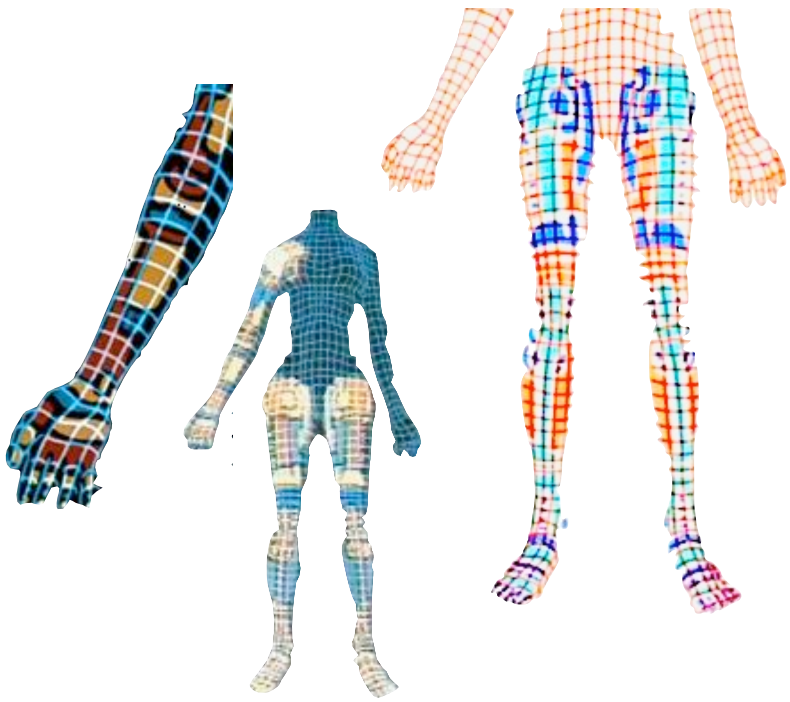
Bionic Arduino

Introduction to Microcontrollers with Arduino



Class 4

20 Nov 2007 - machineproject - Tod E. Kurt

What’s for Today

- About PWM
- Controlling Servos
- About the I2C bus
- Using I2C on Arduino
- About Accelerometers
- Nintendo Wii Nunchuck as Input Device

Recap: Blinky LED

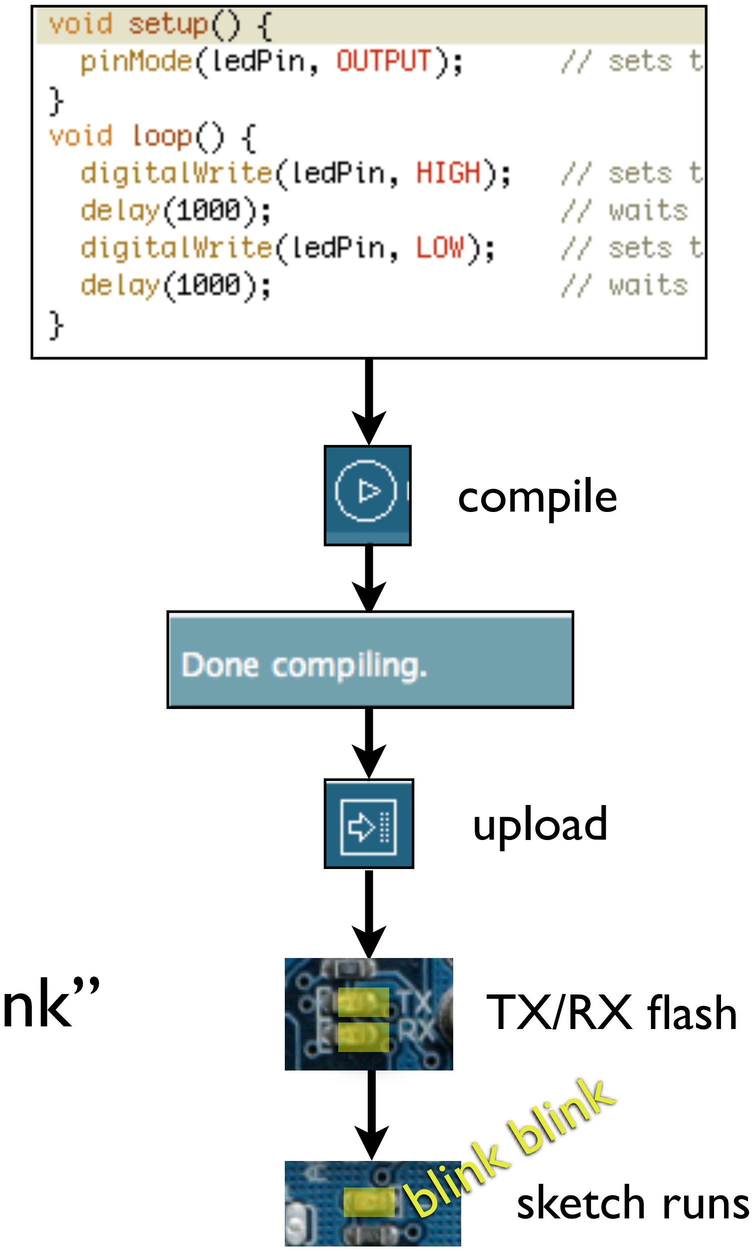
Make sure things still work

```
int ledPin = 13;                 // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);       // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);    // sets the LED on
  delay(1000);                   // waits for a second
  digitalWrite(ledPin, LOW);     // sets the LED off
  delay(1000);                   // waits for a second
}
```

Load “File/Sketchbook/Examples/Digital/Blink”



Change the “delay()” values to change blink rate

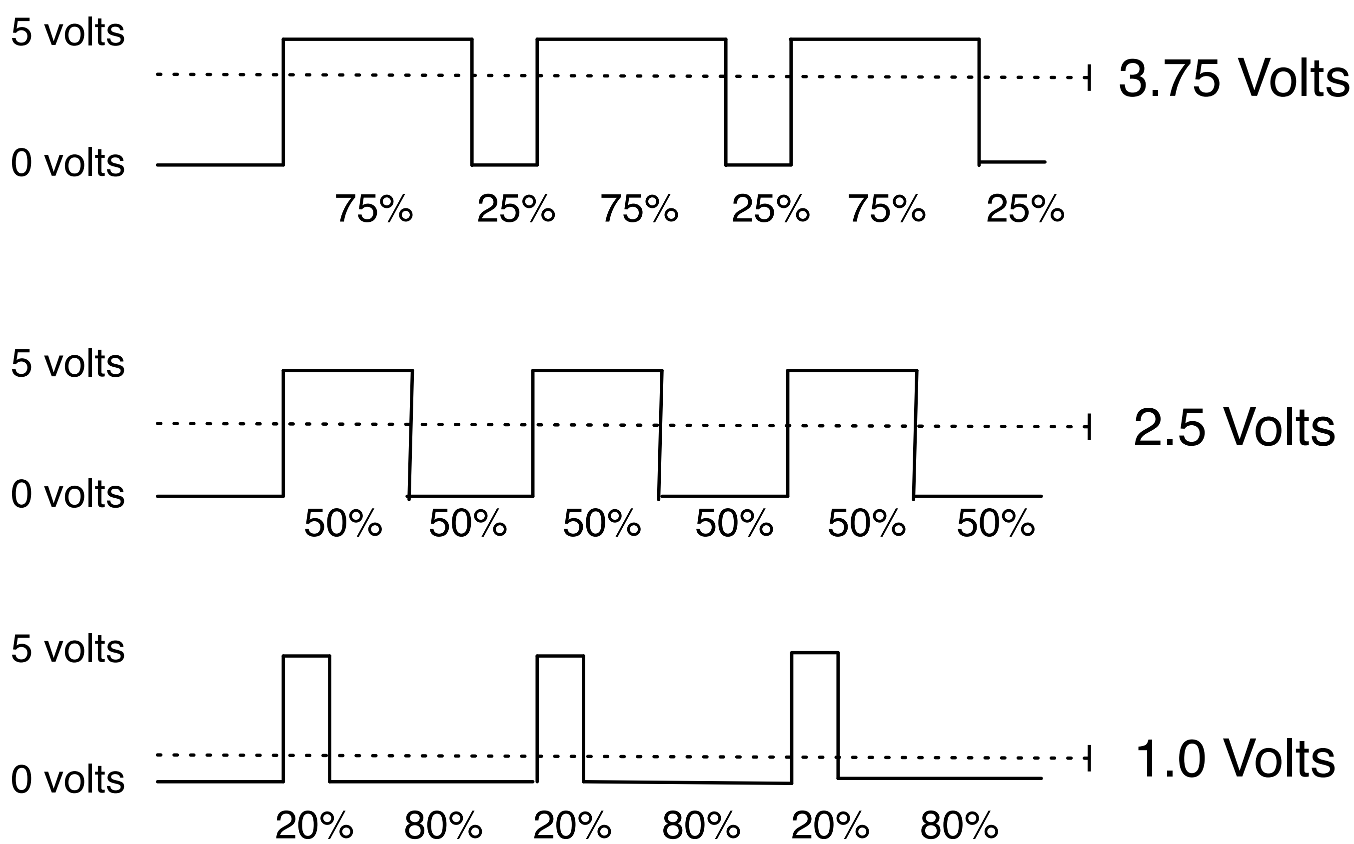
Pulse Width Modulation

- More commonly called "PWM"
- Computers can't output analog voltages
  - Only digital voltages (0 volts or 5 volts)
- But you can fake it
  - if you average a digital signal flipping between two voltages.
- For example...

PWM

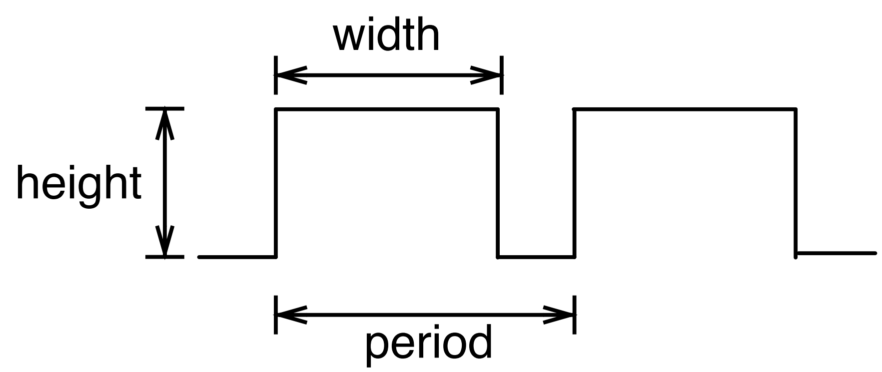
Output voltage is averaged from on *vs.* off time

output_voltage = (on_time / off_time) * max_voltage



PWM

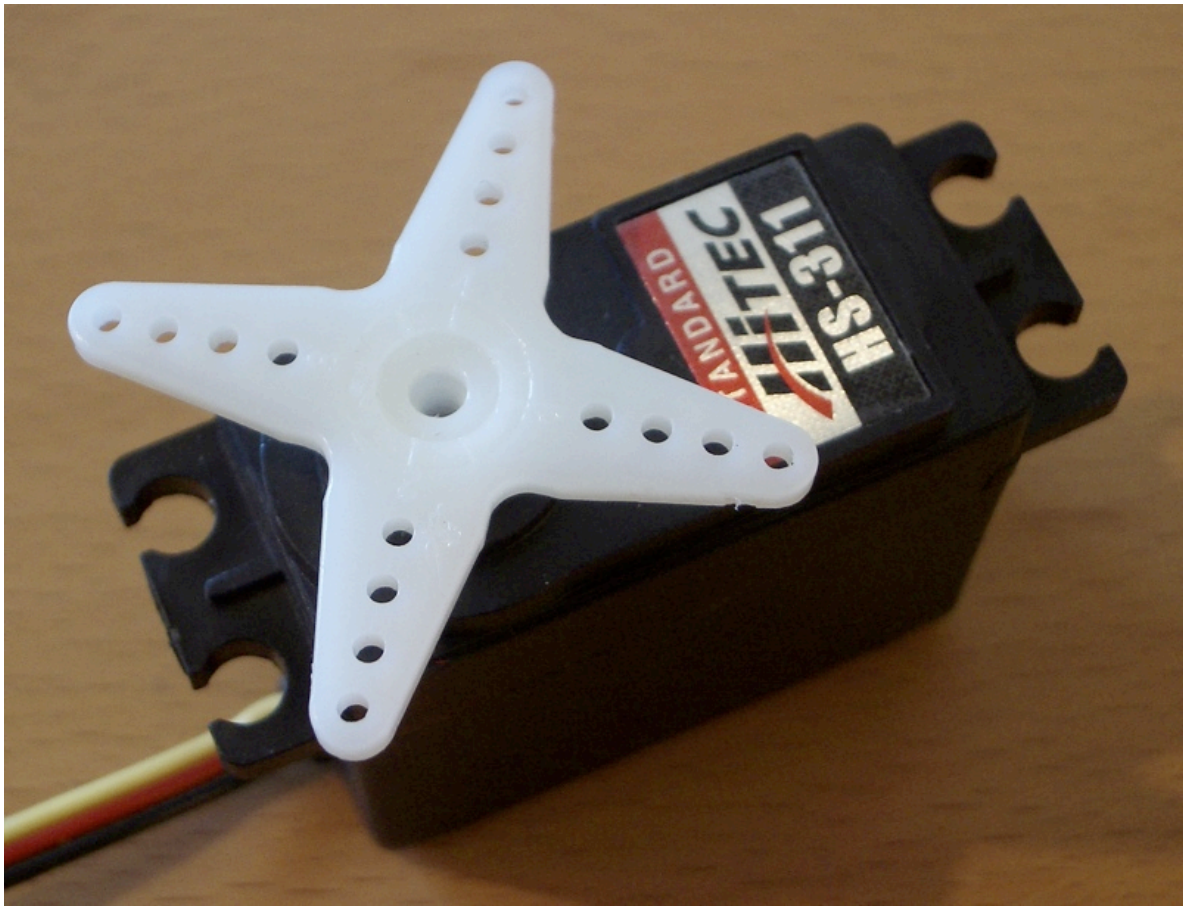
- Used everywhere
  - Lamp dimmers, motor speed control, power supplies, noise making
- Three characteristics of PWM signals
  - Pulse width range (min/max)
  - Pulse period
    (= 1/pulses per second)
  - Voltage levels
    (0-5V, for instance)



You experienced a few applications of PWM already.

Servomotors

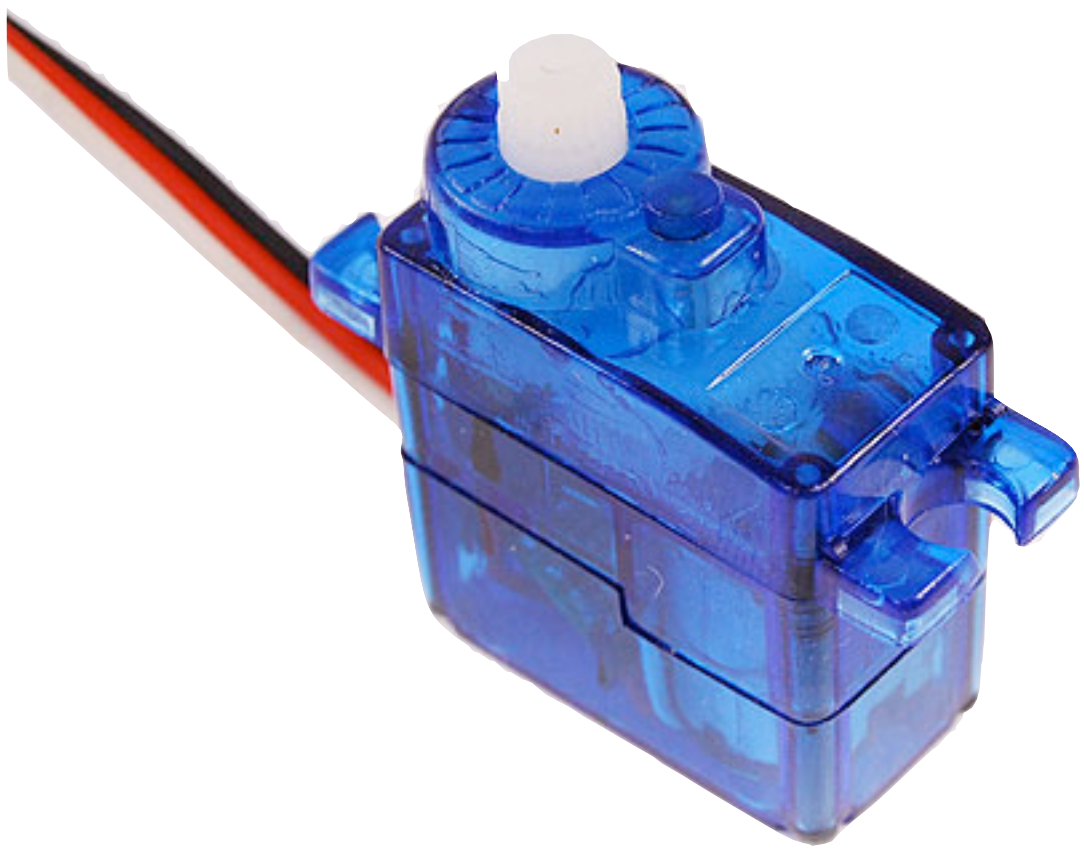
- Can be positioned from 0-180° (usually)
- Internal feedback circuitry & gearing takes care of the hard stuff
- Easy three-wire PWM 5V interface



More specifically, these are R/C hobby servos used by remote control enthusiasts
In general, "servomotor" is a motor with an inherent feedback mechanism that allows you to send position commands to it without requiring you to do the position reading.

Servos are Awesome

- DC motor
- High-torque gearing
- Potentiometer to read position
- Feedback circuitry to read pot and control motor
- All built in, you just feed it a PWM signal



With these little blue ones you have, you can see inside a bit at the internals of the servo.

Servos, good for what?

- Roboticists, movie effects people, and puppeteers use them extensively
- Any time you need controlled, repeatable motion
- Can turn rotation into linear movement with clever mechanical levers

Even clothes use servos now: http://www.technologyreview.com/read_article.aspx?id=17639&ch=infotech

Servos

- Come in all sizes
  - from super-tiny
  - to drive-your-car
- But all have the same 3-wire interface
- Servos are spec'd by:

weight: 9g
speed: .12s/60deg @ 6V
torque: 22oz/1.5kg @ 6V
voltage: 4.6~6V
size: 21x11x28 mm

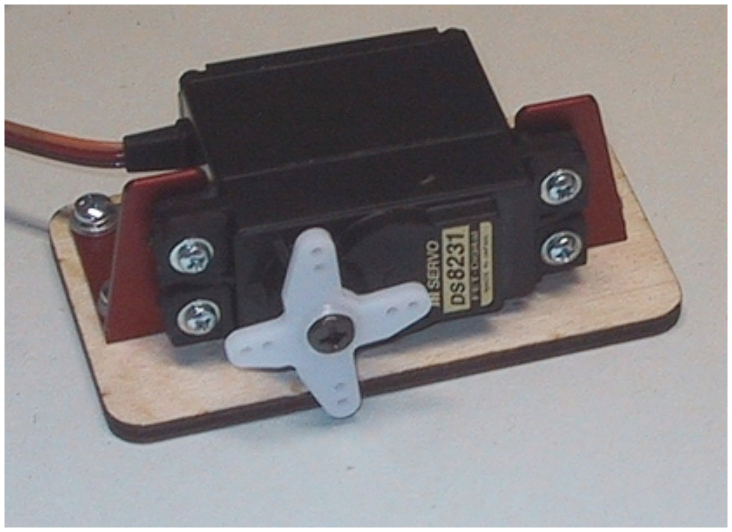
9g

157g

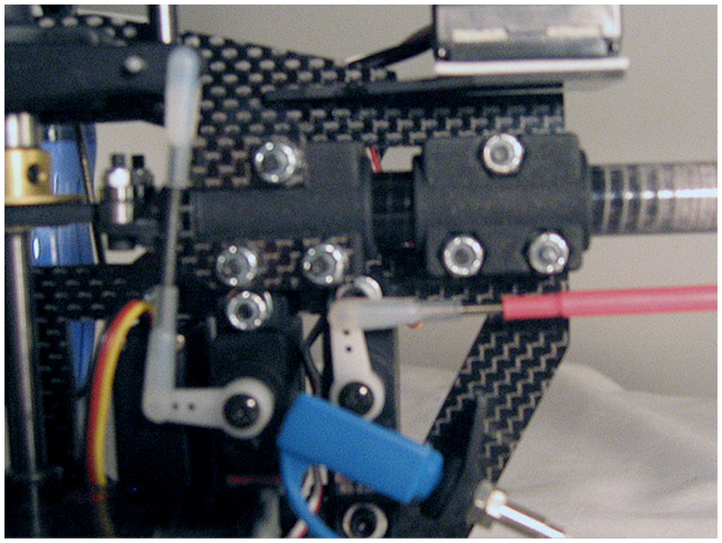
http://rctoys.com/
http://hobbypeople.net/

Servo Mounts & Linkages

Lots of ways to mount a servo

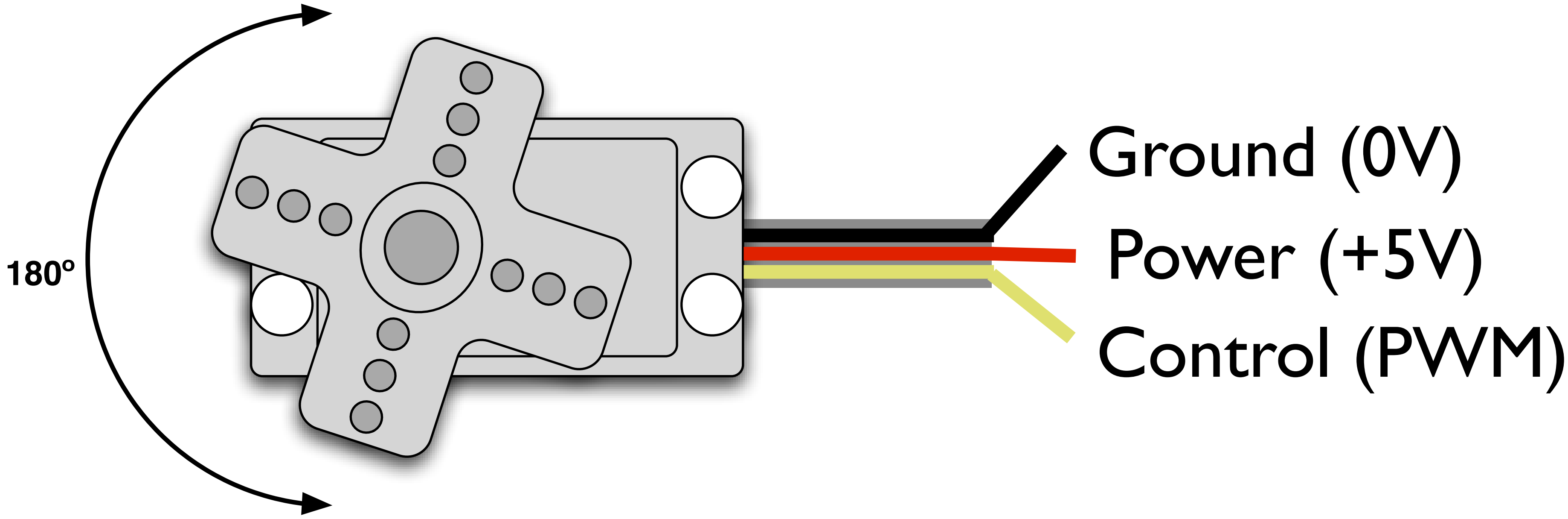


And turn its rotational motion into other types of motion



mounting bracket: http://www.sierragiant.com/prod28.html

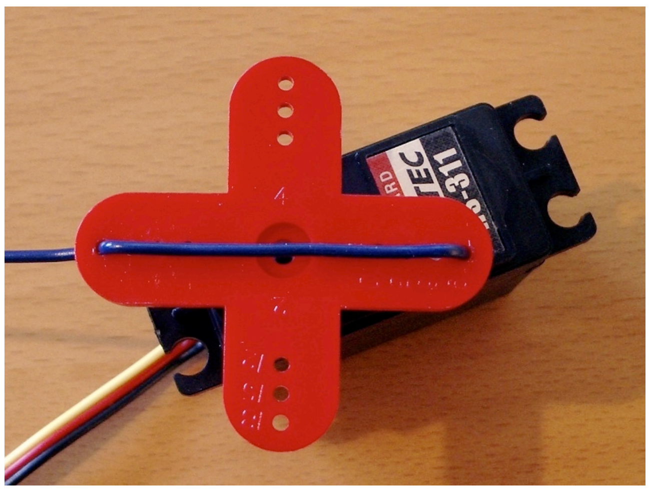
Servo Control



- PWM freq is 50 Hz (i.e. every 20 millisecs)
- Pulse width ranges from 1 to 2 millisecs
  - 1 millisec = full anti-clockwise position
  - 2 millisec = full clockwise position

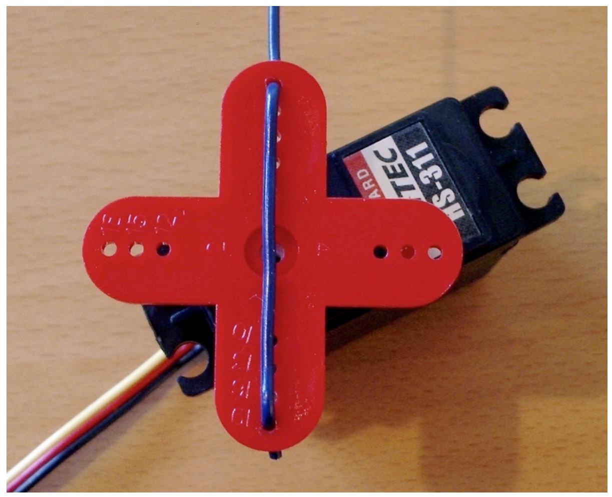
Servo Movement

0 degrees



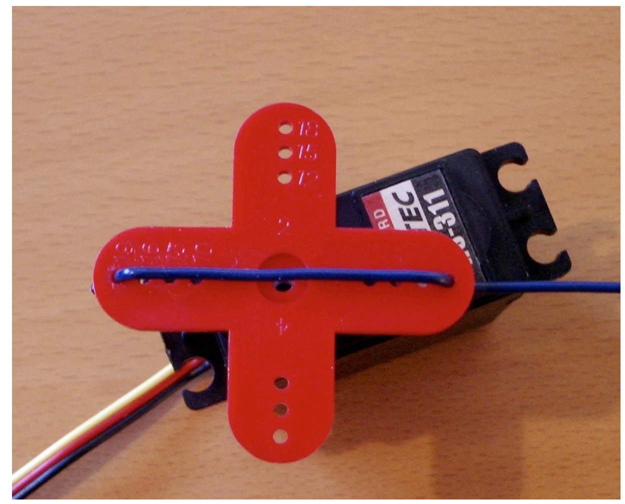
1000 microsecs

90 degrees



1500 microsecs

180 degrees



2000 microsecs

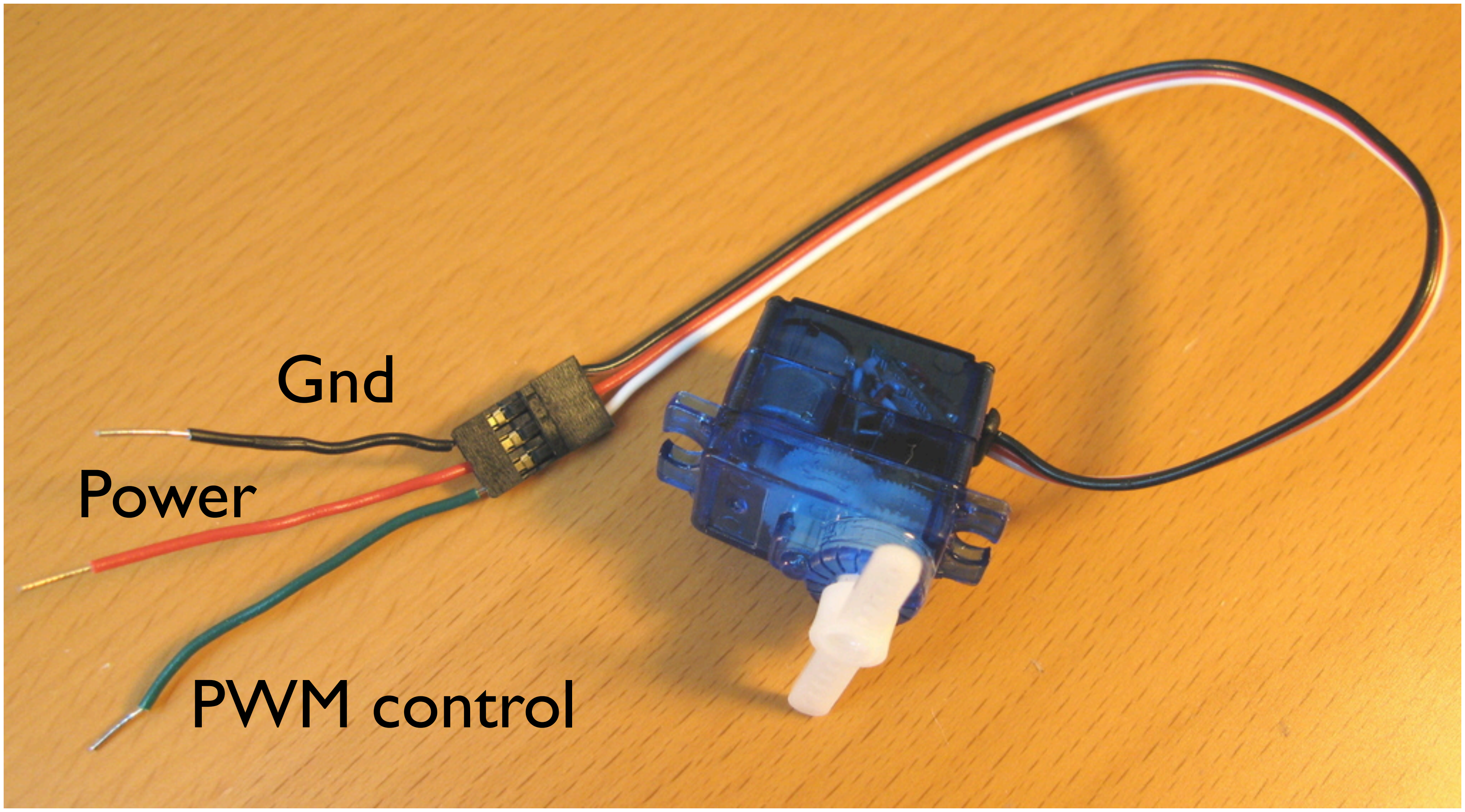
In practice, pulse range can range from 500 to 2500 microsecs

(and go ahead and add a wire marker to your servo like the above)

Put the red “arm” on your servo. Needs a philips screwdriver.
Many commercial servo drivers have a calibration setting to deal with servo variability

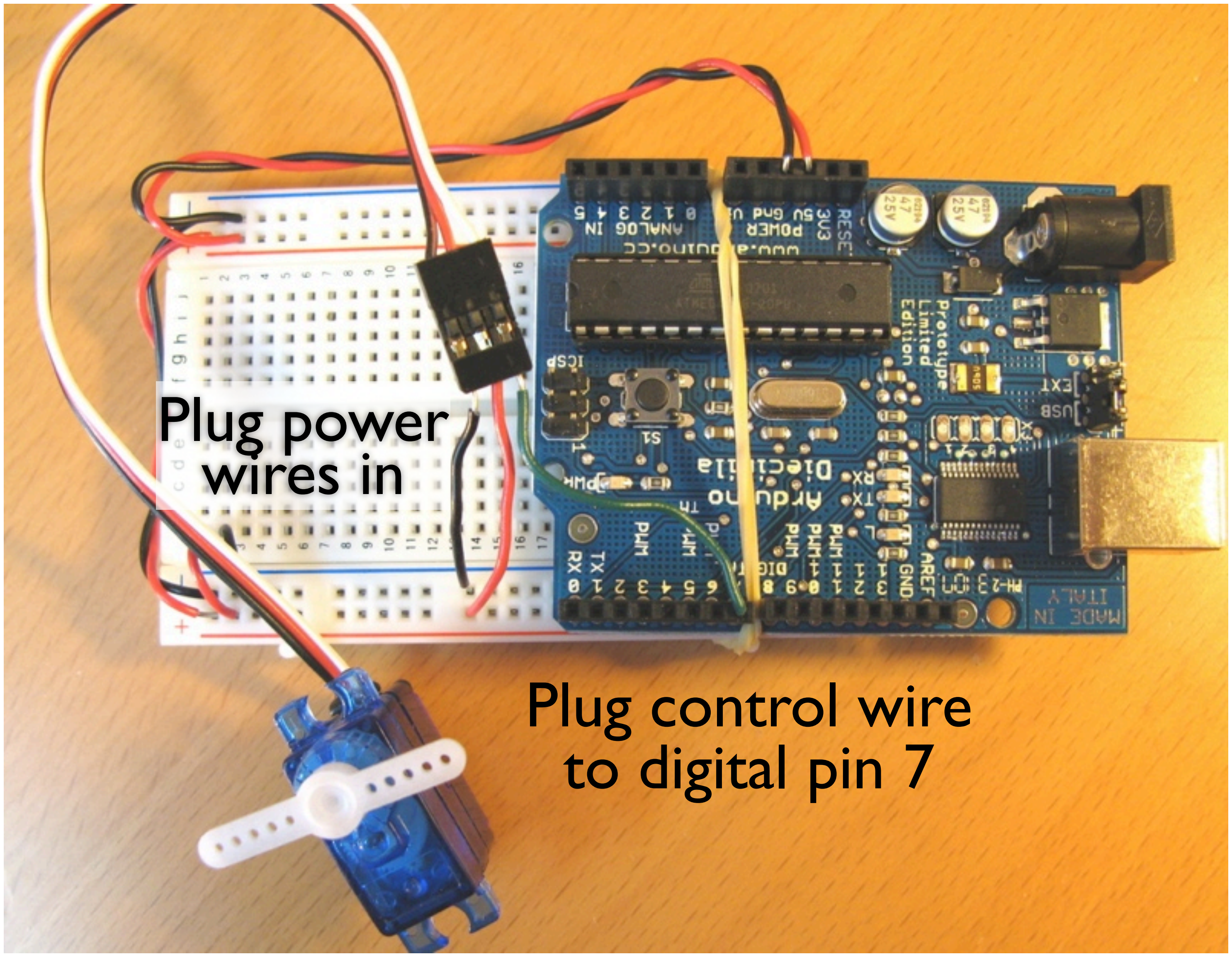
Servo and Arduino

First, add some jumper wires to the servo connector



I recommend matching the color coding of the wires as closely as possible

Servo and Arduino



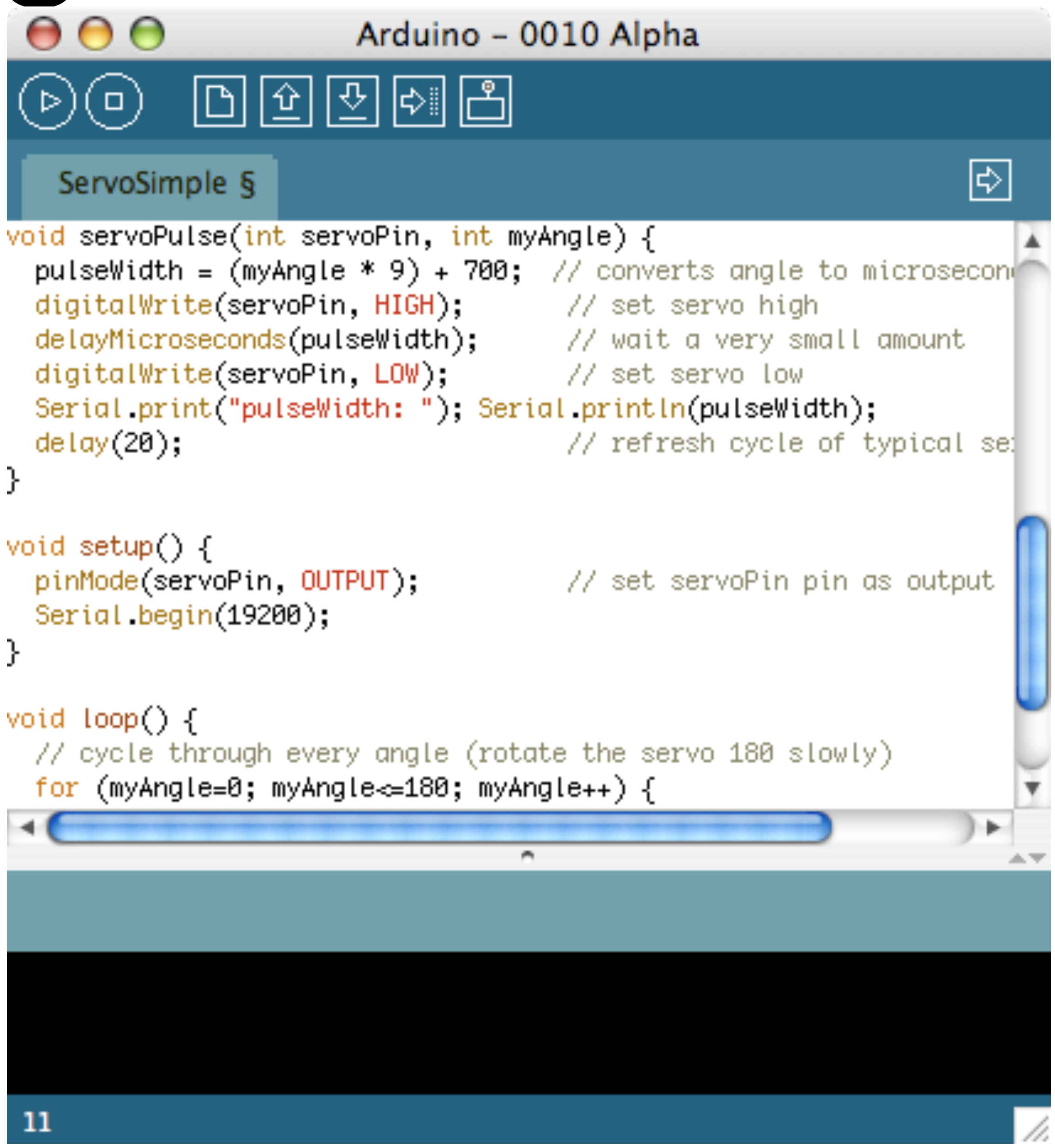
Moving a Servo

“ServoSimple”

Move the servo across its range of motion

Uses `delayMicroseconds()` for pulse width

Uses `delay()` for pulse frequency



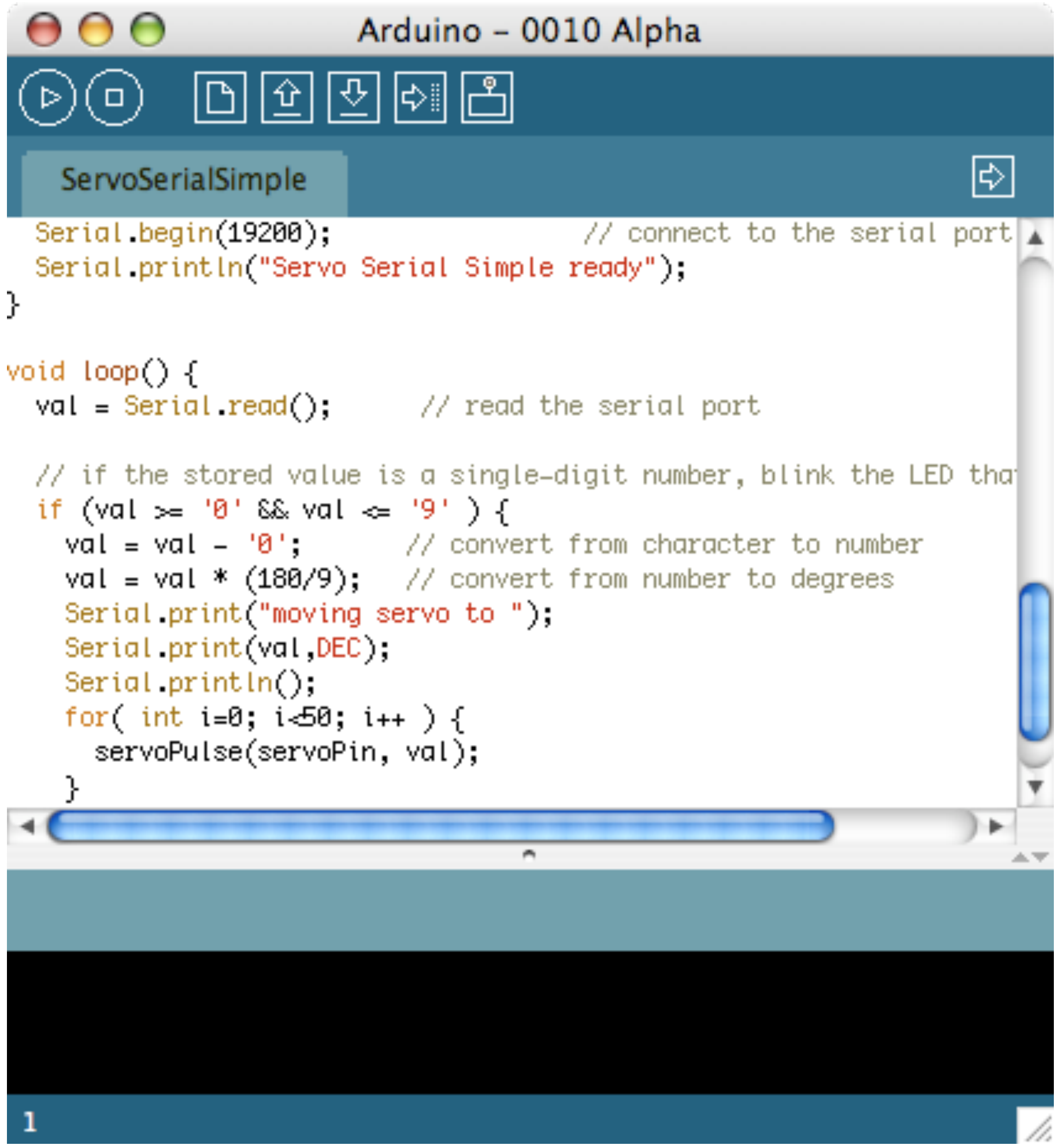
Sketch is in the handout
Created a custom function to handle making servo pulses
New function “delayMicroseconds()”. Like “delay()”, but µsec instead of millisec.
(and actually, just delaying 20 millisec is kinda wrong. should be: 20 – (pulsewidth/1000)
(1000 microseconds = 1 millisecond, and 1000 milliseconds = 1 second)

Serial-controlled Servo

"ServoSerialSimple"

Drive the servo by pressing number keys

Takes the last servo example and adds our standard serial input to it.



Sketch is in the handout.
Why that for loop? Because it takes time for the servo to get to a position and it has no memory.

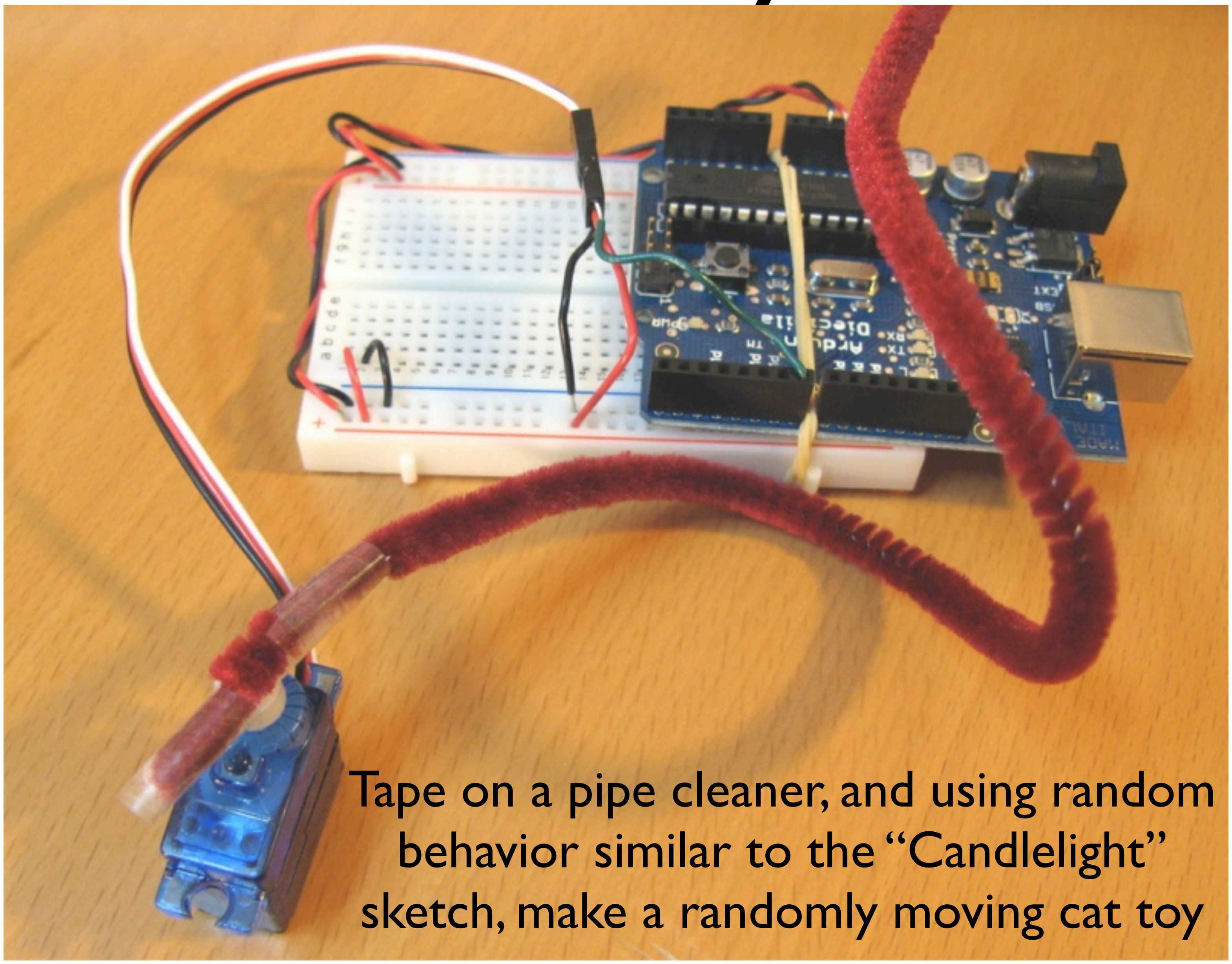
Aside: Controlling Arduino

- Any program on the computer, not just the Arduino software, can control the Arduino board
- On Unixes like Mac OS X & Linux, even the command-line can do it:

```
demo% export PORT=/dev/tty.usbserial-A3000Xv0
demo% stty -f $PORT 9600 raw -parenb -parodd cs8 -hupcl -cstopb clocal
demo% printf "1" > $PORT   # rotate servo left
demo% printf "5" > $PORT   # go to middle
demo% printf "9" > $PORT   # rotate servo right
```

Unix is rad.

Robo Cat Toy Idea



Be sure to securely mount the servo before doing trial runs. Cats are good at taking apart prototype electronics.

Servo Timing Problems

- Two problems with the last sketch
  - When `servoPulse()` function runs, nothing else can happen
  - Servo isn’t given periodic pulses to keep it at position
- You need to run two different “tasks”:
  - one to read the serial port
  - one to drive the servo

If a servo is not being constantly told what to do, it goes slack and doesn’t lift/push/pull

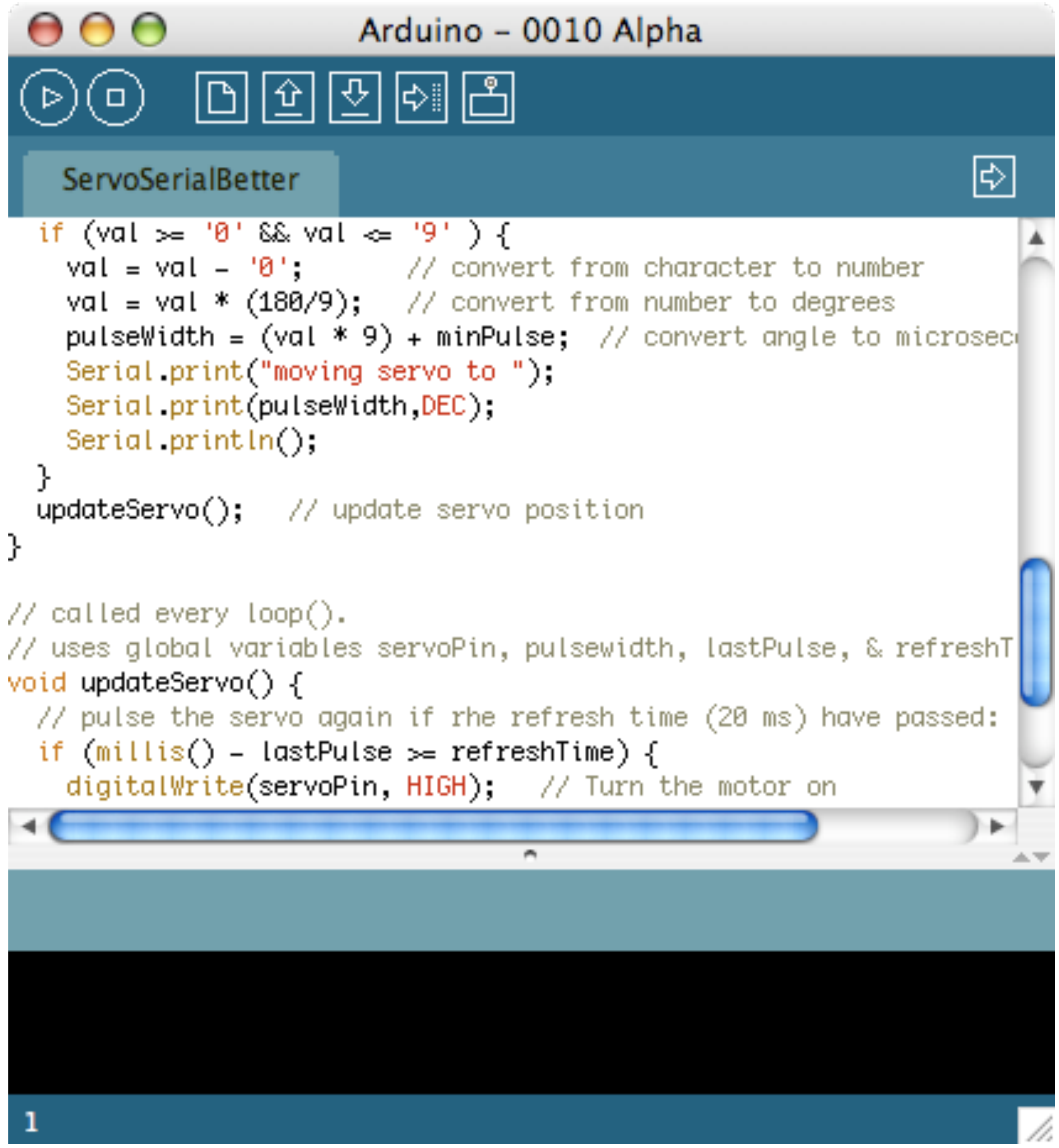
Better Serial Servo

"`ServoSerialBetter`"

Works just like
`ServoSerialSimple`
(but better)

Update the servo when needed, not just when called at the right time

Uses "`millis()`" to know what time it is



Sketch is in the handout.
Trades memory use (the extra variables), for more useful logic.
Can call `updateServo()` as often as you want, servo is only moved when needed.

Multiple Servos

- The `updateServo()` technique can be extended to many servos
- Only limit really is number of digital output pins you have
- It starts getting tricky after about 8 servos though

Multiple “Tasks”

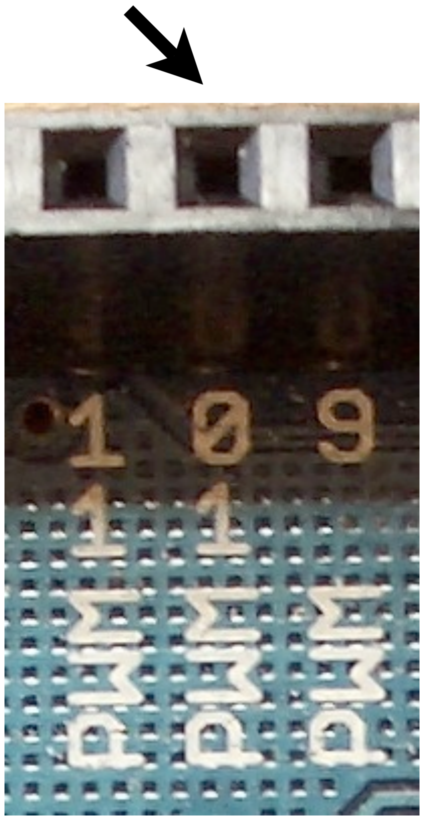
The concept inside `updateServo()` is useful anytime you need to do multiple “things at once” in an Arduino sketch:

- Define your task
- Break it up into multiple time-based chunks (“task slices”)
- Put those task slices in a function
- Use `millis()` to determine when a slice should run
- Call the functions from `loop()`

Inside your task slices, avoid using delay(), for loops, and other code structures that would cause the code to stay inside a task for too long
This is called “cooperative multitasking”, and it’s how OSs in the 80s worked.

Arduino PWM

why all the software, doesn't Arduino have PWM?



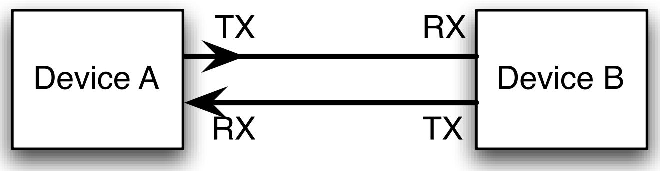
- Arduino has built-in PWM
- On pins 9,10,11
- Use `analogWrite(pin,value)`
- It operates at a high, fixed frequency (thus not usable for servos)
- But great for LEDs and motors
- Uses built-in PWM circuits of the ATmega8 chip -» no software needed

The PWM speed used for analogWrite() is set to 450Hz or 30 kHz currently. I forget which, but it's not something changeable without understanding more about how AVRs work.
So when programming AVRs in C outside of Arduino, PWM speed can be set to just about any value.

Take a Break

Serial Communication

Asynchronous communication

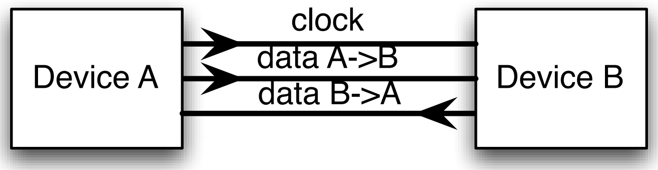


asynchronous – no clock
Data represented by setting
HIGH/LOW at given times

Separate wires for transmit & receive

Each device must have good "rhythm"

Synchronous communication



Synchronous – with clock
Data represented by setting
HIGH/LOW when "clock" changes

A single clock wire & data wire for
each direction like before

Neither needs good rhythm, but one is the conductor

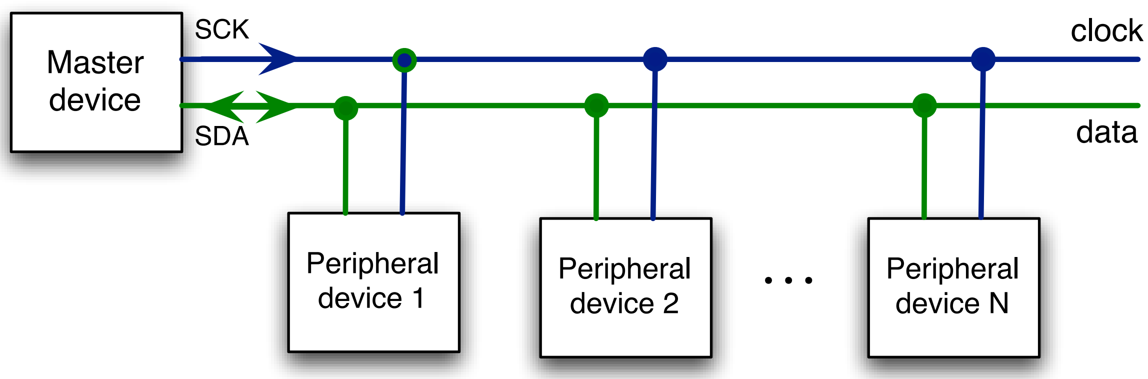
Is one better than the other? It depends on your application. Async is good if there are only two devices and they're both pre-configured to agree on the speed (like your Arduino sketches)

Synchronous is generally better for faster speeds (because you don't need an accurate clock, just the ability to watch the clock wire).

I2C, aka “Two-wire”

Synchronous serial bus with shared a data line

a little network for your gadgets

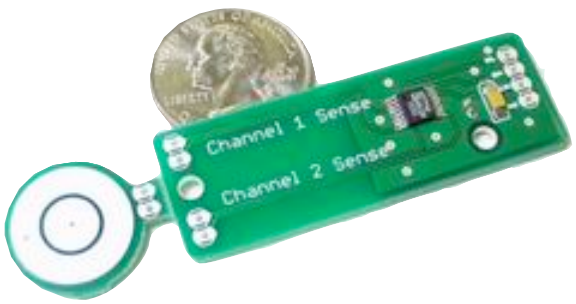


- Up to 127 devices on one bus
- Up to 1Mbps data rate
- Really simple protocol (compared to USB,Ethernet,etc)
- Most microcontrollers have it built-in

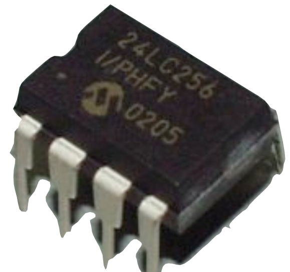
The shared data line means the devices have to agree on when they should “talk” on it. Like how on CBs you say “over” and “over & out” to indicate you’re finished so the other person talk.

See “Introduction to I2C”: http://www.embedded.com/story/OEG20010718S0073
“I2C” stands for “Inter-Integrated Circuit”, but no one calls it that
And if your microcontroller doesn’t have I2C hardware built-in, you can fake it by hand in software (for master devices anyway)

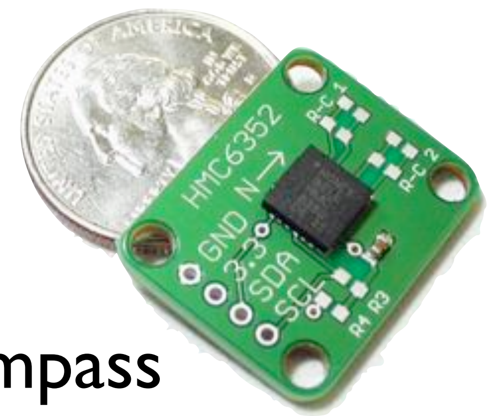
Many I2C devices



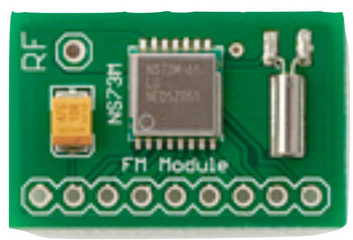
touch sensor



non-volatile memory



compass

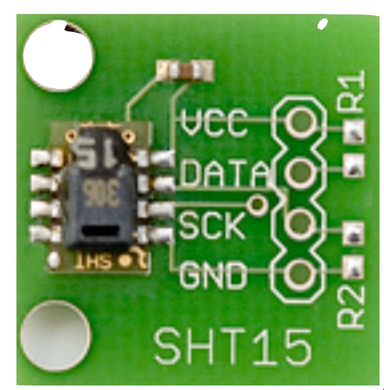


fm transmitter

LCD display

And many others

(gyros,keyboards, motors,...)

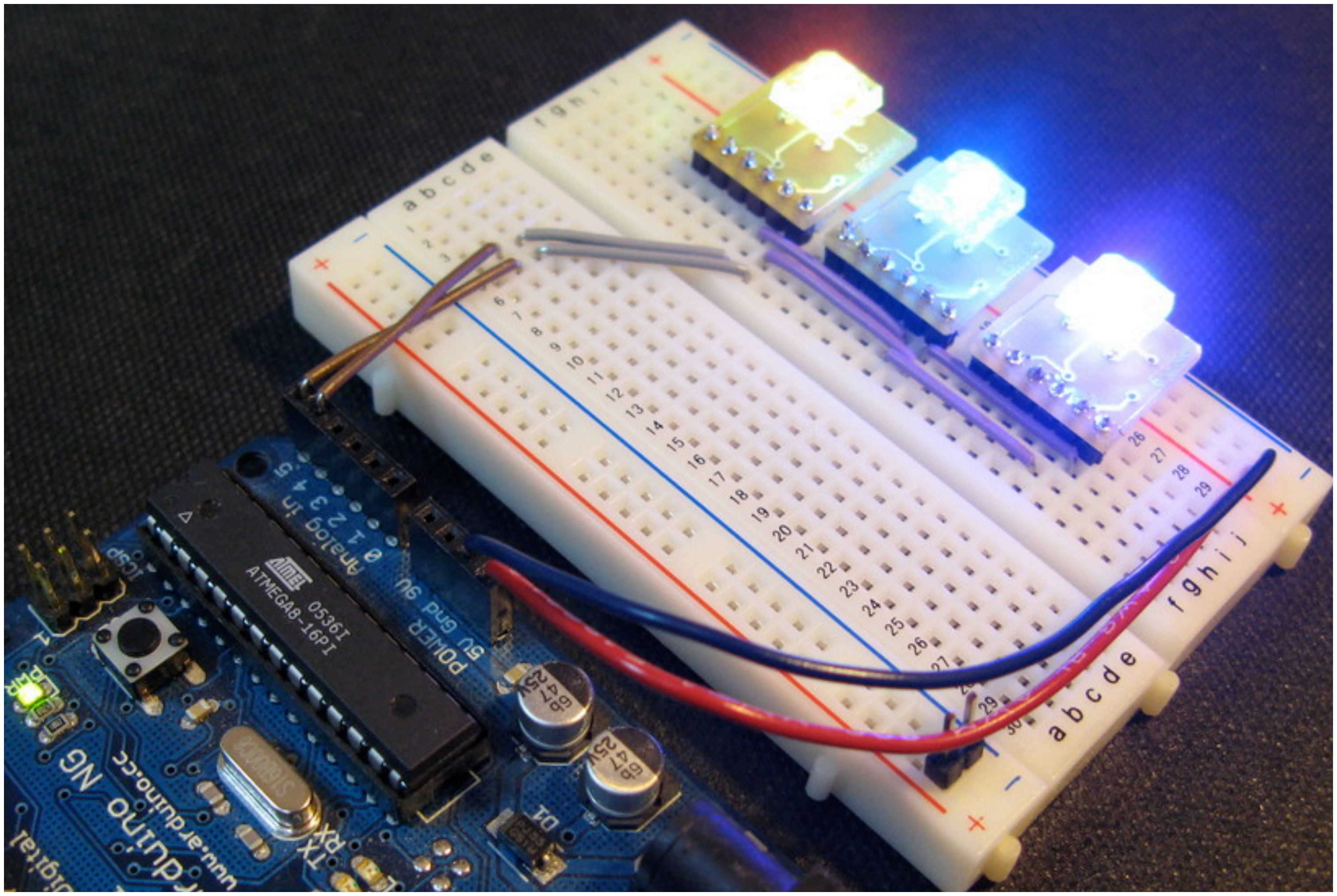


temperature & humidity sensor

Images from Sparkfun.com,except LCD from matrixorbital.com

Obligatory BlinkM Promo

I2C Smart LED



Does all the hard PWM & waveform generation for you

You should be able to buy these from Sparkfun.com in a month or so.

Nintendo Wii Nunchuck

- Standard I2C interface
- 3-axis accelerometer with 10-bit accuracy
- 2-axis analog joystick with 8-bit A/D converter
- 2 buttons
- $20

If you look at the architecture for the Nintendo Wii and its peripherals, you see an almost un-Nintendo adherence to standards. The Wii controllers are the most obvioius examples of this. The Wii controller bus is standard I2C. The Wii remote speaks Bluetooth HID to the Wii (or your Mac or PC)

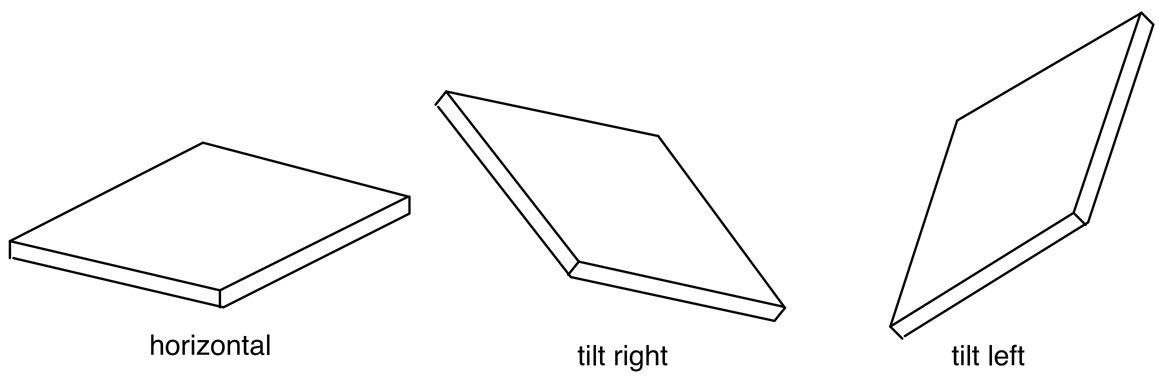
Because it uses standard I2C, it's easy to make the Nunchuck work with Arduino, Basic Stamp or most other microcontrollers.

See: http://www.wiili.org/index.php/Wiimote/Extension_Controllers/Nunchuk
and: http://www.windmeadow.com/node/42
and: http://todbot.com/blog/2007/10/25/boarduino-wii-nunchuck-servo/

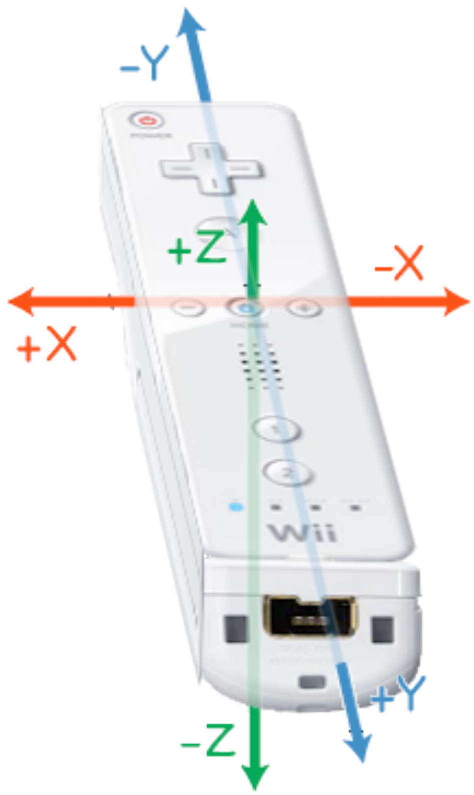
And then there's the Wii Remote, besides Bluetooth HID, it also has accelerometers, buttons, speaker, memory, and is I2C master.

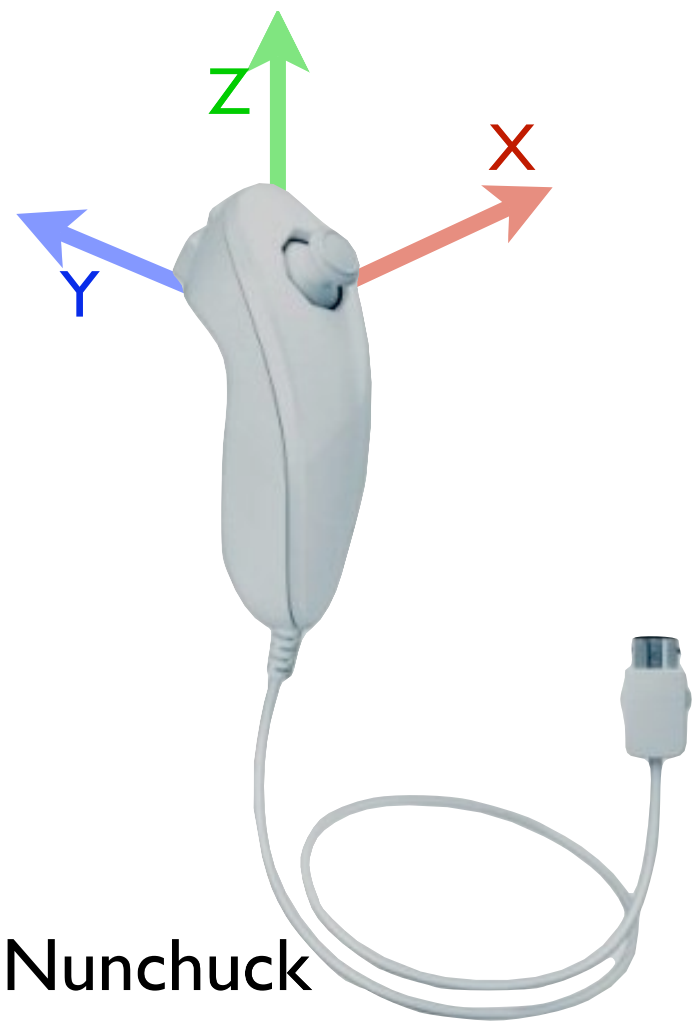
Accelerometer?

- Measures acceleration (changes in speed)
- Like when the car pushes you into the seat
- Gravity is acceleration
- So, also measures tilt



Nunchuck Accelerometer





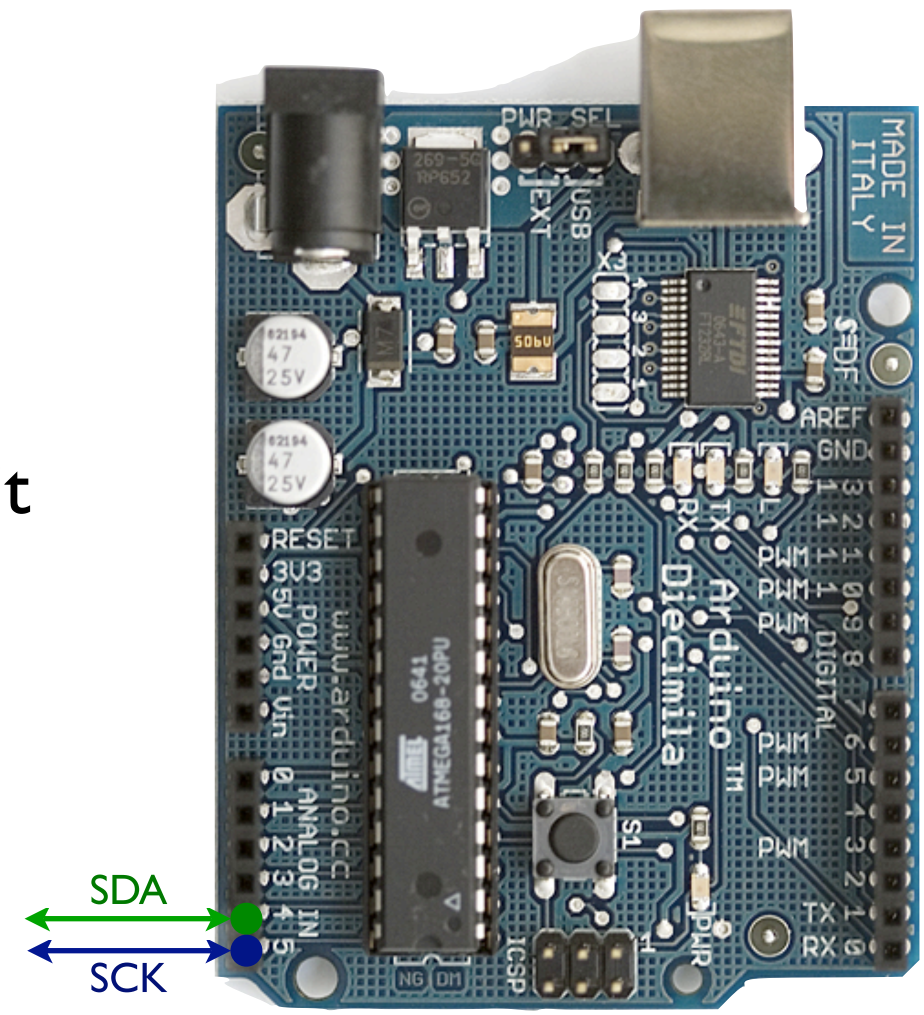
Wii Remote & Nunchuck accelerometer axes

I’m not sure if I have the Nunchuck one right.

Wiimote axis image from http://www.wiili.org/index.php/Wiimote

I2C on Arduino

- I2C built-in on Arduino's ATmega168 chip
- Use "Wire" library to access it
- Analog In 4 is SDA signal
- Analog In 5 is SCK signal



Arduino “Wire” library

Writing Data

Load Wire library

Join I2C bus
(as master)

Start sending

Send data

Stop sending

```
#include <Wire.h>

void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop() {
  Wire.beginTransmission(4); // transmit to device #4
  Wire.send("x is ");        // sends five bytes
  Wire.send(x);              // sends one byte
  Wire.endTransmission();    // stop transmitting

  x++;
  delay(500);
}
```

And what the various commands do are documented in the instructions / datasheet for a particular device.

Arduino “Wire” library

Reading Data

Join I2C bus
(as master)

Request data from device

Get data

```
#include <Wire.h>

void setup() {
  Wire.begin();         // join i2c bus (address optional for master)
  Serial.begin(9600);   // start serial for output
}

void loop() {
  Wire.requestFrom(2, 6);    // request 6 bytes from slave device #2

  while(Wire.available()) {   // slave may send less than requested
    char c = Wire.receive(); // receive a byte as character
    Serial.print(c);         // print the character
  }

  delay(500);
}
```

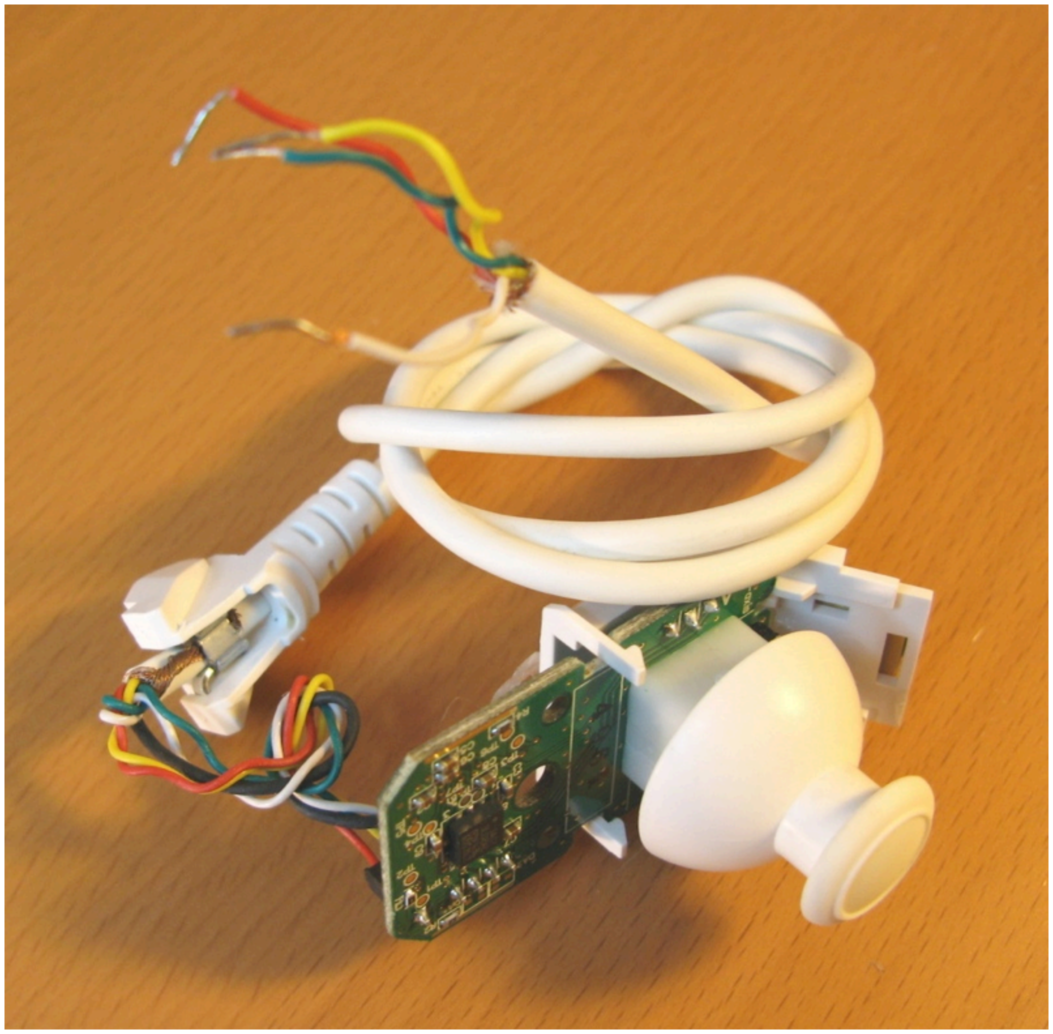
What kinds of interactions you can have depends on the device you’re talking to

Most devices have several “commands”

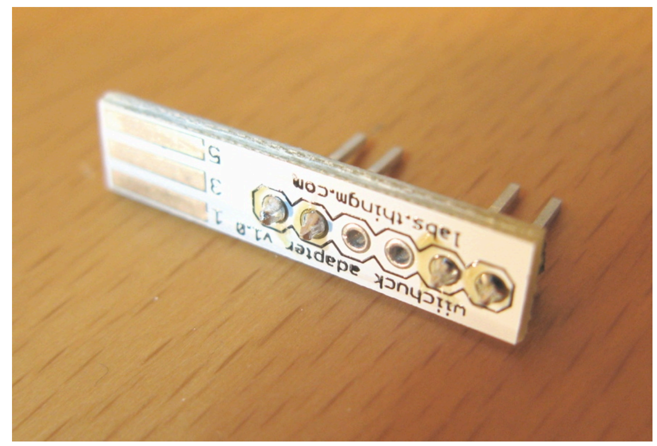
And what the various commands do are documented in the instructions / datasheet for a particular device.

Wiring up the Nunchuck

We could hack off the connector
and use the wires directly

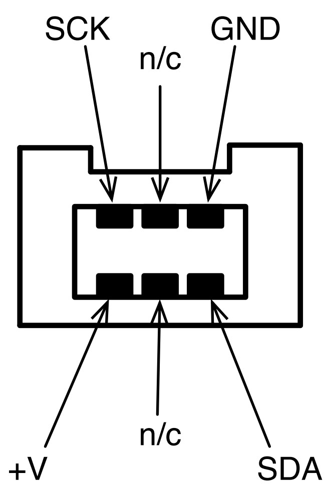


But instead let’s use this
little adapter board



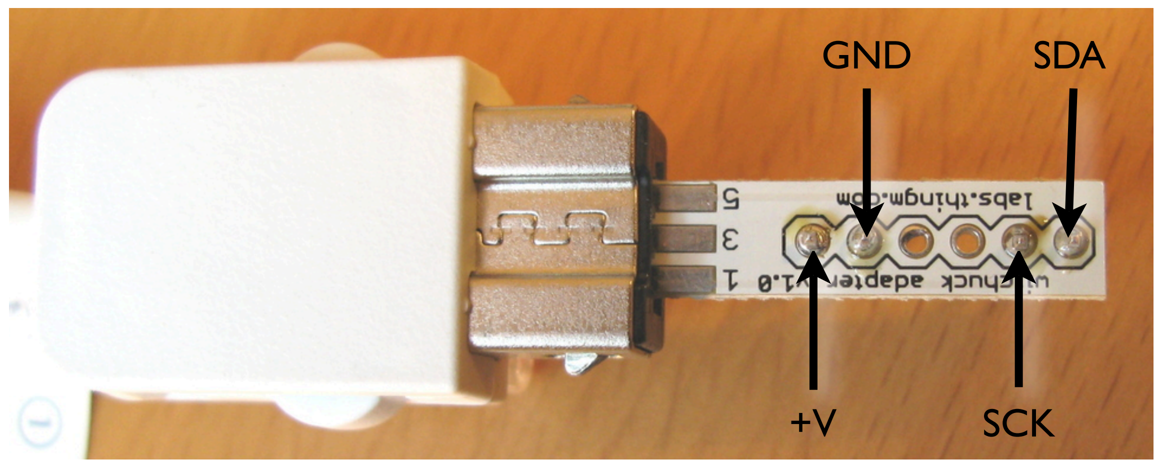
Wii Nunchuck Adapter

Nunchuck Pinout



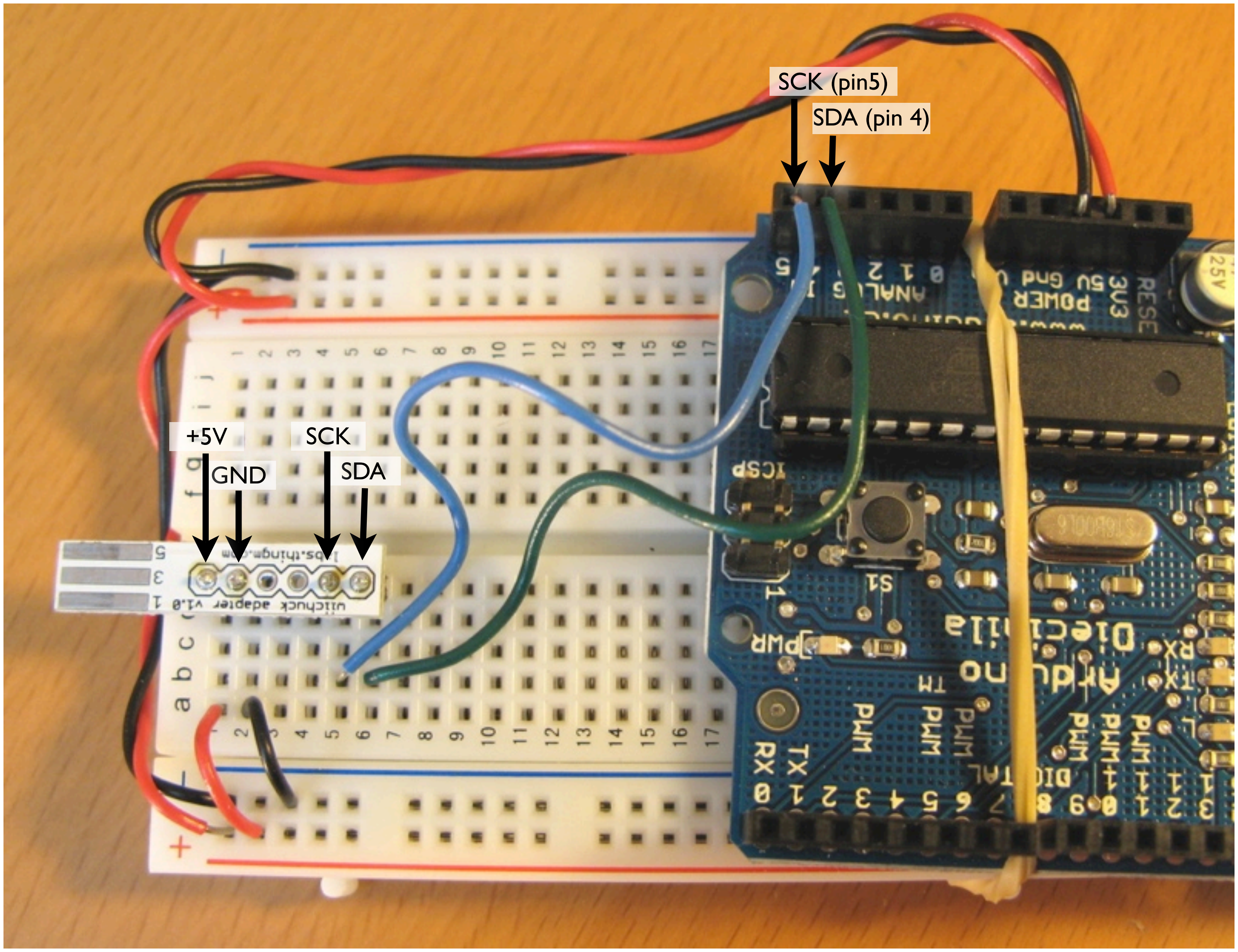
(looking into Nunchuck connector)

Adapter Pinout

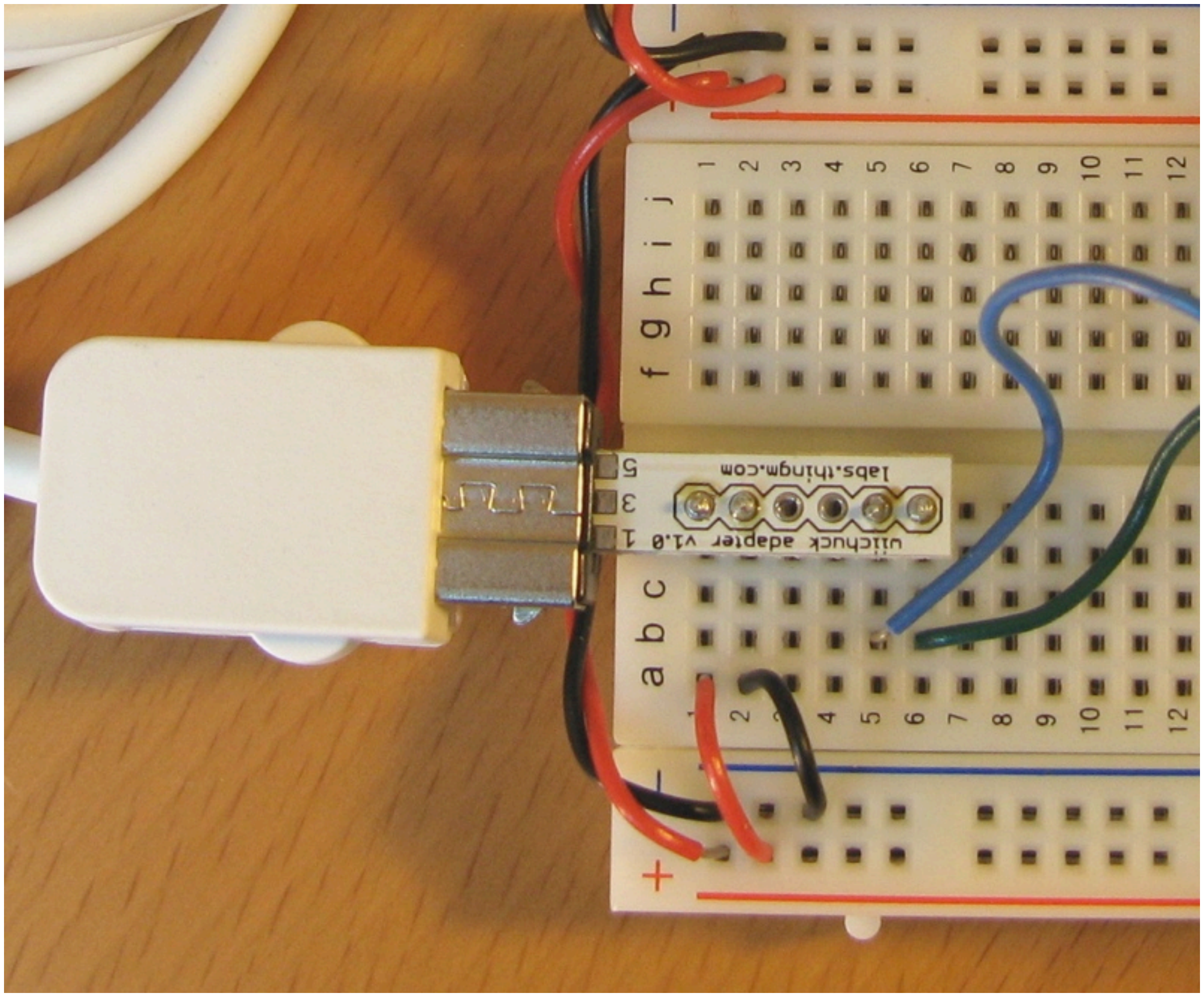


Note there *are* labels on the adapter, but they're wrong. So you'll have to trust the diagrams above

Wiring it Up



Pluggin' in the 'chuck

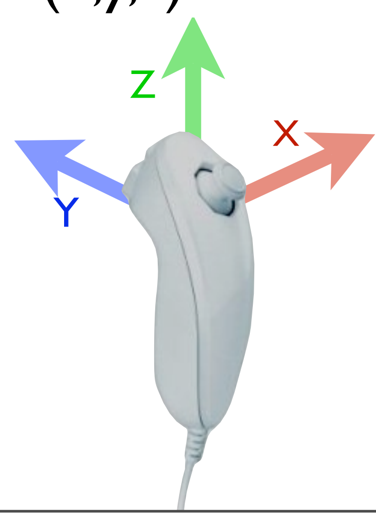


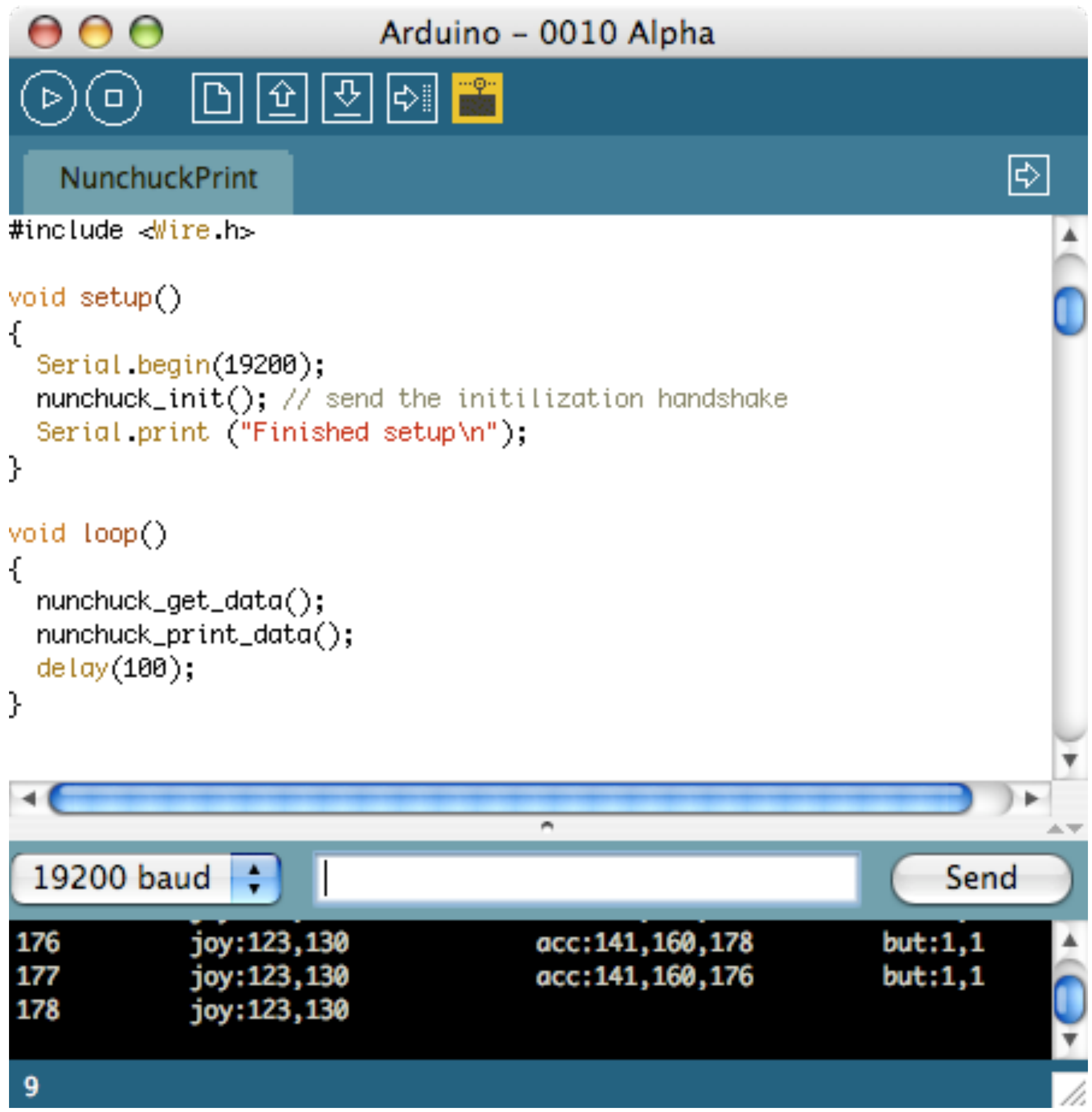
Trying the Nunchuck

“NunchuckPrint”

Read the Nunchuck every 1/10th of a second & print out all the data:

- joystick position (x,y)
- accelerometer (x,y,z)
- buttons Z,C





Uses the beginnings of an Arduino library I’m writing.

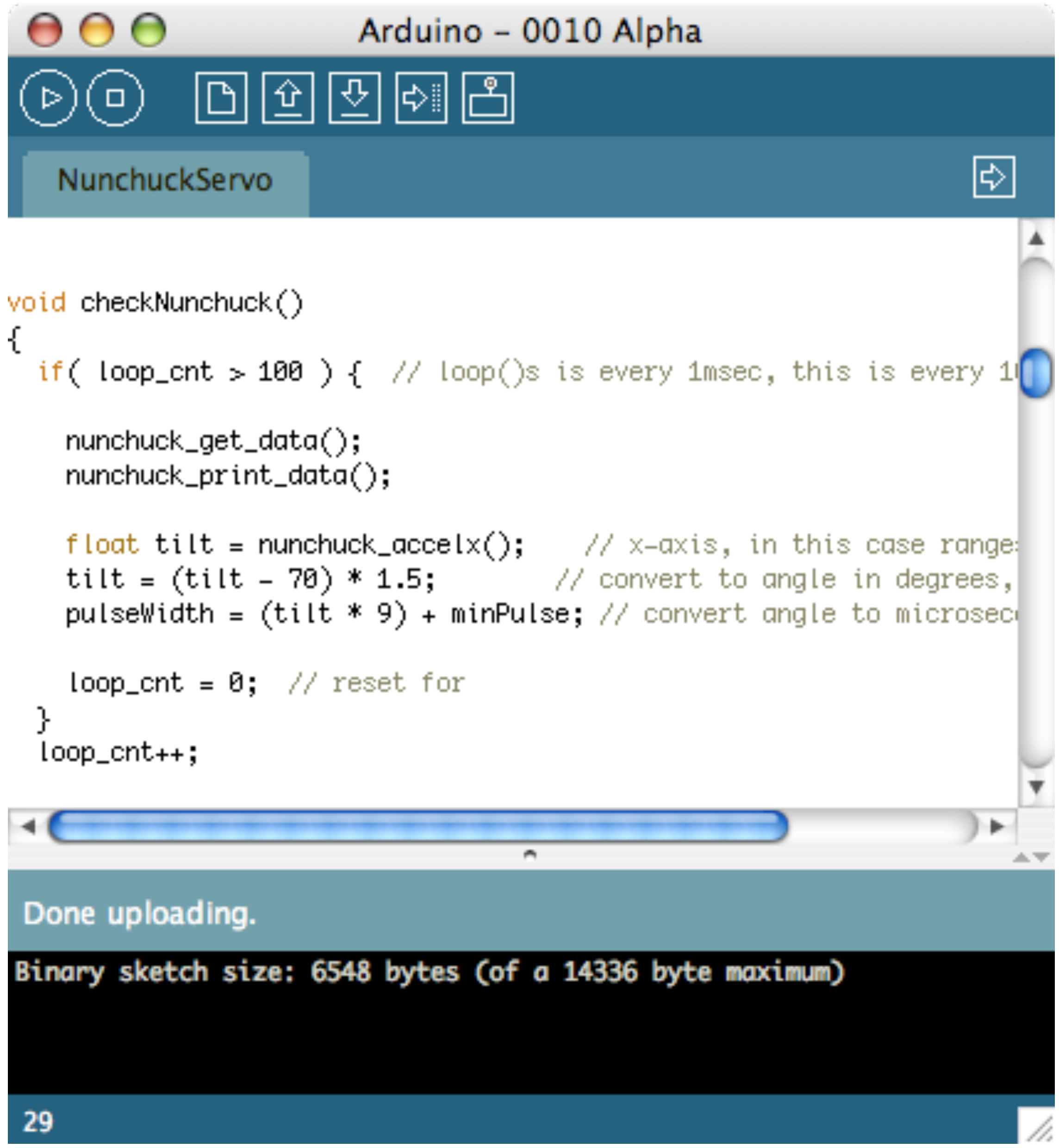
Adding a Servo

“NunchuckServo”

Move the servo by moving your arm

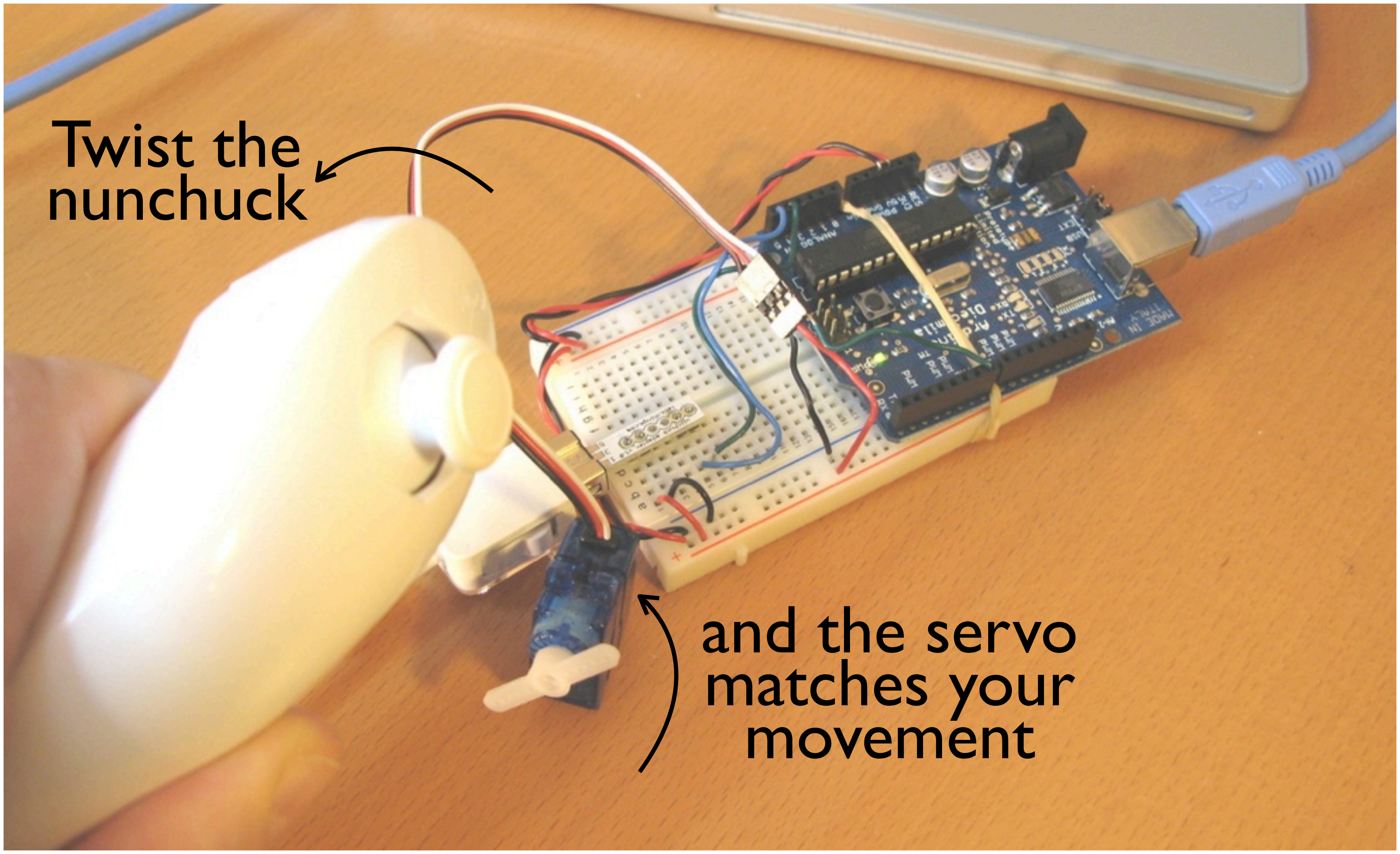
You’re a cyborg!

Also press the Z button to flash the pin 13 LED

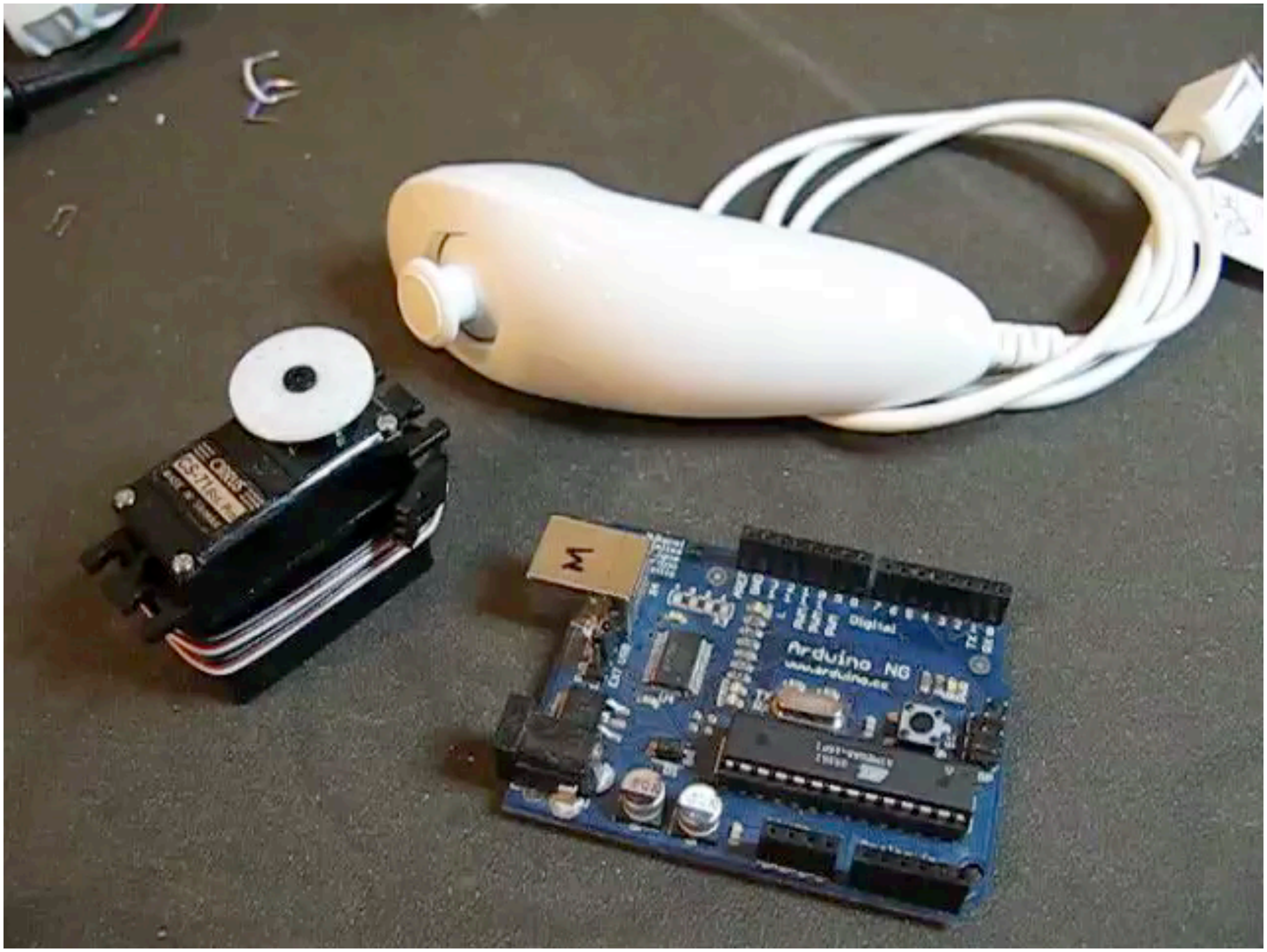


Utilizes the task slicing mentioned before

Nunchuck Servo



Segway Emulator



Same basic code as NunchuckServo.
For details see: http://todbot.com/blog/2007/10/25/boarduino-wii-nunchuck-servo/

Going Further

- Servos
  - Hook several together to create a multi-axis robot arm
  - Make a “servo recorder” to records your arm movements to servo positions and plays them back
  - Great for holiday animatronics

Going Further

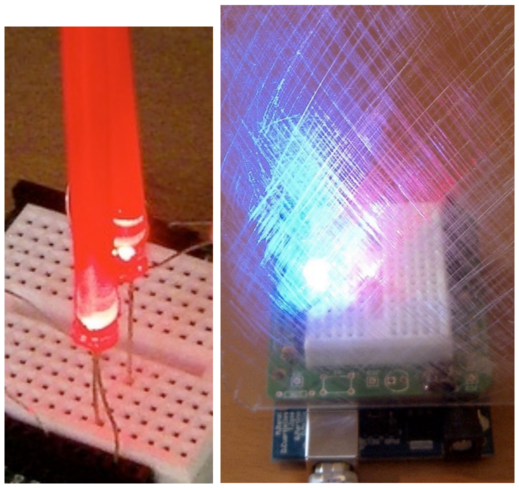
- I2C devices
  - Try out some other devices
  - Just string them on the same two wires used for the Nunchuck
- Cooperative Multitasking
  - Try making a theremin with nunchuck & piezo
  - See if previous examples can be made more responsive

Going Further

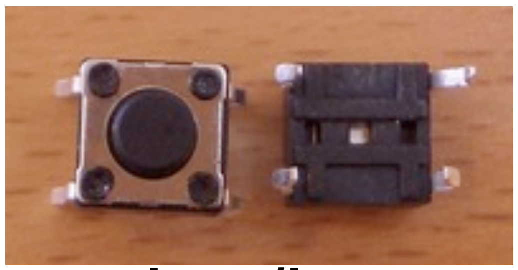
- Nunchuck
  - It’s a freespace motion sensor. Control anything like you’re waving a magic wand!
  - What about the joystick? We didn’t even get a chance to play with that
  - Alternative input device to your computer: control Processing, etc.

Summary

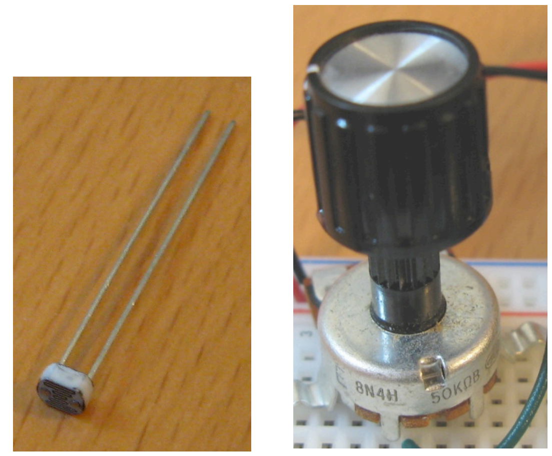
You've learned many different physical building blocks



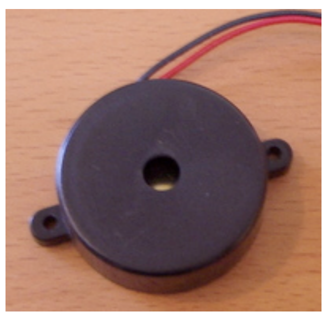
LEDs



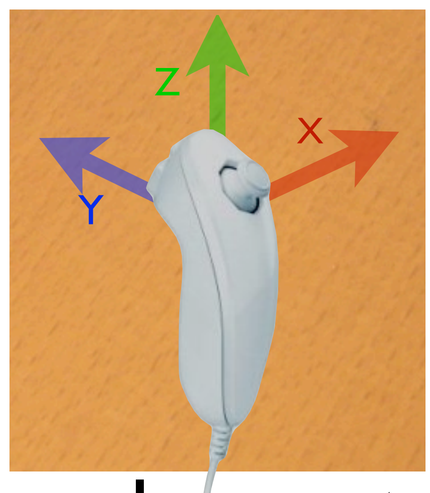
switches/buttons



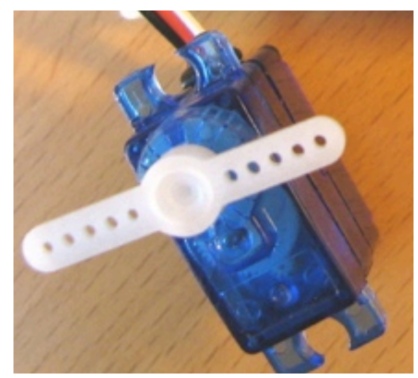
resistive sensors



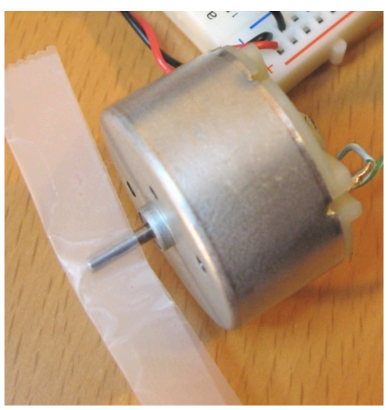
piezos



accelerometers



servos



motors

Summary

And you've learned many software building blocks

serial communication

pulse width modulation

I2C

analog I/O

digital I/O

data driven code

frequency modulation

multiple tasks

Summary

Hope you had fun and continue playing with Arduino

Feel free to contact me to chat about this stuff

END Class 4

http://todbot.com/blog/bionicarduino/

Tod E. Kurt

tod@todbot.com

Feel free to email me if you have any questions.