to, 152, 157
 - printed circuit board design, 418
 - shapes stacking order, 159
- CadSoft, Eagle PCB software, 419
- CA glue (cyanoacrylate adhesive), 97, 139
- Caidin, Martin, author, 16
- calculating capacitance and resistance, 678
- calculating speed for wheel diameter, 301
- Call and Return programming statements, 454
- Cameron, Russell, 597
- capacitance, calculating, 678
- capacitors, 390
 - applications, 393
 - calculating capacitance, 678
 - capacitance value, 392
 - decoupling, 200, 206
 - dielectric breakdown voltage value, 393
 - dielectric material, 391
 - dielectric material, table of, 394
 - microfarads, measured in, 391
 - for noise suppression, 200
 - polarized, 393
 - rating, capacitance, 391
 - rating, dielectric breakdown voltage, 391
 - value reference table, 392
 - in voltage regulation circuits, 206
- Capek, Karel, 28
- carbon-zinc battery, 180

- cardboard
 - cutting, 146
 - heavy-duty, 145
 - screws and nuts, using, 148
- cartesian coordinate arm, 342
- car-type steering, 212
- Case Western Reserve University, 217
- casters
 - ball, 303
 - placement, 298
 - selection, 303
 - stability, 303
 - swivel, 302
- cathode (diode), 395
- CD players, robotics parts from, 36
- CdS (cadmium sulfide) photocell, 390, 591
 - input, interface for, 524
- CDs and DVDs, as robot bodies, 171
- celluloid, as early plastic, 87
- Celtec, plastic, 91
- central processing unit (CPU), in microcontroller, 442
- ceramic, dielectric, 394
- Chaney Electronics Theremin kit, 627
- circuit board, 414
 - Arduino-specific boards, 420
 - cleaning, 415
 - dead bug wiring, 417
 - etchable PDF, from CAD, 421
 - lead-to-lead construction, 417
 - options for creating, 420
 - PCB CAD, creating with, 418
 - perf board construction, 416
 - printed, 418
 - proto boards, using, 422
 - solder breadboard, 415
 - stripboard, 417
 - wire wrapping, 422
 - wire wrapping tools, 423
 - wire wrap sockets, 417
- circuit design
 - best practice, 377
 - decoupling capacitors, 200, 206, 378
- circuit design (*Cont.*):
 - resistors, pull-up/pull-down, 378
 - wire length, 379
- circular saw, cutting with, 65
- clamping and taping
 - clamps and vises, using, 63
 - glued joints, 141
- clevis ends, 276
- clip-on jumpers, 377
- CMOS integrated circuits, and electrostatic discharge, 45
- CMUcam vision sensor, 602
- code block, in Arduino programming, 469
- codewheel, in odometry, 609
- collision avoidance, 570
- color code chart, resistor, 682
- combat robot, 18
 - relays with, 405
- comments, 452
 - in Arduino programming, 473
 - in BASIC Stamp programming, 502
 - in PICAXE programming, 488
- communicating
 - with lighting effects, 649
 - with lights, 639
 - with sounds, 646
 - text display, 646
- communications, interconnectivity, 433
- commutator, in DC motor, 231
- compass, electromagnetic, 617
- programming sensors, 619
- compiling, in programming, 453
- compliance, 301
- components, building blocks as, 6
- computer-aided design (*See* CAD)
- computer control
 - hardwired, discrete components, 428
 - manually operated, 428
 - microcontroller-based, 429
 - mini-ITX motherboard, 432
- computer control (*Cont.*):
 - personal computer, 431
 - single-board computer, 430
 - Smartphones, tablets, PDAs, 434
 - USB port, connecting through, 433
 - variations in, 430
 - wireless connection, 433
- computer mice, as robot bodies, 171
- conditional expressions, 449, 453
 - in Arduino programming, 476
 - in BASIC Stamp programming, 502
 - in PICAXE programming, 490
- conductive foam, 558
- connectors, 375
 - clip-on jumpers, 377
 - custom-made, male and female, 375
- Conner, Sarah, 17, 288
- constants, in programming, 451
- construction foam, 146
- construction materials
 - comparison, 50
 - selection of, 49
 - selection table, 52
- construction plans, basics of, 60
- contact area, touch sensor, 549
- contact bumper, 549
- contact cement glue, 137
- continuity test with multimeter, 366
- continuous rotation R/C servo motor, 261
- controlling outputs
 - with Arduino, 479
 - with BASIC Stamp, 502
 - PICAXE, 494
- controlling R/C servo
 - with Arduino, 473
 - with BASIC Stamp, 508
 - with PICAXE, 493
- coping saw, cutting with, 64

- copper
 - craft and hobby store, 113
 - robot construction, 108
- corrugated plastic
 - as a construction material, 145
 - cutting, 146
- counters, timers, in microcontrollers, 442
- couplers, 278
 - flexible, custom-made, 281
 - ready-made, 278
 - rigid, flexible, 278
 - standoffs and threaded, 280
 - tubing, custom, 279
- C programming language, 440
 - with Arduino, 467
- current
 - limiting, interfacing for, 521
 - Ohm's law, 387
 - sinking, 515
 - sourcing, 515
- current draw
 - amp-hours, 184
 - brownout, 207
 - current sense, 244
 - digital multimeter, using, 199
 - fuse protection, 199
 - of motors, 222
 - rating of batteries, 185
 - stalled, DC motors, 232
 - voltage drop, 228
- custom battery packs, making, 192
- cutting
 - basics, 64
 - circles, 73
 - circuit boards, 416
 - controlling the depth, 66
 - creating a layout for, 152
 - laser cutter, 158
 - miter box, using, 76
 - square into other shapes, 73
 - substrate sheets, 146
 - tool choice and speed table, 65
 - wheel wells, 74
- cutting techniques
 - for metal, 114
 - for plastic, 91
 - for wood, 72
- cianoacrylate adhesive (CA), 164
 - using, 139
- cybernetic anthropomorphous machine (cyborg), 16
- cyclops eye, construction of, 594
- cylindrical coordinate arm, 342
- DAGU Electronics, multiple-cell robotic eye, 597
- data
 - input, 450
 - output, 450
 - selector, 530
 - storage, 433
- data conversion
 - analog-to-digital, 526
 - digital-to-analog, 527
- data demultiplexer, 528
- data type
 - in Arduino programming, 470
 - constants, 451
 - expression, 451
 - literals, 450, 456
 - number limits table, 451
 - variables, 451, 469
- Dave Brown Lite-Flight wheels, 299
- DC motors, 220, 230
 - from junkyards, 35
 - motor control, H-bridge, 244
 - from surplus, 35
- dead band, in R/C servos, 260
- dead bug wiring, 417
- debounce circuit
 - software in microcontroller, 556
 - touch sensor, 555
- decimal fractions, table of, 672
- decoupling capacitor, 378
 - as motor noise filter, 229
 - uses for, 200, 206
- degrees of freedom (*See* DOF)
- Delrin, 88
- demultiplexer (demux)
 - analog, 531
 - binary control signals, table for, 528
- desktop robot, 17
 - relays in, 405
- detracking, derailing, 317
- Devantech
 - CMPS09 compass, 618
 - SRF05, 265, 587
- dielectric (capacitor) materials, 391
- differential steering
 - 4WD robot, 306
 - BasicBot, 304
 - defined, 211
 - switch-controlled, 234
 - track steering, 215
 - wheel traction, 300
- digital multiplexer, 530
- digital proportional motor (R/C servo), 252
- digital R/C servos, 257
- digital sensors, 512
- digital-to-analog conversion, 527
- digital video, 600
 - CMUcam, 602
 - Webcam, 600
- Dinsmore 1490 compass, 617
- Dinsmore Sensors, 617
- diodes, 394
 - applications, 396
 - for battery polarity protection, 198
 - common anode, cathode, LEDs, 398
 - current rating of, 395
 - flyback, in MOSFET motor bridge, 243
 - flyback, in motor control circuits, 240
 - forward voltage drop, 395
 - incremental voltage drop, 396
 - laser, 395
 - LEDs, 396
 - light-emitting, 395
 - peak inverse voltage rating, 395
 - photo, 394
 - polarization of, 395
 - ratings of, 395
 - rectifier, 394
 - reverse polarity protection, 396
 - Schottky, 394

- diodes (*Cont.*):
 - voltage regulation, 202
 - zener, 394, 522
- DIP (dual in-line package), 401
- direct motor current measurement, 225
- discharge rate of batteries, 185
- discrete components, 428
- distance and proximity senses,
 - overview, 26
- distance sensor
 - defined, 572
 - depth and breadth, 572
 - far-object detection, 572
 - infrared light, 580
 - judgment sensor, 582
 - position-sensitive device (PSD), 580
 - ultrasonic, 572
 - wall following, 608
- DOF (degrees of freedom)
 - arm, robotic, 340
 - defined, 19, 320
 - leg joints, 320
 - operating terrain, 321
 - wrist, 344
- double-sided adhesive tape, for
 - battery pack, 194
- double-sided foam tape, using, 149
- DPDT switch, for motor control, 233
- drill bit
 - selection and use, 60
 - types and use, 55
- drill bit and tap sizes
 - imperial, table of, 673
 - metric, table of, 674
- drill chuck, 55
 - using, 62
- drilling
 - aligning holes, 62
 - basics of, 60
 - controlling hole depth, 62
 - creating layout for, 152
 - metal, 116, 118
 - plastic, 93
 - setting proper speed, 61
 - speed and bit table, 63
 - wood, 78
- drill press, and metalworking, 117
- drills, electric, 55
- drive geometry, definition of, 209
- drivetrain components, table of
 - commonly used, 277
- dual in-line package (*See* DIP)
- Dual Lock, by 3M, 148
- Du-Bro foam tires, 275
- Duemilanove Arduino, 463
- durometer, of wheel material, 299
- DWG, DXF (file formats), 158
- Dynalloy Flexinol, 287
- EEPROM data storage, in
 - microcontrollers, 442
- electrical metallic tubing (*See* EMT)
- electrical noise from motors, 229
- electric drills, robot construction
 - and, 55
- electric light kits
 - incandescent, 32
 - low-voltage LEDs, 32
- electric shock and first aid, 48
- electronic circuit, ground in, 190
- electronics
 - abbreviations used in, 679
 - letter symbols used in, 680
 - numbering used in, 680
 - units of measure used in, 681
- electronics parts sources, list of, 670
- electrostatic discharge (*See* ESD)
 - emitters/detectors
 - layout, 606
 - mounting, 612
- EMT (electrical metallic tubing), 113
 - strap for motor mounting, 269
- encoders, optical
 - conditioning, 613
 - mounting, 612
 - multimeter, testing with, 614
 - in odometry, 609
- encoders, optical (*Cont.*):
 - quadrature, 614
 - reflective disc, 609, 610
 - resolution, 611
 - transmissive disc, 609, 611
 - types, 609
- end effector (grippers),
 - defined, 19
 - robotic, 352
- endless round belts, 277
- environment settings, PICAXE
 - programming, 487
- EPS (Encapsulated PostScript), 158
- Erector set, 160
 - gears in, 286
 - as robotic parts, 10
- ESC motor speed controllers, 247
- ESD (electrostatic discharge)
 - defined, 45
 - prevention of, 45
- etchable PDF, 421
- expansion board, Arduino, 465
- expressions
 - order of precedence, 459
 - in programming, 448, 451, 456
- extension arm, 354
- eye injury and first aid, 48
- eye protection, 369
- eyes, for dolls, teddy bears, 32
- eye sensors, 590
- farads, as unit of measure, 391
- far-object detection, 572
- fasteners
 - basic hardware, 57
 - best selection for robotics, 134
 - brackets, using, 135
 - cable clamp, 150
 - double-sided foam tape, 149
 - drive styles, 132
 - head styles, 131
 - hook-and-loop, 148
 - introduction to, 129
 - machine screws, 131
 - machine screws, self-tapping, 134

- fasteners (*Cont.*):
 nuts, using, 132
 plastic ties, 149
 screws, wood and sheet metal, 131
 semipermanent, 147
 sizes, imperial and metric, 130
 tape, 149
 tapping threads for, 134
 Velcro, 148
 washers, using, 133
- FDM (fused deposition modeling), 159
- feedback with lights and sound, 639, 646
- filing, metal, 119
- fingers
 flexible, 359
 parallel, 355
- FIRST (For Inspiration and Recognition of Science and Technology), 11
 Tech Challenge kit, 11
- first aid, and project safety, 47
- Fischertechnik, 161
- flame detection, 655
 fire flickering, 656
 infrared heat, 656
 ultraviolet light sensors, 656
- flash, burrs, metal finishing and, 119
- flash program storage, in micro-controllers, 442
- flat washers, 133
- Flexinol (shape memory alloy), 287
- flex resistor, as pressure sensor, 562
- flex sensor, force-sensitive resistor, 390
- flippers, locomotion with, 217
- flow control
 in Arduino programming, 476
 in BASIC Stamp programming, 502
 in PICAXE programming, 490
- flowchart, in programming, 447, 487
 in PICAXE, 487
- flush wire cutters, 369
- foamboard, 32
 cutting, 146
 screws and nuts, using, 148
 used in rapid prototyping, 145
- Foam Core, 145
- foam PVC, 90
- foam rubber sheets, 32
- Forbidden Planet*, 14
- force-sensitive resistors, 390, 561
- forklift, robotic, cylindrical coordinate work envelope, 342
- Forrest Mims Engineer's Notebook*, 524
- forums and blogs sources, list of, 671
- found parts, explanation of, 170
- frame
 assembly, 77
 constructing using metal, 116
 constructing using plastic, 95
 construction steps for, 76
 metal materials for, 115
 miter box, using, 76
 from PVC irrigation pipe, 171
- Frankenstein
 likened to robotics, 3
 nuts and bolts to keep head on, 129
- Freeduino, Arduino-compatible board, 465
- Fritzing, ECAD software, 420
- full-bridge (motor control), 241
- fused deposition modeling (*See* FDM)
- fuse protection
 multimeter, 365
 resettable PPTC, 199
 selection of, 199
 slow-blow glass-type, 199
- Futaba
 connector style, 256
 drive sprockets, plastic, 316
- Futaba (*Cont.*):
 S3003 servo, modifying for continuous rotation, 264
- gait (walking)
 alternating tripod, 327
 defined, 327
 hexapod, 328
 metachronal, 327
- gates, defined, 236
- gauge
 as metal thickness measurement, 109
 wire, 370, 682
- gear reduction, 284
- gears, 277
 4WD robots, 308
 function, 283
 gearbox, gears in, 283
 introduction to, 283
 reduction ratio, 284
 specifications, 285
 types of gear teeth, 285
- gear trains, 283
 used in R/C servo, 254
- germanium diode, 395
- global declaration, in Arduino programming, 469
- global positioning satellite (*See* GPS)
- Global Specialties EXP-350 solderless breadboard, 412
- glue
 construction toys, 163
 cross-reference table of, 143
 cyanoacrylate adhesive, 139
 dots, sticks, transfer tape, 150
 hot, for plastics, 98
 hot-melt, 140
 household adhesive, application of, 98
 introduction to, 136
 joints, reinforcing, 141
 mounting R/C servos, 271
 paper, 32
 plastic bonding, characteristics table, 97
 for plastics, 96

- glue (*Cont.*):
 RTV adhesive, silicone-based, 164
 setting and curing, 136
 Shoe Goo, 164
 solvent cement, application of, 97
 Super Glue, 137
 two-part epoxy, 137, 138
- gm-cm motor specification, definition of, 224
- Google Sketch, 155
- GPS (global positioning satellite), 624
- graphics programs, creating layouts with, 154
- gripper
 bonus projects, 359
 defined, 19
 plastic tool clamp, construction of, 356
 robotic, 352
 two-pincher, advanced, 354
 two-pincher, basic, 353
 two-pincher, parallel, 355
- grit, defined, 79
- ground, defined, 190
- ground loop, prevention of, 379
- GWS, S-35 continuous rotation servo, 261
- gyro sensors, 621, 624
- hacking, of hardware
 appliances for robotics parts, 35
 First Alert smoke alarm, 658
 mini 4WD trucks, 166
 mouse, encoder wheels, 611
 R/C vehicles, 166
 smoke detector, 658
 snowmobile, metal treads, 310
 sources for gears, 286
 toy extension arm, 354
 toys and kits, 165, 166
 toy tanks, treads, 310
- hacksaw, 64
 cutting metal, 114
 robot construction and, 55
- half-bridge, 237
 Hamamatsu R2868 Flame Detector UVTron sensor, 656
- hardening, metal treatment for, 110
- hardwood versus softwood, 69
- H-bridge, 241, 244
 for BASIC Stamp, 507
 MOSFET transistor, using, 242
 in R/C vehicles, 166
- HDPE, defined, 88
- heat sensor, 568, 664
- heat treating metal, 110
- Heinlein, Robert, 15
- Hewitt, Robin, 601
- Hex3Bot robot, 9, 328
- hexagon, bases, 73
- Hitec
 connector style, 256
 HS-422, modifying for continuous rotation, 262
 R/C servos, 254
- hobby parts sources, list of, 670
- holonomic steering, 213
- Honeywell C7027 flame detector, 656
- hook-and-loop fasteners, 148
- hot-melt glue, 140
- household glue
 applying, 138
 characteristics, 137
- household plastics, constructing robots from, 98
- humanoid robot, 320
 defined, 19
- hydraulic power
 arm actuation with, 343
 robotics use of, 24
- Hyzod (plastic), 88
- I2C interface, 518
- IDE (integrated development environment), 467
 in Arduino programming, 468
- idler wheels, 278
- in-circuit programming, of microcontrollers, 444
- indirect motor current measurement, 226
- in-field programming, of microcontrollers, 444
- infrared detector
 adjusting sensitivity, 573
 gripper, mounting on, 573
- infrared distance sensor, 580
 analog or digital, 582
 analog output ranging, 583
 beam width, 581
 coding, 584
 distance judgment, 582
 electrical hookup, basic, 582
 working distance, 581
 using, 583
- infrared emitter/detector pairs, 606
- infrared light proximity sensor, 572
- infrared proximity detector (See IRPD)
- infrared receiver/demodulator, 538
 PICAXE, interface for, 538
- infrared remote control, 537
 DC motors, 541
 operating robot, 542
 universal, 537
- Inkscape, 158
 using for layout, 155
- input
 analog interfacing, 522
 buffer, 521
 current-limiting interface, 521
 digital, 520
 direct connection, 520
 protection with zener diode, 522
 sensors, 514
 signal amplification, 523
 switch, 520
- input, adding more
 demultiplexer, 528
 multiplexer, 529
 serial-to-parallel shift register, 528
- integer data type, 452

- integrated circuits
 - dual in-line package (DIP), 401
 - identifying, 401
 - microcontroller, 402
- integrated development board, 441
- integrated development environment (See IDE)
- internal resistance of batteries, 186
- Internet parts sources, list of, 669
- interrupts, in microcontrollers, 442
- Inventa, gear sets, 286
- invertible robot, PlyBot as, 85
- iRobot Create platform, 10
- IRPD (infrared proximity detector), 574
 - enhanced circuitry, 578
 - microcontroller, connecting to, 576
 - modulated, 574
 - using auxiliary microcontroller, 577
- jigsaw, cutting with, 65
- JohnnyRobot plastic treads, 310, 316
- joint, reinforcement of, 141
- joystick, 534
 - analog, IBM PC, 535
 - Arduino, 536
 - buttons, table for, 535
 - teaching pendant, 535
 - USB, 535
- jumper wires, solderless breadboard, 409
- K&S Engineering metal structural components, 116
- kerf, defined, 66
- K'NEX, constructions with, 163
- L293D motor control IC, 244
 - table, 246
- L298 motor control IC, 246
- laser cutting, 158, 159
- layout
 - computer programs, 154
 - direct, 153
 - paper, 153
 - paper, copying, 153
 - paper, transfer, 154
- lb-ft motor specification, definition of, 224
- LCD (liquid-crystal display) panel
 - characteristics, 647
 - interface types, 647
 - text display, 646
- lead-free solder, 372
- leaf switch touch sensor, 548
- LED (light emitting diode), 396
 - in Arduino programming, 471
 - colors, 397
 - common anode, cathode, 398
 - feedback circuit with, 639, 641
 - flashing patterns, 640
 - forward voltage drop, 386
 - limiting current to, 386
 - multicolor, 651
 - as photodiode, 593
 - powering, 397
 - size, shape, output, 397
 - specifications, 396
- LED, infrared
 - optical sensor, 604
 - resistor values, 605
- LED display
 - feedback, 642
 - numerals, displaying, 642
 - pictures and shapes, displaying, 644
- LEGO
 - balloon tires, 275
 - bricks, 88
 - constructions with, 163
 - gears in, 286
 - motor and component mounting, 164
 - Technic, rubber tank treads, 311
- legs
 - balance, 319
 - brackets, R/C servo, 323
- legs (*Cont.*):
 - locomotion with, 210, 216
 - multisegment, 217
 - operating terrain, 321
 - power supply, 324
 - R/C servos, 324
 - with SMA linear actuation, 293
 - X-Y joint, 322
- lenses and filters, using, 600
- lever switch, as touch sensor, 548
- Lexan (plastic), 88
- library code, in Arduino programming, 469
- light-emitting diode (See LED)
- light feedback, 639
- lighting effects, 649
 - bonus projects, 654
 - multiple LEDs, 650
 - superbright and ultrabright LEDs, 650
- light sensor
 - cyclops eye, building, 594
 - lenses and filters, 600
 - light-receptive robot, 595
 - light spoilage, 599
 - multiple-cell robotic eye, 596
 - optical filtration, 593
 - photodiodes, 593
 - photophilic and photophobic, 596
 - photoresistors, 591
 - spectral response, 593
- light spoilage, 599
 - with optical encoders, 613
- lightweight composites, as a construction material, 51
- LilyPad Arduino microcontroller board, 465
- linear voltage regulation, 204
- line following, 603
 - emitters/detectors layout, 606
 - optical sensors, 604
 - programming, 607
 - reflective tape, 604
- liquid-crystal display panel (See LCD panel)
- literal data type, 456

- literals, in programming, 450
- LM34 temperature sensor, 664
- LM386 audio amplifier IC, 631
- LM1881 sync separator IC, 602
- lock washers, 133
- locomotion
 - drive and mechanical considerations for, 209
 - legs, 23, 318
 - techniques in robotics, 209
 - tracks, 23
 - wheels, 22
- logical 0, LOW and HIGH, definition, 235
- logic probe, 368
- loop, in programming, 449, 455
- Lost in Space* robot, 14
- Lovejoy three-piece jaw coupler, 278
- Lucite (plastic), 88
- Lynxmotion
 - arm kits, 350
 - Phoenix, 18
 - plastic treads, 310
 - Servo Erector Set, 10
 - wheel flange, 274
 - X-Y joints, 323
- Mabuchi FA-130-size motor, 306
- machine screws, 131
- machine vision, 600
- magnetic encoder, 609
- Magnevation SpeakJet, 632
- Mars Rover *Sojourner*, 15
- mat board, as construction material, 146
- math operators, table of, 457
- McMaster-Carr
 - ball caster/transfer, 80
 - casters, 303
 - double-sided foam tape, 150
 - industrial-grade fasteners, 148
- MDF (medium-density fiberboard), defined, 72
- Meccano, 160
- mechanical construction, techniques for, 59
- mechanical encoder, 610
- medium-density fiberboard (*See* MDF)
- melting plastic, toxic fumes from, 45
- metal
 - bending, 119
 - characteristics and applications table, 110
 - conduit, 113
 - as construction material, 51
 - cutting using backsaw and miter box, 116
 - extruded aluminum, 111
 - heat treatments, 110
 - measuring thickness of, 109
 - painting, 120
 - properties, and thickness, 109
 - summary of use in robotics, 107, 111
 - tapping holes in, 118
 - where to find, 110
- metal, properties
 - alloys, 107
 - stress, 119
 - thickness, 114
- metal finishing
 - filing, 119
 - painting, 120
 - removing flash and burrs, 119
 - sanding, 120
- metal oxide substrate, 45
- Metropolis* (movie), depiction of robots in, 21
- microcontrollers
 - Arduino, 463
 - Arduino, low-level programmable, 439
 - AVR programming, 445
 - BASIC Stamp, 496
 - BASIC Stamp, integrated language programming, 439
 - debounce delays, 556
 - form factors, 440
 - hardware interrupts, 442
 - input/output pins, 442
 - integrated development environment, 467
 - integrated-language programmable, 439, 482
- microcontrollers (*Cont.*):
 - low-level programmable, 439
 - output interface, 519
 - parallel interface, 516
 - PICAXE, 482
 - PICAXE, integrated-language programmable, 439
 - pin functions, 443
 - programmers for, 438, 444
 - programming in-field, 444
 - programming languages, 446
 - programming limitations of, 443
 - programming steps, 440
 - programming, one-time, 444
 - R/C servo, controlling, 258
 - serial-based hardware, interfacing with, 518
 - serial communications, 517
 - serial interface, 517
 - SMA, actuating, 290
 - speed, 445
 - types of, 437
 - USB, connecting through, 525
- microcontrollers, programming languages
 - BASIC, 439
 - C, 440, 467
 - Pascal, 440
- microcontrollers, use in robotics, 437
- micrometer, machinist, 109
- microphone
 - amplifier, 635
 - best placement for, 635
 - electret condenser, 635
 - as touch sensor, 569
- Microswitch, 548
- Mims, Forrest M., III, 524
- mini-ITX motherboard, 432
- Mini T-bot, constructing, 172
- miter box, and frame assembly, 76
- MIT HandyBoard, 431
- mobile robots (comparison to stationary), 13
- modulated light, use in IRPD sensor, 574

- modules, robotics as building-blocks, 6
- Mondo-Tronics Muscle Wire, 287
- MOSFET transistors, 240, 241
 defined, 241
 electrostatic discharge, 45
 H-bridge, 242
 N-channel, P-channel, difference in, 242
 switch motor control, 241
- most significant bit (See MSB)
- motion detection sensor, accelerometer, 623
- motors
 4WD robot, 307
 AC, DC explained, 219
 centerline drive mount, 298
 common drivetrain components, 277
 continuous, stepping motor explained, 220
 current draw, 222
 front-drive mount, 298
 gear reduction, using with, 286
 holes for, 267
 loading, 222
 linked drive or separate motors, 307
 main specifications, 221
 micro-miniature and Plasto-Bot, 102
 mounting brackets, 268
 noise with, 200, 229
 operating voltage, 221
 R/C servo, mounting, 270
 servo, 220
 shaft types, 282
 speed, RPM, 222
 Tamiya worm gear, 80
 testing current draw, 224
 torque, 223
- motors, DC
 Arduino, using with, 479
 BASIC Stamp, controlled by, 507
 bridge module control, 244
 brushed and brushless, 231
- motors, DC (*Cont.*):
 drivetrain components to shaft, mounting, 272
 ESC motor speed controllers, 247
 MOSFET transistor, controlled by, 241
 mounting techniques, 266
 permanent magnet, 230
 PICAXE, remote control, 541
 pulse width modulation, 478
 ratings, 232
 relay controlled, 234
 relay half-bridge, 237
 reversible, 231
 serial control, 246
 speed, controlling, 247
 switch, controlled by, 232
 transistor, control, 239
 transistor, full-bridge, 241
 wheels, mounting, 273
- motors, gearbox, 283
- motors, R/C servo
 analog and digital, 325
 Arduino, 473
 BASIC Stamp, controlled by, 508
 controlling, 258
 drivetrain components to shafts, mounting, 272
 lubrication, 277
 mechanical linkages, attaching, 276
 modified, 261
 modified, limitations of, 265
 mounting, 270
 PICAXE, controlling with, 493
 serial controller, 325
 torque ratings, 326
 wheels, mounting, 274
- MOVITS robot kits, 165
- MP3 file sound playback, 631
- MSB (most significant bit), 451
- multimeter
 accuracy, 365
 batteries, testing, 366
 continuity test, 366
 functions, 365
 introduction to, 363
- multimeter (*Cont.*):
 optical encoders, testing, 614
 potentiometers, testing, 389
 range, selecting, manual or automatic, 364
 resistor, verifying value of, 367
 resistors, testing value of, 384
 safeguards, 367
 testing, 366
 test leads and supplies, 365
 using, basics, 366
- multiple-cell robotic eye, 596
 testing program, 597
- multiplexer, 529
 analog, 531
 bumper switches, controlling, 553
 digital, 530
- multisegment legs, locomotion with, 217
- Muscle Wire (shape memory alloy), 287
- music, output, 629, 631
 mux (data multiplexer), 529
- My First Robot, directions to, 667
- nail plate, as robot construction material, 124, 174
- navigational sensor, 624
- N-channel, P-channel MOSFET, difference in, 242
- near-object detection, 571
- netbook, used in robotics, 432
- nibbler tool, cutting metal with, 115
- NiCd battery, 181
- nickel cadmium (NiCd) battery, 181
- nickel metal hydride (NiHM) battery, 181
- Nickel Titanium Naval Ordnance Laboratory (Nitinol), 287
- Nitinol, 287
- N-m motor specification, definition of, 224
- noise, electrical
 decoupling capacitors for, 200, 206

- noise, electrical (*Cont.*):
 ground loops, 379
 prevention of, 378
- nominal battery voltage, 183
- noncontact sensing, 570
- nonholonomic steering, 213
- nonslip surfaces, and foam rubber, 32
- noxious gas
 detector, 661
 location, 663
 specific gravity table, 663
 warm-up period, 662
- Nubotics Wheel Watcher, 616
- numbered and fractional drill bits, table of, 675
- Number Five, example robot using tracks, 215
- numbering units, electronics, 680
- number limits of data types, table of, 452
- numeric values
 in programming, 449
 signed and unsigned, 451
- nut drivers, robot construction and, 56
- nylon, characteristics, 88
- octagon, bases, 73
- odometry, 609
 calculating distance and speed, 610
 errors, understanding, 617
 pulse accumulator, 612
- odometry codewheel
 mounting, 612
 quadrature encoding pulses, 614
 reflective and transmissive, 610
 resolution of, 611
- ohm, 382
- Ohm's law, 387, 677
 calculations, zener diode, 203
 in measuring motor torque, 227
 and resistors, 387
- Oilite bushing, 255
- omnidirectional steering, 214
- one-time programmable microcontroller (*See* OTP)
- operating voltage of motors, 221
- optical encoder, 609
- optical filtration for light sensors, 593
- optical sensor
 line following, 604
 mounting, 612
- opto-isolator, 521
- order of precedence, operators, 459
- Or truth table, 458
- oscillation, floating voltage, 378
- oscillator, voltage-controlled (VCO), 531
- OTP (one-time programmable microcontroller), 444
- output
 motors, 514
 power-handling requirements, 515
 sound and voice, 515
 visual indication, 515
- output interface, and microcontrollers, 519
- OWIKIT robot kits, 165
- oz-in motor specification, definition of, 224
- pads, foam rubber, 32
- painting
 metal, 120
 plastic, 98
 wood, 79
- paper foil, capacitor dielectric, 394
- Parallax
 BASIC Stamp, 496
 BOE-Bot robot kit, 429
 Continuous Rotation Servo, 261
 noxious gas detector, 661
 Ping, 265
 servo motor, modified, 124
- parallel interface
 communications, 435
 LCD panels, 647
 microcontrollers, 516
- parallel-to-serial port changing, 531
- parts
 from craft stores, 32
 from hobby and model stores, 31
 from online robotics retailers, 31
- parts storage
 antistatic, 46
 organization, 37
- Pascal programming language, 440
- PBASIC, 496
 BASIC Stamp, 500
 special functions, 505
- PCB (printed circuit board), 418
- PC motherboard, use in robots, 430
- peak inverse voltage, 395
- pentagon, bases, 73
- permanent magnet motor, 230
- personal computer (PC)
 Mini-ITX, 432
 Mini-ITX motherboard, 432
 netbook, mounted on robot, 432
 USB port, 433
 use in robots, 430
- Perspex (plastic), 88
- Philips head screws, 132
- photocell, 390
- photodiodes, as light sensors, 593
- photophilic and photophobic reactions, 596
- photoresist, 418
- photoresistors
 in fire detection, 656
 light sensors, 390, 591
 line following, 605
- phototransistors
 flame detection, 656
 light sensors 592
 mounting for odometry, 613
 optical sensors, 604
 resistor values, 605
- PIC (programmable integrated circuit), 496

- PICAXE
 08M, 577, 578
 08M, introduction to, 485
 18M2, introduction to, 486
 BASIC language, 487, 488
 functions for robotics, 492
 infrared proximity detector, enhanced, 578
 infrared proximity detector, simple, 577
 introduction to, 482
 pins and legs, 483
 program construction, 489
 remote control, 539
 serial port connection, 484
 serial port, programming through, 526
 piezoelectric film, 565
 bend sensor, constructing, 567
 leads, attaching to, 566
 as mechanical transducer, 566
 piezoelectric touch sensor, 563
 experimenting with, 563
 interface circuit, building, 564
 pillow block, couplers with, 281
 pin jumpers
 custom, 410
 solderless breadboard, 410
 pin mapping, Arduino, 467
 pins (inputs, outputs)
 Arduino, 470
 BASIC Stamp, 498, 502
 expanding, 528
 PICAXE, 483
 PISO (parallel-in, serial-out) IC, 532
 bumper switches, controlling, 554
 pitch, of gears, 285
 Pitsco
 TETRIX, 11, 350
 X-Y joints, 323
 planking (wood), use in robots, 71
 plastic
 bending and forming, 96
 characteristics and usability table, 89
 plastic (*Cont.*):
 as construction material, 50
 finishing, 96
 frames with, 95
 painting, 98
 ties as fasteners, 149
 varieties, 87
 plastic bases, from straight cuts, 94
 plastic project boxes, 171
 PlastoBot
 construction of, 100
 design variations, 105
 introduction to, 100
 using, 104
 Plexiglas (plastic), 88
 pliers, robot construction and, 55
 PlyBot
 cutting and drilling, 81
 introduction to, 80
 using, 85
 plywood
 thickness table, 70
 use in robots, 70
 pneumatic power
 arm actuation with, 343
 use in robotics, 25
 pneumatic shears, cutting metal with, 115
 point-to-point perforated circuit board, 416
 polar coordinate arm, 341
 Pololu
 casters, 303
 DC motors, 165
 FA-130-size motor, 306
 miniature motors and wheels, 101
 motor kits, mounting, 266
 servo motor, modified, 124
 polycarbonate, characteristics, 88
 polyester polypropylene, as dielectric material, 394
 polyethylene, characteristics, 88
 polystyrene, characteristics, 88
 polyvinyl chloride (PVC), characteristics, 89
 port changing, 531
 posing wire, bendable, 32
 position-sensitive device (*See* PSD)
 potentiometer
 Arduino, wiring, 471
 interface for, 524, 388
 multimeter, testing with, 389
 rating, value, 390
 taper, 388
 used in R/C servo motor, 253
 using, 389
 powder coating, 121
 power MOSFET transistor, 241
 power sources for robotics, review of, 179
 power systems used in robots, 24
 power tools, using, 67
 PPTC fuses, 199
 pressure angle, of gears, 285
 pressure sensitivity, 548, 558
 pressure sensor, 558
 antistatic conductive foam, 558
 microcontroller, connecting to, 560
 on/off pressure pad, 561
 reading resistance, 559
 printed circuit board (*See* PCB)
 priority encoder, bumper switches, controlling, 552
 programmable integrated circuit (*See* PIC)
 programming
 And truth table, 458
 assigning value to variable, 455
 bitwise port, 533
 branch, 449
 bumper switches, 557
 call and return statements, 454
 C language, case statement, 454
 code block, 469
 code library, 469
 comments in, 452
 conditional branching, 490
 conditional expression, 453
 conditional statement, 449

- programming (*Cont.*):
 - data input/output, 450
 - electromagnetic compass sensor, 619
 - environment settings, 487
 - expressions, 448, 456
 - flowchart, 447, 487
 - for/next statement, 454
 - fundamentals, 446
 - global declaration, 469
 - Go statement, 454
 - if statement, 453
 - integer, 452
 - languages for microcontrollers, 446
 - line-following robot, 607
 - loop, 449, 454, 455
 - numerical values, 449
 - Or truth table, 458
 - PBasic, 500
 - PICAXE, remote control, 539
 - PICAXE BASIC, 488
 - pulse counting, odometry, 614
 - routine, 447
 - Select Case, 453
 - simulator, PICAXE, 487
 - sketch, Arduino, 468
 - source code, 453
 - string, 449
 - strings, using operators with, 458
 - syntax checker, 487
 - ultrasonic distance sensor, 587
 - unconditional branching, 454
 - variables, 447
 - While/Wend statements, 455
- programming microcontrollers, steps in, 440
- programming, operators
 - And and Or, 458
 - math, 457
 - multiple, 459
 - order of precedence, 459
 - relational, 457
 - string, 458
- proto shield, for Arduino, 465
- prototype
 - quick-turn, 159
 - rapid, 144
- proximity sensor
 - adjusting sensitivity, 573
 - defined, 571
 - gripper mounted on, 573
 - infrared light, 572
 - wall following, 608
- PSD (position-sensitive device), defined, 580
- pseudo-code, 446
- pull-up resistor, 516
- pulse accumulator, and measuring distance, 612
- pulse and frequency management, 436
- pulse counter
 - measuring distance, 612
 - programming for odometry, 614
- pulseIn Arduino programming statement, 478
- pulse width modulation (*See* PWM)
- pushrod, 276
- PVAc glue, 137
- PVC
 - benefits of, 90
 - defined, 89
 - foam, 90
 - rigid expanded, 90
 - sheets, common thickness of, 91
 - as substrate, 91
 - weight and thickness table, 91
- PWM (pulse width modulation), 252, 436, 247
 - defined, 244
 - sound produced with, 628
- quadrature encoding, 615
- quick-turn prototyping, 159
- R2-D2 robot, 14
- radio-controlled vehicles, hacked for parts, 166
- radio frequency navigation sensors, 624
- radio signal remote control, 543
- RAM and EEPROM data storage, in microcontrollers, 442
- RAM data storage, in microcontrollers, 442
- rapid prototyping, 144
- razor saw, cutting with, 65
- RBB online support, main entry, 667
- R/C servo, 220, 249
 - analog, digital, 257
 - Arduino, using with, 473
 - ball bearing construction, 255
 - basic control, 249
 - brackets, mounting with, 270
 - components used inside, 250
 - connectors and wiring, 256
 - continuous rotation, 260, 261
 - controlling speed of, 252
 - dead band, 260
 - drivetrain components to shaft, mounting, 272
 - general specifications, 255
 - glue, mounting with, 271
 - horns for, 272
 - limiting rotation, 251
 - linear movement, controlling, 276
 - mechanical linkages, attaching, 276
 - microcontroller, controlled by, 258
 - modifying, 261
 - pulse, controlling by, 251
 - pulse duration modulation, 252
 - pulse width range, 252
 - recommended voltage, 259
 - screws, mounting with, 270
 - sensor turret, 265
 - serial controller, 259
 - transit time, 255
 - types and sizes, 253
- rechargeable/nonrechargeable batteries, 180, 191
 - alkaline, 181

- relational operators, table of, 457
- relay
 - characteristics, 404
 - common types, 404
 - driver electronics, 238
 - motor control, 234
 - motor direction control, 236
 - motor on/off control, 235
 - rating, 405
 - specifications for, 238
- remote control, 534
 - broadcast video, 543
 - infrared, 537
 - joystick, 534
 - radio signal, 543
 - teaching pendant, 535
 - telerobotics, 543
- RepRap, 159
- repurposing parts, 35
 - bicycle wheels, 299
 - CDs, 171
 - computer mice, 171
 - consumer electronics device, battery pack, 191
 - Erector set, 160
 - Fischertechnik, 161
 - flooring and countertop samples, 175
 - joystick, 534
 - K'NEX, 162
 - LEGO Technic, rubber tank treads, 311
 - metal hardware parts, 172
 - plastic tool clamp, 356
 - plumbing supplies, 176
 - radio-control, battery packs, 191
 - skateboard/inline roller skate wheels, 275
 - solderless breadboards, 171
 - storage containers, 170
 - from toys, 167
 - trash cans, 170
 - TV remote control, 537
 - wheelbarrow wheels, 300
 - wheelchair wheels, 299
- resistance
 - calculating, 678
 - Ohm's law, 387
- resistors
 - calculating resistance, 678
 - color code table, 383, 682
 - dividing voltage, 385
 - fixed, 382
 - force-sensitive, 390, 561
 - limiting current, 385
 - limiting current, Ohm's law calculation, 388
 - multimeter, testing value, 367
 - photoresistors, 390
 - potentiometers, 388
 - power dissipation, 384
 - pull-up, 516
 - rating, unit of value, 382
 - series, potentiometers, 389
 - series connection, 385
 - value, testing of, 367, 384
 - value tolerance, 383
 - variable, 388, 390
 - wattage, 384
- Restriction of Hazardous Substances (RoHS), 379
- reverse polarity protection, and diodes, 396
- reversible motor, 231
- revolute coordinate arm, 341
 - construction of, 345
- Revolution Education PICAXE microcontroller, 483
- revolutions per minute (*See* RPM)
- rigid expanded PVC, 90
- Robby the Robot (*Forbidden Planet*), 14
- RoboRealm vision analysis software, 601
- robot
 - definition, 16, 27
 - first use of term, 28 (*see also* robots)
- Robot B-9 (*Lost in Space*), 14
 - as example of robot using tracks, 215
- robot bases
 - metal, 114
 - plastic, 93
 - wood, 72
- robota, 28
- Robot Builder's Bonanza*
 - online support (*See* RBB online support)
- robot-human interaction, 639
 - lighting effects, 649
- robotic eyes, 590
- robotic senses, overview, 25
- robotics parts sources, list of, 669
- robot locomotion systems, defined, 209
- robot navigation, introduction to, 603
- robots
 - hardwired, discrete components, 428
 - manually operated, 428
 - personal computer as brains, 431
 - single-board computer, 430
- robots, programmable
 - microcontroller, 429
 - Smartphones, tablets, PDAs, 434
 - USB port, connecting through, 433
 - variations in, 430
 - wireless connection, 433
- robot shop, ideal locations, 58
- robot-to-human interaction (feedback), 639
- RoHS (*See* Restriction of Hazardous Substances)
- roller chain, 277
- rotary cutter, 32
- rotor, use in DC motor, 230
- routine, in programming, 447
- RoverBot bonus project, 306
- rover robot, defined, 18
- RPM (motor speed), defined, 222
 - calculating speed in R/C servo motor, 255
- R.U.R., 28
- safety
 - cutting tools, 43, 65
 - ear protection, 59
 - eye protection, 44, 59

- safety (*Cont.*):
 - multimeter, 367
 - soldering, 44
- sanding
 - metal, 120
 - wood, 79
- sandpaper
 - grit and use tables, 79
 - plastic, for sanding, 96
- sawing
 - basics, 64
 - circles, 73
 - circuit boards, 416
 - controlling cutting depth, 66
 - creating layout for, 152
 - metal, using a hacksaw, 114
 - miter box, using, 76
 - substrate sheets, 146
 - techniques for plastic, 92
 - tool choice and speed table, 65
 - wheel wells, 74
- Scalable Vector Graphics (*See* SVG)
- schematics
 - battery symbols, 190
 - reading symbols, 381
- Schmitt trigger inverter IC, 555, 613
- screwdrivers, robot construction and, 54
- screws, wood and sheet metal, 131
- screws and nuts, for robot construction, 57
- sealed lead-acid (SLA) battery, 182
- security camera, as vision sensor), 602
- segmented chain, 308
- segmented robots, 319
- self-contained autonomous robot, defined, 15
- semipermanent fasteners, 147
- sensor
 - digital and analog, 512
 - heat, 664
 - introduction to, 512
 - light and vision, 590
 - navigation, 603
 - sensor (*Cont.*):
 - proximity and distance, 570
 - sound, 625
 - touch, 547
 - types, 514
 - sensor turret, 265
 - X-Y joint, 338
 - serial communications
 - Arduino, 477
 - interface, 435
 - microcontrollers, 517
 - serial interface
 - hardware, 518
 - LCD panels, 648
 - microcontrollers, 517
 - serial monitor window, in Arduino, 477
 - serial motor control, 246
 - serial peripheral interface (SPI), 519
 - serial port
 - BASIC Stamp, connecting to, 499
 - microcontroller, connecting through, 526
 - PICAXE, connecting to, 484
 - protocols, and microcontrollers, 518
 - serial servo controller, 325
 - serial servo controller (*See* SSC)
 - serial-to-parallel port changing, 531
 - servo (*See* R/C servo)
 - set, saw teeth, 66
 - shape memory alloy (*See* SMA)
 - Sharp
 - GP2D12 infrared sensor, 583
 - infrared distance-measuring module, 581
 - sheet metal, working with, 115
 - shield
 - expansion board, Arduino, 465
 - printed circuit board layout, 420
 - shiftOut, Arduino programming statement, 478
 - shift register, 528, 531
 - cascading, 533
 - shock and vibration sensor,
 - accelerometer, 623
 - signal amplification, 523
 - silicon diode, 395
 - silicone glue, 137
 - silver mica capacitor dielectric, 394
 - Simpson Strong-tie, 112, 124
 - Mini T-bot construction, 172
 - strapping T weight table, 173
 - single-board computer
 - form factors, 431
 - kits, 431
 - operating system, 430
 - use in robots, 430
 - sinking current, 515
 - Sintra (plastic), 21, 91
 - SIPO (serial-in, parallel-out) chip, 532
 - siren sound effects, 627
 - Six Million Dollar Man, The*, as cyborg example, 16
 - sketch
 - Arduino programming, 468
 - Arduino programming libraries, 469
 - skid
 - applications for, 303
 - materials, 302
 - stability, 302
 - slew rate, in R/C servo motor, 255
 - slotted head screws, 132
 - SMA (shape memory alloy), 287
 - actuated with batteries, 290
 - actuated with LM555 timer IC, bonus project, 290
 - actuated with microcontroller, 290
 - bias force, 289
 - composition and characteristics, 287
 - crimp-on connectors, 288
 - defined, 287
 - linear actuation, 293
 - mechanical applications, 291
 - terminating, mechanically, 288
 - smartphones, tablets, PDAs, use in robots, 430

- smoke detection, 657
 - limitations, 661
- smoke detector, microcontroller interface, 660
- Sojourner*, Mars rover, 15
- solar power, use in robotics, 24
- solder
 - clamp, 369
 - rosin core, 44, 372
 - rosin flux remover, 372
 - silver, silver-bearing, 372
 - types, 372
 - vacuum, 372
- soldering, 371
 - cleaning, 372, 375
 - heat sink for, 372
 - how-to tips, 374
 - instructions for, 371
 - iron, 368
 - lead-bearing and lead free, 372
 - pencil, 368
 - station, 373
 - temperature setting, 373
- soldering tools, 371
 - cleaning, 375
- solderless breadboard, 407
 - circuit, constructing, 411
 - contact points, 408
 - mounting, 412
 - pin jumpers, 410
 - shield, Arduino, 465
 - size and layout, 408
 - using tips, 413
 - wire, connecting, 409
 - wire, jumper, 409
 - wire, length table, 409
- solderless breadboard shield, 465
 - as robot body, 171
- solvent cement, 138
- Sony remote control, 538
- sound amplifier, 631
- sound effects, 625
 - kits, 627
 - pulse width modulation, 628
- sound feedback, 646
- sound files, music, prerecorded, 631
- sound modules
 - microcontroller, controlled by, 626
 - preprogrammed, 625
- sound, output
 - microcontroller use, 628
 - musical scale notes table, 629
 - sirens, warning sounds, 627
- sound, playback, with microcontroller, 631
- sound sensor, 634
 - bonus projects, 637
 - microcontroller, connecting to, 636
 - threshold, 637
- sound statement, PICAXE, 629
- sourcing current, 515
- SpeakJet sound IC, 632
- spectral response of light sensors, 593
- speech synthesis, 632
- speed
 - calculating, and wheel diameter, 301
 - gear use to decrease, 284
 - linear, calculating, 300
 - of motor (RPM), 222
- SPI interface, 519
- “Spock’s Brain” episode (*Star Trek*), 427
- spray paints, and wood, 79
- spring whisker, 550
- SpringRC continuous rotation servo, 261
- SPST switch, for motor control, 233
- SSC (serial servo controller), 325
- stainless steel, 108, 113
- Star Trek*, 427
 - Captain Pike’s limited vocabulary, 450
- static and dynamic balance, 319, 328
- stationary versus mobile robots, 13
- steel
 - bending, 119
 - filing, 120
- steel (*Cont.*):
 - gauge, 135
 - nail plate, 124
 - painting, 120
 - robot construction with, 108
 - stainless, 108
 - tie plates, 112
 - tubes, pipes, and angles, 113
- steering circle, defined, 298
- stepping motors, 220
- sticky tape, using, 149
- STK500 programmer, 445
- strapping T, size and weight table, 173
- string, in programming, 449
- stripboard, 417
- Styrofoam, 145
- substrates, as construction material, 144
- substrate sheets, cutting, 146
- Super Glue, 97, 139, 164
- surplus, getting parts from, 34
- SVG (Scalable Vector Graphics), 158
- switches
 - Arduino, 479
 - BASIC Stamp, 506
 - bounce, 520, 555
 - bumper, 551
 - center-off, 403
 - contact area of, 549
 - contact bumper, 549
 - debounce, 555
 - identifying, 403
 - leaf, 548
 - momentary, 402
 - NC (normally closed), 404
 - NO (normally open), 404
 - PICAXE, 494
 - poles, 402
 - throw, 402
 - as touch sensor, 548
- switching voltage regulation, 205
- switch motor control, 232
 - MOSFET, 241
- symbol, variable, 488
- tactile feedback, 547
- tail wheel stability, 303

- Tamiya
 - ball caster, 173
 - ball caster, for PlyBot, 84
 - interlock mounting, 272
 - Jr. toys, for hacking parts, 167
 - Ladder-Chain Sprocket set, 309
 - motor kits, mounting, 266
 - Remote Control Bulldozer kit, 311
 - Track and Wheel set, 309, 311
 - Tracked Vehicle Chassis Kit, 311
 - Twin Motor gearbox, 165, 237
 - Twin-Motor kit, 306
 - worm gear motors, 80
- tank-style robot, 216, 309
 - metal hardware plates, 174
 - tracks, 316
- tantalum electrolytic capacitor, 394
- tape measure
 - robot construction and, 54
 - using, 60
- tapping
 - metal, 118
 - plastic for fasteners, 135
 - threads for fasteners, 134
- T-bone robot, description of, 85
- teaching pendant, 535
- teleoperated robot, defined, 14
 - accelerometer, controlling with, 624
 - operated by, 433
 - video feedback, 543
 - Waldo, 15
 - wireless cameras, 544
- temperature sensor, 664
- temperature setting for soldering, 373
- tempering, metal treatment for, 110
- tethered versus self-contained robots, 15
- TETRIX, Robotic Design System, arms, 350
- thermopile sensor, and flame detection, 657
- tilt sensor, 620
 - accelerometer, 620, 621
 - dual-axis, 621
- timers, counters, in microcontrollers, 442
- timing belts, 277
- tin, and robot construction, 108
- TinBot
 - construction, 124
 - introduction to, 122
 - using, 127
- Toki Corp. BioMetal (shape memory alloy), 287
- tool clamp gripper, plans for, 355
- tools
 - air, 68
 - hand, 54
 - power, 67
 - for robot construction, 54
- tooth geometry of gears, 285
- torque
 - in digital servos, 258
 - gears to increase, 284
 - legs, lifting, 324
 - measurement, direct and indirect, 225
 - rating, running and stalled, 224
 - R/C servo motor, 255
 - servos, ratings, 326
 - specification, 223
 - wheel diameter, relating to, 300
- touch sensor
 - accelerometer, 623
 - air pressure, 568
 - bumper, 551, 568
 - conductive coatings, 562
 - contact area of, 549
 - fabric, 569
 - flex resistor, 562
 - force, 561
 - introduction to, 547
 - piezoelectric, 563
 - piezoelectric bumper bar, 568
 - piezoelectric film, 565, 567
- touch sensor (*Cont.*):
 - pressure, 558
 - switch, 548
 - tactile feedback, 547
 - wall following, 608
 - whiskers, 549
- toxic fumes, during soldering, 44
- toys, hacking, 161
- tracked robot base, 311
- tracks, locomotion with, 210, 215
- traction
 - compliance, 301
 - detracking, 317
 - wheel selection for, 301
- transistor, 399
 - connections, BJT, 400
 - control of motors, 239
 - identifying, 399
 - limiting current to, 385
 - MOSFET, 400
 - motor control, DC, 239
 - NPN, PNP, 400
 - rating, 399
 - signal and power, 399
- transit time, R/C servo motor, 255
- trash cans, as robot bodies, 170
- treads, 309
 - detracking, 317
 - differential steering, 215
 - foam rubber, 32
 - locomotion with, 210, 215
 - materials, 309
 - plastic, 316
 - rubber, 316
 - rubber, flexible, 310
 - slipping, 215, 316
 - steering, 316
 - Tamiya, 311
 - tank steering, 216
 - from toy vehicles, 166, 167
 - tracked robot base, 311
 - tricycle steering, 213
 - Trinity College Fire Fighting Contest, 665
 - TurboCAD, 156
 - two-part epoxy, applying, 139

- treads (*Cont.*):
 two-pincher gripper, construction plans for, 353
 two-wire interface, 518
- ultrasonic distance sensor, 572, 585
 programming, 587
 specifications, 588
 time-of-flight calculation, 586
 using, 587
- ultrasonic ranging, 585
- ultraviolet light sensors, and flame detection, 656
- unconditional branching, 454
- units of measure, in electronics, 681
- US Digital odometer kits, 617
- USB
 Arduino, programming with, 466
 BASIC Stamp, programming with, 499
 hubs, 526
 microcontroller, connecting through, 525
 port programming, 433
 U.S. Naval Ordnance Laboratory, 287
- Valiant Technologies, Inventa, 162
- variables
 assignment, 455
 in programming, 447, 451, 469
- VCO (voltage-controlled oscillator), 531
- VCRs, robotics parts from, 36
- vector graphics, 154
 Bezier curves, 155
 file formats, 158
 Inkscape, 155
 shapes stacking order, 159
- Velcro, 148
 for battery pack, 194
- Vex Robotics Design System, 10
 plastic treads, 310, 316
- video signal
 analog video transmitter, 544
 through data radio, 543
 standards for, 602
- video vision, 600
 analog, 602
 digital, 600
 image analysis, 600
 microcontroller-based, 601
 PC-based robot, 600
 Webcam, 600
- voltage
 Arduino, 466
 brownout, 184
 comparator, 523
 dielectric breakdown, 391
 dividing with potentiometers, 389, 524
 dividing with resistors, 385
 drop, 228
 floating, 378
 forward voltage drop, diodes, 395
 incremental voltage drop, diodes, silicon, 396
 monitor, for batteries, 208
 Ohm's law, 387
 peak inverse, 395
 piezoelectric-produced, 563
 for walking robot motors, 324
 working, capacitor rating, 393
- voltage-controlled oscillator (*See* VCO)
- voltage regulation, 201
 linear, 204
 multiple systems, using, 206
 silicon diodes, 202
 switching, 205
 zener diodes, 203
- voltage regulator, in Arduino, 466
- volt-ohm meter, introduction to, 363
- walking robot
 construction materials, selection of, 321
 defined, 18
- walking robot (*Cont.*):
 Hex3Bot, 328
 power supply, 324
- wall following, 608
- washers, using as fasteners, 133
- watts, and Ohm's law, 387
- WAV file sound playback, 631
- Webcam, 600
- weight distribution, motors and wheels, placement of, 298
- wheels
 bottle caps, repurposed, 171
 car-type steering, 212
 caster, tricycle base, 213
 centerline drive motor mount, 298
 custom-made for R/C servo motors, 275
 diameter and width, 300
 differential steering, 211, 300
 foam tires, lightweight, 275
 front-drive motor mount, 298
 holonomic steering, 214
 idler, 278
 locomotion with, 210
 materials for, 299
 modifying to match motor, 273
 modifying with setscrews, 273
 mounting, DC motors, 273
 mounting, R/C servo motors, 274
 multidirectional, 298
 O-rings, made from, 275
 placement and turning circle, 300
 skateboard/inline roller-skate, 275
 speed, calculating, 301
 stability, support, 302
 synchronized omnidirectional steering, 214
 toy vehicles, repurposed from, 167
 traction, 300
 tricycle steering, 213
 wells, benefits of, 101
 width considerations, 300
- Wheel Watcher, quadrature encoding, 616

- whegs (wheel legs), locomotion
 - with, 217
- whiskers
 - fiber-optic, 568
 - spring, 550
 - as touch sensors, 549
- windings, use in DC motors, 230
- wire
 - circuit board, wrapping, 423
 - connectors, 375
 - gauge, 370, 682
 - insulation, 370
 - single stranded and solid conductor, 371
- wireless applications
 - Arduino, 464
 - cameras, 544
- wire strippers, 369
- wire wrapping, 422
 - tools, 423
- wood
 - as construction material, 50
 - from craft stores, 32
 - finishing, 78
 - painting, 79
 - planking, dimensions table, 71
 - selection, hardwood/softwood, 69, 175
- work envelope
 - defined, 19
 - robotic arms, 340
- working voltage, capacitors, 393
- wrist, DOF, 344
- X-Y joint
 - constructing, 336
 - custom-made, 335
 - as sensor turret, 338
 - wrist, 344
- zener diodes, and voltage regulation, 203

Notes

Notes
