Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators

Cynthia Ferrell[1]
ferrell@ai.mit.edu

Massachusetts Institute of Technolgy

A revised version of the thesis submitted in partial fulfillment of the requirements of the degree of Master of Science in Electrical Engineering and Computer Science

[1]This report describes research done at the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by a NASA Graduate Student Researcher Program Fellowship administered through the Jet Propulsion Laboratory, by Jet Propulsion Laboratory grant 959333, and in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014–91–J–4038.

Abstract

This thesis presents methods for implementing robust hexapod locomotion and fault tolerance capabilities on an autonomous robot with many sensors and actuators. The controller is based on the Subsumption Architecture and is fully distributed over approximately 1500 simple, concurrent processes. The robot, Hannibal, weighs approximately 6 pounds and is equipped with over 100 physical sensors, 19 degres of freedom, and 8 on board computers.

We investigate the following topics in depth: control of a complex robot, insect-like locomotion control for gait generation and rough terrain mobility, and tolerance of sensor and actuator failures. The complexity of the robot and the controller is managed using a *local control with cooperation* paradigm. In this approach, the control problem is distributed evenly among the legs. Because the legs are physically coupled through the robot and through the terrain, the legs communicate with each other to work as a team. Robust, flexible locomotion is implemented using ideas from insect locomotion and strategies used by insects to traverse natural terrain. As a result, Hannibal's locomotion exhibits many insect-like properties. Fault tolerance is implemented via a network of processes responsible for detecting failures and minimizing the impact of failures on the robot's performance. By exploiting concurrency and distributedness, the system monitors, detects, and compensates for component failures simultaneously.

The controller was implemented, debugged, and tested on Hannibal. Through a series of experiments, we examined Hannibal's gait generation, rough terrain locomotion, and fault tolerance performance. These results demonstrate that Hannibal exhibits robust, flexible, real-time locomotion over a variety of terrain and tolerates a multitude of hardware failures.

Thesis supervisor: Prof. Rodney A. Brooks

Contents

1 Introduction **9**

1.1 The Challenge . . . 9

1.2 The Response . . . 11

1.3 Organization of Thesis . . . 13

2 Hannibal **14**

2.1 The Physical Robot . . . 14

2.1.1 Legs . . . 14

2.1.2 Body . . . 15

2.1.3 Size . . . 16

2.2 Sensors . . . 16

2.3 Computing . . . 19

2.3.1 Master processor . . . 20

2.3.2 Satellite control processors . . . 20

2.3.3 Serial I^2C bus . . . 21

3 Controller Overview **22**

3.1 Issues of Controller Design . . . 22

3.2 Robot Behavior Control Philosophy . . . 23

3.3 Overview of the Controller . . . 24

3.3.1 Sensor-Actuator Level . . . 25

3.3.2 Basic Locomotion Level . . . 27

3.3.3 Rough Terrain Level . . . 28

3.4 Managing Complexity . . . 28

3.4.1 Layering . . . 28

3.4.2 Distributedness . . . 29

3.5 Summary . . . 30

4 Basic Locomotion **32**
4.1 The Hexapod Locomotion Challenge 32
4.2 Definition of terms 33
4.3 Insect locomotion 35
4.3.1 Wilson 35
4.3.2 Pearson 37
4.3.3 Cruse 40
4.4 Cruse Control 43
4.5 Modified Cruse Control 47
4.6 Pacemaker Control 51
4.6.1 Gait Behavior 56
4.6.2 Turning Behavior 59
4.6.3 Direction of Travel Behavior 62
4.6.4 Lesion Compensation Behavior 64
4.7 Performance 65
4.7.1 Definition of Terms and Stability Formulas 65
4.7.2 Stability Performance 67
4.7.3 Speed Performance 68
4.7.4 Turning Performance 69
4.7.5 Directional Performance 70
4.7.6 Lesion Compensation Performance 72
4.8 Legged Locomotion 73
4.8.1 Phoney Pony 73
4.8.2 OSU Hexapod 75
4.8.3 SSA Hexapod 76
4.8.4 Genghis 78
4.8.5 Case Western Hexapod 81
4.9 Contributions 83

5 Rough Terrain Locomotion **84**
5.1 The Rough Terrain Challenge 84
5.2 Insect Locomotion Over Rough Terrain 86
5.3 Inter-leg Communication 88
5.3.1 Virtual Sensor Activated Behaviors 90
5.4 Rough Terrain Network 92
5.4.1 Loading considerations 92
5.4.2 State activated behaviors 97

5.4.3 Hormone activated behaviors 99
5.5 Performance 100
5.5.1 Tests 100
5.5.2 Results 102
5.5.3 Evaluation 103
5.6 Comparison with insect rough terrain locomotion 104
5.7 Comparison with legged robot locomotion over rough terrain . 104
5.7.1 OSU Hexapod 105
5.7.2 Preambulating Vehicle II 106
5.7.3 Adaptive Suspension Vehicle 107
5.7.4 Ambler 108
5.7.5 Ghengis 110
5.8 Contributions 112

6 Fault Tolerance 113
6.1 The Fault Tolerance Challenge 113
6.2 Confinement of Errors 115
6.3 Levels of Fault Tolerance 116
6.3.1 Replication of Hardware 116
6.3.2 Redundant Control Behaviors 117
6.3.3 Robust Virtual Sensors 119
6.4 Adaptivity vs Redundancy 120
6.5 Fault Tolerance Network 122
6.5.1 Detection 123
6.5.2 Masking 132
6.5.3 Recovery 134
6.5.4 Reintegration 138
6.5.5 Catastrophic Failures 141
6.6 Performance 145
6.6.1 Tests 145
6.6.2 Results 146
6.6.3 Evaluation 148
6.7 Comparison with other systems 150
6.7.1 Robustness 151
6.7.2 Replication of Hardware 151
6.7.3 Redundant Control Behaviors 152
6.8 Contributions 153

7 Conclusion 154
7.1 Review of Significant Results 154
7.1.1 Control of Complex Robot 154
7.1.2 Fault Tolerance 155
7.1.3 Robust Hexapod Locomotion 157
7.1.4 Micro-Rover Missions 157
7.2 Real Robots for Real Problems 158

List of Figures

1.1 Hannibal 10

2.1 Leg degrees of freedom 15
2.2 Body degree of freedom 16
2.3 Leg sensors 18
2.4 Connection of smart subsystems 19

3.1 Levels of competence of Hannibal's controller. 24
3.2 Global control organization 29

4.1 Locomotion definitions 34
4.2 Insect gaits 36
4.3 Pearson's model for individual leg control 38
4.4 Pearson's model for gait coordination 39
4.5 Cruse's model for individual leg control 40
4.6 Cruse's model for gait coordination 41
4.7 Cruse's mechanisms for gait coordination 42
4.8 Individual leg control circuit for Cruse Control 44
4.9 Gait coordination circuit for Cruse Control 45
4.10 Gait phase shift from Cruse Control 46
4.11 Gait coordination circuit for Modified Cruse Control 47
4.12 Gait coordination mechanisms for Modified Cruse Control . . . 48
4.13 Modified Cruse Control gaits 50
4.14 Lesion switchboard 51
4.15 Lesion results of Modified Cruse Control 52
4.16 Individual leg control circuit for Pacemaker Control 53
4.17 Gait coordination circuit for Pacemaker Control 53
4.18 Oscillator implementation 54

4.19 Pacemaker Control mechanism 55
4.20 Pacemaker Control gaits 57
4.21 Gait transition circuit 58
4.22 Gait transition results 60
4.23 Turning circuit 61
4.24 Direction of travel circuit 63
4.25 Lesion compensation circuit 64
4.26 Stability definitions 66
4.27 Hannibal's stability margins 68
4.28 Speed performance 69
4.29 Turn performance 70
4.30 Directional speed performance 71
4.31 Lesion speed performance 72
4.32 Phoney Pony control circuit 74
4.33 SSA control circuit 77
4.34 Genghis basic locomotion control circuit 79
4.35 Case Western hexapod control circuit 81

5.1 Obstacle size definitions 85
5.2 Terrain depression definitions 87
5.3 Priority scheme for inter-leg conflicts 89
5.4 Circuit for walking with leg loading 91
5.5 Find-foothold circuit 93
5.6 Circuit for obstacle traversal 94
5.7 posture definitions 94
5.8 Medium sized obstacle traversal 95
5.9 Slope definitions 96
5.10 Inclination circuit 96
5.11 Large obstacle avoidance circuit 97
5.12 Time based terrain adaptation circuit 99
5.13 Rough terrain performance 102
5.14 Genghis rough terrain circuit 111

6.1 System hierarchy 115
6.2 Replication of hardware 116
6.3 Redundant control behaviors 118
6.4 Robust virtual sensors 120

6.5 Sensor states versus leg phases 125
6.6 Sensor monitor processes 127
6.7 Response of sensor monitor processes 128
6.8 Monitor consensus processes 129
6.9 Response of monitor consensus processes 130
6.10 Injury agent 131
6.11 Response of injury agent 131
6.12 Robust ground-contact virtual sensor 132
6.13 Virtual sensor masks local failure 133
6.14 Recovery to transient error 134
6.15 Dynamic re-calibration agent 135
6.16 Response of dynamic re-calibration agent 136
6.17 Recovery to permanent local failure 137
6.18 Reintegration processes 139
6.19 Response to repaired component 140
6.20 Detection of global failure 141
6.21 Masking of global failure 142
6.22 Recovery agents for global failures 143
6.23 Recovery response to global failure 144
6.24 Reintegration response to repair of global component 145
6.25 Fault Tolerance performance 146
6.26 Fault tolerance response times 147

Acknowledgements

The MIT AI Lab provided a very inspirational, helpful, and supportive environment throughout every stage of this thesis from conception to implementation to documentation. Special thanks to my thesis advisor, Rod Brooks, for granting me "quality time" when I needed it, providing me with helpful suggestions and ideas, and for being the Supreme Non-Squasher. The quality of this thesis benefited from helpful comments and suggestions provided by those who read earlier versions of this work: Rod Brooks, Anita Flynn, Gill Pratt, and Mike Binnard. I am also very grateful to Anita Flynn for being an outstanding role model, and for making the Mobot Lab a place where wild ideas become reality. Thanks to Mike Binnard for going on food runs for me when I refused to leave my computer. Thanks to the whole mobot group for their support and enthusiasm.

I would like to sincerely thank my entire family for their continuing support. Finally, endless gratitude to my husband, Robert. His unconditional love, support, and respect never failed to make me realize that life was actually quite good no matter how trying times became.

Chapter 1

Introduction

1.1 The Challenge

Our lab has argued the case in favor for using multiple autonomous micro-rovers on the order of 1 to 2 Kg to explore planets on the grounds of mission cost effectiveness, robustness, and flexibility ((Angle & Brooks 1990), (Brooks & Flynn 1989)). Autonomous robots execute their task independent of human assistance by performing their own sensing and control. As a result, autonomous rovers do not require sizable (and therefore expensive) ground crew support. Also, the principal cost of a planetary mission is payload mass. Since smaller rovers have less mass, it is feasible to send more of them cost effectively. Multiple rovers enhance mission robustness since if a few rovers fail, there are still others to perform the mission. Furthermore, missions can be more flexible since groups of rovers can perform different tasks at various locations.

Planetary exploration using autonomous robots is an interesting control problem. The surface of Mars or the Moon is quite rugged. Undoubtedly, as rovers explore the planet, they will encounter dangerous situations. Rovers must operate in real-time so they can quickly respond to these hazardous circumstances. Also, as rovers wander over the terrain, they will encounter cliffs, rocks, slopes, and crevices. This requires rovers to sense rugged terrain features and adapt their behavior appropriately. Rovers must also be tolerant of hardware failures since they cannot not be repaired once components fail.

Hannibal (shown in figure 1.1) was designed and built by our lab as an

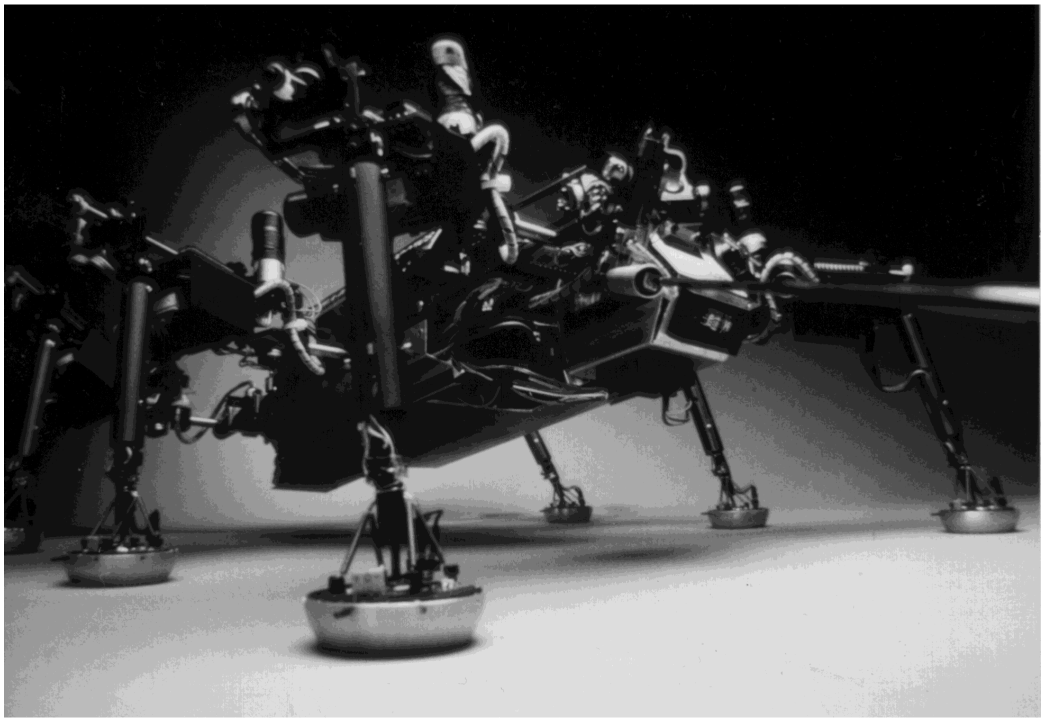


Figure 1.1: Hannibal.

experimental platform to explore planetary micro-rover control issues (Angle 1991). When designing the robot, careful consideration was given to mobility, sensing, and robustness issues. Much has been said concerning the advantages of legged vehicles over wheeled vehicles in regard to their mobility over rough terrain ((Song & Waldron 1989), (Hirose 1984)). Since rough terrain locomotion is fundamental for a micro-rover, Hannibal was engineered with six 3 degree of freedom legs and a 1 degree of freedom body. Rovers must have sufficient terrain sensing capabilities to locomote safely and effectively over rugged terrain. To meet this requirement, Hannibal's legs are encrusted with a multitude of sensors. For robustness and reliability considerations, many of these sensors provide complementary information. Not surprisingly, Hannibal is quite complex for its size. It is approximately the size of a bread box and is equipped with 19 degrees of freedom, over 100 physical sensors, and 8 computers.

1.2 The Response

This work presents the behavior control system implemented on Hannibal. No simulations were used in its development–all code was directly implemented and tested on Hannibal. The controller enables Hannibal to locomote over rough terrain in real-time and tolerate hardware failures. These capabilities are implemented from the bottom-up using a subsumption-based approach where distributed networks of simple and concurrently acting behaviors form the robot's levels of competence. The lowest level of the control system consists of hardware related processes such as fault tolerance, terrain sensing, and driving the actuators. The middle layer consists of flat terrain locomotion behaviors that address mobility, stability, speed, and leg failure issues. The top layer is comprised of rough terrain behaviors that enable Hannibal to negotiate obstacles of varying sizes, terrain depressions of various depths and widths, and slopes of varying steepness. In the course of designing these layers, we explored the following topics in detail:

- *Real-time control of a complex robot.* Hannibal is an intricate robot which operates in a complex environment. The controller must process information from over 100 physical sensors to operate 19 degrees of freedom concurrently. The controller becomes complicated as more processes are written to improve and expand Hannibal's capabilities. The issues of scalability and modularity in controller design are amplified. Despite this formidable task, we want the robot to operate in real-time using relatively minimal computing power. The complexity of this project forces us to explore these issues beyond other autonomous robot controllers in the field ((Brooks 1989), (Donner 1987)).

- *Robust hexapod locomotion.* To effectively traverse rough terrain, Hannibal's locomotion control must be flexible, robust, and adaptive. Rugged terrain exhibits numerous kinds of terrain features. The controller must be able to detect these terrain features and evoke the robot to perform the appropriate evasive maneuvers. Insects display impressive hexapod locomotion capabilities using a distributed controller, so we have looked to insect locomotion research for inspiration ((Cruse 1990*b*), (Dean 1991*a*), (Pearson 1976), (Wilson 1966)).

- *Tolerance to hardware failures.* Having many sensors and actuators is a double edged sword. More components increase the hardware capabilities of the robot; however there is also more that can fail and subsequently degrade performance. Sensor drift, transient erroneous sensor values, sensor failures, and actuator failures degrade the robot's performance. The controller must purposefully recognize when these failures occur, so it can specifically tailor its use of sensors and actuators to minimize the impact of failures on the robot's performance. When on Mars, rovers do not have the luxury of repair when components fail. Thus, we want the robot's performance to degrade as gracefully as possible when components fail. Surprisingly little work has been done to advance the state of fault tolerant autonomous robots given the importance of this problem.

Hannibal's controller successfully integrates several topics into a single system using a common framework. Previous work exploring fully distributed, insect-like locomotion controllers has only been addressed for flat terrain (Beer, Chiel, Quinn & Espenschied 1992), (Quinn & Espenschied 1993), (Donner 1987)). Few rough terrain walkers are completely autonomous. For those that are, they either require significant computing power (Krotkov, Simmons & Thorpe 1990) or implement only a subset of Hannibal's rough terrain capabilities (Brooks 1989). We are not aware of any autonomous robot, walking or otherwise, that is fault tolerant to sensor or actuator failure. Hannibal's controller implements all these capabilities in a fully distributed, subsumption-based approach.

1.3 Organization of Thesis

The body of this thesis is divided into 5 chapters:

Chapter 2: Hannibal. This chapter presents a detailed description of robot used in this research. It covers the physical design, sensing, and computing capabilities of Hannibal.

Chapter 3: Controller Organization. This chapter discusses issues of Hannibal's controller design (such as scalability, modularity, flexibility, adaptivity, and robustness), and how we address them. It also presents the basic organization of the controller and provides a brief description of the control layers.

Chapter 4: Basic Locomotion. This chapter presents the behaviors responsible for flat terrain locomotion. The design of these behaviors is strongly inspired by insect locomotion control strategies. It defines the task, describes the implementation in detail, presents our results, and evaluates the performance of the system. It concludes by comparing this part of the controller to other statically stable walking robots.

Chapter 5: Rough Terrain Locomotion. This chapter presents the behaviors responsible for rough terrain locomotion. The design of these behaviors is inspired by insect rough terrain locomotion control strategies. It defines the task, describes the implementation in detail, presents our results, and evaluates the performance of the system. It concludes by comparing this part of the controller to other statically stable walking robots.

Chapter 6: Fault Tolerance. This chapter presents Hannibal's fault tolerance capabilities. It defines the task and presents the issues involved in designing fault tolerant systems. It describes the implementation in detail, presents our results, and evaluates the performance of the system. It concludes by comparing this part of the controller to related work.

Chapter 7: Conclusion. This chapter reviews the major results of the thesis, and suggests areas of future work.

Chapter 2

Hannibal

This chapter presents the physical, sensing, and computing aspects of Hannibal. Hannibal, and a duplicate robot named Attila, were designed and built under the supervision of Prof. Rodney Brooks in the Mobile Robotics Lab at MIT (Angle 1991). Hannibal is a small autonomous robot which performs its own sensing and computing on board[1]. This robot is perhaps the most sophisticated and complex robot for its size. A photograph of Hannibal is shown in figure 1.1.

2.1 The Physical Robot

2.1.1 Legs

Hannibal was designed with legs instead of wheels for greater mobility over rugged terrain. Each of Hannibal's six legs has 3 degrees of freedom (DOF) to allow arbitrary foot placement on a surface (see figure 2.1). The *lift axis* is horizontal. A rotational actuator raises and lowers the foot by rotating the leg about this axis. The *swing axis* is vertical which is important for the efficiency of the robot (Song & Waldron 1989). When the leg makes a step, most of the motion of the leg is rotation about the swing axis and therefore parallel to gravitational forces. As a result, relatively little work is done against gravity as the leg swings through the environment. A rotational actuator advances the leg along the direction of travel by rotating the leg

[1]Hannibal receives its power from an external power supply.

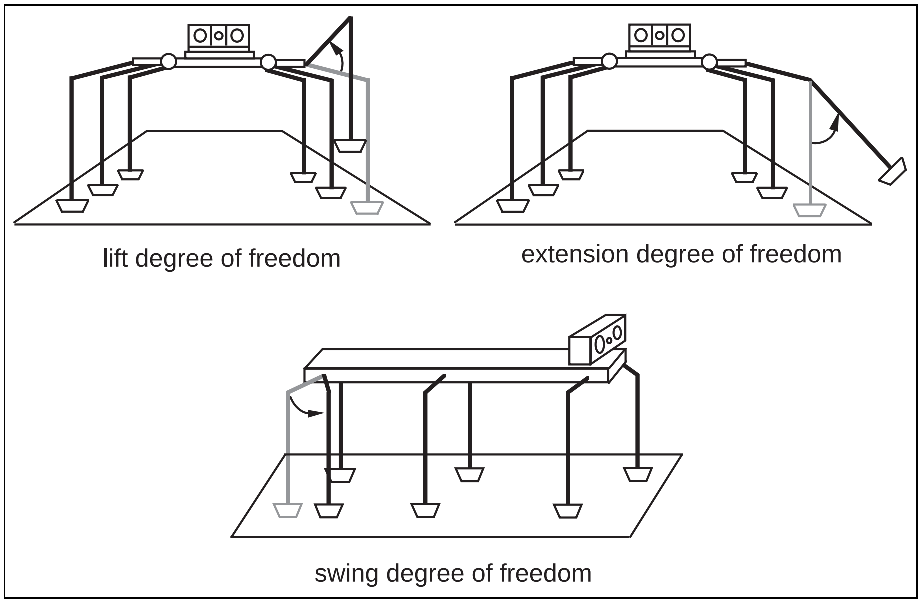


Figure 2.1: Each leg has three degrees of freedom.

about the swing axis. The *extension axis* is orthogonal to the lift and swing axes. A linear actuator extends and contracts the leg along this axis.

2.1.2 Body

A global degree of freedom mounted on Hannibal's body links the rotation of all six legs about their axles together (see figure 2.2). A spine actuator is responsible for rotating the legs about this global degree of freedom. The purpose of this DOF is to ensure the legs are always vertical[2]. Angle & Brooks (1990) argues that the load on each of the leg motors is independent of robot inclination if the legs are always vertical. As the robot's inclination increases during a climb, the global rotation of the legs brings the center of mass of the robot closer to the surface being climbed, and keeps it within the polygon of support for any inclination.

[2]This is only true if the robot is facing up or down hill.

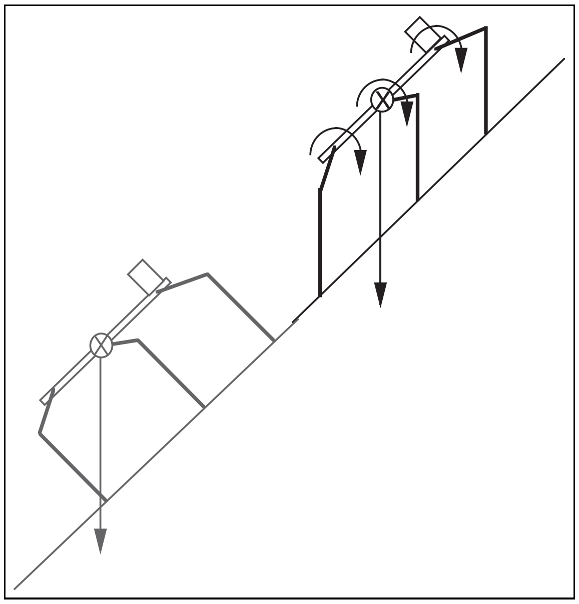


Figure 2.2: The body has one degree of freedom.

2.1.3 Size

Hannibal enjoys several benefits from its small size as argued in (Angle 1991). Hannibal measures 14 inches long, stands eight inches high, and weight 6 pounds. Because Hannibal is small, it has relatively low mass which gives rise to several advantages, including reduced dynamic effects, which simplifies control. Another advantage is lower power consumption–a smaller robot can be driven with smaller, lower power motors. The greatest advantage is a favorable strength-to-weight ratio of the robot's structure. The strength of a structure scales by its cross sectional area, while the weight of a structure scales by its volume. As a structure is proportionally scaled up, its weight increases at a faster rate than its strength. Hence, it is relatively easy to make a small structure quite strong with little mass.

2.2 Sensors

Hannibal has many sensors that provide complementary information. This serves several purposes. First, multiple sensors provide the robot with more information about its environment, which helps the robot behave more intelligently. Second, multiple complementary sensors increase sensing robustness– if one sensor fails, its complementary sensors can provide the robot with similar information. Third, they enhance sensing reliability since the information from each sensor can be used to confirm the results of the other complementary sensors. This increases the confidence in the net sensor output.

Hannibal receives a tremendous amount of sensor information. The robot has over 100 physical sensors of 5 different types to provide the robot with over 60 sensory signals. The following types of sensors are mounted on Hannibal:

- Leg mounted force sensors: these are foil strain gauges that can be used to measure loads on the leg and to detect leg collisions. There is a set of strain gauges for each DOF of the legs. They are manufactured by Micro Measurements.

- Joint angle sensors: These are potentiometers that measure the joint angle for each DOF of the leg.

- Joint velocity sensors: The joint angle sensors are differentiated in analog for each DOF of the leg.

- Foot contact sensor: This is a linear potentiometer mounted on the ankle that measures the deflection of the foot as it presses against the ground.

- Inclinometer: This sensing unit is made up of a +/- 45 degree roll sensor and a 360 degree pitch sensor. This sensor is manufactured by Spectron.

Hannibal's sensors are used to sense the immediate terrain, provide the robot with knowledge of its physical configuration, and servo the motors which control its DOFs to specified positions. Most of Hannibal's sensors are mounted on its legs. These sensors provide the robot with terrain information as the legs sweep through the environment. Figure 2.3 shows where these sensors are mounted on the legs and how they are labeled in this thesis. The strain gauges mounted on the legs measure the force the external world is exerting on them. Hannibal uses them to sense vertical loading of the legs and to detect collisions the legs suffer as they move through the

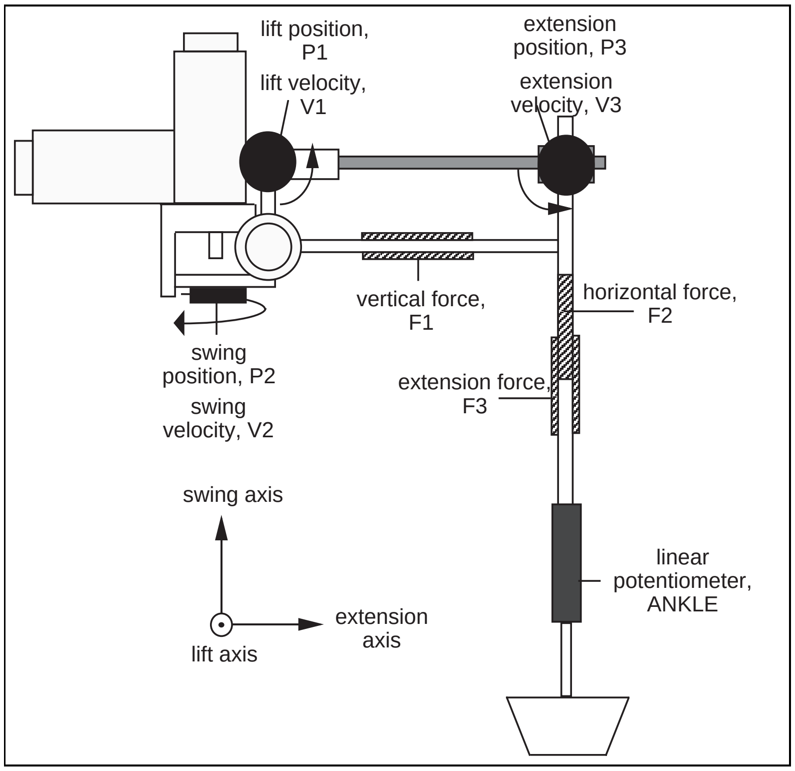


Figure 2.3: Leg mounted sensors with labels.

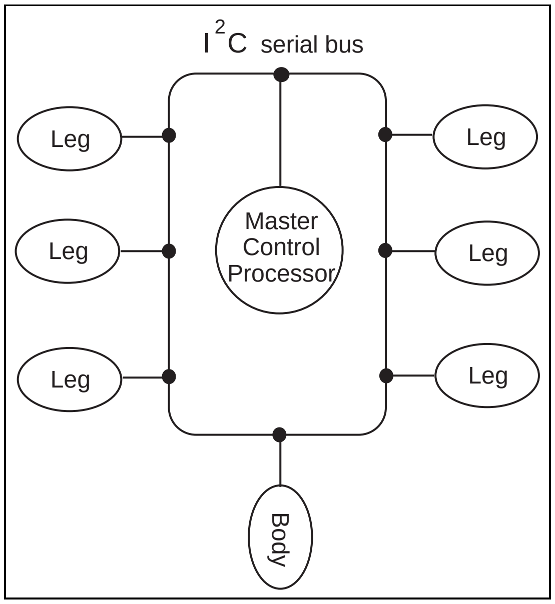


Figure 2.4: Hannibal's smart subsystems are linked by a serial network.

environment. The ankle potentiometer is used to sense foot loading. The robot uses this information to find secure footholds as it walks through the environment. The potentiometers are used to measure joint angles of all the robot's degrees of freedom. This information is used to servo control the motors to specified positions and to inform the robot of its physical configuration. The joint velocity sensors are used to control the motors. Velocity information in conjunction with target position information can be used to roughly sense leg collisions. For example, if a leg prematurely stops before it reaches its target position, the leg may have experienced a collision along the way. The inclinometer is used to sense the robot's pitch and roll. The spine potentiometer and the inclinometer, both mounted on Hannibal's body, are used to control Hannibal's spine actuator to keep all the legs vertical.

2.3 Computing

Hannibal is a complex robot. It receives over 60 sensor signals and orchestrates 19 DOFs[3] to locomote over rough terrain. To make the design and control of this robot manageable, Hannibal was divided into smart subsystems and then linked together with a medium speed serial network (I^2C bus). Each leg and the body are individual subsystems that posses a set of sensors and actuators. As shown in figure 2.4, Hannibal is organized with a single central processor to which 7 subsystem[4] control processors are attached. This allows high bandwidth communication and motor servoing to be handled locally, with only high level communication mon the serial network.

2.3.1 Master processor

The master processor is a 15 MHz Signetics 68070. The computational architecture is very similar to a Motorola 68010, but the 68070 has hardware support for the I^2C serial bus. This processor serves as the computational engine of the robot and runs the behavior control code. It receives sensor information from the robot's subsystems over the serial bus and uses this information to intelligently control the robot's behavior. It dictates the actions the robot takes by sending position, velocity, and force control commands to the actuators of the subsystems.

2.3.2 Satellite control processors

Each local satellite processor is a Signetics 87c751. The processor has two responsibilities. The first is to acquire the sensor information of its subsystem and send it to the master processor. The second is to receive commands from the master processor to servo the actuators of its subsystem. Servo control code running on this processor at 100 Hz servos the subsystem's actuators to the commanded position with velocity and/or force considerations. The processor uses the command information from the master processor and the sensor information of its subsystem to generate the pulse width modulation wave forms which drive the actuators.

[3]The robot has two additional actuators for its pan-tilt head but they were not used in this project.

[4]There are ten subsystems in the complete design.

2.3.3 Serial I^2C bus

The I^2C serial bus is a 100 Kbaud communication system which is used to connect the various subsystems on Hannibal. Bus transactions occur 30 % of the time which allows the satellite processors to exchange information with the master processor at 10 Hz. The bus is idle for 70 % of the time to allow the satellite processors to execute their servo control code. The I^2C bus is not fast enough to do real-time motor servoing, and this limitation forces modularity on the robot.

Chapter 3

Controller Overview

3.1 Issues of Controller Design

Hannibal is a complex robot which operates in a complex environment. Hannibal's controller must process over 60 sensor signals and command 19 actuators such that Hannibal effectively locomotes over rough terrain in real-time. To satisfy this formidable task, Hannibal's controller must satisfy several requirements:

- The controller must *scale* well to efficiently and effectively utilize numerous sensors, actuators, and processes. Adding more sensors, actuators, or processes to the system increases the load on the controller. We do not want this additional load to result in computational bottlenecks. If the controller scaled poorly, attempting to enhance the robot's capabilities by adding more sensors, actuators, or processes could degrade the robot's performance rather than enhance it.

- The controller must be *modular* so that additional sensors, actuators, or processes can be integrated readily. Specifically, we want to be able to quickly add or change the processes which interpret new sensor information, utilize new actuators, or add new processing capabilities to the system.

- The controller must be *flexible*. It must characterize numerous terrain features and control the robot to perform a variety of maneuvers to successfully traverse rough terrain. In addition, the controller must control the robot's legs and body simultaneously to effectively locomote over natural terrain.

- The controller must be *robust* and *adaptive*. The robot's behavior should be resilient to both environmental changes and internal changes. Environmental changes are attributed to irregular terrain. The robot must be able to detect these terrain variations and adapt to them accordingly. Internal changes are attributed to various hardware failures, erroneous sensor readings, or sensor drift. The robot must recognize these faults and minimize their effect on the robot's performance.

3.2 Robot Behavior Control Philosophy

Brooks (1986) proposed the idea of *subsumption* as an alternative approach to the robot behavior control problem. This approach decomposes the control problem into task achieving layers. Each slice in the vertical division is level of competence. The main idea is that we can build layers of a control system corresponding to each level of competence and simply add a new layer to an existing set to move to the next higher level of overall competence. The lower layers run continually and are unaware of higher layers. However, when the higher layers wish to take control they can subsume the roles of lower levels.

The subsumption approach creates tight couplings between sensors and actuators on the robot, separated by only very limited amounts of reasoning in the form of simple rules. The approach is embodied in the *Subsumption Architecture* which uses finite state machines augmented with timing elements (AFSMs) to construct simple rules. The AFSMs communicate through message passing, mutual suppression (one AFSM stops all inputs to another for a fixed time period), and inhibition (one AFSM stops all outputs of another for a fixed time period).

Combinations of AFSMs (also called *processes*) form *behaviors* (also referred to as *agents*), the building blocks of the *Behavior Language* (Brooks 1990). The Behavior Language is a high level language for writing subsumption programs. Behaviors run concurrently and asynchronously, perform their own perception, monitor their input wires, perform computation, control actuators, or send messages out their output wires. It is important to design the controller such that conflicting behaviors are not active at the

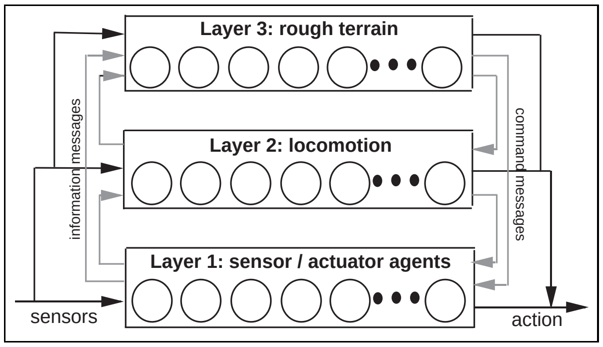


Figure 3.1: Levels of competence of Hannibal's controller.

same time. To deal with this situation, behaviors have the ability to inhibit outputs or suppress inputs of other behaviors. Inhibition of outputs is used when the inhibiting behavior does not want the inhibited behavior's outputs influencing the system. Behaviors can prevent other behaviors from becoming active by suppressing inputs used for activating other behaviors. Individual behaviors are connected to form task-achieving modules (the robot's observable behaviors), and these task-achieving modules can be grouped together to form layers. Each layer is a level of competence which corresponds to the robot's abilities.

The subsumption approach is a set of philosophical concepts about robot behavior design which stresses the issues of reactivity, concurrency, and real-time control (Mataric 1992). This approach is robust since failure of any layer does not affect the layers below. Additionally, this organization allows for modular addition and removal of behaviors, and thus for incremental design and debugging. Most importantly, it allows for a tight loop between sensing and action which can be performed quickly and with much less computation.

3.3 Overview of the Controller

Numerous papers argue in favor of behavior based control for modularity, flexibility, robustness, and adaptability considerations ((Brooks 1986), (Maes 1990), (Rosenblatt & Payton 1989)). In addition, the subsumption approach has been successfully demonstrated on several robots in our lab such as Toto (Mataric 1990), Herbert (Connell 1989), Squirt (Flynn, Brooks, Wells &

Barrett 1989), and Genghis (Brooks 1989). Given these previous successes, we implemented a subsumption-based controller on Hannibal written in the Behavior Language. Hannibal's controller is perhaps the most complex behavior based controller currently in existence. This gives us the opportunity to test the scalability of this approach.

Hannibal's controller consists of three levels of competence as shown in figure 3.1. The lowest level performs basic functions such as processing sensory information and commanding actuators. The middle layer implements flat terrain locomotion capabilities. The highest layer implements rugged terrain locomotion capabilities. A brief overview of the composition of these levels is presented below. Later chapters will describe structure and function of these levels in detail.

3.3.1 Sensor-Actuator Level

This level processes sensory information and sends commands to the actuators. This level consists of several types of agents:

Virtual Sensor Agents

Task driven sensing is performed by the virtual sensor agents. The virtual sensors use information from multiple sensors of different and complementary types to produce qualitative assessments of the robot's interaction with the world. There is a separate virtual sensor for each condition the robot must detect, such as ground contact or collisions. The computations performed by the virtual sensors are fast and simple–typically using thresholds. Using multiple sensors has several advantages: First, each sensor output contains noise and measurement errors; however multiple sensors can be used to determine the same property with the consensus of other complementary sensors. In this way, sensor uncertainty can be reduced. Second, the output of a single sensor may be ambiguous and misleading; however other complementary sensors can be used to resolve this ambiguity. Third, multiple sensor data can be integrated to provide information which might otherwise be unavailable or difficult to obtain form any single type of sensor. Finally, if some sensors fail, it is important the robot have other sensors it can rely on for similar information. Here is a list of Hannibal's main virtual sensors:

- ground-contact virtual sensor

- step-in-hole virtual sensor
- over-rock virtual sensor
- inclination virtual sensor
- lift-velocity virtual sensor
- swing-velocity virtual sensor
- extend-velocity virtual sensor

Actuator agents

The actuator agents are the only agents that send commands to the actuators– each agent commands one actuator. All requests for that actuator are sent to the corresponding agent, but only one request is satisfied. Currently the priority for which request is serviced is hardwired. We use actuator agents to keep track of potential behavior conflicts over the use of the same actuator. The following actutator agents are implemented:

- lift agent
- swing agent
- support agent
- step agent
- spine agent
- extend agent

Fault Tolerance processes

The fault tolerance processes and agents are responsible for recognition, masking, and recovery from component failures. Other processes a responsible for reintegrating the use of repaired components into the system. Together, these processes and agents make the robot tolerant of sensor and actuator failures. They are described in detail in chapter 6. Below is an abbreviated list of Hannibal's fault tolerance processes and agents:

- sensor-state processes
- sensor-monitor processes
- monitor-concensus processes
- injury agents
- failure-masking processes
- transient-failure-recovery processes
- dynamic-calibration agents
- permanent-failure-recovery processes
- repaired-component-integration processes

Static Calibration processes

The static calibration processes are responsible for calibrating Hannibal's sensors at start up. Here is a short summary of the types of static calibration agents:

- loaded-calibration agents
- unloaded-calibration agents

3.3.2 Basic Locomotion Level

This level encompases the behaviors needed to locomote effectively over flat terrain given stability, speed, mobility, and leg failure considerations. This level is described in detail in chapter 4. The behaviors of this layer are strongly inspired by models of insect locomotion control. Below is an abbreviated list of the basic locomotion agents.

- oscillator agent
- speed (or gait) agent
- oscillator-phase agents

- turn agents
- direction-of-travel agents
- lesion-compensation agents

3.3.3 Rough Terrain Level

This level encompasses the behaviors needed to locomote effectively over rough terrain. This level is presented in detail in chapter 5. These behaviors enable Hannibal to locomote over terrain with obstacles of various sizes, depressions of various widths and depths, and undulations. Here is a list of Hannibal's primary rough terrain behaviors:

- load-walk behavior
- traverse-low-obstacle behavior
- find-foothold behavior
- step-over-gap behavior
- avoid-cliff behavior
- slope-adaption behaivor
- caution behavior
- lift-body behavior
- step-high behavior
- impatient behavior

3.4 Managing Complexity

3.4.1 Layering

Hannibal's controller is made up of several levels of competence, and these levels contain many processes, agents, and behaviors. In general, the lower

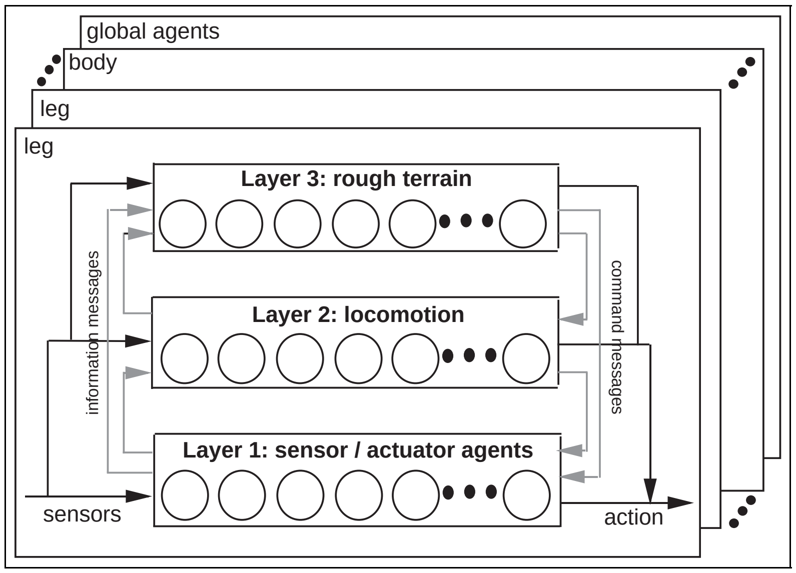


Figure 3.2: Hannibal's control task is distributed among several subsumption-based controllers.

levels are more fundamental to the function of the robot and can operate without the presence of higher levels. When present, the higher levels dominate the lower levels. Furthermore, there is stratification within the levels of competence where higher agents dominate lower agents. Lower level agents pass information messages to higher level agents, and higher level agents send command messages to the lower level agents. Thus, Hannibal's controller is layered on two different scales.

3.4.2 Distributedness

As presented in chapter 2, Hannibal consists of several subsystems. Each subsystem has its own sensors, actuators, and servo control. All subsystems run concurrently for Hannibal to locomote effectively. To simplify the control of a robot of this complexity, we decomposed the global control problem into several local control problems. We did this by implementing a subsumption-based controller for each susbsytem which is responsible for govering the behavior of that subsystem. Therefore, instead of controlling a robot with over 60 sensory signals and 19 degrees of freedom with a single subsumption-based controller, the task is distributed among several subsumption-based controllers–each responsible for a subset of the overall control problem. This is illustrated in figure 3.2. Thus, Hannibal's control structure is distributed on two levels: the subsumption-based controller is distributed among behaviors, and several subsumption-based controllers are distributed among the subsystems.

With this approach, Hannibal can be modeled as a robot which is composed of several *sub-robots*. Each of Hannibal's subsystems (legs and body) is an autonomous sub-robot posessing its own set of sensors, actuators, servo control, and behavior control[1]. However, these sub-robots cannot function independently from one another because they are physically coupled to each other through the robot and through the terrain. Consequently, these sub-robots must cooperate with one another so that the overall system can achieve its goals. To achieve cooperation between the sub-robots, the sub-robots communicate with each other. This is done either directly between the sub-robots, or indirectly through special *global* agents which communictate a unified message to all relevant subsystems. The theme of *local control with cooperation* runs throughout this thesis and is elaborated upon in later chapters.

3.5 Summary

Designing and implementing Hannibal's controller is an exercise in managing complexity. I found the subsumption approach was effective for designing an intricate controller, and for controlling a robot of Hannibal's complexity. Currently, the controller consists of approximately 1500 processes. In building a controller of this complexity from the bottom-up, we have demonstrated the modularity of the design. This modulartity appears on every scale of the controller: AFSMs form agents, agents form task-achieving behaviors, task-achieveing behaviors form levels of competence, levels of competence form a subsumption-based controller, subsumption-based controllers run within intelligent subsubsystems, and these subsystems cooperate to control the behavior of the overall robot. The controller scales well since Hannibal oper-

[1]The behavior control code is distributed among the subsystems in software. Physically all behavior control code runs on one processor.

ates in real-time despite the large number of concurrently running processes[2]. Furthermore, the controller is quite flexible. Each subsystem processes its sensory information and reacts to the specific circumstances confronting it simultaneously with the other subsystems. Finally, the behavior control is robust and adaptive to changing circumstances. The robot readily adapts to irregular terrain and is tolerant of hardware component failures.

[2]Because all of Hannibal's behavior control runs on a single processor, these processes are simulated to run concurrently.

Chapter 4

Basic Locomotion

This chapter presents the development and implementation of Hannibal's locomotion network. We present various insect locomotion models which influenced the design of our locomotion controller. We also present a variety of insect-model-based controllers we implemented on Hannibal to generate a variety of stable insect-like gaits. Of these controllers, we chose the best implementation as the foundation of our locomotion controller. We expand the capabilities of the controller to implement turning, changing direction, and lesion compensation. Once the description of the controller is finished, we present our results and discuss the performance of the system. We conclude by relating Hannibal's locomotion scheme to other statically stable walking robots.

4.1 The Hexapod Locomotion Challenge

Hannibal must have basic locomotion capabilities before it can effectively traverse natural terrain. *Basic locomotion* encompasses the capabilities needed to locomote effectively over flat terrain. These capabilities extend beyond stable locomotion. The abilities to turn and change direction are important for mobility considerations. The ability to change gait is important for speed, energy, and stability considerations. Lesion compensation is important in the event of leg failure–if Hannibal's loses the use of a leg, Hannibal's gait must change to maintain stability while walking with fewer legs.

To address this challenge, we added a layer to Hannibal's control architecture to give Hannibal basic locomotion capabilities. The Basic Locomotion Layer is built on top of the Sensor-actuator Layer. Insect locomotion research has provided valuable insight as to how to implement basic locomotion capabilities on Hannibal.

To mimic insect locomotion, the locomotion controller is fully distributed and adheres to a *local control with cooperation* paradigm. As argued in chapter 3, this paradigm is also an effective means for managing Hannibal's numerous sensors and actuators. Locomotion control is distributed evenly among the six legs. Each local leg controller is responsible for generating the cyclic motion of its leg. The local leg controllers run simultaneously; however, they are not independent of one another. The six legs must work together as a team for the robot to locomote effectively. To achieve inter-leg cooperation, the local leg controllers communicate with each other to synchronize and coordinate leg behavior. Using this approach, we have implemented the following basic locomotion capabilities on Hannibal:

- Locomote in a statically stable manner.
- Change speed by switching among a variety of insect-like gaits (wave gaits).
- Change direction of travel.
- Turn with varying sharpness.
- Lesion compensation for any single leg or the two middle legs.

4.2 Definition of terms

Below are several terms we use through out this chapter. Please also refer to the accompanying figure, (figure 4.1).

1. *Protraction*: The leg moves towards the front of the body.
2. *Retraction*: The leg moves towards the rear of the body.
3. *Anterior*: Situated toward the front of the body.
4. *Posterior*: Situated toward the rear of the body.

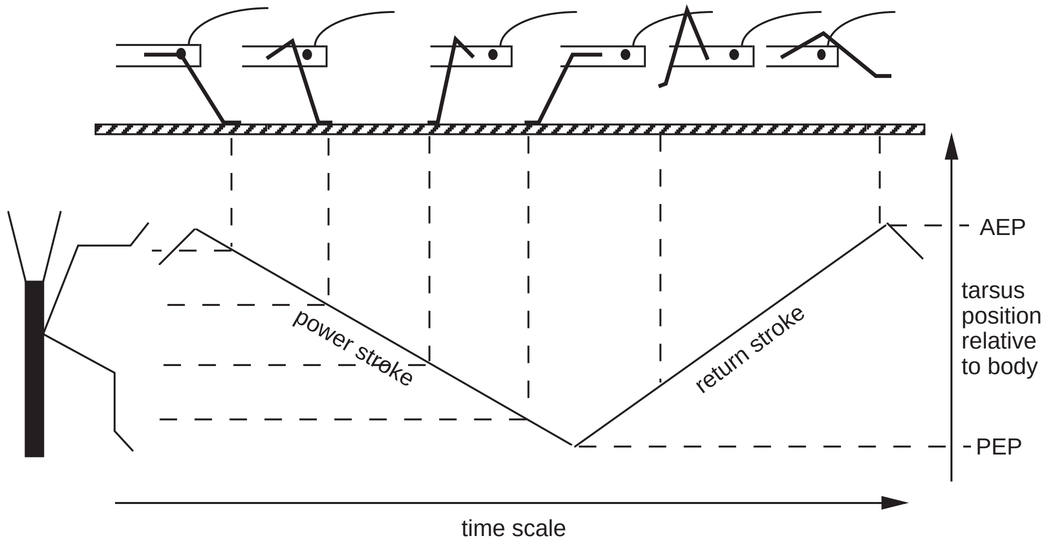


Figure 4.1: During the recovery phase, the leg lifts and swings to the start position of the next power stroke. In forward locomotion, the leg moves towards the AEP during the recovery phase. During the support phase, the leg supports and propels the body along the direction of motion. In forward walking, the leg moves toward the PEP during the support phase. Adapted from (Cruse 1990).

5. *Contralateral*: Situated on directly opposite sides of the body.
6. *Ipsilateral*: Situated on the same side of the body.
7. *Rostral*: Directed toward the front part of the body.
8. *Caudal*: Directed toward the hind part of the body.
9. *Power stroke*: The leg is on the ground where it supports and propels the body. In forward walking, the leg retracts during this phase. Also called the stance phase or the support phase.
10. *Return stroke*: The leg lifts and swings to the starting position of the next power stroke. In forward walking, the legs protracts during this phase. Also called the swing phase or the recovery phase.
11. *Anterior extreme position (AEP)*: In forward walking, this is the target position of the swing degree of freedom during the return stroke.
12. *Posterior extreme position (PEP)*: In forward walking, this is the target position of the swing degree of freedom during the power stroke.

4.3 Insect locomotion

Insect locomotion is exceptionally robust, adaptive, and versatile. We would like Hannibal to walk with the same qualities. Toward this goal, we have used various models of insect locomotion control to develop Hannibal's locomotion control.

4.3.1 Wilson

Wilson (1966) presents a descriptive model for characterizing all of the commonly observed gaits of insects, including those resulting from amputation. Some of these gaits are shown in figure 4.2. These rules are adequate for describing the qualitative features of leg coordination in most insects when they walk on smooth horizontal surfaces (note they do not account for all the empirical data).

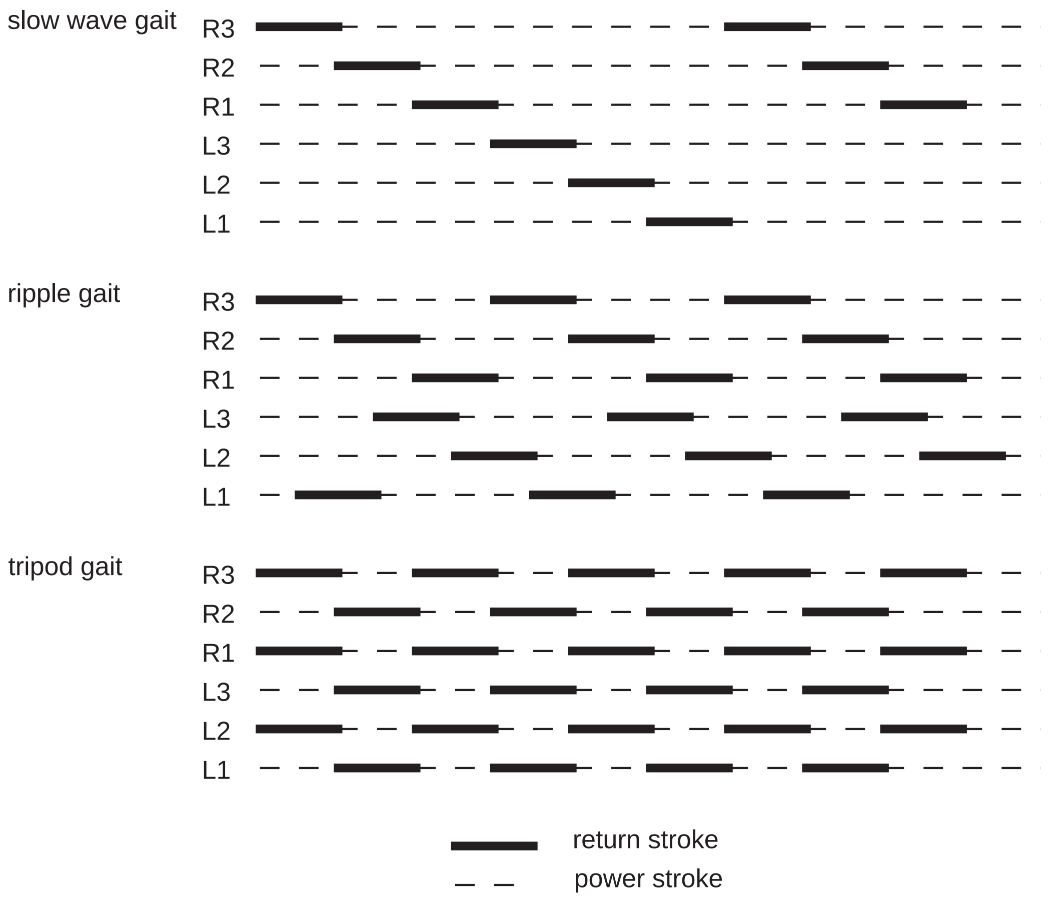


Figure 4.2: Some commonly observed gaits of insects. All are members of the family of wave gaits. Adapted from (Wilson 1966).

- A wave of protractions runs from posterior to anterior. No leg protracts until the one behind is placed in a supporting position.
- Contralateral legs of the same segment alternate in phase.
- Protraction time is constant.
- Frequency varies (retraction time decreases as frequency increases).
- The intervals between steps of the hind leg and middle leg and between middle leg and foreleg are constant, while the interval between the foreleg and hind leg steps varies inversely with frequency.

4.3.2 Pearson

Keir Pearson and his collaborators investigated the neural systems that control walking in the cockroach ((Wilson 1966), (Pearson 1976)). They developed neurological models to explain the control of an individual leg and the coordination between legs. The resulting gaits are consistent with Wilson's descriptive model.

The complex unit of action that controls the stepping pattern of a single leg combines three elementary units of action–the oscillator, the servomechanism, and the reflex (see figure 4.3). The oscillator generates the stepping pattern of the leg by controlling the activation of the flexor motor neurons and the extensor motor neurons. The flexor motor neurons protract the leg when activated, and the extensor motor neurons retract the leg when activated. At the peak of its cycle, the oscillator generates the leg's swing phase by activating the flexor motor neurons and inhibiting the extensor motor neurons. The duration of this protraction command is independent of the oscillator's period, which accounts for Wilson's third rule: "Protraction time is constant". Throughout the remainder of the oscillator's cycle, the oscillator generates support phase by activating the extensor motor neuron. The retraction rate varies with the frequency of the oscillator, which accounts for Wilson's fourth rule: "Retraction time varies". The rhythmic pattern established by the oscillator is modified by a servomechanism circuit and a reflex circuit. The servomechanism circuit uses sensory signals fed back to the central nervous system from joint receptors and/or stretch receptors. The sensory feedback adjusts the strength of the supporting and pushing

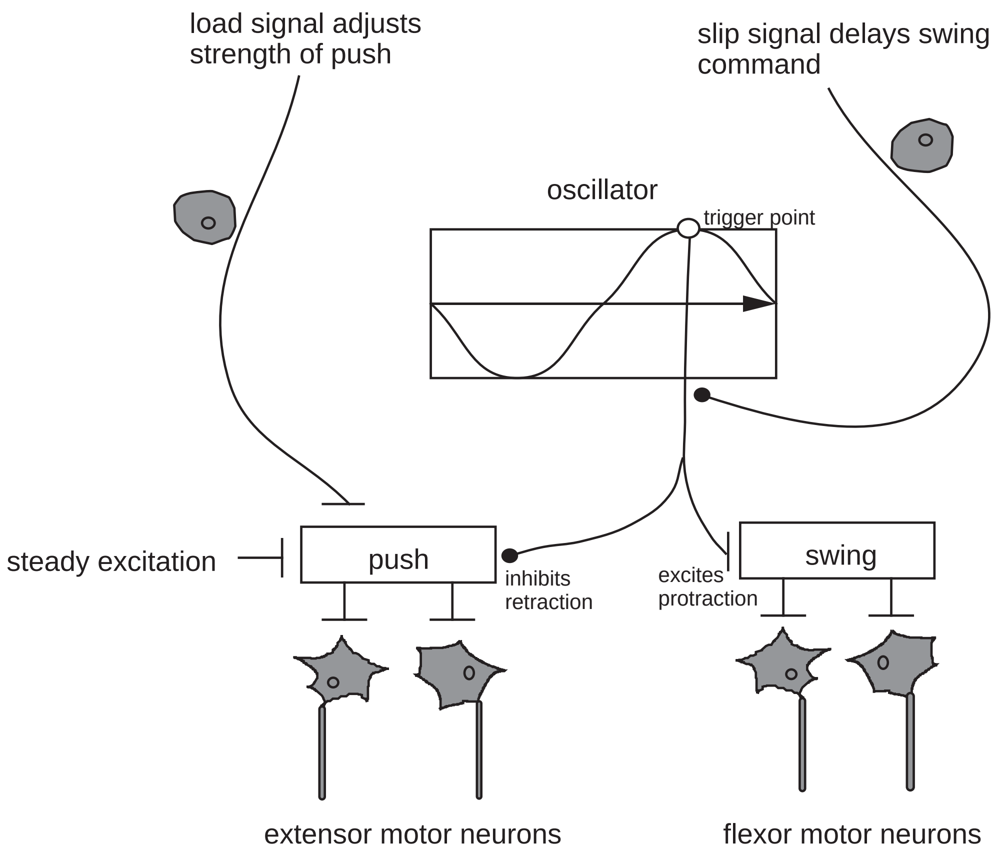


Figure 4.3: Pearson's neural circuit for controlling the stepping motion of a single leg. An oscillator provides the stepping rhythm. It triggers a swing command near the peak of its cycle. The swing command excites the motor-neuron circuit that swings the leg forward, and inhibits the push circuit. The push circuit presses the foot to the ground and draws it back. A steady excitatory input keeps the push circuit active whenever it is not inhibited by the swing command. Adapted from (Wilson 1966).

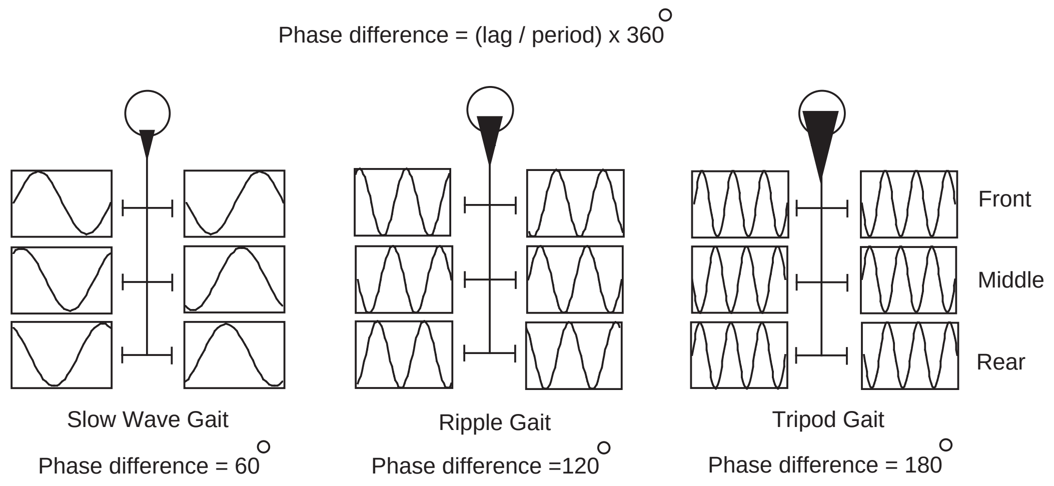


Figure 4.4: Various gaits emerge from changing the frequency of the local leg oscillators.

contractions to match variations in load. The reflex circuits delay or prevent the command that swings the leg forward. The input to these reflexes comes from receptors that detect whether another leg has taken up some of the load. For example, if a middle leg is amputated, it fails to take up the load at the normal time. This failure delays the protraction of the front leg. When the rear leg hits the ground it takes up some of the load, and this releases the delayed protraction of the front leg. As a result the front and hind legs step 180 degrees out of phase instead of in phase (transitions from a tripod gait to a slower wave gait).

The unit that controls walking is comprised of six leg-stepping units (one for each leg) and a command neuron(s) . These leg-stepping units are coordinated by coupling signals that pass back and forth between the oscillators. The oscillators that control the legs directly across form one another maintain a constant 180 degrees phase relationship This accounts for Wilson's second rule: "Contralateral legs of the same segment alternate in phase". In contrast, the three oscillators along either side of the body maintain a temporal lag. The fixed lags between ipsilateral oscillators account for the first part of Wilson's fifth rule "the intervals between steps of the hind leg and middle leg and between middle leg and the front leg are constant." The fact that the front oscillator lags the middle, which in turn lags the rear oscillator, ac-

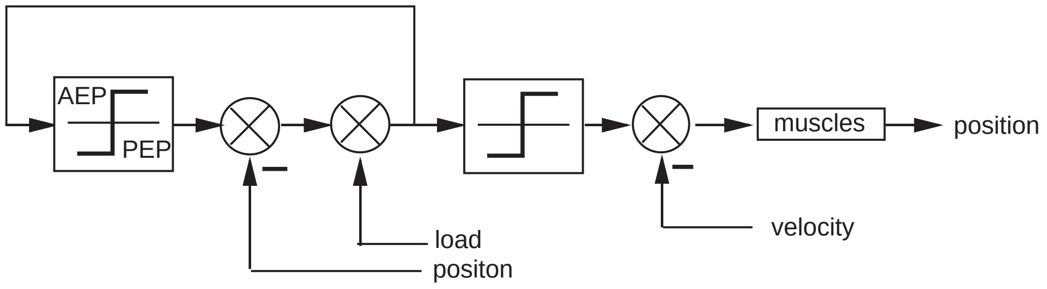


Figure 4.5: Cruse's circuit for controlling the motion of an individual leg. The left side of the circuit determines whether the system adopts the power stroke or the return stroke. The left relay characteristic produces the two alternative target positions, AEP or PEP, when its input value is positive or negative, respectively. The value of the target position is compared with the actual leg position. The output of the left side of the circuit is sent as a reference input to the right side of the circuit. The right side of the circuit is a velocity-controlling feedback system. Adapted from (Cruse 90).

counts for Wilson's first rule: "A wave of protractions runs from posterior to anterior". The final component of the walking circuit is the command neuron or neurons, which sets the pace of walking by changing the period of the oscillators. With a strong command signal the oscillators cycle rapidly (makes the period shorter). Oscillators with a fixed-lag coupling must change their phase relationship when their period changes. Consequently, the changes in gait are simply changes in the phase relationship between the three oscillators on either side. See figure 4.4.

4.3.3 Cruse

Holk Cruse ((Cruse 1976*a*), (Cruse 1976*b*), (Cruse 1979), (Cruse 1980*a*), (Cruse 1990*b*)) has studied locomotion of several animals. Among other models, he developed two models for the locomotion of walking stick insects *Carausius morosus*. The first is a model for the control of individual legs; the second is a model for the coordination between legs. The resulting gaits are consistent with Wilson's descriptive model.

Cruse's model for the control of individual legs of the walking stick insect is presented in (Cruse 1980*b*) and shown in figure 4.5. Each leg has a step

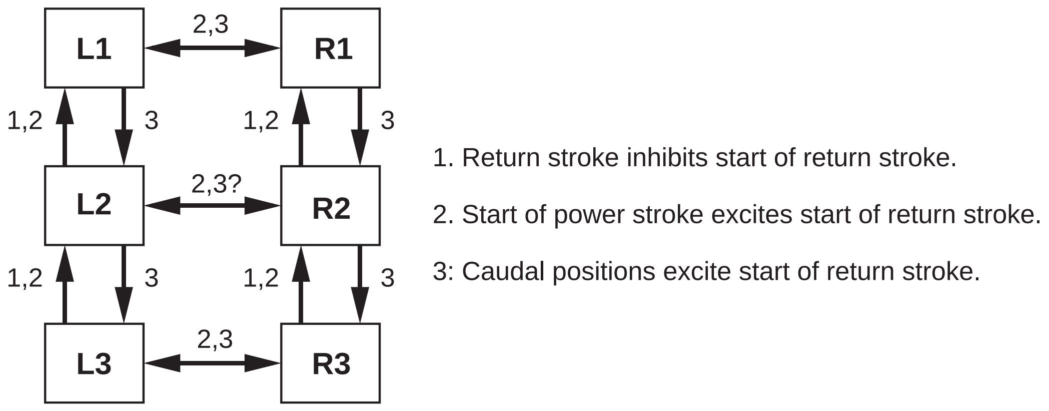

L1
R1
L2
R2
L3
R3
2,3
2,3?
2,3
1,2
3
1,2
3
1,2
3
1,2
3
1. Return stroke inhibits start of return stroke.
2. Start of power stroke excites start of return stroke.
3: Caudal positions excite start of return stroke.

Figure 4.6: Cruse's circuit for leg coordination. This figure summarizes the coordinating mechanisms operating between the legs of a stick insect. Adapted from (Cruse 1990).

pattern generator that is responsible for the transition between the power stroke and the return stroke. The step pattern generator transitions from the return stroke to the power stroke when the leg reaches the AEP, and it transitions from the power stroke to the return stroke when the leg reaches the PEP, the load on the leg is small, and the adjacent legs are in their supporting phase. These conditions insure the leg doesn't lift until the body is supported by other legs. The value of the target position (AEP or PEP) is sent as a reference position to a velocity controlling feedback system.

In (Cruse 1980*a*), the author presents 6 mechanisms responsible for the coordination between legs observed in walking stick insects. These mechanisms are redundant in re-establishing coordination in the case of minor disturbances. This thesis presents the three primary mechanisms. These mechanisms effect the threshold for beginning the return stroke by adjusting the PEP of the receiving leg. The PEP adjustment is based on the sum of the interleg influences affecting that leg. The threshold for beginning the power stroke (AEP) is fixed. The influences are sent between legs as shown in figure 4.6. Figure 4.7 shows what these influences look like as a function of leg position.

- $Mechanism_1$: Rostrally directed influence inhibits the start of the return stroke in the anterior leg by shifting PEP to a more posterior position. This is active during the return stroke of the posterior leg.

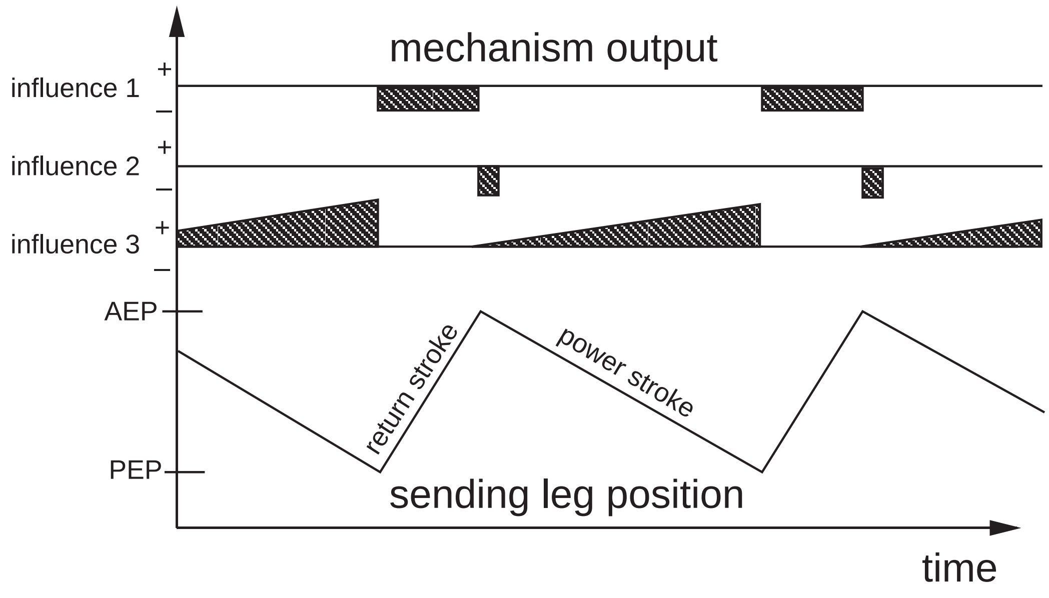


Figure 4.7: Relation of the coordinating mechanisms to leg position.

- $Mechanism_2$: Rostrally directed influence excites the start of the return stroke in the anterior leg by shifting the PEP to a more anterior position. This is active during the start of the power stroke of the anterior leg.

- $Mechanism_3$: Caudally directed influence excites start of a return stroke in the posterior leg. The start of the return stroke is more strongly excited (occurs earlier), the farther the anterior leg is moved rearward during the power stroke. This causes the posterior leg to perform the return stroke before the anterior leg begins its return stroke. This is active during the power stroke of the anterior leg.

How do these three mechanisms stimulate stability and back to front metachronal waves? As an example, let's discuss their influence on the right front leg, R1. For this discussion we use the leg labeling convention of figure 4.6, and we say $mechanism_1$ exerts $influence_1$, $mechanism_2$ exerts $influence_2$, and $mechanism_3$ exerts $influence_3$. R1 receives four influences: $influence_2$ and $influence_3$ from L1 and $influence_1$ and $influence_2$ from R2. $Influence_2$ from R2 and L1 inhibits R1 from beginning its swing phase while R2 and L1 are in support phase. This encourages stability by discouraging R1 from lifting before all of its adjacent legs are supporting the body. $Influence_3$ from L1 stimulates R1 to start its swing phase while L1 is in support phase. This influence strengthens as L1 approaches its PEP so that R1 will perform its return stroke before L1 finishes its power stroke. $Influence_1$ from R2 provokes R1 to begin its return stroke after R2 performs its return stroke. This establishes back to front metachronal waves. R1 sends three influences: $influence_2$ and $influence_3$ to L1 and $influence_3$ to R2. Sending $influence_3$ to L1 and R2 contributes to stability by exciting L1 and R2 to swing while R1 supports the body. Sending $influence_2$ to L1 encourages 180 degree phasing between contralateral legs of the same segment.

Cruse's model for leg coordination has been tested in simulation ((Dean 1990), (Dean 1991*a*), (Dean 1991*b*), (Dean 1992*a*), (Dean 1992*b*)) and on a robot (Beer et al. 1992). Dean successfully tested four of the six mechanisms presented in Cruse (1990*a*). He used kinematic leg models in his simulation, so the effects of friction and inertia were not addressed. The four mechanisms include the three mechanisms described above and a targeting mechanism that adjusts the AEP of the posterior legs (this mechanism is responsible for the follow-the-leader gait observed in walking stick insects). Beer et al. (1992) successfully tested the three mechanisms described above on a hexapod robot with 2 DOF legs. Whereas Dean's goal was to reproduce experimentally observed aspects of walking stick insect gaits, Beer and his colleagues goal was to produce effective robot locomotion. By removing various mechanisms Beer and his colleagues determined that $mechanism_2$ promotes normal back to front metachronal waves, and $mechanism_3$ promotes 180 degree phasing between cross-body leg pairs. Dean found that the mechanisms rapidly recoordinate the gait in response to leg perturbations. Beer et al. (1992) and Dean (1991*b*) found $mechanism_3$ is the most effective single coordinating mechanism and $mechanism_2$ was the least effective. Both papers report Cruse's mechanisms produce stable metachronal coordination over a wide range of step periods provided all the legs retract at the same velocity.

4.4 Cruse Control

Hannibal's first locomotion network implements $mechanism_1$, $mechanism_2$ and $mechanism_3$ presented in Cruse (1990*a*) and in the preceding section.

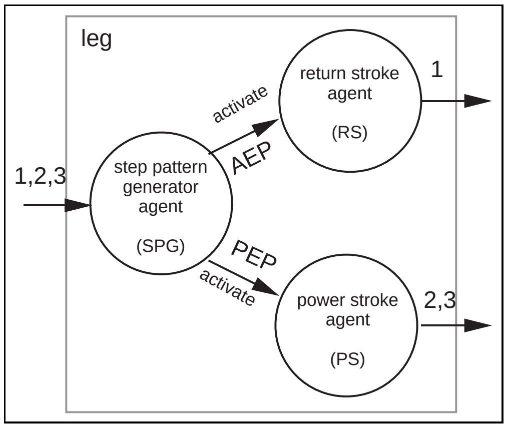


Figure 4.8: The cyclic stepping pattern of each leg was implemented using this circuit. The step pattern generator agent produced the step cycle by exciting the return stroke agent and the power stroke agent in turn. The arrows with numbers represent the direction the coordination mechanisms were sent between the legs. The numbers represent Cruse's coordination influences as labeled in this section.

We hoped these mechanisms would enable Hannibal to walk with stable metachronal gaits over a wide range of step periods and would make these gaits robust to leg disturbances. Each leg had a network of three agents as shown in figure 4.8: a step pattern generator agent, a return stroke agent, and a power stroke agent.

- The step pattern generator agent is responsible for transitioning between the return stroke and the power stroke. The transition from the return stroke to the power stroke occurs when the leg position is greater than or equal to the AEP. In this implementation, the AEP is fixed. The transition from the power stroke to the return stroke occurs when the leg position is less than or equal to the PEP. It computes the PEP from $influence_1$,$influcence_2$ and $influence_3$ from peripheral legs.

- The return stroke agent is responsible for lifting and swinging the leg to the starting position of the power stroke. It is activated by the step pattern generator. While this agent is active, it exerts $mechanism_1$. It receives the AEP from the step pattern generator agent.

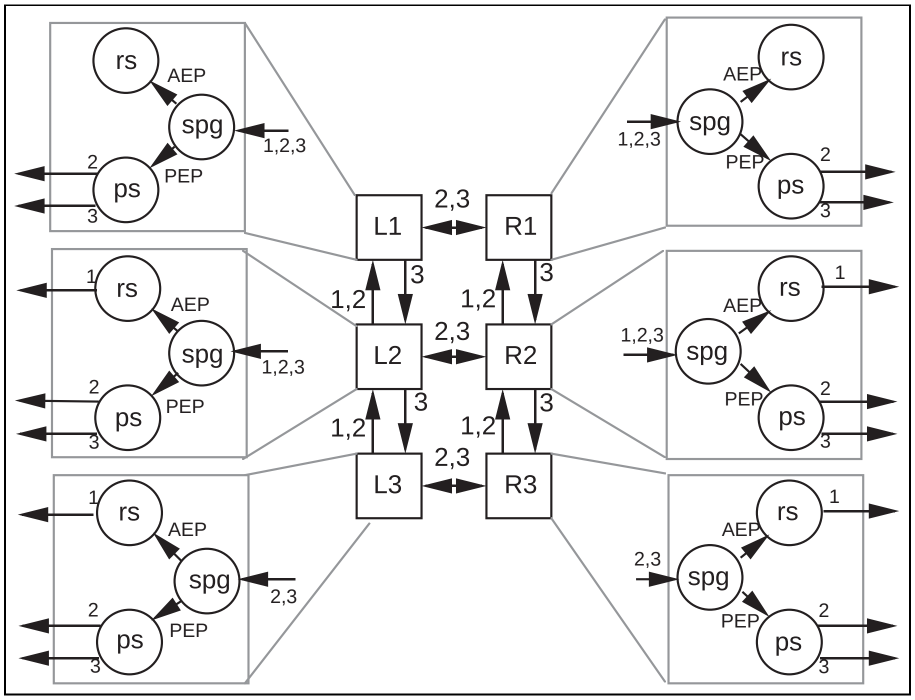


Figure 4.9: This circuit implemented Cruse Control on Hannibal. A step cycle circuit was implemented on each leg to produce the step pattern for that leg. The coordination mechanisms (represented as numbers with arrows) applied the appropriate constraints on the sequencing of the legs to produce insect-like gaits.

- The power stroke agent is responsible for supporting and propelling the body by steadily moving the leg to the PEP. While this agent was active it exerts $mechanism_2$ and $mechanism_3$. It is activated by the step pattern generator agent. It receives the updated PEP from the step pattern generator agent.

Figure 4.9 shows the gait coordination circuit. The form of the inhibitory/excitatory influences are shown in figure 4.7. The return stroke agent implements $mechanism_1$ by sending a negative constant in the rostral direction during its activation and 60ms after its deactivation. The power stroke agent implements $mechanism_2$ by sending a positive constant in the rostral direction starting 60ms after its activation and ending 60ms later. The

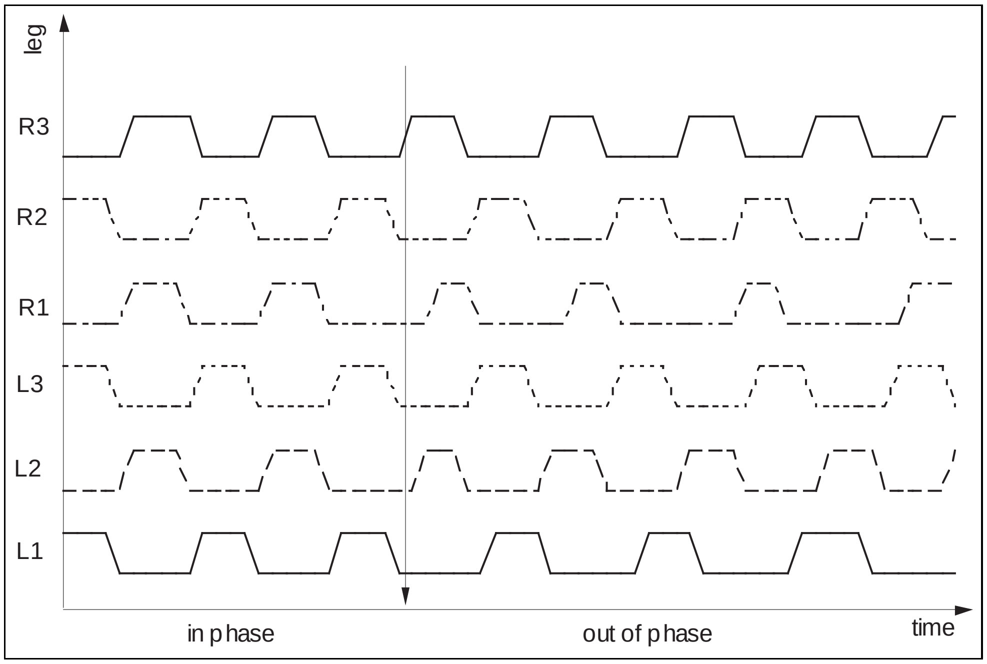


Figure 4.10: After implementing Cruse Control on Hannibal, we found that insufficient velocity control of the legs caused the gaits to shift in and out of phase.

power stroke agent implements $mechanism_3$ by sending a positive monotonically increasing ramp function in the caudal direction during its activation. The new PEP of a leg is computed from the influences it receives by the formula

$$PEP = PEP_{default} + \Sigma Influence_1 + \Sigma Influence_2 + \Sigma Influence_3 \quad (4.1)$$

where the standard stride length is given by

$$Stride_{standard} = AEP - PEP_{default} \quad (4.2)$$

This approach was somewhat successful on Hannibal. When Hannibal's legs were unloaded it was able to transition between gaits as a function of step period frequency. However, this approach failed to work once Hannibal was put on the ground. The models proposed by Cruse require that all the legs retract with the same velocity. Unfortunately Hannibal's velocity

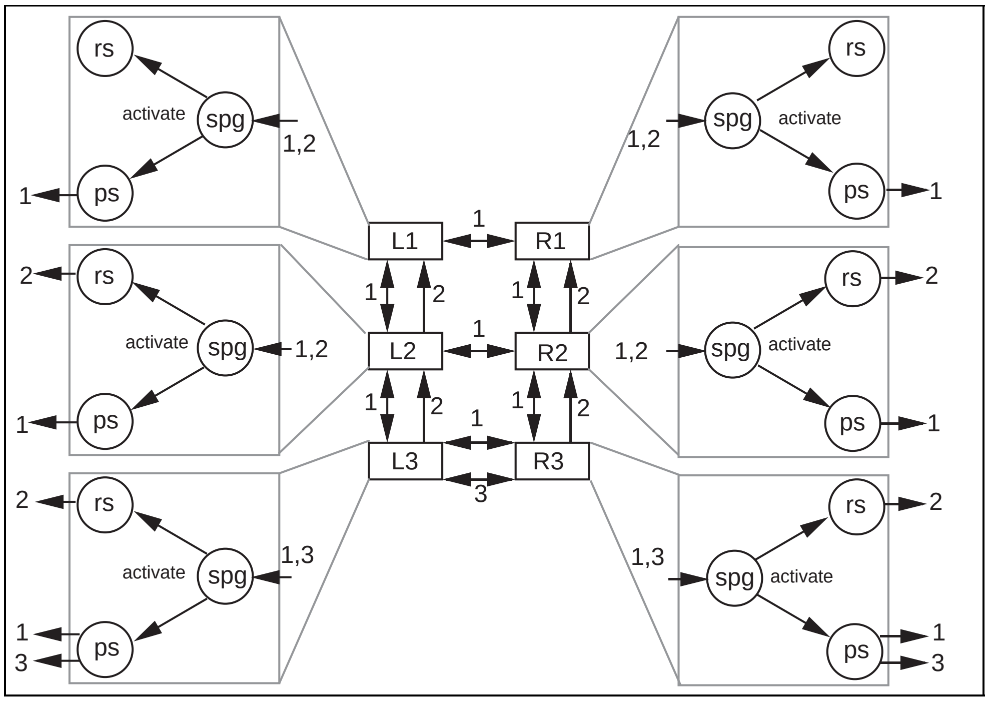


Figure 4.11: The Modified Cruse Control gait coordination circuit.

control is insufficient, so once Hannibal's legs were loaded they did not retract with the same velocity. This caused Hannibal to change gaits randomly and occasionally become unstable (as shown in figure 4.10).

4.5 Modified Cruse Control

Given Hannibal's velocity control is inadequate for Cruse control, we modified the three mechanisms to make them compatible with this shortcoming. The modified mechanisms are shown in figure 4.12. Figure 4.11 shows how the modified mechanisms are routed between the legs.

- Modified $mechanism_1$: During the recover stroke send a `wait` message to all adjacent legs. This enforces stability by not allowing adjacent legs to lift at the same time.

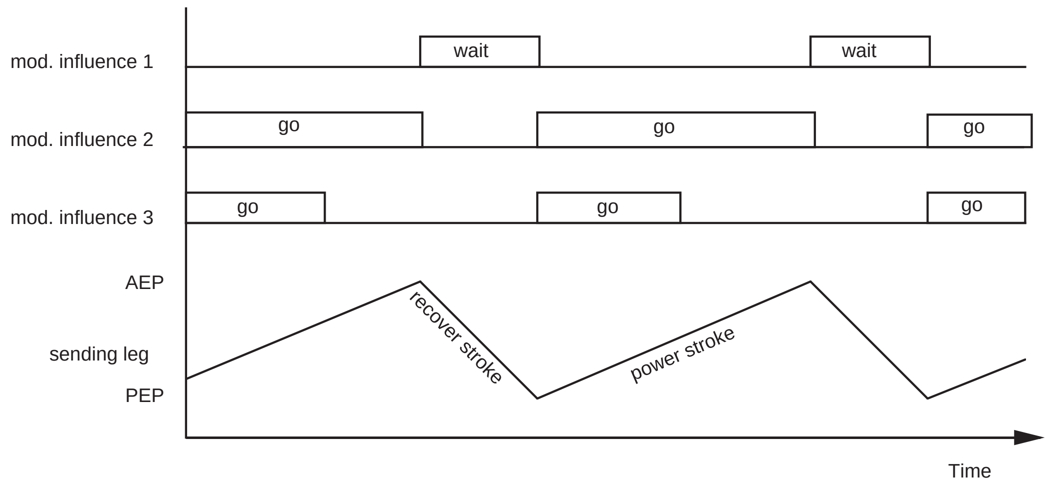


Figure 4.12: The influences used in the Modified Cruse Control gait coordination mechanism.

- Modified *mechanism*$_2$: During the start of the stroke phase send a `go` message in the rostral direction. This enforces back to front metachronal waves along each side of the body.

- Modified *mechanism*$_3$: During the stroke phase send a `go` message in the contralateral direction of the same segment. This enforces 180 degree phasing between adjacent contralateral legs.

The step pattern generator agent, the return stroke agent, and the power stroke agent of each leg were also changed.

- The step pattern generator agent is responsible for transitioning between the power stroke and the return stroke. It sets a `recover` flag `true` when it receives a `go` message, and it clears the `recover` flag when it initiates a recover stroke. It no longer transitions from the power stroke to the return stroke when the leg position is less than or equal to the PEP (except for the special case of the hind legs). Instead, it makes the transition when its `recover` flag is `true` and it stops receiving `wait` messages from the peripheral legs. However, it still transitions from the recover stroke to the power stroke when the leg position is greater than or equal to the AEP.

- The return stroke agent is responsible for lifting and swinging the leg to the starting position of the power stroke. It is activated by the step pattern generator agent. While this agent was active, it exerts modified $mechanism_1$. The AEP is fixed.

- The power stroke agent is responsible for supporting and propelling the body by steadily moving the leg to the PEP. It is activated by the step pattern generator agent. While this agent is active it exerts $mechanism_2$ and $mechanism_3$. The PEP is fixed.

Modified Cruse control is similar in concept to Cruse control. $Mechanism_1$ of both methods effects stability, $mechanism_2$ of both methods effects back to front metachronal waves, and $mechanism_3$ of both methods effects 180 degree phasing of adjacent contralateral legs. In Cruse control, these effects emerge from adjusting the PEP of the legs because swing phase coordination of the legs depends on the time at which the legs reach their PEPs. This is why all legs must retract at the same velocity for Cruse control to work. In contrast, modified Cruse control forces the start of the recover stroke as soon as the adjacent legs are supporting the body and the adjacent posterior leg finishes its recover stroke. In the special case of the hind legs, the return stroke starts when the leg reaches the PEP, all adjacent legs are in the supporting position, and the adjacent contralateral leg has completed its return stroke. Consequently, this approach is less sensitive to retraction velocity and is better suited to Hannibal. The Modified Cruse control approach produces a range of stable gaits as shown in 4.13.

Lesion compensation is implemented by treating the step pattern generator of the lesioned leg as a switchboard for the messages of the peripheral legs (see figure 4.14). An example of this is shown in figure where L2 is the lesioned leg. With L2 gone, L1 and L3 are ipsilaterally adjacent legs. For extra support, L3 is considered contralaterally adjacent to R2 and R3. L1's and L3's `wait` messages are routed through L2 to each other, L3's and R2's `wait` messages are routed through L2 to each other, and L3's `go` message is routed through L2 to L1. Effectively, the influences of the lesioned leg are removed from the network and replaced by the influences of its adjacent legs. Figure 4.15 shows how re-routing the coordination influences through the lesioned leg affects the gait.

The results of the modified Cruse control are mixed. On the positive side, Hannibal's gait coordination is robust to inconsistent retraction velocities,

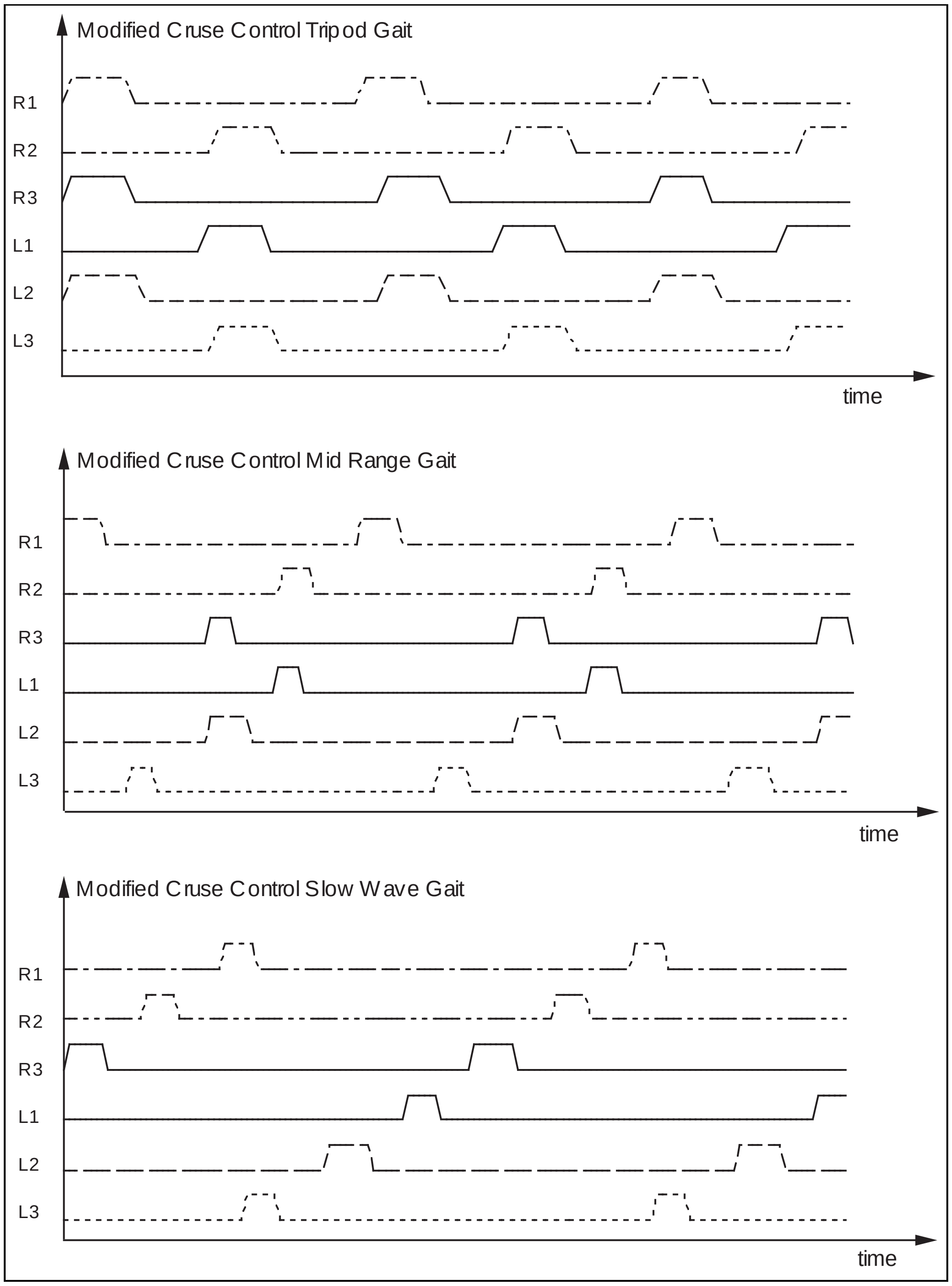


Figure 4.13: Various gaits produced using Modified Cruse Control.

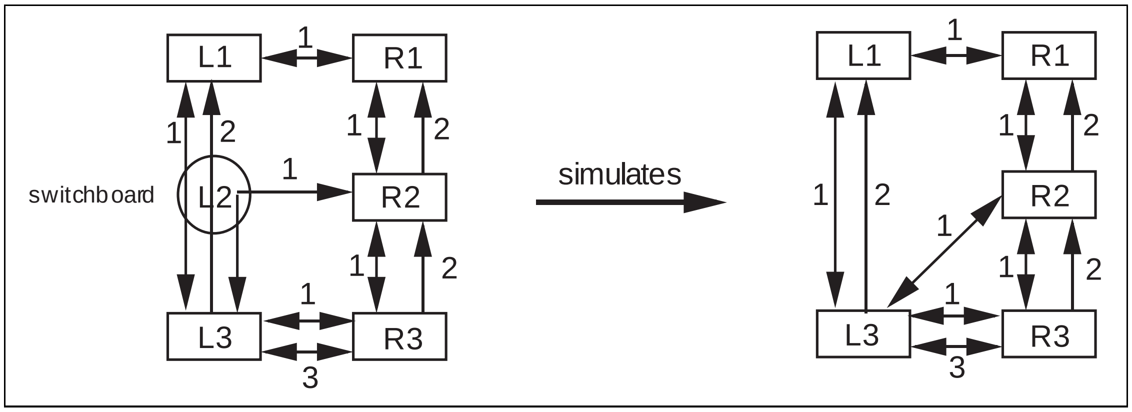


Figure 4.14: By re-routing the leg coordination influences through the lesioned leg, gait coordination was maintained despite the loss of the leg.

it's able to transition between stable metachronal gaits from a slow wave gait through a tripod gait, the gaits are robust to leg perturbations, and the lesion mechanism successfully re-routes messages. On the negative side, mid-range gaits were difficult to reproduce, and poor retraction velocity control presents a different problem. Because Hannibal cannot consistently control the duration of the power stroke, some legs reach the PEP too soon and wait there until it's ok to start the return stroke. For the time period between reaching the PEP and starting the return stroke the leg does not propel the body. Consequently, the leg effectively becomes "dead-weight" that the other supporting legs have to drag along[1].

4.6 Pacemaker Control

To conquer Hannibal's velocity control problem, we included pacemaker (oscillator) agents in the control network. The motivation for the pacemakers is to synchronize the step cycle stages of the legs. The step cycle stages include lift/swing, step/swing, and a sequence of support stages. During the

[1]Hannibal's leg design causes the stroke trajectory to move in a fairly tight arch. The AEP and PEP are chosen to make the stride as long as possible before arching becomes unacceptable. Ergo if the leg continued to retract after reaching the PEP, the leg would be worse than "dead weight" because its contribution significantly propels the body perpendicular to the direction of motion. This causes unacceptable torquing on the body.

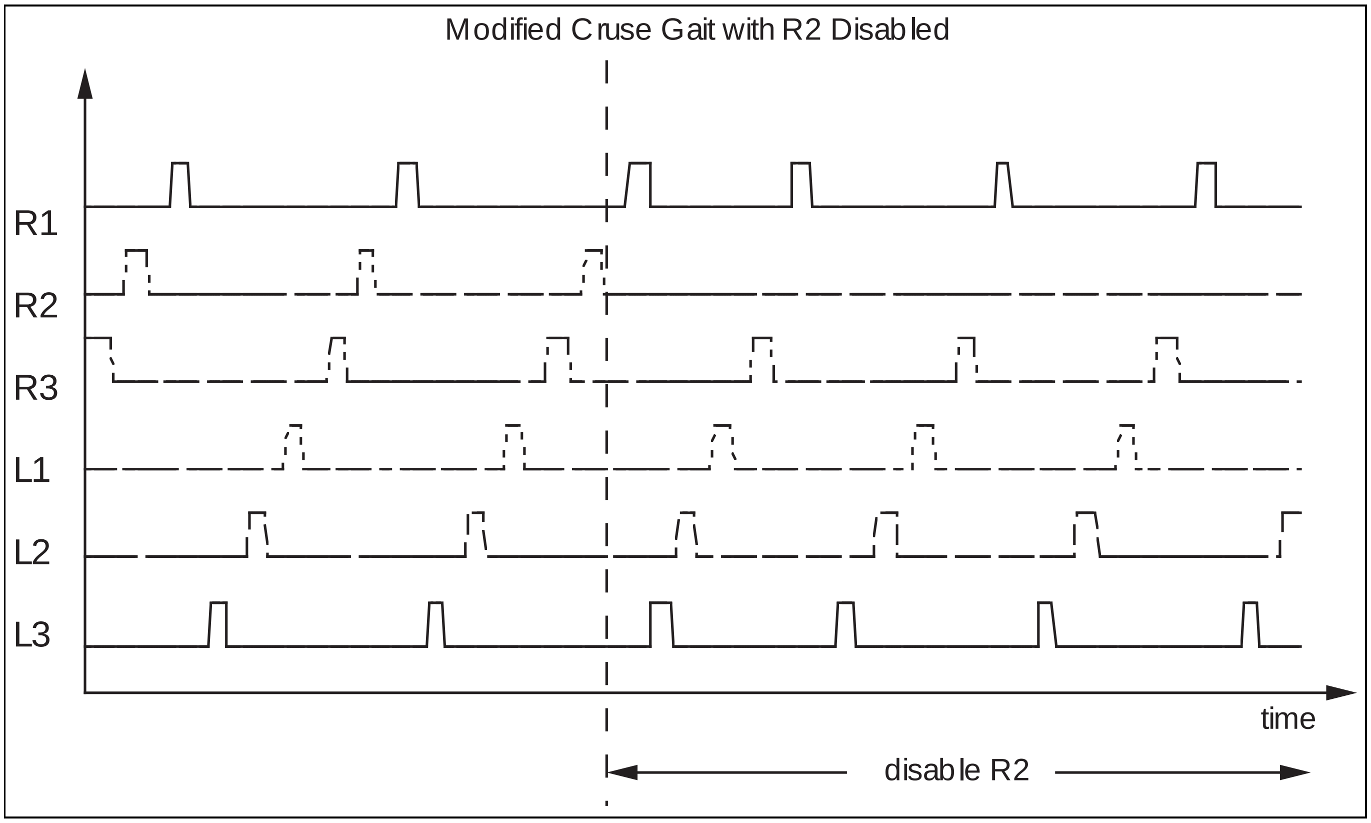


Figure 4.15: Run-time data of the robot's gait before and after the right middle leg (R2) is lesioned. The retraction velocity was set such that the robot performed a slow wave gait. Notice that after leg R2 was removed, leg R1 began its recovery phase immediately after leg R3 finished its recovery phase.

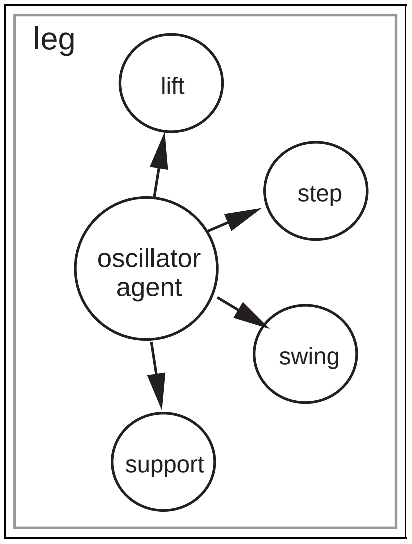


Figure 4.16: Individual leg control circuit for the pacemaker scheme.

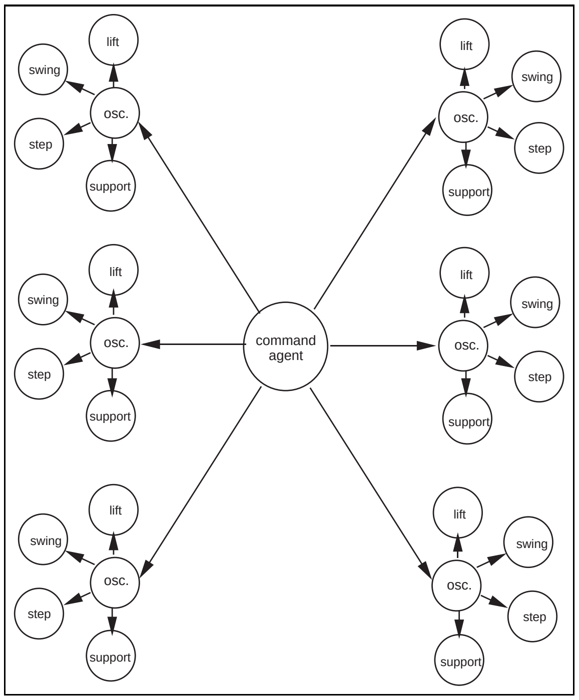


Figure 4.17: Gait coordination circuit for the pacemaker scheme. This circuit is strongly inspired by the work of Pearson.

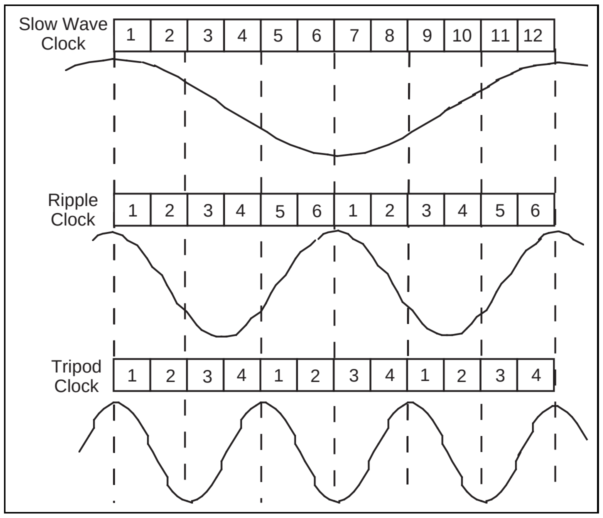


Figure 4.18: Implementation of leg oscillators on Hannibal. Each oscillator is modeled as a clock which cycles through its values at regular time intervals. The peak of the oscillator phase corresponds to `osc-clock = 1`. The period of the clock corresponds to the period of the oscillator.

lift and step stages, the leg moves to the AEP. During each support stage the leg moves an incremental distance towards the PEP. This increment is determined so that the leg reaches the PEP during the last of the series of support stages. Ergo all supporting legs propel the body in synchrony, and the duration of the power stroke is equal for all the legs. The step cycle frequency is adjusted by a command neuron agent. The inclusion of oscillator agents and a command neuron agent were inspired by Pearson's work.

As described by Pearson, the oscillator agent is responsible for generating the cyclic motion of the leg. The oscillator agent excites a lift agent, a swing agent, and a step agent to produce the return stroke, and it excites the support agent to produce the power stroke. Figure 4.16 shows the individual leg control circuit, and figure 4.19 shows the influences sent between these agents to produce the step cycle pattern. The lift, swing, and step agents serve a similar function as the Flexor neurons, and the support agent serves a similar function as the Extensor neurons.

- The oscillator agent: Through studying the neural system of the walking cockroach, Pearson and his collaborators experimentally determined

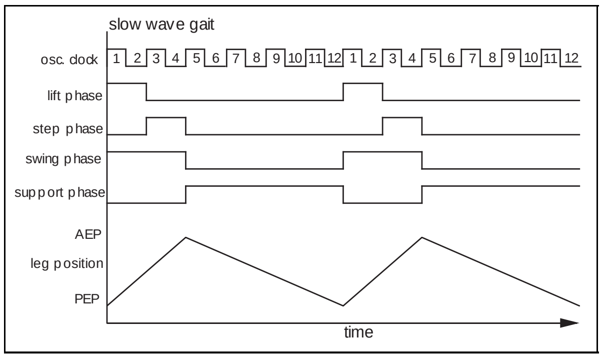


Figure 4.19: The influences used in the pacemaker gait coordination mechanism. The leg lifts and swings for the first clock cycle, steps and swings for the second clock cycle, and supports and propels the body for the remainder of the clock period.

the existence of a pacemaker in each leg. On Hannibal, the oscillator agent is similar to a pacemaker. The oscillator agent is modeled as a clock that cycles through its values at regular time steps (see figure 4.18). In the current implementation, the clock value changes every 0.6 seconds. It generates the cyclic movement of the leg by activating the lift, swing, step, or stroke agents as a function of its clock value.

- The lift agent: While active, this agent commands the up-down actuator to continually lift the leg.

- The step agent: While active, this agent commands the up-down actuator to lower the leg until the leg supports the body.

- The swing agent: While active, this agent commands the protract-retract actuator to move to the AEP.

- The support agent: While active, this agent commands the protract-retract actuator to move an incremental amount towards the PEP each clock cycle. The increment is equal to $S \div N$ where S = abs(PEP - AEP) and N = number of support stages. Consequently the transition from the stroke phase to the swing phase occurs on the clock pulse after the leg reaches the PEP.

The gait coordination network is shown in figure 4.17. The emergent gaits of this network posses similar characteristics to Cruse's, Pearson's, and Wilson's model for insect gaits. These gaits are shown in figure 4.20. Regarding individual leg control:

- The oscillator agent keeps protraction time constant and decreases retraction time as step cycle frequency increases.
- The leg position reaches the AEP when transitioning from the return stroke to the power stroke.
- The leg position reaches the PEP when transitioning from the power stroke to the return stroke.

Concerning the coordination between legs, the oscillator agents are synchronized and initialized such that:

- A wave of protraction runs from posterior to anterior.
- No leg protracts until the leg behind is placed in a supporting position.
- The supporting legs propel the body such that no leg is "dead weight".
- The duration of the power stroke is the same for all legs.
- Contralateral legs of the same segment alternate in phase.
- The intervals between steps of the hind leg and middle leg and the intervals between middle leg and foreleg are constant.
- The interval between the foreleg and hind leg steps varies inversely with frequency.

4.6.1 Gait Behavior

Hannibal changes gait as a function of oscillator frequency. The network responsible for gait transitions is shown in figure 4.21. Changing the number of clocks per step cycle (clocks/cycle) of the oscillator changes the phase between the metachronal waves along each side of the body. This causes different wave gaits to emerge. The transition between gaits is smooth and

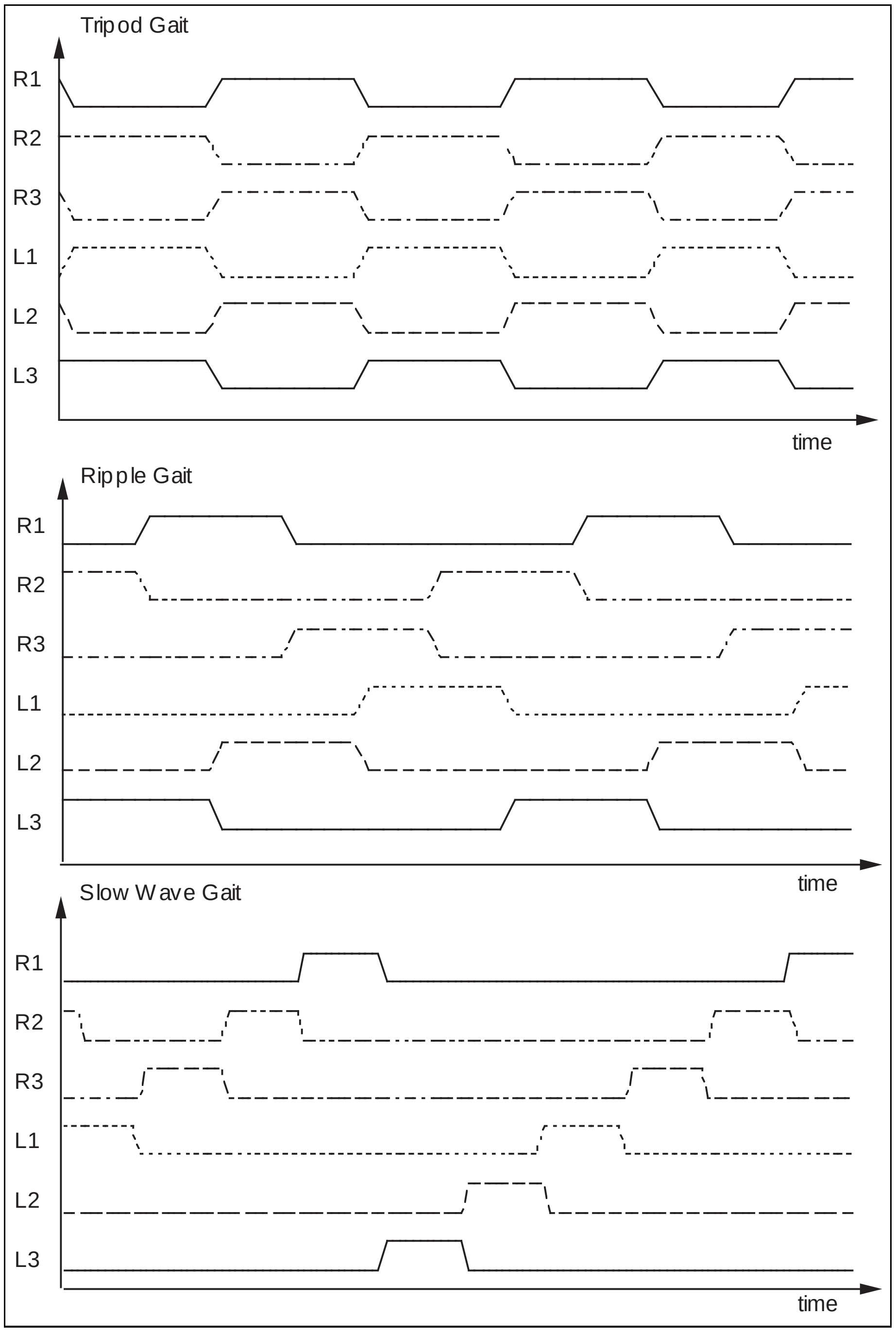


Figure 4.20: Run-time data of various gaits implemented on Hannibal using the pacemaker scheme.

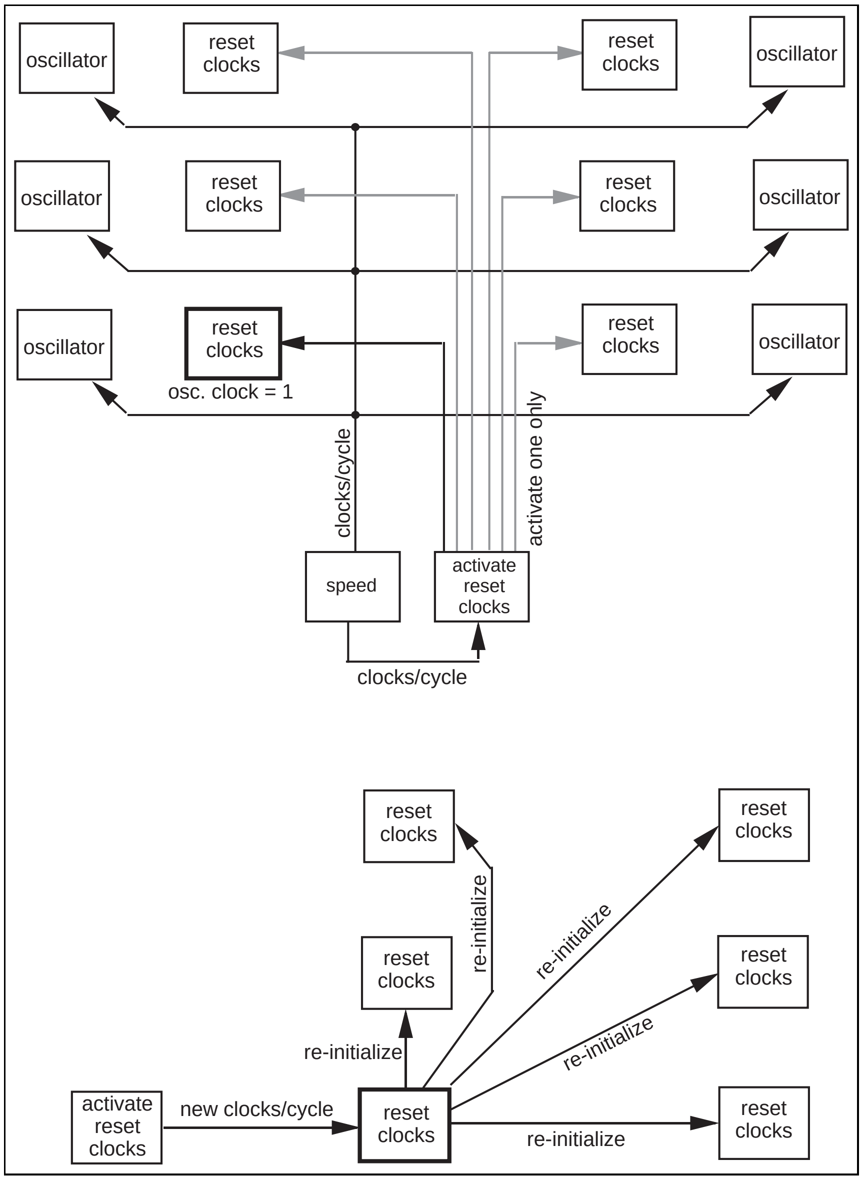


Figure 4.21: The network governing speed transitions. A new speed request corresponds to changing the period of the local oscillators and re-establishing the correct phasing between them. The top figure shows the left rear leg, L3, in the lift phase (`osc-clock = 1`) when the new speed is requested. As a result, the reset clocks agent of L3 is activated. The bottom figure shows the reset clocks agent of L3 re-initializing the oscillator clocks of the other legs given L3's current step cycle phase and the new oscillator period.

immediate. Currently, Hannibal uses three gaits: a slow wave gait, a ripple gait, and a tripod gait. Figure 4.22 shows the robot transitioning between various gaits. Wilson (1966) reports the slow wave gait is the slowest gait and the tripod gait is the fastest gait observed in insects. These gaits are implemented by a global speed agent and local reset clocks agents. The global speed agent is similar to the command neuron(s) of Pearson's model. The reset clocks agents are similar to the coupling signals sent between oscillators in Pearson's model.

- The speed agent: Whenever Hannibal wants to change speed, this global agent sends a new clocks/cycle value to all the oscillators. Higher level agents command speed through this agent.
- The activate-reset-clocks agent: If a new clocks/cycle value is sent to the oscillators, this agent finds the furthest posterior leg in the lift stage and activates the local reset-clocks agent of that leg.
- The reset-clocks agent: Pearson found the local pacemakers of the legs send coupling signals between each other to coordinate the step pattern generators. Hannibal implements a similar mechanism when changing speed. Because a different clocks/cycle value is sent to the oscillators whenever the robot changes speed, the values of the oscillators must be re-coordinated to maintain a proper gait. When active, the reset-clocks agent re-initializes the oscillators of the other legs with respect to its `osc-clock = 1`.

4.6.2 Turning Behavior

Hannibal turns by adjusting two parameters: turn-direction and turn-sharpness. The turn-direction parameter determines whether Hannibal turns to the right, to the left, or makes no turn. The turn-sharpness parameter determines the radius of the turn. Figure 4.23 illustrates the network that gives the robot turning capabilities.

- Global turn agent: keeps all the legs in agreement as to what kind of turn to make by telling the leg turn agents the turn-direction and the turn-sharpness. Higher level agents command turn-direction and turn-sharpness through this agent.

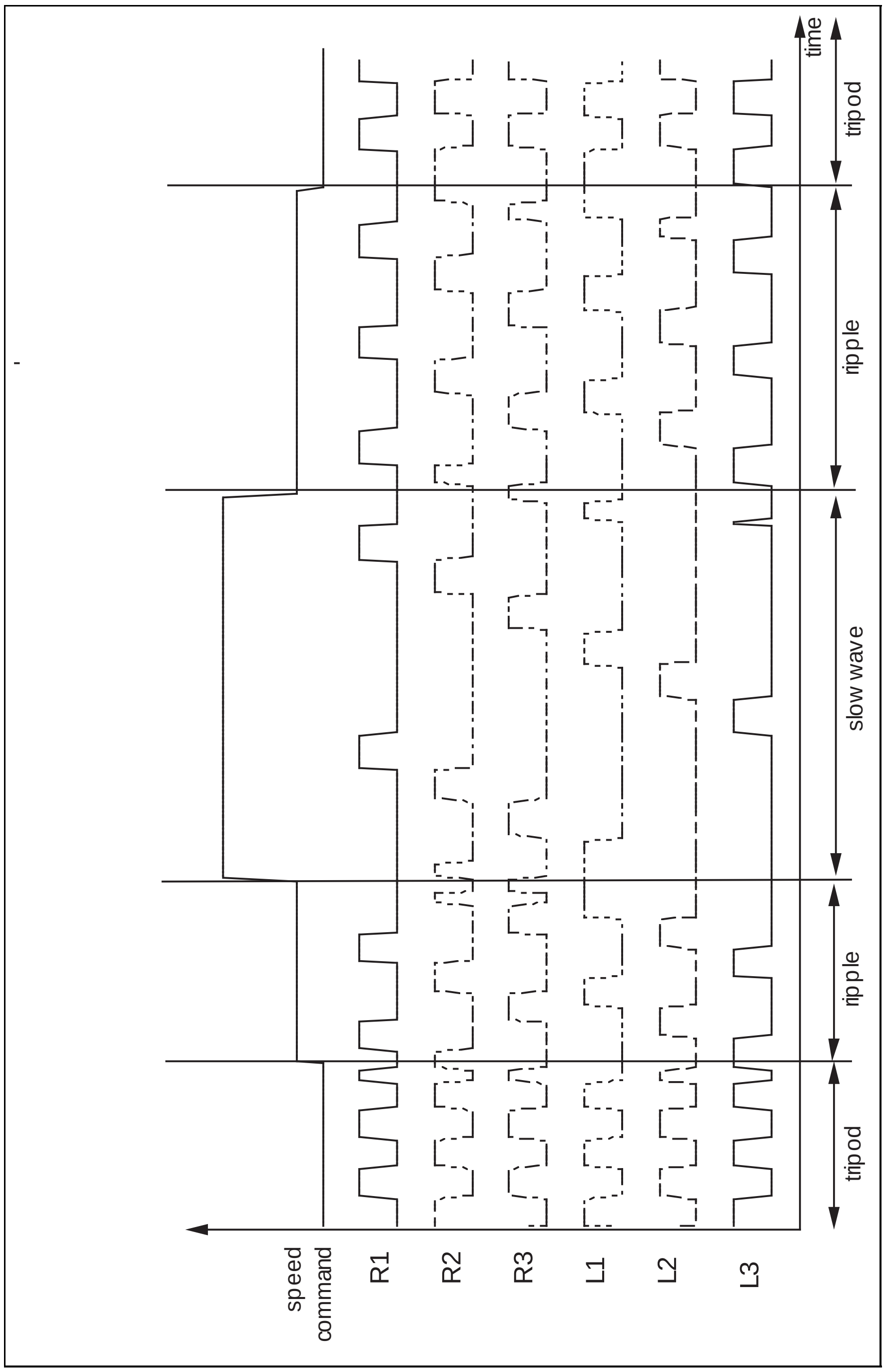


Figure 4.22: Run-time data of the robot transitioning between gaits as a function of oscillator period.

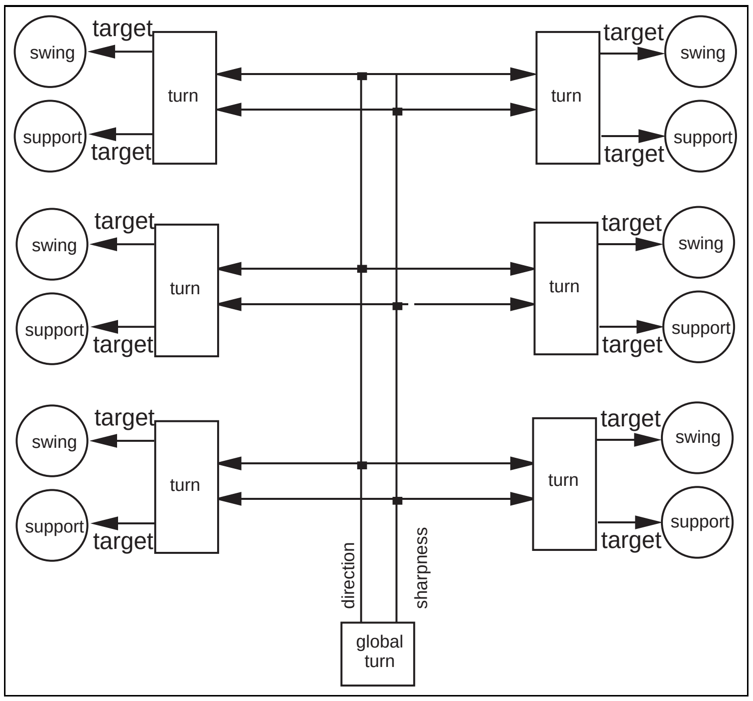


Figure 4.23: The network governing turning. Turning is achieved by adjusting the stride length of the legs along one side of the body. The local turn agents receive rotation and sharpness values from the global turn agent. According to these parameters, the local turn agents compute new AEP and PEP values for their leg. To execute the turn, the local turn agents send these new target values to their leg's swing agent and the support agent.

- Local turn agents: implement the turn by changing the PEP and AEP according to the turn-direction and turn-sharpness parameters. They receive the return stroke target and power stroke target values from the local direction agents and modify them as necessary to perform the turn. If the turn-direction is `straight` the AEP and PEP are left alone. If turn-direction is `right` the turn agents on the right side of the body modify their AEP and PEP values by the turn-sharpness parameter, and the turn agents on the left side of the body leave the AEP and PEP values alone. If turn-direction is `left` the turn agents of the legs on the left side of the body modify their AEP and PEP by the turn-sharpness parameter, and the turn agents on the right side of the body leave the AEP and PEP values alone. To modify the AEP and PEP, the local turn agents add the sharpness value to the PEP and subtract the sharpness value from the AEP. The turn-sharpness value can vary the radius of curvature from a gentle arch to turning in place. The turn agent of each leg sends the resulting return stroke target value to the swing agent and the resulting power stroke target value to the support agent.

4.6.3 Direction of Travel Behavior

Hannibal changes direction by swapping the swing phase and support phase target positions between the PEP and AEP. To walk forwards, the return stroke targets the AEP and the power stroke targets the PEP. To walk backwards, the return stroke targets the PEP and the power stroke targets the AEP. Figure 4.24 illustrates the network that gives the robot the ability to change its direction of travel.

- The global direction agent: keeps all the legs in agreement of the direction of travel by sending the local direction agents a direction parameter (forward or backwards). Higher level agents command direction of travel through this agent.

- The local direction agents: implement the commanded direction of travel by setting the target positions of the power stroke and return stroke according to the direction parameter. For each leg, its direction agent sends the swing phase target position to its swing agent and the

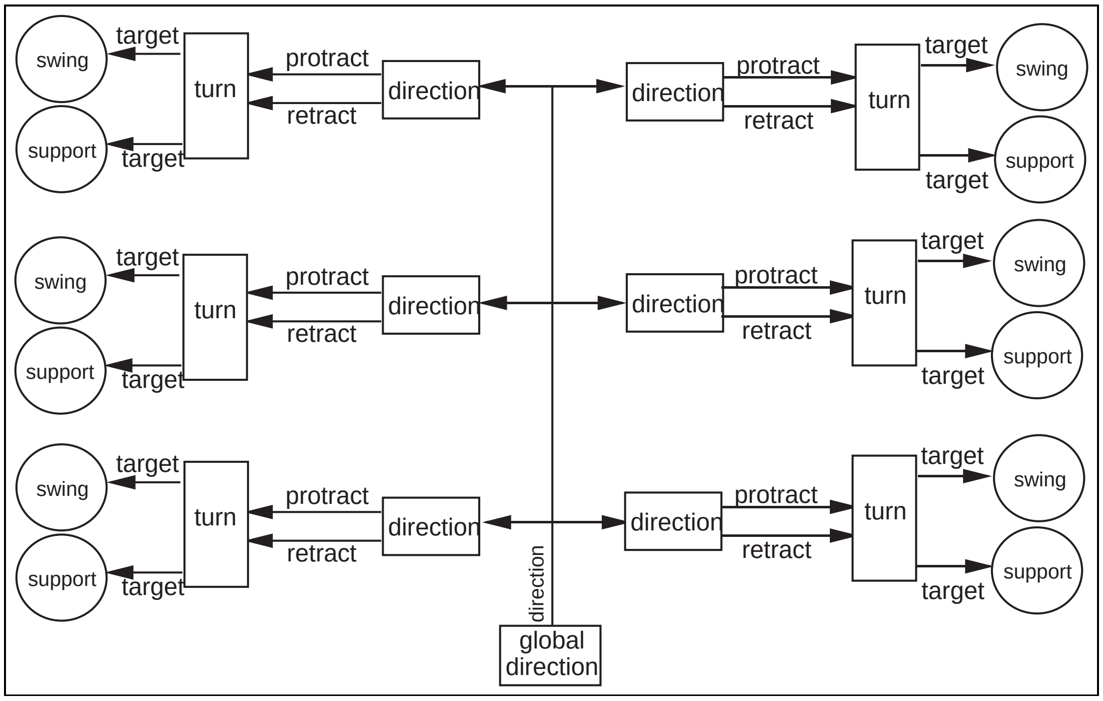


Figure 4.24: The network governing the direction of travel. Changing the direction of travel entails exchanging the target values of the swing agent and support agent. The local direction agents receive the commanded direction value from the global direction agent. Based on this value, the local direction agents send the swing target position and stroke target position to the local turn agents. If the robot does not want to turn, the local turn agents pass these values to the swing agent and support agent. Otherwise, the local turn agents modify these values so that the robot may change direction and turn simultaneously.

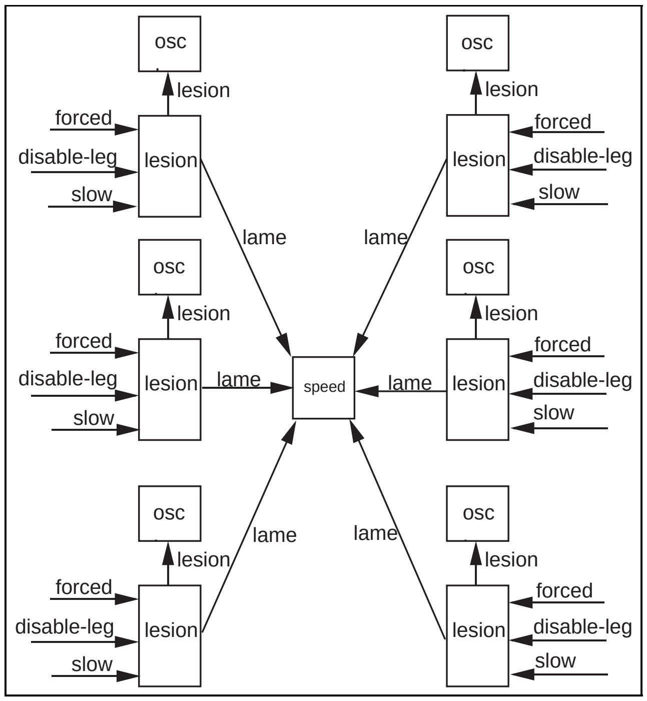


Figure 4.25: This network enables the robot to walk in a stable fashion despite leg damage. When the lesion agent of a leg is active, the lesion agent sends a message to the global speed behavior to make the robot adopt the slow wave gait. This gait is stable even with fewer than six legs are working. In addition, the lesion agent places the motors of its leg in brake-mode and removes the influences of its from the rest of the network.

support phase target position to its support agent (through the local turn agent).

4.6.4 Lesion Compensation Behavior

If a leg becomes in-operable, Hannibal changes gait so that it can walk in a stable fashion with fewer legs. The network responsible for this is shown in figure 4.25. Experimental findings regarding the effect of lesions on cockroach gaits is presented in (Wilson 1966). He reports when the two middle legs of a cockroach are removed, the animal resorts to a slower gait where a sufficient number of legs are supporting the body at any given time. If the middle two legs of a cockroach are removed the slow wave gait is still stable, but the tripod gait and ripple gait are unstable. Consequently, the cockroach adopts a slow wave gait. The slow wave gait remains stable in the event of losing any single leg or losing both middle legs.

- The lesion agent: Each leg has a lesion agent that becomes active if a leg is in-operable (chapter 6 discusses how the robot decides when a leg is useless). The lesion agent sends a message to the speed agent to evoke the slowest wave gait. The slow wave gait is stable with the loss of any single leg or the loss of the two middle legs. It also disables the leg by putting all its motors in brake mode.

4.7 Performance

4.7.1 Definition of Terms and Stability Formulas

The following definitions and theorems are presented in (Song & Waldron 1989). Most of this work was established by McGhee and his co-workers.

Definitions

For the following definitions, the leg number of a 2n-legged animal is assigned as 1,3,5,...,2n-1 on the left side and 2,4,6,...,2n on the right side from the front to the rear.

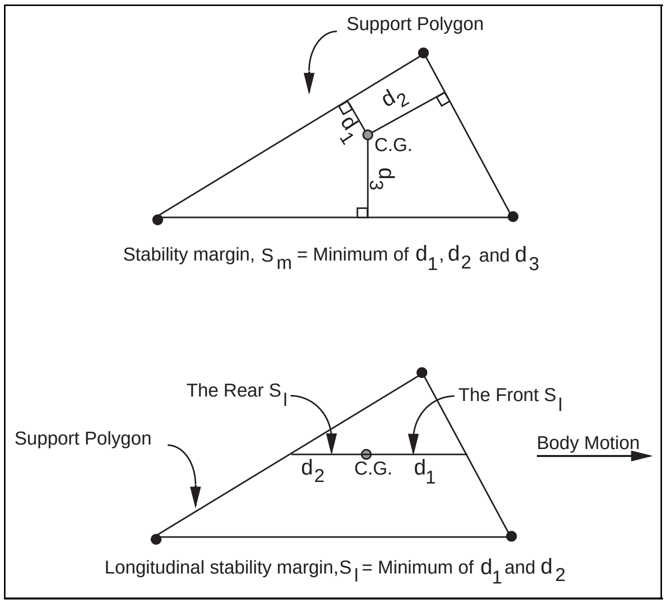
1. The *cycle time, T*, is the time for a complete cycle of leg locomotion of a periodic gait.

2. The *duty factor* β_i, is the time fraction of a cycle time in which leg i is in the support phase. $\beta_i = \frac{t_i}{c_i}$ where t_i is the time of support phase of leg i and c_i is the cycle time of leg i.

3. The leg *stroke*, R, is the distance through which the foot is translated relative to the body during the support phase.

4. The *stroke pitch*, P, is the distance between the centers of strokes of the adjacent legs on one side.

5. A *regular gait* is a gait with the same duty factor for all legs.



Support Polygon
d2
d1
C.G.
d3
Stability margin, Sm = Minimum of d1, d2 and d3
The Rear Sl
The Front Sl
Support Polygon
Body Motion
d2
C.G.
d1
Longitudinal stability margin, Sl = Minimum of d1 and d2

Figure 4.26: Graphical depiction of the stability margin and longitudinal stability margin.

6. A gait is *symmetric* if the motion of the legs of any right-left pair is exactly half a cycle out of phase.

7. A *wave gait* is a regular and symmetric gait where the placing of each foot runs from the rear leg to the front leg along either side of the body as a wave, and each pair of legs is 180 degrees out of phase.

8. A *support pattern (or support polygon)* of an animal or a walking machine is a two dimensional point set in a horizontal plane consisting of the convex hull of the vertical projection of all foot points in support phase. The contact between foot and ground is idealized to a point contact without slip. In a real, distributed foot contact, the contact point can be interpreted as the center of pressure.

9. The *stability margin*, S_m, is the shortest distance of the vertical projection of center of gravity to the boundaries of the support pattern in the horizontal plane. See figure 4.26.

10. The *front stability margin* and the *rear stability margin* are the distances from the vertical projection of the center of gravity to the front and rear boundaries of the support pattern, respectively, as measured in the direction of motion. The *longitudinal stability margin*, S_l, is the shorter of these two. See figure 4.26.

11. The *longitudinal gait stability margin*, S, or the *gait stability margin* in brief, for a periodic gait, G, is the minimum of S_l over an entire cycle of locomotion. A gait is *statically stable* if $S \geq 0$. Otherwise it is *statically unstable*.

Stability Formulas

- For a 2n-legged wave gait with a duty factor in the range $1/2 \leq \beta < 1$, the longitudinal gait stability margin can be determined from the following equation where P is the pitch, R is the stroke and

  $$R_b = (\beta/(3\beta - 2)) \times P \tag{4.3}$$

  If $1/2 \leq \beta$, or if $\beta > 2/3$ and $R \leq R_b$,

  $$S_1 = (n/2 - 1) \times P + (1 - 3/(4\beta)) \times R \tag{4.4}$$

  If $\beta > 2/3$ and $R > R_b$,

  $$S_2 = (n/2 - 1/2) \times P + (1/(4\beta) - 1/2) \times R \tag{4.5}$$

- For an 2n-legged backward wave gait, the gait stability margin is

  $$S = (n/2 - 1) \times P - R/(4\beta) \tag{4.6}$$

  for $1/2 \leq \beta < 1$.

4.7.2 Stability Performance

To measure Hannibal's stability we computed its gait stability margin for a variety of gaits. The gait stability margin is described in definition 11. We used equation 4.4 to compute the gait stability margin of the slow, ripple, and

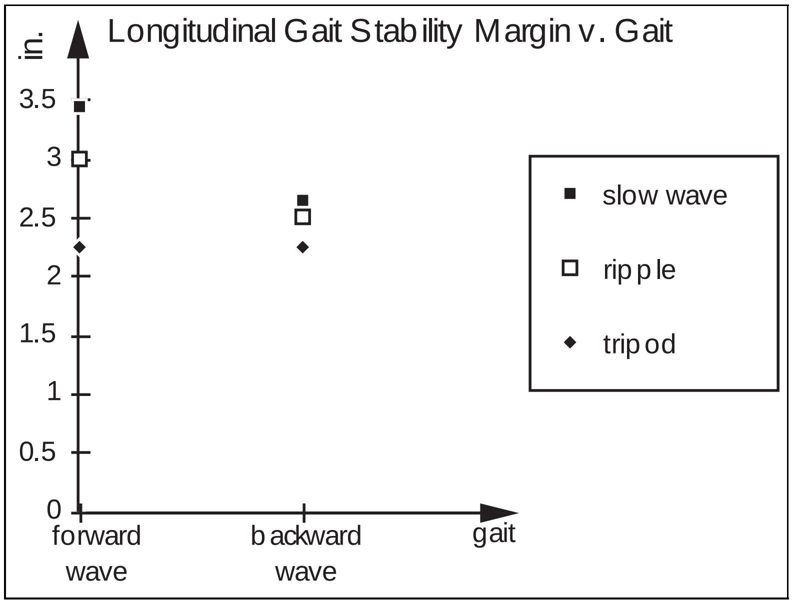


Figure 4.27: Hannibal's gait stability margin for various gaits.

tripod wave gaits. A positive gait stability margin corresponds to a stable gait (where a larger magnitude indicates greater stability). A negative gait stability margin corresponds to an unstable gait (where a larger magnitude indicates less stability).

Figure 4.27 presents Hannibal's gait stability margin for the three types of wave gaits. Not surprisingly, all gaits are stable. In fact, it has been determined that the family of wave gaits provides the optimum stability for hexapods (Song & Waldron 1989). The figure shows a trend where slower gaits (gaits with a larger duty factor) have a greater gait stability margin than faster gaits (gaits with a smaller duty factor).

4.7.3 Speed Performance

To quantify Hannibal's speed, we measured the time required for Hannibal to walk 3 feet. We conducted this speed test for the slow wave gait, the ripple gait, and the tripod gait. The speed test for each gait consisted of ten trials. During the trials we adjusted the spine DOF to see if there was any effect on the robot's speed. By adjusting the spine DOF we can make the

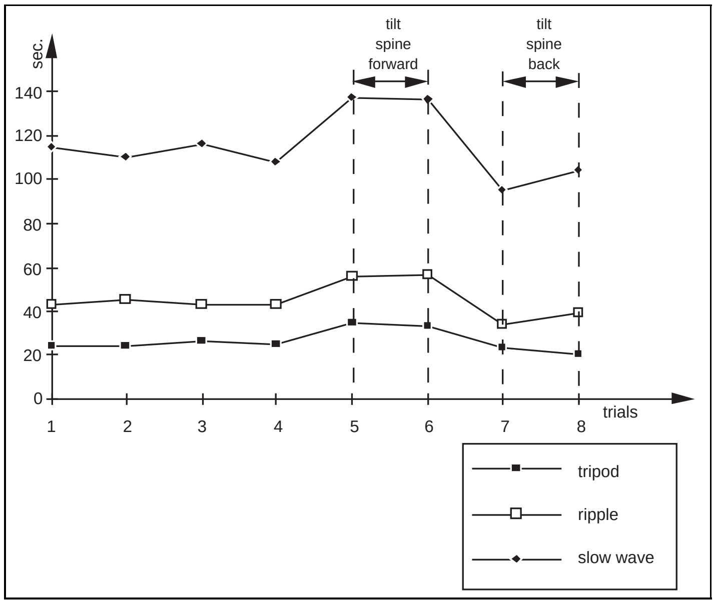


Figure 4.28: Speed performance of Hannibal.

robot lean forward or backward.

The results of the speed tests are shown in figure 4.28. The robot's feet slip as it walks, so not surprisingly, the measured speed is slower than the theoretical speed. Adjusting the spine DOF to the lean-forward position increased the robot's speed. In this configuration, gravity assists the swing actuator advance the leg forward which effectively lengthens the stride. The lean-forward speed is fairly close to the theoretical speed. Adjusting the spine DOF to the lean-back position caused the robot to walk slower. In this configuration, gravity acts against the actuator as the actuator swings the leg forward.

4.7.4 Turning Performance

To quantify the relation between the turn sharpness parameter value to the actual turn sharpness, we measured how far the rear of the robot moves until the robot turns ninety degrees. The rear of the robot traces a quarter of a circle (approximately) when it completes its path; hence we are measuring the length of this arc. We conducted this test for a variety of turn sharpness

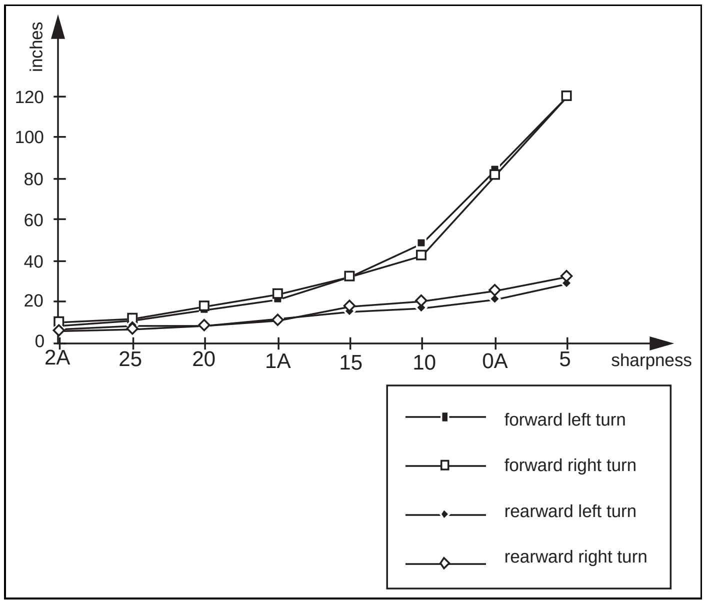


Figure 4.29: Relation of turn sharpness to sharpness parameter.

parameter values. `Turn sharpness = 2A` corresponds to turning in place and `turn sharpness = 0` corresponds to walking straight. The test was performed for clockwise and counter-clockwise turning directions.

The results of the turn sharpness tests are shown in figure 4.29. As expected, the actual sharpness of the turn became greater as the turn sharpness parameter value increased. The turn sharpness for a clockwise rotation is comparable to the turn sharpness for a counter-clockwise rotation.

4.7.5 Directional Performance

We computed Hannibal's gait stability margin when it walks backward for the three different gaits using equation 4.6. The results are presented in figure 4.27. Again there is a trend that as the duty factor increases the gait stability margin increases. It is interesting to note that Hannibal actually performs a different gait when it walks backward - it uses a backward wave gait. For the backward gait the metachronal waves progress against the direction of travel, whereas for the wave gait the metachronal waves progress along the direction of travel. The backward wave gaits are less stable than the wave

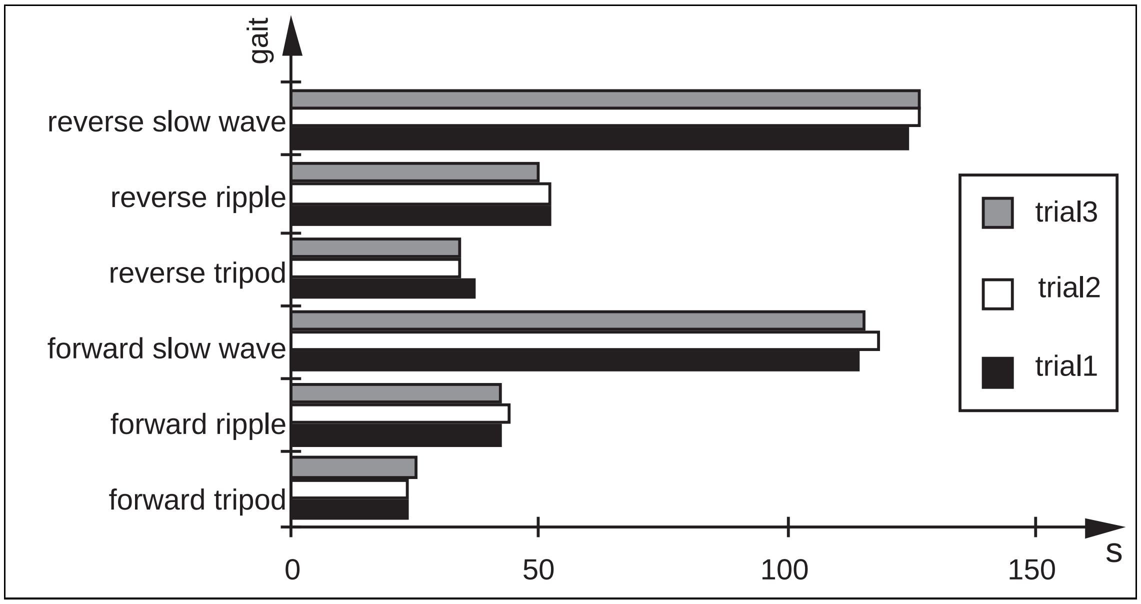


Figure 4.30: Comparison of times required to walk three feet either backwards or forwards.

gaits (with the exception of the tripod gait). The reason for this is as follows: the minimum stability margin (for both the wave gait and for the backward wave gait) occurs when a rear foot is lifted. Hence the stability margin is determined by the position of the supporting middle leg directly ahead of the lifted rear leg (we call it the "key" leg). For a wave gait, the key leg is at its most backward position where it provides the maximum stability margin. On the contrary, for the backward wave gait, the key leg is at its most forward position where it provides the minimum stability margin.

We measured the relationship between the turn sharpness parameter value to actual turn sharpness as the robot walks backwards. We performed the same set of tests as for the forward case. The results are shown in figure 4.29. Again, the turn sharpness for a clockwise rotation is comparable to the turn sharpness for a counter-clockwise rotation. It is interesting that the robot turns much more sharply when walking backwards than when walking forwards. We hypothesize the decrease in stability of the backward wave gait counteracts the robot's momentum along its direction of travel. As a result, the robot covers a shorter distance along the direction of travel per step.

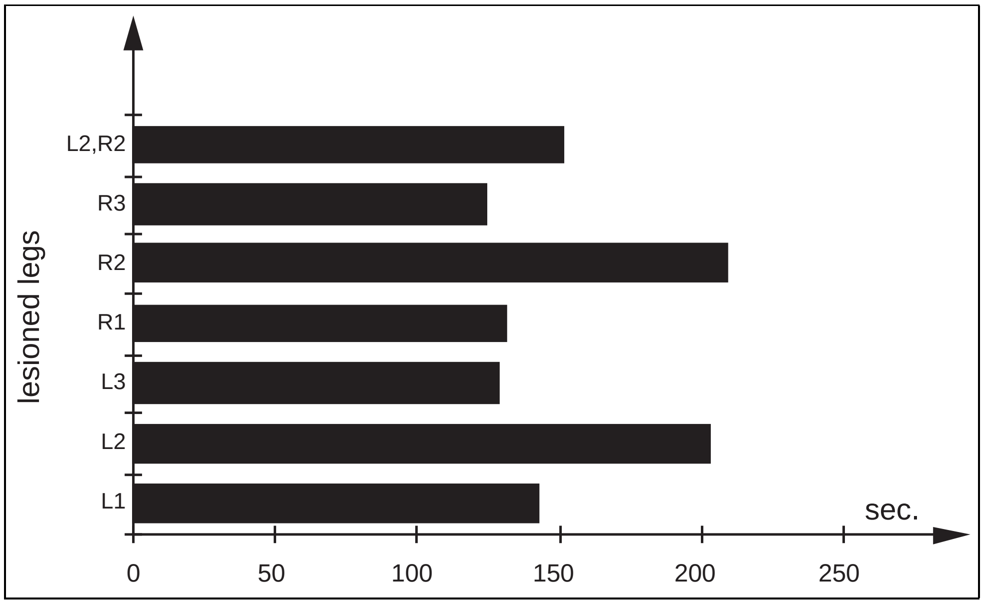


Figure 4.31: Time taken to walk three feet with various legs disabled.

Consequently, the distance it travels to complete a 90 degree turn is less.

We measured the time for the robot to walk backwards for three feet. The results are shown in figure 4.30. The robot walks slower backwards than it does forwards. We hypothesize the decrease in stability of the backward wave gait counteracts the robot's momentum along its direction of travel. This could cause the robot to walk slower when using the backward wave gait.

4.7.6 Lesion Compensation Performance

We computed the gait the stability margins for the following cases: the robot has an outer leg removed, a middle leg removed, or both middle legs removed. The gait stability margins were computed using geometry given the robot uses a slow wave gait when a leg is lesioned. Insects can locomote with stability when a leg is cut off provided it walks with the slow wave gait. Hannibal uses the same strategy to maintain stability given the loss of a leg.

The robot is on the boarder line of stability if the lesioned leg is completely removed. The reason is that the remaining legs are not long enough to provide more compensatory support for the lesioned leg. To increase the stability of the robot if a leg is removed would require changing physical dimensions of either the robot's body, the legs, or both. To get around this problem, Hannibal places the motors of the lesioned leg in brake-mode and allows the foot of the lesioned leg to rest on the ground. By doing so, the lesioned leg provides a minimal amount of support–enough to keep the robot stable.

We determined the speed of the robot as it walks with any single leg disabled or with both middle legs disabled. The test involved measuring the time required for the robot to walk three feet. The results are shown in figure 4.31. Not surprisingly, the robot was significantly slower than when it used the slow wave gait with all legs intact.

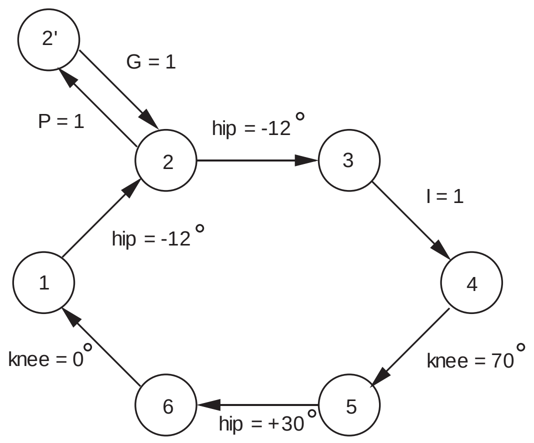
4.8 Legged Locomotion

Research in legged locomotion is divided into several categories. Significant progress has been made in the areas of dynamic legged locomotion for robots (Raibert & Hodgins 1993) as well as for insects (Full 1993). Hannibal falls under the category of statically stable locomotion. This section compares and contrasts Hannibal's flat terrain locomotion with that of other computer controlled, statically stable walkers. Many of the presented approaches are inspired by biology. We address rough terrain locomotion in chapter 5.

4.8.1 Phoney Pony

Frank and McGhee's group at USC built the Phoney Pony in the mid '60s(McGhee 1976). The Phoney pony was a quadruped with 2 degree of freedom (DOF) legs. Both the hip joint and knee joint were driven through a worm-gear, and each joint could be in one of 3 states: forward rotation, rearward rotation, and locked. The phoney pony was the first fully autonomous walker, and the first walker to use a digital computer to control electronic linkages.

Joint coordination of the Phoney Pony was implemented using *finite state control* which is a biologically motivated control scheme proposed by McGhee and Tomovic in 1966. In finite state control, each leg has an identical finite-state control circuit that generates the cyclic movement of the leg. Each control state of the leg cycle has corresponding hip rotation and knee rotation values. As shown in figure 4.32, transitions between states depend on var-



P pause signal
G synchronization signal
I interlock signal

control state	Hip rotation	Knee Rotation
1	rearward	locked
2	rearward	locked
2'	locked	locked
3	locked	locked
4	forward	rearward
5	forward	locked
6	locked	forward

Figure 4.32: Locomotion controller for the Phoney Pony. Adapted from (McGhee 1976).

ious conditions. Contralateral synchronization is implemented by the state transitions $2 \rightarrow 2'$ when the pause signal, P, is set to one, and $2' \rightarrow 2$ when the synchronization signal, G, is set to 1. The P and G signals establish the correct hip rotation difference between contralateral legs during the support phase. The transition $3 \rightarrow 4$ when the interlock signal, I, is set to 1 maintains stability by keeping three legs in the support phase at any time. Joint angle sensors or limit switches initiate state transitions $1 \rightarrow 2, 2 \rightarrow 3, 4 \rightarrow 5, 5 \rightarrow 6$, and $6 \rightarrow 1$. With this approach, the Phoney Pony was programmed to perform a crawl gait. With a slightly modified controller, a trot gait was also implemented.

The control scheme implemented on the Phoney Pony is similar to the control scheme of Hannibal in several respects. Both controllers are biologically motivated. Both approaches implement control with finite state machines (although the augmented finite state machines on Hannibal are more sophisticated). In both cases, the legs operate as monostable oscillators. For the Phoney Pony the cyclic motions were generated by a finite state circuit on each leg. For Hannibal, the oscillator agent on each leg generates the cyclic motion. In both schemes the legs send synchronization signals to each other to maintain proper leg sequencing. For the Phoney Pony, the synchronization signal was based on the difference between hip rotation of contralateral legs. For Hannibal, agents local to the legs maintain the proper phasing between leg oscillators.

4.8.2 OSU Hexapod

In the mid '70s, McGhee and colleagues built the OSU Hexapod at Ohio State (McGhee & Iswandi 1979). The walker was roughly the size of a pony and had six three DOF legs. The OSU Hexapod was a experimental means of follow up on McGhee's earlier theoretical findings on the combinatorics and selection of gait (McGhee 1976). The Hexapod was eventually programmed to negotiate simple obstacles, but we will only discuss basic locomotion capabilities here.

Basic locomotion of the OSU Hexapod was implemented in a hierarchical structure. Motion planning was the responsibility of the higher level, and joint angle coordination was the responsibility of the lower level. The motion planner chose leg placements such that the *gait stability margin* is maximized. This requires that the center of mass of the walker remain in the polygon of support while the legs sequence through the gait (Song & Waldron 1989). For periodic support state sequences, the family of wave gaits (the same gaits observed in insects) optimizes the longitudinal stability margin. Joint angle coordination was implemented with *model reference control* (also known as *algorithmic control*). With this approach, the OSU Hexapod was programmed to walk with a number of gaits, turn, and walk sideways. In this control scheme, the computer's primary task was to solve the inverse kinematic equations for the leg positions chosen by the planner; this approach was computationally extensive. The solutions to these equations were joint angle commands for the 18 electric motors driving the legs. Servo control is a significant capability the OSU Hexapod had, but the Phoney Pony did not.

Locomotion control of the OSU Hexapod and Hannibal are dramatically different, but the goal of stable locomotion is the same. The OSU Hexapod's controller was designed in a top-down approach, whereas Hannibal's controller was designed from the bottom up. The control scheme of the OSU Hexpod is hierarchical and functionally decomposed, whereas the control scheme of Hannibal is distributed and decomposed into task achieving agents. For example, regarding the OSU Hexapod controller, the higher level first derives a set of wave gait foot placements based on maximizing the longitudinal stability margin; then the lower level converts these planned foot placements into joint angle commands. In contrast, Hannibal's distributed control network is designed so that different wave gaits emerge from the interaction of the speed agent, reset clocks agents, and oscillator agents. Joint angle coordination is implemented differently as well. The OSU Hexapod performs inverse kinematics on foot placements to produce leg joint angles. In contrast, Hannibal's lift, swing, step, and support agents determine the command joint angles using either the default values of these agents or the incoming values from other agents (such as Hannibal's turn and direction agents). Overall, the OSU Hexapod's control is computationally expensive compared to Hannibal's control.

4.8.3 SSA Hexapod

In the early 80s, Marc Donner implemented distributed locomotion control on the SSA Hexapod (Donner 1987). The SSA hexapod was built at CMU by Dr. Ivan Sutherland of Sutherland, Sproull, and Associates. Its three

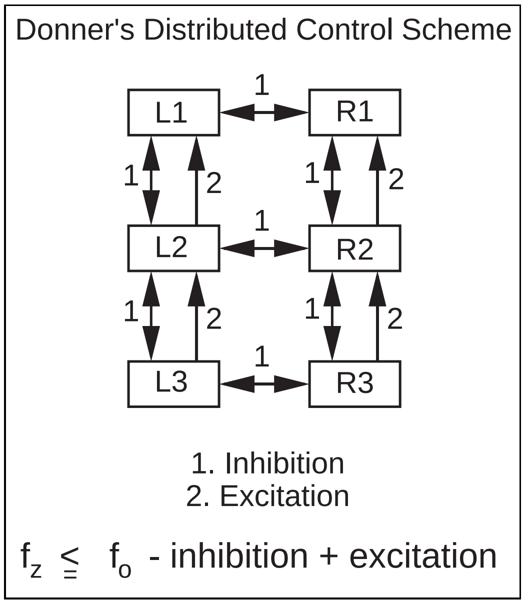

Donner's Distributed Control Scheme
L1
R1
L2
R2
L3
R3
1
2
1. Inhibition
2. Excitation
$f_z \leq f_o$ - inhibition + excitation

Figure 4.33: Locomotion controller for the SSA Hexapod. Adapted from (Donner 1987).

DOF legs were hydraulically actuated and had position and force sensing for each DOF (among other sensors). Donner's walking algorithm demonstrated locality of control with no global information and little computation. He implemented the walking algorithm in OWL, a specially designed language he wrote for real-time performance and concurrency control.

Donner's walking algorithm was inspired by insect locomotion. Individual leg control was implemented by a separate and mostly autonomous process responsible for generating cyclic stepping movements. Leg coordination was implemented by an excitation mechanism and an inhibition mechanism. Each leg process sends/receives these influences to/from neighboring leg processes as shown in figure 4.33. The excitation and inhibition influences effect when a leg makes the transition from the support phase to the swing phase. The transition occurs when the leg has made more than half a stride and the vertical force, f_z, satisfies the relation $f_z \leq f_o - inhibition + excitation$. The excitation influences sent ipsilaterally encourage back to front metachronal waves. A leg sends an excitation number to its frontward neighbor when it finishes its swing phase, and removes this excitation number when it begins its swing phase. Consequently, the excitation encourages an adjacent anterior leg to begin its swing phase when the adjacent posterior leg is in the support phase. The excitation sent contralaterally between the rear legs encourages 180 degree phasing between pairs legs on opposite sides of the body. This is accomplished by having each rear leg excite its crosswise neighbor when it reaches the halfway point in its drive stroke. The inhibition mechanism encourages stability. Each leg that is in its swing phase sends an inhibition signal to its neighbors thereby discouraging them from entering their swing phase. The leg removes this inhibition when it enters its support phase. The SSA hexapod walked as well as the physical constraints of the machine allowed using this approach. The approach also permitted the machine to locomote when a middle leg was removed.

Hannibal's and the SSA's controller are similar in several respects. Regarding global comparisons, both controllers were implemented in a language specially designed to run concurrent processes in real-time. Both controllers demonstrate locality of control with no global information and little computation. The locomotion processes of each machine are executed concurrently and are implemented as finite state machine circuits. Concerning specific comparisons, it is interesting that Donner's controller is almost identical to Hannibal's Modified Cruse controller given they were derived independently from different models of insect locomotion. However, the mechanisms implemented by the Modified Cruse controller enforce certain relationships between the legs, whereas the mechanisms implemented by Donner's controller encourage these relationships.

4.8.4 Genghis

Genghis is a small hexapod (35cm long, 25cm across) built in the mid '80s by the Mobile Robotics Group at MIT (Brooks 1989). Genghis is the predecessor to Hannibal; it has six legs, each with 2 DOF (lift and shoulder). Brooks programmed Genghis to traverse rough terrain and follow people, and later Maes programmed Genghis to learn a tripod gait. This section covers the implementation of flat terrain locomotion on Genghis (its rough terrain abilities is addressed in the next chapter 5).

Basic walking on flat terrain is implemented by 32 AFSM's (see figure 4.34). Two of these AFSM's are for global coordination:

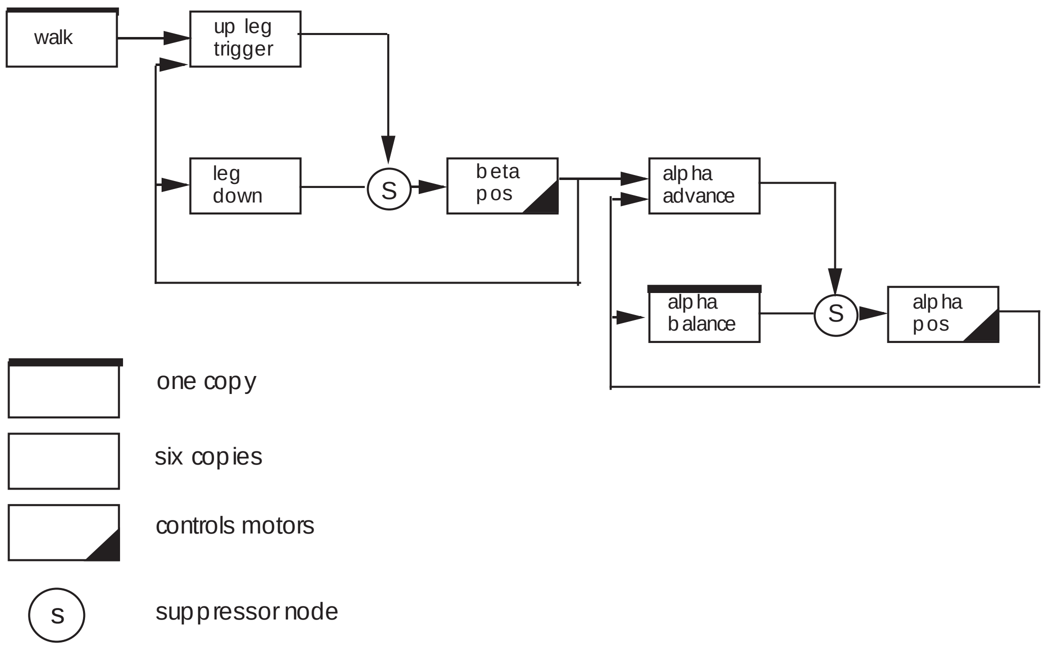


Figure 4.34: Basic locomotion network for Genghis. Adapted from (Brooks 1989).

- Walk machine sequences the lifting of the individual legs.
- Alpha Balance machine drives the sum of leg swing angles to zero. Forwards is positive angle, straight out is 0 angle, and backwards is negative angle. When a leg advances, this machine causes all the supporting legs to push backwards a little bit.

Each leg has 5 AFSM's which account for the remaining thirty AFSMs:

- Beta Pos commands the lift motor
- Alpha Pos commands the advance motor
- Up Leg Trigger is activated by the Walk machine. When it receives an input from the Walk machine, it causes its leg to lift for a predetermined time period. This is accomplished by blocking the output of the leg down machine to the Beta Pos machine.
- Leg Down continually tells the Beta Pos machine to put the leg down. This message gets through to the Beta Pos machine except when the Up Leg Trigger machine blocks it.
- Alpha Advance tells the Alpha Pos machine to swing the leg forward. It is active when the leg is commanded to lift and blocks the Alpha Balance message from reaching Alpha Pos. So, whenever the leg is lifted, it is reflexively swung forward as well.

It is not surprising that the control scheme of Hannibal and Genghis are similar given they were developed in the same lab and are programmed in the Subsumption Architecture. We shall focus on the differences instead. Genghis' locomotion network is less distributed. Genghis' locomotion controller is only partially distributed; the leg reflexes are local, but a centralized gait sequencer (the Walk machine) is used to generate stable gaits. In contrast, Hannibal's locomotion controller is completely distributed. Genghis' locomotion controller is less flexible. Implementing new gaits on Genghis requires changing the Walk machine in the control network. So far, a Tripod Walk machine and a Ripple Walk machine have been implemented on Genghis. In contrast, Hannibal's gaits emerge from the same network simply by changing the duration of the support phase. Genghis' controller is less

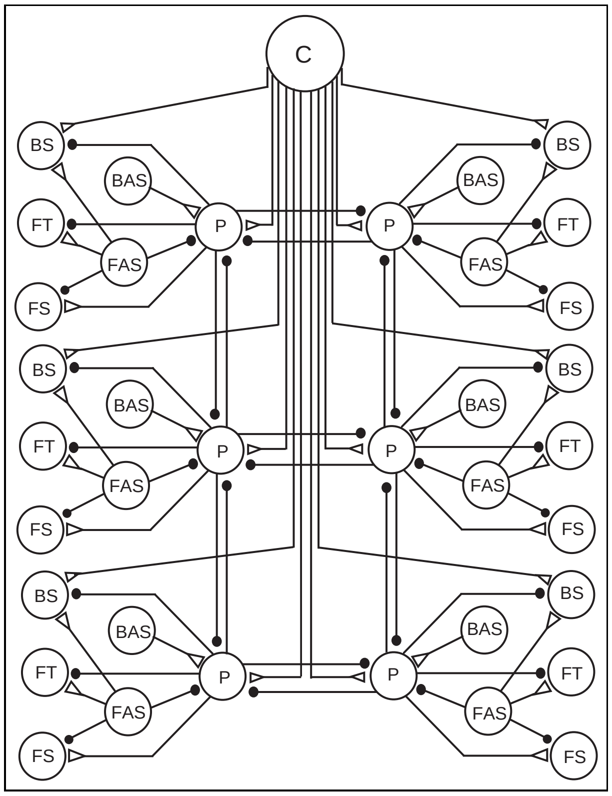


Figure 4.35: Neural controller for the Case Western Hexapod. Adapted from (Beer and Chiel 1993).

modular. On genghis, turning and directional behaviors are buried in other agents which makes them more difficult to interface with. Hannibal's turn behaviors and direction behaviors are distinct processes so they are more accessible.

4.8.5 Case Western Hexapod

Beer and colleagues implemented a neural network control architecture on a small hexapod robot with 2 DOF legs ((Beer & Chiel 1993), (Chiel, Quinn, Espenschied, & Beer 1992), (Quinn & Espenschied 1993)). The neural controller was developed by Beer and was inspired by Pearson's *flexor burst-generator* model of cockroach locomotion. The goal was to generate robust hexapod locomotion using a neural network controller.

The neural network controller designed by Beer is shown in figure 4.35. In this model, the same neural network is implemented on each leg. At the center of each leg controller is a pacemaker neuron whose output rhythmically oscillates. A pacemaker burst initiates a swing by inhibiting the foot and backward swing motor neurons and exciting the forward motor neurons. This causes the foot to lift off the ground and the leg to swing forward. Between pacemaker bursts, the foot is down and tonic excitation from the command neuron moves the leg backward. The output of the central pattern generator is tuned by feedback from 2 sensors that signal when the leg is reaching the AEP or the PEP. Approaching the AEP encourages a pacemaker to terminate a burst by inhibiting it. Approaching the PEP encourages a pacemaker to initiate a burst by exciting it. Inserting mutually inhibitory connections between the pacemaker neurons of adjacent legs generates statically stable gaits, and phase-locking the pattern generators on each side of the body enforces metachronal waves. Using this network, the Case Western Hexapod is capable of producing a continuous range of wave gaits by varying the tonic level of activity of the command neuron. The system was also robust to lesion studies performed by removing certain sensors or connections.

Beer's neural network is much truer in spirit to Pearson's model of insect locomotion control than Hannibal's agent network. However, both control schemes have similar organization and function. Organizationally, both systems have an oscillator on each leg that generates step patterns, a command neuron that determines the frequency of the oscillators, and a network circuit repeated on each leg. Granted the implementation of these components differs between the robots (Beer's is closer to Pearson's model), but the overall agents are similar. Both control schemes generate a range of wave gaits by varying the oscillator frequency. Both control schemes are robust to leg perturbations. Beer's approach has the nice property that adjacent legs put constraints on each other, so even if the control circuit in a leg is perturbed (removing sensor nodes or connections) the other legs coerce the perturbed leg into functioning properly. Similarly, Hannibal's oscillators are synchronized and coordinated such that if a leg is damaged, the oscillators remain coordinated with each other.

4.9 Contributions

The work described in this chapter makes three contributions toward the advancement of autonomous hexapod control. First, we implemented and tested several fully distributed, biologically motivated locomotion controllers on Hannibal. By doing so, we have demonstrated how various locomotion schemes used by insects can be applied to legged robots. In addition, we have further confirmed that a distributed control scheme using simple, concurrently running processes is a viable approach to controlling hexapod robots in real-time with relatively little computational power ((Brooks 1989), (Quinn & Espenschied 1993), (Donner 1987)). Second, we implemented a wide assortment of basic locomotion capabilities on Hannibal using the *local control with cooperation* paradigm. By doing so, we have explored the effectiveness of this approach in controlling a complex system (the robot) which consists of many concurrently running subsystems (the legs). Third, we implemented a fully distributed, biologically motivated locomotion controller on Hannibal that exhibits more *basic* locomotion capabilities than other hexapods using similar control schemes ((Beer et al. 1992), (Donner 1987)). By doing so, we have advanced the state of the art of fully distributed, biologically motivated locomotion controllers.

Chapter 5

Rough Terrain Locomotion

This chapter presents Hannibal's rough terrain capabilities. We present several strategies used by insects to traverse rough terrain. Several of these tactics inspired the design of Hannibal's rough terrain skills. We discuss how Hannibal uses inter-leg communication to traverse rough terrain. This is a significant aspect of Hannibal's control scheme, so we give it extra attention. Afterwards, we present the implementation of Hannibal's rough terrain network. Following this, we describe the tests we used to evaluate Hannibal's rough terrain performance and present the results. We conclude by comparing Hannibal's rough terrain control with that of insects and other legged robots.

5.1 The Rough Terrain Challenge

Hannibal's task is to locomote over natural terrain. Naturally occurring terrain has holes, cliffs, obstacles of various sizes, and undulations. To locomote over varying terrain, the robot needs ample sensory information to recognize changes in the terrain. The robot must orchestrate sequences of actions to negotiate obstacles and avoid hazards. Clearly, the control problem is significantly more complicated than for basic locomotion. Effectively managing Hannibal's numerous sensors and actuators and controlling the robot in real-time are important for the robot's success and safety.

We added another layer to the control architecture to give Hannibal rough terrain capabilities. The Rough Terrain Level is built on top of the Basic lo-

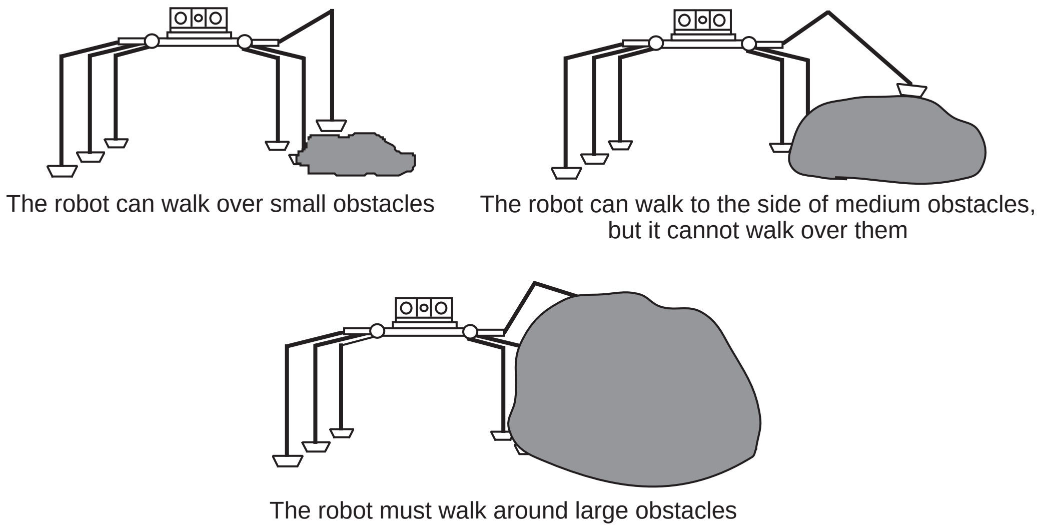

The robot can walk over small obstacles
The robot can walk to the side of medium obstacles, but it cannot walk over them
The robot must walk around large obstacles

Figure 5.1: Hannibal encounters obstacles of various sizes as it locomotes over natural terrain. "Small" obstacles are objects low enough for the robot to walk directly over. "Medium" obstacles are objects low enough for the robot to step on top of but too big to walk directly over. Consequently, Hannibal must walk to the side of medium sized objects. "Large" obstacles are too big for the robot to step over. As a result, the robot must walk around large obstacles.

comotion Level and is currently the top layer of the control architecture. The processes within this layer are responsible for adapting to terrain variations, negotiating obstacles, and avoiding hazards. Several of the rough terrain tactics implemented on Hannibal were inspired by rough terrain tactics used by insects. To activate the appropriate rough terrain behaviors, the system must recognize when it is confronted by challenging terrain and what challenge consists of. The system obtains this information from its sensors (via the output of the virtual sensors), indirectly through the state of agents, and indirectly by observing the behavior of the system over time.

Rough terrain control is fully distributed and exploits local control of the legs with inter-leg cooperation. Local leg control is responsible for handling challenges that confront individual legs such as stepping over an obstacle or finding a foothold. Effective inter-leg cooperation is vital because the legs are physically constrained to each other through Hannibal's body and through the terrain. Legs communicate with each other not only to coordinate and synchronize their behavior, but also to alert each other of hazards and to recruit the help of the other legs when necessary. Using this approach, we have implemented the following capabilities on Hannibal:

- Walk over small and medium sized obstacles
- Avoid large obstacles blocking the robot's path
- Search for footholds
- Walk over holes
- Avoid cliffs
- Adapt gait to terrain roughness
- Adapt to slopes

5.2 Insect Locomotion Over Rough Terrain

Pearson & Franklin (1984) presents results of cinematographic analysis of locusts walking on a variety of terrains. They wanted to determine the tactics

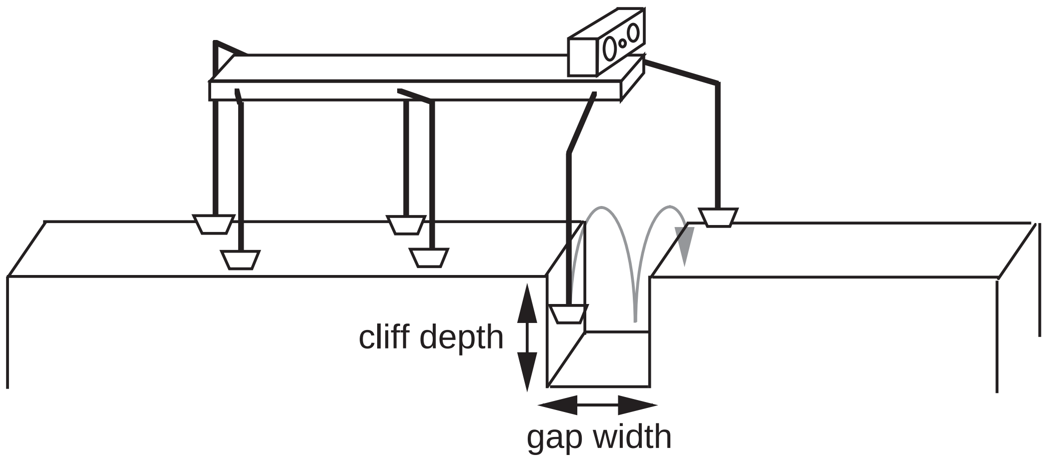


Figure 5.2: Natural terrain has depressions of various sizes. We define a "cliff" to be a terrain depression of which the robot can neither touch the bottom nor step across. We define a "gap" to be a terrain depression of which the robot cannot touch the bottom but can step across.

used by single legs to find a site for support, and the patterns of leg coordination when walking on rough terrain[1]. They report three distinct tactics used by single legs to find support sites on rough terrain:

- *Searching movements*: These are rapid, rhythmic, up-and-down movements initiated when the leg fails to find any support at the end of a swing phase. The searching movements caused the animal to pause in walking, and caused the insect to stop walking when searching is extensive. They also witnessed this reflex when the insect made postural adjustments.

- *Elevator reflex*: This consists of a rapid elevation and extension of the leg to lift above an object when it contacts the object during the swing phase. When the leg steps down, the foot usually is placed on the object. It occurs in all three pairs of legs, but is seen the most clearly in the middle and front legs.

[1]They also present results for the method the insects used for stepping over ditches and over elevated objects, but these results are not presented here. These rough terrain tactics used by the locusts are far beyond Hannibal's rough terrain capabilities to justify comparison.

- *Local searching movements*: These are small, rhythmic shifts of the foot on a potential supporting surface. Its function is to find a local region for a suitable support site. Pearson & Franklin (1984) propose that if the load on the leg does not quickly increase after the tarsus touches the surface, the foot is quickly lifted and replaced on the surface at another point. This process continues until the critical load is borne by the leg.

In regards to gait coordination, they found the insect did not adopt a rigid gait when walking on rough terrains. The wide range of stepping patterns was due mainly to variation in the timing of stepping in opposite legs of the same segment. Their findings suggest that when the locusts walk on rough terrain they do not adopt a strategy for coordination that differs in principle from the one used on flat surfaces. Rather, each leg appears to act independently in finding a support site, and the basic modes of coordination of opposite legs, and the posterior-to-anterior sequence of stepping in ipsilateral legs are preserved. However, they did witness that the middle legs of the insect stepped either exactly in phase or 180° out of phase, which was not observed on flat terrain.

5.3 Inter-leg Communication

Inter-leg communication is essential for Hannibal to successfully traverse rough terrain. Each leg is programmed to operate as a individual subsystem. However the legs are physically constrained to each other through Hannibal's body and through the terrain. Consequently the task of traversing rough terrain can be viewed as a team effort where the legs must work together for the global system (Hannibal) to accomplish the task. Inter-leg cooperation is achieved through inter-leg communication.

Hannibal's rough terrain network implements several types of inter-leg communication. For example, if a leg is stepping in a hole or over an obstacle, it tells the other legs to pause while it deals with the complication. It does this by having its step agent send a message that inhibits the oscillators of the other legs until the leg achieves ground contact (see figure 5.4). This type of communication insures the robot's stability by preventing the robot from advancing to the next step cycle before all recovered legs support the body. A leg can also recruit the help of other legs. For instance, when a

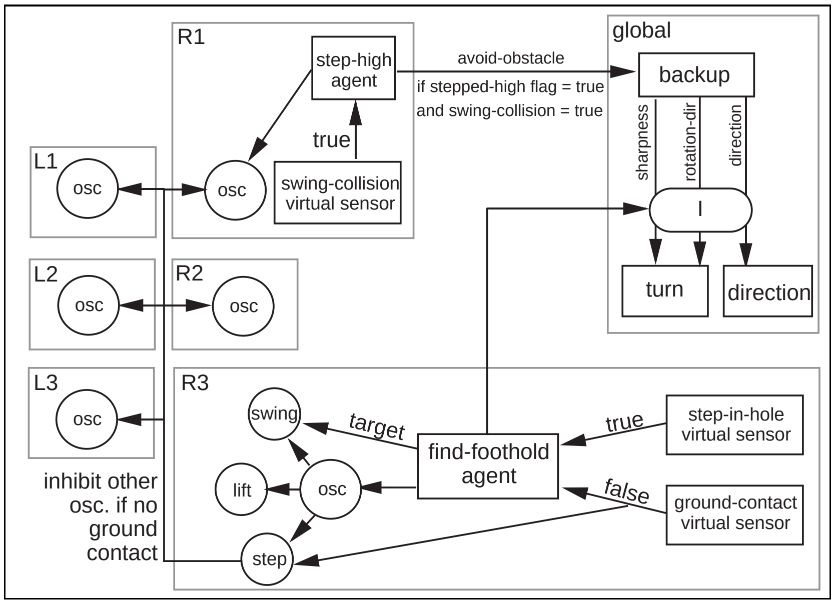

R1
step-high agent
avoid-obstacle
if stepped-high flag = true
and swing-collision = true
global
backup
sharpness
rotation-dir
direction
true
L1
osc
osc
swing-collision virtual sensor
I
turn
direction
L2
osc
R2
osc
L3
osc
R3
swing
target
find-foothold agent
true
step-in-hole virtual sensor
lift
osc
false
ground-contact virtual sensor
inhibit other osc. if no ground contact
step

Figure 5.3: Inter-leg behavior conflicts are handled by a pre-programmed priority scheme. Behaviors that achieve goals with a higher priority inhibit behaviors that achieve goals with a lower priority. For example, stability has a higher priority than obstacle avoidance. As a result, the find-foothold behavior inhibits the backup behavior. Once the leg finds a foothold, the robot is allowed to backup.

leg is trying to step over an obstacle, it asks the supporting legs to raise the body to help it clear the obstacle. It does this by sending a message to activate the lift-body behaviors in the supporting legs (see figure 5.6). A leg can also alert the other legs of a dangerous situation. For example, if a leg is stepping over a cliff, it tells the other legs to back the robot up. It does this by sending a message to the global direction and global turn behaviors which coordinate the direction and turn behaviors of all the legs.

Communication between locally controlled legs is essential for the legs to maintain a cohesive effort. This is especially the case if different legs want the robot to do different things. As shown in figure 5.3, one leg may hit a large obstacle and want the robot to back up, whereas another leg may be searching for a foothold and want the robot to hold still until it finds one. This is essentially behavior conflict, but on a larger scale than is typically encountered by other robots. Instead of having conflicting behaviors in competition over an actuator, conflicting behaviors on different legs are in competition over the action of all the legs. An inter-leg priority scheme, implemented using inhibition and suppression mechanisms (Brooks 1990) is used in cases like these to determine which leg has dominance. In general the stability of the robot has the highest priority, so in this example the legs wait until the searching leg finds a foothold before they perform the backup maneuver.

The rough terrain network handles slopes, gaps in the terrain, cliffs, low obstacles, and obstacles too large to step over. The system must recognize when it is confronted by a hazard and what the hazard is. The system obtains this information directly through sensor values, indirectly through the state of agents, and indirectly by observing the behavior of the system over time. Each of these cases is addressed below.

5.3.1 Virtual Sensor Activated Behaviors

The virtual sensor activated behaviors react reflexively to rough terrain. As presented in chapter 3, virtual sensor agents combine actual sensor data to detect a specific type of walker-terrain interaction. In the case of rough terrain, virtual sensors detect hazards such as slopes, obstacles, or holes in the robot's path. For each type of hazard, there is a virtual sensor designed to detect it and a rough terrain behavior designed to handle it. When a virtual sensor detects a hazard, it activates the corresponding rough terrain

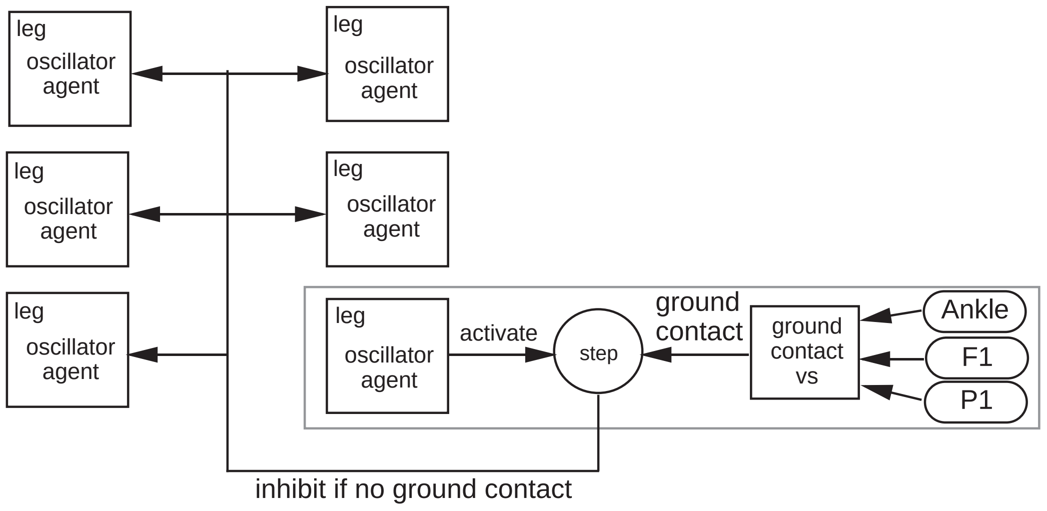


Figure 5.4: To enforce stable locomotion, the step agents of the recovering legs prevent the robot from advancing to the next step cycle unless the recovering legs support the body. Here, the right rear leg is in the recover phase. The step agent of this leg inhibits the oscillators of the supporting legs until the output of ground-contact virtual sensor of this leg is `true`. When this is the case, the step agent releases the oscillators of the other legs.

behavior.

5.4 Rough Terrain Network

5.4.1 Loading considerations

When walking, the robot should not transition to the next step cycle until all recovering legs have contacted the ground. This way, the robot does not begin the next step cycle until its legs are supporting the body. The *ground-contact virtual sensor* is responsible for detecting when a recovering leg is loaded. Its output is `true` when the leg is sufficiently loaded, otherwise its output is `false`. The *step agent* of each recovering leg inhibits the oscillators of the supporting legs until the recovering leg achieves ground contact (see figure 5.4). Once the recovering leg is sufficiently loaded, the output of the ground-contact virtual sensor is `true`, and its step agent releases the oscillators of the supporting legs. When all recovering legs are loaded, the oscillators are free to begin the next step cycle.

Holes in the terrain

When a leg steps into a hole it tries to find a foothold (see figure 5.2). The *step-in-hole virtual sensor* is responsible for detecting holes in the terrain. Its output is `true` when the foot reaches its lowest position but fails to make ground contact, otherwise its output is `false`. When it determines the leg is stepping in a hole, it activates the *find-foothold behavior*. The find-foothold behavior activates the recover phase of the leg and sets the AEP equal to one foot diameter beyond the current foot position. If the foot does not find a foothold looking in the current direction, it reverses the search direction once it reaches the search limit point. The behavior searches between the anterior and posterior search limit points until the search is successful or until it is deactivated by another behavior. While performing the search, the step agent of the leg inhibits the motion of the other legs by pausing their oscillators. When the foot makes contact with the terrain the search is successful, the find-foothold behavior releases the oscillators, and walking resumes. See figure 5.5.

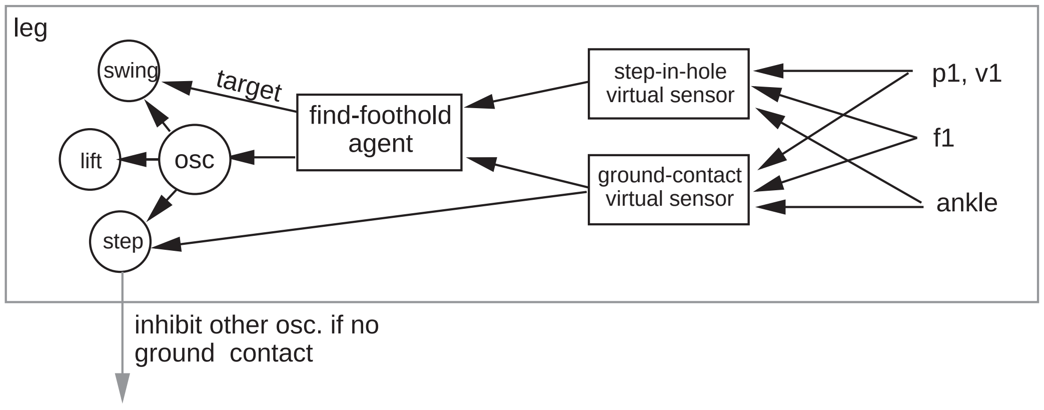

leg
swing
target
find-foothold agent
step-in-hole virtual sensor
p1, v1
f1
lift
osc
ground-contact virtual sensor
ankle
step
inhibit other osc. if no ground contact

Figure 5.5: For each leg, the find-foothold agent is activated once the output of the step-in-hole virtual sensor is `true`. While active, it generates the searching pattern by making the leg's oscillator agent repeat the recover phase with new target values. Meanwhile, the leg's step agent makes the robot pause by inhibiting the oscillators of the supporting legs. The find-foothold agent is de-activated when the output of the ground-contact virtual sensor is `true`.

Traversable obstacles

When a leg hits an obstacle during the swing phase, it tries to step over the obstacle. The *swing-collision virtual sensor* detects obstacles in the robot's path. Its output is `true` when the leg collides with an obstacle during the recover phase, otherwise the output is `false`. When its output is `true`, it activates the *step-high behavior* of the same leg. Because the elbow DOF is significantly slower than the other DOF's, the step-high behavior inhibits the motion of other legs until the leg completes the step-high maneuver. This is accomplished by pausing the oscillators of the other legs. While the other legs are waiting, the step high behavior lifts the leg as high as it can by fully extending the elbow. Once the leg is fully raised, it moves to the AEP and clears the obstacle if the obstacle is small enough. After the leg reaches the AEP, the behavior prepares for the step phase by contracting the elbow until the ankle is vertical. Once the ankle is vertical, the step-high maneuver is complete, the step-high behavior releases the oscillators, and walking is resumed. See figure 5.6.

When a leg hits an obstacle, it tells the supporting legs to lift the body

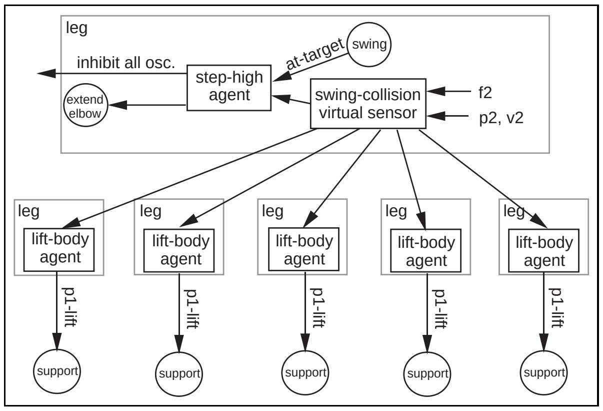


Figure 5.6: For each leg, the step-high agent is activated when the output of the swing-collision virtual sensor is `true`. When active, the step-high agent causes the foot to rise off the ground by activating the extend-elbow agent. Once the elbow is fully extended, the leg swings to its target value. The swing agent de-activates the step-high agent when the leg reaches the target value. Once this occurs, the elbow is contracted and the leg steps down.

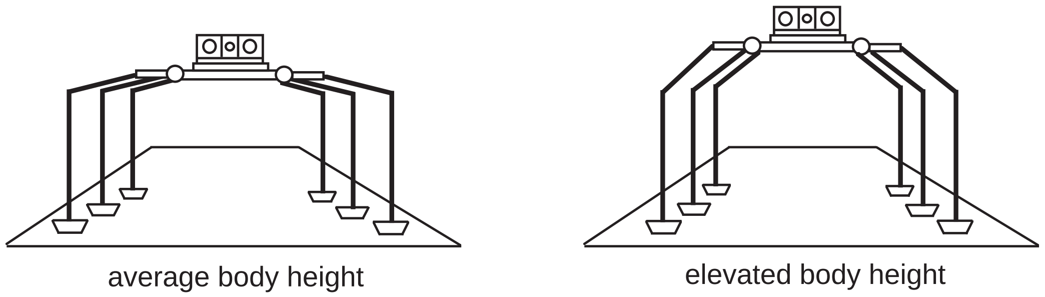


Figure 5.7: The robot's body can be elevated by activating the lift-body agents of the supporting legs. For example, when the output of a leg's swing-collision virtual sensor is `true`, the swing-collision virtual sensor activates the lift-body agents of the supporting legs. This helps the collided leg clear the obstacle.

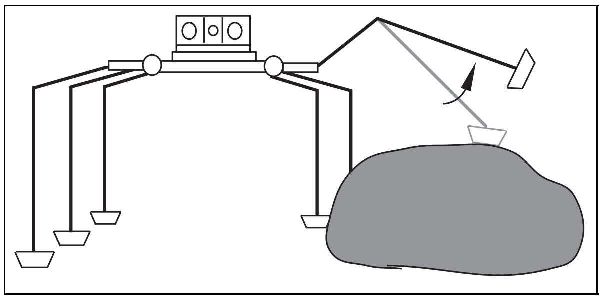


Figure 5.8: When the robot's foot is stepping on a medium sized obstacle, the foot needs to be lifted higher during the recover phase. Otherwise, the foot will drag along the top of the obstacle as the robot swings the leg forward.

higher. If the output of the swing-collision virtual sensor is `true`, it activates the *lift-body agents* of the supporting legs. Each lift-body agent extends the shoulder DOF of the supporting legs. When the lift-body agents of the supporting legs are active simultaneously, the robot's body height is increased (see figure 5.7). This helps the collided leg clear the obstacle while it tries to step over the obstacle.

Walking over an obstacle

When a leg is stepping on a fairly high obstacle, the robot lifts its leg higher during the recover phase, so its foot doesn't drag along the surface of the obstacle (see figure 5.8). The *step-over-rock virtual sensor* detects when the foot is stepping on an obstacle. It's output is `true` when the leg is loaded as a result of contracting the elbow instead of lowering the shoulder, otherwise its output is `false`. When the leg is stepping on an obstacle, the step-over-rock virtual sensor tells the step-high agent of the same leg to clear the obstacle during the recover phase of the next step.

slopes

When the robot walks on sloping terrain, it keeps its center of mass withing its polygon of support. The *inclination virtual sensor* senses the inclination of the robot. Its output is `true` when the inclination-error is sufficiently

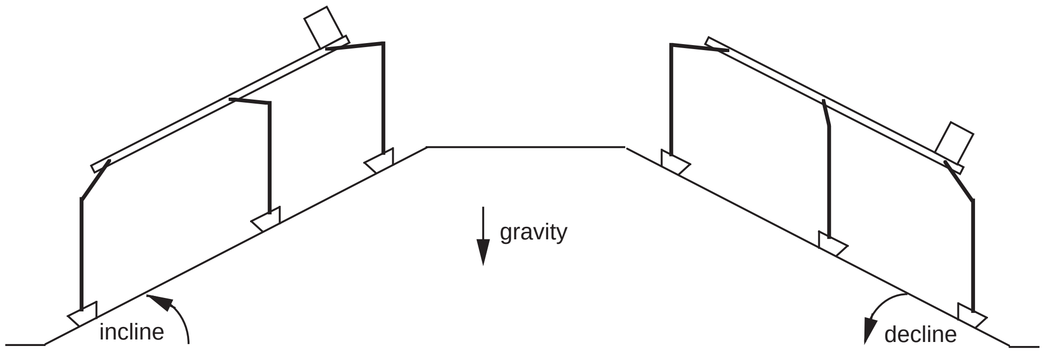


Figure 5.9: Hannibal servos its spine motor to keep its legs vertical when walking over undulating terrain. This maintains the robot's stability by keeping its center of mass withing the polygon of support.

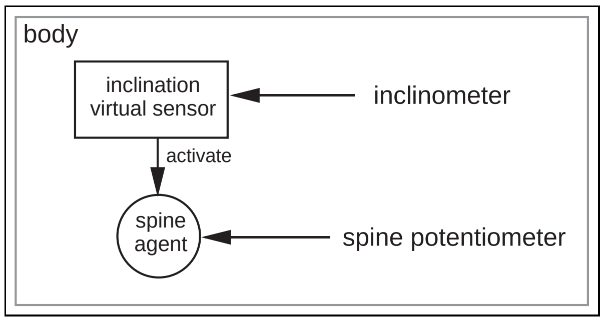


Figure 5.10: The inclination virtual sensor sends the pitch-error value to the spine agent. When this error is too large, the spine agent is activated and reduces the pitch-error by servoing the spine actuator. When the inclinometer's pitch value equals the reference pitch value (pitch-error = 0), the robot's legs are vertical.

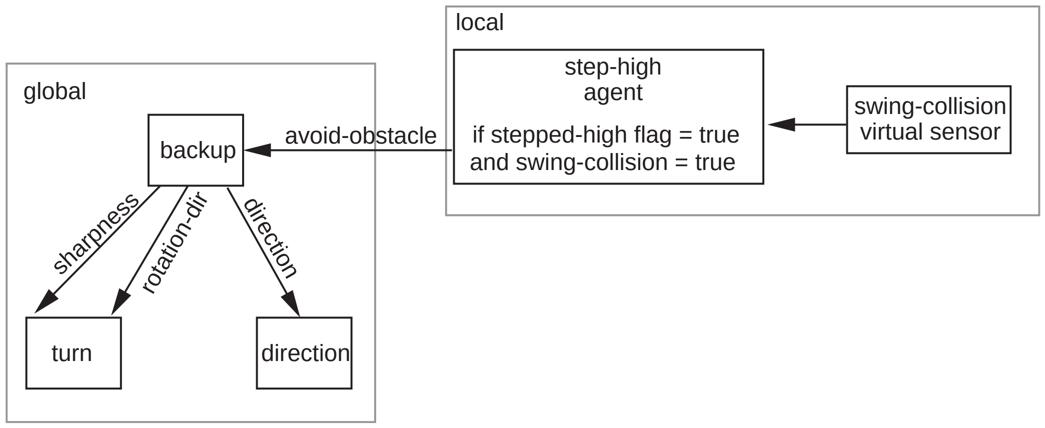


Figure 5.11: This network causes the robot to back away from obstacles that are too large to step over. The backup agent is activated when a leg collides with and obstacle after its foot is lifted as high as possible off of the ground. The global backup agent sets the number of steps the robot walks backwards and the turn sharpness based on the `avoid-obstacle` message. The direction of rotation is set such that the robot turns away from the obstacle.

large, otherwise it is `false`[2]. When the inclination virtual sensor output is `true`, it activates the *spine agent*. The network is shown in figure 5.10. The spine agent servos the spine potentiometer until the inclination sensor error is sufficiently small. This effectively servos the supporting legs until they are vertical as shown in figure 5.9. When the supporting legs are vertical, the center of mass of the robot is within the polygon of support.

5.4.2 State activated behaviors

The state activated behaviors respond to terrain information obtained through observing the state of other behaviors.

Large obstacles

[2]The spine actuator moves too slowly to make the quick, small adjustments necessary to continuously servo the inclination error to zero. To get around this problem, the inclination virtual sensor places a small "dead-zone" around the vertical inclination position.

The robot avoids obstacles too large to step over based on the state of the step-high agents of the front legs (or the rear legs if the robot is walking backwards). When either front leg collides with an obstacle during the recover phase, the step-high agent of that leg sets its `stepped-high` flag. If the leg collides with an obstacle when the `stepped-high` flag is set, the leg assumes the obstacle is too high to walk over and tells all the legs to avoid the obstacle by sending an `avoid obstacle` message to the backup behavior. The backup behavior is a global behavior that coordinates the direction, turn direction, and turn radius of the legs. When active, it causes the robot to walk backwards, angle away from the obstacle, and then resume walking forward. The network is shown in figure 5.11.

Cliffs

The robot retreats from cliffs based on the state of the find-foothold behavior. When the find-foothold behavior is active and either front leg finds ground contact searching rearwards, the leg assumes it is stepping over a cliff and tells all the legs to retreat by sending an `avoid cliff` message to the backup behavior. The backup behavior causes the robot to backup from the cliff, about face, and resume walking away from the cliff.

Resume forward direction of travel

Given the robot is walking backwards, it resumes walking forward based on the state of the direction agent, and either the state of the find-foothold behavior, or the state of the step-high agent. First consider the case where the find-foothold behavior is active when the robot is walking backwards and the leg finds ground contact searching in the forward direction. Under these circumstances the leg assumes it has backed up to a cliff and tells all the legs to change direction. Next consider the case where the robot is walking backwards and a rear leg collides with an obstacle while its `stepped-high` flag is set. Under these circumstances the robot assumes the obstacle is too high to step over and tells all the legs to change direction. The leg tells the robot to resume walking forward by sending a `walk-forward` message to the global direction agent.

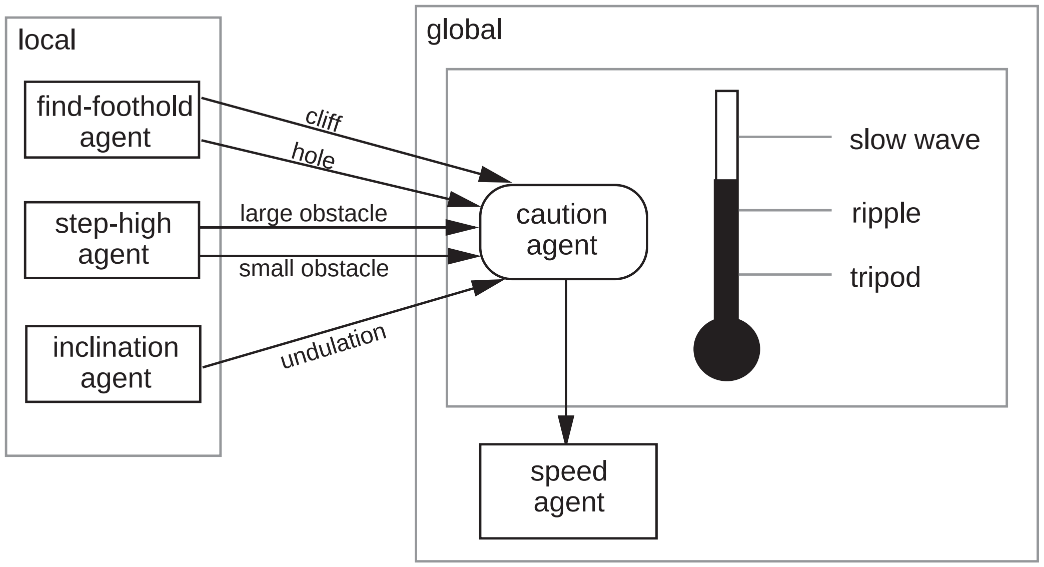


Figure 5.12: When the rough terrain agents of any leg become active, they increase the caution level of the robot. As the caution level rises, the robot adopts a slower gait. Hence, as the terrain becomes more challenging, the robot switches to gaits with larger gait stability margins.

5.4.3 Hormone activated behaviors

The hormone activated behaviors respond to terrain information obtained through observing the behavior of the system over time.

Cautious behavior

The robot's *caution level* influences the robot to change gait as a function of terrain roughness. Each time the robot encounters a hazard such as stepping into a hole or hitting an obstacle, the activated rough terrain agent increases the robot's caution level (see figure 5.12). If no hazards are encountered, the caution level decays. At the lowest caution level, the robot assumes the terrain is relatively flat and uses the tripod gait. This gait is the fastest of the wave gaits and simultaneously lifts the maximum number of feet. At the mid range caution levels the robot assumes the terrain is fairly rough and uses the ripple gait. The ripple gait is the robot's medium speed gait and simultaneously lifts fewer feet in the air than the tripod gait. At the highest caution level, the robot assumes it is in extremely rough terrain and uses the slow wave gait. This gait is the robot's slowest gait and keeps the maximum number of feet on the ground. The robot also uses the slow wave gait when it is in immediate danger. For example, if the robot thinks it is stepping over a cliff, it increases the caution level to the slow wave gait range. This causes the robot to use the slow wave gait when backing up from the cliff.

Impatient behavior

The robot's *impatience level* excites the robot to resume walking if the robot has been stationary for a sufficiently long time. It accomplishes this by temporarily lesioning the leg responsible for the hold up. When a leg is lesioned, the other legs ignore its inhibition of their oscillators. If a front leg is lesioned while walking forwards, or a back leg is lesioned while walking backwards, the robot assumes the terrain is too difficult for it to proceed in that direction and resumes walking in the opposite direction. For all other cases, the robot assumes the terrain is traversable in the direction of travel. For example, if a middle leg steps in a hole, it stops the other legs and searches for a foothold. For each cycle of the oscillator that the robot stands still, the impatience level rises. When the impatience level rises above its threshold, the robot is "frustrated" enough to ignore the inhibitory influences of the leg trying to find a foothold, lesions that leg, and resumes walking without it. While the robot is walking, the impatience level decays. When the impatience level is sufficiently low, the robot resumes using the lesioned leg.

5.5 Performance

5.5.1 Tests

Individual obstacle tests

These tests challenged Hannibal's ability to handle a single type of obstacle in its path. They were designed to test the performance of specific virtual sensors as well as specific rough terrain behaviors. They were used to determine if Hannibal could correctly characterize the type of obstacle in its path and respond to it. To test the robot's performance with respect to a particular hazard, the test terrain was flat with the exception of the terrain feature. Only one leg encountered the challenge at a time. The following terrain features were tested:

- Obstructive objects of varying heights and shapes
- Cliffs
- Holes or gaps in the terrain
- Inclines
- Declines

Multiple obstacle tests

These tests challenged Hannibal's ability to handle multiple terrain features in its path. The hazards could be of the same type or of different types. The robot could encounter the hazards sequentially or concurrently. The tests were designed to test the flexibility of the controller. They were used to ascertain Hannibal's ability to adapt its behavior to changing circumstances, test the legs ability to handle local terrain features concurrently, and test the legs ability to behave in a unified manner–particularly when behaviors on different legs may be in conflict. We did not bother to test all permutations of terrain features as there are too many to test. Instead, we chose a few sample terrains to test various capabilities. The following terrains were used (other terrains were used as well but served redundant purposes):

- *Small plateau* This terrain causes the robot to frequently encounter cliffs while walking forwards, backwards, or turning. The terrain tests the ability of the robot's legs to maintain a cohesive effort since behaviors on different legs want the robot to move in different directions.

- *Multiple small obstacles* This terrain causes the robot's legs to traverse obstacles concurrently. It tests the robot's ability to handle local terrain features simultaneously.

- *Crevice* This terrain causes the robot's legs to sequentially step over the gap. It tests the ability of the legs to address a hazard in rapid succession.

terrain feature	traversable size: minimum size	maximum size
small obstacle size	< 1/3 average body height	1/2 average body height (elevated) 1/3 average body height (normal stance)
medium obstacle size	1/3 average body height	5/6 average body height
large obstacle size	5/6 average body height	> 5/6 average body height
gap width	< 3/4 stride length width	3/4 stride length width
cliff depth	1/4 average body height	> 1/4 average body height
incline slope	< 15 degree slope	15 degree slope
decline slope	< 20 degree slope	20 degree slope

Figure 5.13:

- *Small object on plateau* This terrain causes the robot to handle different types of challenges. It tests the robot's ability to coordinate obstacle avoidance which the legs perform this individually, with hazard avoidance which the legs perform this as a team.

Naturally occurring terrain

We tested Hannibal's ability to traverse naturally occurring terrain in two environments. The first environment is a simulated lunar surface (also known as the "sandbox"). Our lab built the sandbox to test Hannibal and our other robots in a more realistic setting. The sandbox consists mostly of gravel, with some sand, and rocks of various sizes. The second environment is Mars Hill in Death Valley, CA. This location has an uncanny resemblance to terrain images taken of Mars. It is characterized by a fairly firm, undulating, sandy surface with scattered rocks.

5.5.2 Results

Figure 5.13 quantifies Hannibal's performance at traversing various aspects of rough terrain. These results were gathered from the individual obstacle tests. The values are primarily determined by the physical capabilities of the robot. Hannibal successfully traversed the sample terrains of the multiple obstacle tests provided the terrain features remained within the values of figure 5.13.

Similarly, Hannibal successfully traversed natural terrain provided the terrain features remained within the values of figure 5.13.

5.5.3 Evaluation

Hannibal successfully handles a wide assortment of terrain features. The rough terrain controller pushes the envelope of Hannibal's physical capabilities. Obviously if the robot had greater physical capabilities such as larger work spaces for the legs, better leg kinematics, greater strength to weight ratio, etc. the robot could traverse more challenging terrain. These issues are beyond the scope of this work, and we will not address them further. It would be interesting to implement a similar control network on future generation legged robots for comparison.

Local leg control with inter-leg communication has proven to be an effective means for Hannibal to traverse rough terrain. Each leg is responsible for handling the rough terrain challenge it faces concurrently with the other legs. Consequently, the legs can simultaneously handle obstacles local to themselves. This ability was demonstrated in the multiple small obstacles test. Inter-leg communication has demonstrated its importance in several respects. First, each leg behaves as a scout for the other legs and alerts them of common dangers. In this manner, the local terrain view of each leg is shared with the other legs. This ability was demonstrated in the tests using cliffs and large obstacles. Second, inter-leg communication enables the legs to act as a team. By working together, each leg accomplishes more than could if it had to fend for itself. For example, by asking the supporting legs to elevate the body, a leg can clear an obstacle it would not be able to clear otherwise. This was demonstrated in the individual obstacle tests using obstructive object of varying heights and shapes. Third, inter-leg communication enables the legs to maintain a unified effort, and the inter-leg priority scheme maintained the unified effort even when behaviors of different legs were in conflict. This ability was demonstrated by the small plateau test.

5.6 Comparison with insect rough terrain locomotion

Several of the same rough terrain tactics observed in locusts are implemented in Hannibal's rough terrain network. The find-foothold agent in conjunction with the step-in-hole virtual sensor implement the searching movements and local searching movements observed in locusts. The step-in-hole virtual sensor activates the find-foothold agent when the load on the foot is not sufficient and the foot is fully extended. Consequently, the find-foothold agent is activated either when Hannibal steps in a hole, or when there is insufficient loading on the foot after it completes the recovery phase. While the find-foothold agent is active it causes Hannibal to pause or stop (by inhibiting the motion of the other legs) until the search for a foothold is is successful. This behavior is also observed in locusts. The step-high agent in conjunction with the swing-collision virtual sensor implements the elevator reflex observed in locusts. Hannibal uses similar tactics as the locusts for gait coordination. Each leg of Hannibal is responsible for handling the holes and obstacles it encounters, but the strategy for gait coordination remains the same. Consequently, the basic modes of coordination are preserved, but the timing of stepping between the legs varies as the legs compensate for the terrain.

5.7 Comparison with legged robot locomotion over rough terrain

A common goal of legged locomotion research to develop walkers capable of traversing terrain too rough for wheeled vehicles. Several dynamically stable legged machines have been designed to study rough terrain locomotion (Raibert & Hodgins 1993). However, this section compares and contrasts Hannibal only to statically stable rough terrain walkers.

Rough terrain locomotion has been tackled using two different approaches: *closed loop* and *open loop*. In the closed loop approach, sensory information about the state of the machine and its interaction with the environment is used to compute the machine's actions during run time. Consequently, closed loop systems are able to adapt to the unexpected provided they have adequate sensing capabilities. In contrast, the open loop approach requires all important knowledge about the machine and the environment be given a priori. This knowledge is used to compute the machine's interaction with its environment before run time. Consequently, open loop systems rely on accurate information about the world and require good models of the interaction between the system and the world if accurate planning is to be achieved.

5.7.1 OSU Hexapod

The OSU Hexapod, introduced in chapter 4, was the testbed for the control algorithm proposed in McGhee & Iswandi (1979). This paper formalizes the rough terrain problem for legged vehicles (the free gait problem) and demonstrates the feasibility of generating a solution to the problem in real-time with a computer using a simulation. Its control strategy is centralized and open loop.

To solve the free gait problem, the control algorithm makes three assumptions. First, the algorithm assumes it is given a terrain map that provides global knowledge of the terrain. Second, it assumes it is given a motion trace (a path defining the body trajectory provided by a navigation system) that does not contain any obstacles too large to step over. This is a significant assumption for real-time control because computing the motion trace is quite expensive–it requires detailed knowledge about the kinematics of the machine and considers a combinatorially large set of possible leg motion sequences. Third, it assumes the task of placing a foot on a chosen foothold is easy for a statically stable system. With these assumptions, the solution of the free gait problem is reduced to the heuristic selection of reachable and suitable footholds along the motion trace; suitable footholds are terrain locations that provide adequate support, maintain stability of the system, and moves the system toward the goal. Raibert & Hodgins (1993) term this approach the *body motion-then-footholds paradigm*. The heuristic used for the OSU Hexapod is to lift the legs with the least kinematic travel available in the direction of travel, while putting legs into support with the largest available travel. This heuristic extended each support state to increase the probability that it would overlap with the next support state. The OSU Hexapod demonstrated its ability to negotiate small obstacles and climb down a step while operated under joystick control.

The solution to the free gait problem assumes a functional decomposition of the rough terrain problem into perception, modeling, planning, and control modules. Each module requires the output of the preceding module, so control proceeds sequentially. The solution to the free gait problem presented in McGhee & Iswandi (1979) is part of the planning phase (navigation is the other part). It assumes the perception, modeling, and navigation tasks are complete and give their results (the terrain map and the motion trace) to the free gait planner. It also assumes the results of the free gait planner (the sequence of footholds) can be easily executed by the control module. These are weighty assumptions as the perception, modeling, navigation and control tasks are difficult and computationally extensive tasks in their own right. Consequently, although McGhee & Iswandi (1979) argue the free gait planner can perform in real-time, the whole system may not.

Hannibal's approach to solving the rough terrain problem is radically different from the approach of the free gait planner. Hannibal's control system does not assume the existence of a terrain map or a motion trace, and it does not try to solve the free gait problem. Hannibal's control scheme decomposes the rough terrain problem into task achieving modules that execute concurrently and in real-time (Brooks 1986). The system is built from the bottom up where higher layers add more capabilities to the system. Given Hannibal only has tactile sensing ability, its highest control layer enables Hannibal to to wander over rough terrain in real-time. It chooses foot placements and leg adjustments based on its real-time interaction with the terrain. Eventually higher layers of control and more sophisticated sensors (compass, vision, etc.) could be added to give Hannibal the ability to navigate purposefully.

5.7.2 Preambulating Vehicle II

The Preambulating Vehicle II (PVII) is a quadruped developed at the Tokyo Institute of Technology (Hirose 1984). The PVII is fairly small, weighing 10 kg and measuring 870 mm in length. Each of its legs is a 3 DOF pantograph that translates the motion of each actuator into a pure Cartesian translation of the foot (each actuator translates the foot along one axis). This special linkage removed the burden of computing kinematic solutions to foot trajectories thereby significantly simplifying control. Each foot had two contact sensors at the bottom and around each foot.

The PVII uses closed loop control, and traverses rough terrain by implementing reflex-like control. The sensors on each foot sense when the foot is pressing against anything and how hard. This enables the robot to use its feet as sensing probes to initiate three reflex-like algorithms. If a foot switch signals contact as the foot advances forward, a reflex-like algorithm causes the foot to be pulled back, lifted, then advance forward again. Another reflex-like algorithm causes support legs to push downward if a load cell in the foot indicates it is not bearing an adequate vertical load. The third reflex-algorithm causes the leg to lift higher if the foot makes contact with an object during the swing phase. An oil-damped pendulum measuring body inclination triggers a fourth reflex which causes the relative altitude of the feet to be adjusted so the body remains level[3].

The control scheme of the PVII and Hannibal are similar in two respects. The first similarity is reflexive control. The PVII's reflex-algorithms enable the quadruped to climb up and down steps of arbitrary height and step length without a model of the terrain and without human intervention. Similarly, several of Hannibal's rough terrain agents such as the find-foothold agent or step-high agent give the legs reflexive behaviors. The second similarity is simplification of leg control. The legs of the PVII were specifically designed to simplify the joint angle coordination problem. In essence, the linkage design computes the inverse kinematic solutions to move the foot along a desired trajectory instead of the computer. Hannibal simplifies the joint coordination problem by continuously sensing the legs' interaction with the terrain. For example, instead of computing the angles for a specific trajectory which causes the leg to step over a ditch, Hannibal uses its leg sensors to tell it when it is stepping on a good foothold. If it is not stepping on a good foothold, it uses its sensors to help find a good foothold.

5.7.3 Adaptive Suspension Vehicle

The Adaptive Suspension Vehicle (ASV) was built in the mid '80s at Ohio State (Song & Waldron 1989). The ASV is a large hexapod vehicle designed for self contained locomotion on natural terrain. It stands 10ft tall, 15ft long, and weighs 3 tons. The 3 DOF legs are pantographs that used displacement hydraulic pumps to drive the joints.

The ASV traveled on rough terrain without global information. An operator rides in ASV to provide steering and speed commands while computers

[3]In later work (Hirose 84), the free gait problem was investigated in simulation using a hierarchical controller. The solution to the free gait problem and how it relates to Hannibal's control scheme has already been discussed.

control the stepping motions of the legs. It used force sensors in the leg actuators and a force distribution control algorithm to accommodate variations in the terrain–without foothold selection or planning. In the absence of a visual sensor or human foothold selection, the ASV has demonstrated its ability to travel on gentle slopes, walk through a muddy cornfield, and walk over railroad ties.

Hannibal and the ASV share the task of rough terrain traversal; however, the purpose of the two systems is quite different. Consequently, the two systems were designed to address different aspects of the rough terrain locomotion problem. The intent of the ASV was to build a legged "truck" that an operator could drive in rugged terrain. The constraint of carrying an operator has two important implications for the control of the ASV. First, stability is an extremely important issue if a person is on board the vehicle. Second, the vehicle should offer a fairly smooth ride so the operator doesn't get jostled about. Hence the control of the ASV concentrated on force distribution to address these considerations. In contrast, Hannibal was designed to be completely autonomous. Stability is an important issue, but falling for Hannibal is not as devastating as it would be for the ASV. In fact, one advantage of Hannibal's small size is the superior strength to weight ratio over larger systems. The superior strength to weight ratio helps the robot to survive a fall.

5.7.4 Ambler

The Ambler is an autonomous hexapod built at CMU in the early '90s ((Krotkov & Simmons 1992), (Krotkov et al. 1990), (Nagy, Whittaker & Desa 1992)). The Ambler is huge, standing approximately 18 feet tall and weighing about 2500 kg. Its 6 legs are arranged in two stacks on central shafts. Each leg has 3 DOF: a revolute shoulder, a revolute elbow, and a prismatic vertical axis. A six axis force sensor is mounted on each foot and a 3-D scanning range finder is mounted on top of the robot.

The Ambler locomotes over rough terrain using a traditional centralized control scheme. Locomotion is functionally decomposed into perception, modeling, planning, and control. The perception module uses the 3-D scanning range finder to gather terrain data. The modeling module uses this data to construct and maintain a local terrain map that can be used for locomotion guidance and short range navigation. The planning module is divided into two parts: the gait planner and the leg recover planner. The gait planner chooses footholds and body advances based on the local terrain map. It computes cost maps to indicate the "goodness" of each potential foothold on a 10 cm grid. Different cost maps are made of the terrain based on different considerations such as stability and reachability. The gait planner combines cost maps using weighted sum and selects footholds on the grid with the lowest cost in the composite cost map. The leg recovery planner determines the trajectory that gets the foot to the desired foothold without hitting obstacles. It creates a configuration space for the elbow and shoulder joints by growing terrain obstacles and other legs by the radius of the foot plus an uncertainty factor, and then searches the space using the A^* algorithm for the minimum cost path to the goal. These trajectories are sent to the control module where they serve as the intended set of walker motions that comprise the next gait cycle. The control module evaluates the effect of these planned motions on the walker-terrain interaction. This is done by predicting and assessing the foot forces resulting from the planned motion. If the predicted foot forces are acceptable, the walking cycle proceeds: first the body attitude is leveled into acceptable range, second the body altitude is brought to desired height, third the vertical actuators are locked out, fourth the body is propelled horizontally, and fifth the leg executes the recovery phase. Once the leg steps and makes ground contact, the new foothold is evaluated for stability for subsequent walker motions. If foothold is acceptable, the next planned gait cycle proceeds. Otherwise the gait planner chooses a new foothold location and determines the corresponding leg motions to realize the new foothold. There is also a reactive element of the control module whose function is to bring the body close to level in case of dynamic support failure. Using this control scheme, the Ambler is able to walk across obstacle fields with rolling sandy terrain, ditches, 30° ramps, and boulders that fit under the Ambler's legs and body.

The control concerns of Hannibal are diametrically opposed to the control concerns of the Ambler. Many of these differences result from the drastically different scale of the two physical systems. The Ambler stands 18 ft tall. Stability is of critical importance to the Ambler because of its enormous size. To insure stability, the Ambler plans every step and body movement with great care. It builds terrain maps so the Ambler can carefully select its footholds. Foot placements are chosen so the center of mass of the system always remains in the conservative polygon of support. Foot placement forces are analyzed in detail before and after the step is taken to insure the terrain supports the robot, and the foot will not slip. Leg trajectories are planned so no leg collisions occur during the return stroke, and only one leg performs a return stroke at a time. Body altitude and attitude are carefully monitored so the system does not become unstable. To perform the required computation in reasonable time, the perception, modeling, planning, and control are implemented in a task control architecture using several dedicated processors. In extreme contrast, Hannibal stands 8 inches tall and assumes stability. Its low, broad stance is extremely stable, and falling is not a concern because the spine motor is designed to rotate the legs to a standing position if the robot finds itself on its back. [4] The emergent wave gaits generated by the locomotion network keep the center of mass within the polygon of support. Since it's survivability does not require the robot to carefully plan its gaits, planning and modeling are not necessary for the robot to wander over rough terrain. Instead it uses many sensors to continuously monitor its interaction with the terrain, and uses this information to govern its actions.

5.7.5 Ghengis

The locomotion network presented in chapter 4 is extended to enable Ghengis to traverse rough terrain as shown in figure 5.14. The rough terrain capabilities presented in Brooks (1989) include force compliance, pitch stabilization, and walking over low obstacles.

- The *beta force machine* and *beta balance machine* written for each leg implement active compliance with the terrain. The force machine monitors the forces on the beta motor during the step phase. The beta balance machine sends out a `lift-up` messages when this force is too high. It has a small dead-band where it sends out zero move messages that pass through the defaulting switch on the up-leg trigger machine and eventually suppresses the leg down machine.

- The *alpha collide machine* is responsible for lifting the leg higher on the next step if the leg suffers a swing collision. It monitors the alpha forces during the swing phase and writes a higher value to the height

[4]It is feasible that the robot would survive a moderate fall given it's strength to mass ratio. The strength to mass ratio improves as the system decreases in size.

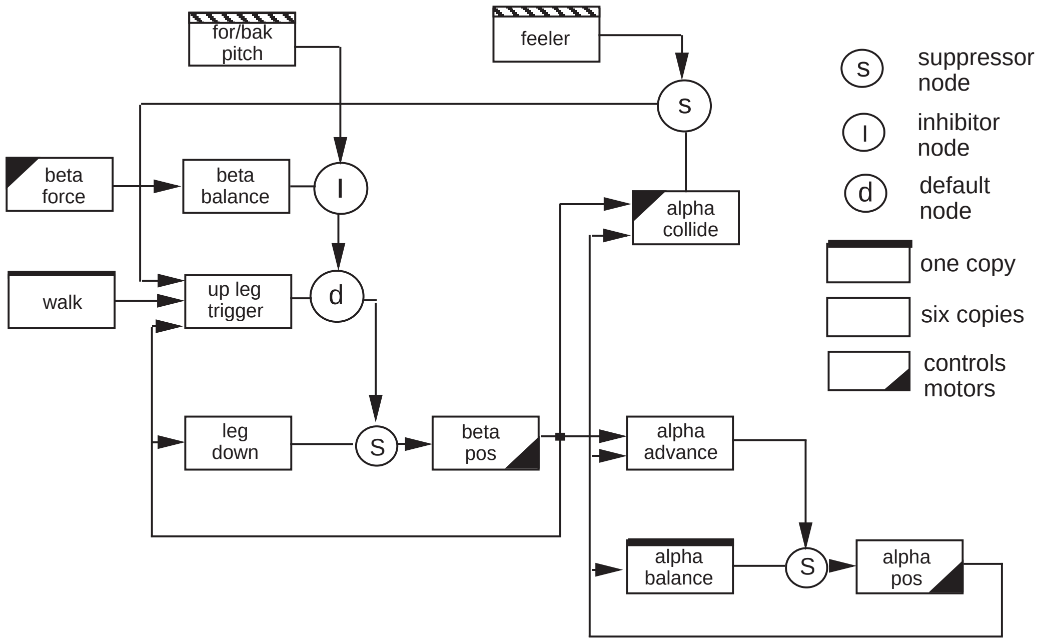


Figure 5.14: Genghis rough terrain network.

register of the up-leg trigger machine if the alpha force is large. The up-leg trigger machine resets this value on the next step.

- The *feeler machine* written for each whisker is responsible for lifting a front leg higher for the next step if the whisker on the same side suffers a collision.

- The *forward pitch machine* and *backward pitch machine* are responsible for minimizing the pitch of the robot. These machines monitor the high-pitch conditions on the pitch inclinometer and inhibit the local beta balance machine output in the appropriate circumstances.

Inter-leg communication is the most significant difference between Ghengis' and Hannibal's rough terrain networks. As described earlier in this chapter, inter-leg communication enhances the cooperation between the legs. Ghengis does not have this capability, and its legs function less as a team because of it. For example, when Ghengis approaches a tall obstacle the local behavior of the leg causes it to lift higher. However, if the leg does not succeed in clearing the obstacle, the other legs do not know that the front leg is having problems. Consequently, they continue to walk forward thereby repeatedly shoving the front leg into the obstacle.

In general, Genghis implements fewer rough terrain capabilities than Hannibal. Genghis has no means for sensing terrain depressions, so it falls into holes and walks over ledges. Genghis is also more limited in the steepness of slopes it can handle. Genghis does not have a spine DOF like Hannibal, so Genghis' center of mass falls outside its polygon of support once the terrain becomes too steep. Genghis also does not have any time-varying behaviors such as adapting its gait to suit terrain difficulty. As a result, Genghis is more limited than Hannibal in the types of terrain it can handle.

5.8 Contributions

The work described in this chapter makes several contributions toward the advancement of autonomous hexapod control. First, we have implemented various rough terrain tactics used by insects on Hannibal. By doing so, we have demonstrated how fully distributed, biologically inspired rough terrain capabilities can be applied to legged robots. Second, by testing Hannibal on various test terrains, we have demonstrated that *local control with cooperation* is an effective means of controlling a complex system (systems consisting of several concurrently running subsystems) in a complex environment. Third, we have implemented rough terrain capabilities on Hannibal comparable to those of sophisticated walkers, but on a dramatically smaller scale and requiring significantly less computing power. In doing so, this work advances the state-of-the-art of fully distributed, biologically inspired legged locomotion control for rugged terrain.

Chapter 6

Fault Tolerance

This chapter presents Hannibal's fault tolerance capabilities. In the following sections, We define the task and present the issues involved in designing our fault tolerant system. Following this, we motivate our approach and present our implementation in detail. Then, we describe the tests we used to evaluate the performance of our system and discuss our results. We conclude by comparing our system to similar work in the field.

6.1 The Fault Tolerance Challenge

For Hannibal, having many sensors and actuators is a double edged sword. Multiple sensors provide more reliable sensing and a richer view of the world. More actuators provides more degrees of freedom. However, more components also means there's more that can fail and subsequently degrade performance. Physical failures can be attributed to either mechanical failure, electronic failure, or sensor failure. Subtle changes in the state of the robot such as sensor signal drift also degrades performance. Hannibal's task is to locomote in rough and hazardous environments. As it scrambles along, it can subjected to repeated bumps, snags, and stresses. This places significant wear and tear on Hannibal's hardware. Not surprisingly, components fail or uncalibrate over time.

Hannibal experiences various types of sensor and actuator failures. Other types of failures can occur, such as computing system failures or software failures, but these failures occur far less frequently. Sensor failures are the most common failure on this robot. Hannibal experiences a sensor failure approximately once every two weeks. These failures have a variety of causes. For instance, sensor wires break because of the stress the moving joints subject them to. Sensor mounting problems such as insufficiently clamped joint angle potentiometers cause signal drift. The reference value of the strain gauges drift over time as well. The metal case of the joint angle potentiometers are not electrically isolated well from rest of system which can cause erroneous sensor readings. The second most frequent type of failure are actuator failures. Actuator failures occur about once every three months. The shaft of the shoulder motor has a tendency to break off over time. Sometimes motor clampings loosen so the motor rotates within its mount. As the motor rotates in its mount, the motor's drive signal wires twist and eventually break from stress.

Fault tolerance is implemented on Hannibal using a distributed network of concurrently running processes. To tolerate hardware failures, a set of fault tolerance processes are written for each component. These processes are responsible for detecting faults of their respective component, and minimizing the impact of the failure on the robot's performance. By exploiting concurrency and distributedness, the system monitors, detects, and compensates for component failures simultaneously.

The fault tolerant network addresses the following objectives:

- *Fast response time to failures*: Hannibal operates in a hazardous environment. Consequently, Hannibal must detect and remedy failures quickly or else its safety may be jeopardized. This means failures must be detected and compensated for *before* system performance degrades to an unacceptable level.

- *Graceful degradation of performance*: Hannibal's performance must degrade gracefully as failures accumulate. This requires Hannibal to maintain the highest level of performance possible given the functional state of the hardware.

- *Access to all reliable resources*: More sensors and actuators enhance system performance provided they are functional. Hence, Hannibal should reincorporate the use of repaired components.

- *Fault coverage*: The robot can suffer from a variety of failures. Failures can be permanent or transient. Some failures have a local effect while

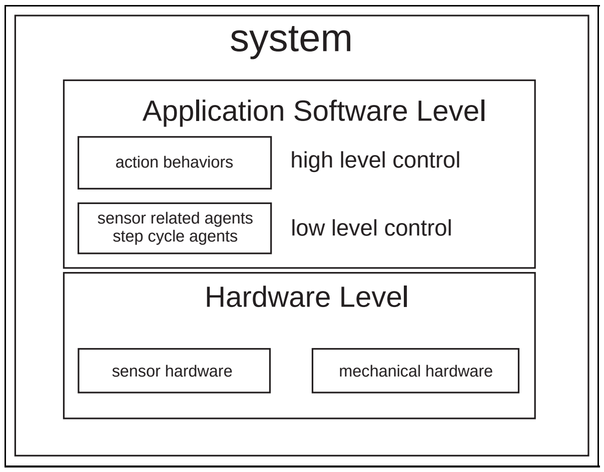


Figure 6.1: Sensor and actuator failures effect these levels of Hannibal's system hierarchy.

others have a global effect on performance. Sensors may uncalibrate. Furthermore, the robot should be able to recover from different combinations of failures. Failures can occur individually, concurrently with other failures, or accumulate over time.

6.2 Confinement of Errors

Sensor and actuator faults affect various levels of Hannibal's system hierarchy as shown in figure 6.1. The sensor-actuator hardware level is the lowest level, low level control is the intermediate level, and behavior control is the highest level. The low level control is equivalent to the Sensor-actuator Level of the control architecture, and the high level control is equivalent to the Basic Locomotion and Rough Terrain layers of the control architecture. Clearly sensor or actuator failures affect the hardware level of the system. Sensor failures affect the low level control because the virtual sensor agents use information from real sensors to compute their results. Consequently, sensor failures may cause the virtual sensor results to be incorrect. If this is indeed the case, then the high level control is also affected by sensor failures. The virtual sensor agents are responsible for activating the correct behavior at the appropriate time. However, if the results of the virtual agents are incorrect,

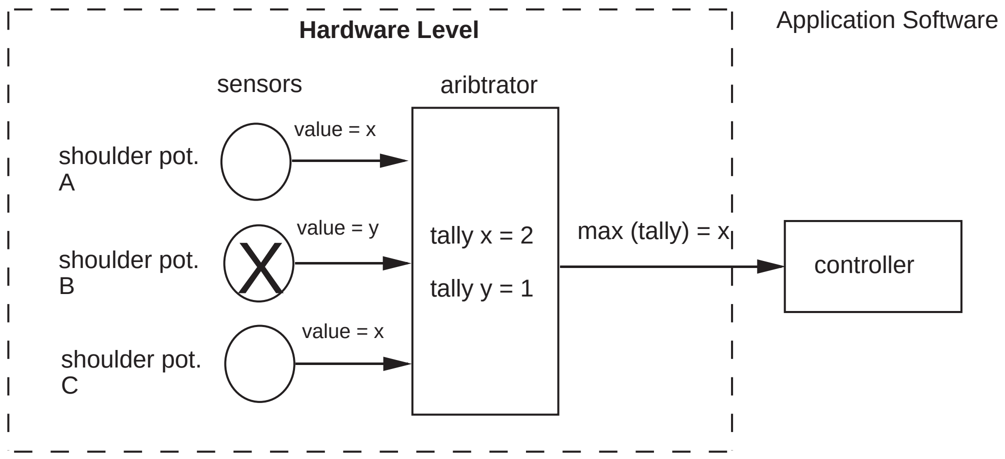


Figure 6.2: Replication of components is often used at the hardware level to make the system tolerant of hardware failures.

then the wrong behaviors will be activated.

It is important to detect and confine errors to the lowest possible level in which they occur. If an error is not confined to the level in which it originated, then higher levels must detect and compensate for the fault. As an error propagates up the levels of the system hierarchy, it affects increasing amounts of system state. Longer response times to error correction means the error manifestations become more diverse. Hence, detecting and confining errors to the lowest possible level of the system hierarchy maximizes the effectiveness of the recovery procedures and minimizes the impact of the error on system performance.

6.3 Levels of Fault Tolerance

Given that hardware failures affect various levels of the system, fault tolerance techniques can be implemented at each level. Below, we present possible fault tolerance strategies for each level, and describe the merits and shortcomings of each method.

6.3.1 Replication of Hardware

Given we are concerned with sensor and actuator failures, reliable sensing and actuating hardware is desirable. Replication of hardware is commonly used to enhance hardware reliability. Several systems, such as those presented in (Gray 1990) and (Siewiorek & Johnson 1981), use this approach to achieve robustness. In Hannibal's case, this would correspond to replicating sensors and actuators. For a sensing example, multiple potentiometers could be used to sense a particular joint angle as shown in figure 6.2. An arbiter gathers the joint angle values of each potentiometer and sets the accepted joint angle value to be the most common value. The application software uses this value as the joint angle. Failure of a joint angle sensor is detected if its value does not agree with the other sensors' values. However, the failure is masked because the most common value is accepted as the actual value. Assuming the majority of the potentiometers work properly, recovery from the failure is immediate because the software is given the correct joint angle value. Consequently, hardware failures are detected and confined to the hardware level of the system hierarchy.

Ideally we could detect, confine, and correct hardware errors at the hardware level. By doing so, the application software need not be alerted to these failures. However, this approach has some drawbacks. First, replication of sensors and actuators is expensive. Second, it assumes that a majority of replicated components are working. Ideally, Hannibal should perform reliably with a minority of functional sensors. Third, Hannibal's size and weight constraints restrict how may sensors and actuators can be mounted on the robot. Therefore it is impractical to enhance sensing or actuating capabilities on Hannibal by using this approach. However, Hannibal was designed with multiple sensors and actuators that provide complementary capabilities. For example, Hannibal can use several different sensors to detect ground contact, and it can lose the mechanical function of a leg and still walk. Hannibal's control system must be clever in the way it uses its existing sensors and actuators to compensate for sensor or actuator failures.

6.3.2 Redundant Control Behaviors

Fault tolerance techniques using redundant sets of control strategies have been investigated in Payton, Keirsey, Kimple, Krozel & Rosenblatt (1992). The redundant strategy approach implements fault tolerance in high level control and addresses high level failures. For example, a high level failure oc-

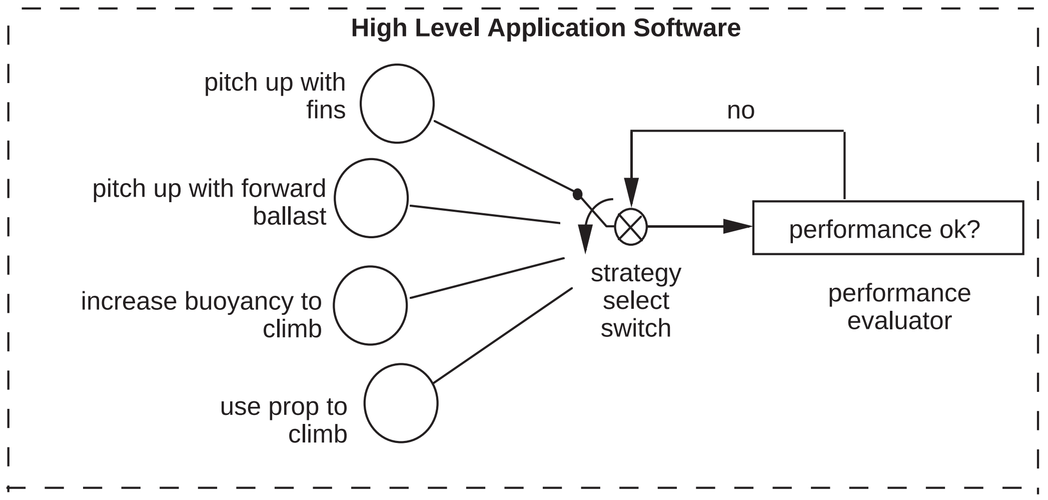


Figure 6.3: Robust behavior can be implemented using redundant strategies.

curs when the robot encounters a situation it was not explicitly programmed to handle. In the redundant strategy approach, the controller is designed with redundant strategies to perform a task. A performance model exists for each strategy, and a failure is detected if the behavior's actual performance is worse than the behavior's expected performance. If the first strategy tried does not suffice, the controller eventually tries another strategy. The controller goes through its repertoire of strategies until it finds a strategy with acceptable performance instead of unsuccessfully trying the same thing over and over. Payton et al. (1992) gives an example of an autonomous submarine trying to avoid the ocean floor (figure 6.3). There are several strategies the submarine can use to avoid the bottom: pitch up using fins, pitch up using the forward ballast, increase buoyancy, and use prop to back away from the bottom. The preferred strategy is to pitch the vehicle up using the fins. If the fin actuator is broken the submarine will not adequately avoid the bottom using this approach. After the controller determines the submarine is not avoiding the bottom well enough, it recovers by switching to the pitch up with forward ballast strategy. If the performance is still unsatisfactory, the controller continues to try other strategies until the performance is acceptable.

The redundant strategy approach is not well suited for hardware failures. First, it does not specifically address the cause of the problem, it only addresses the symptoms. It may take several tries before the controller finds a strategy that works. For example, let us say Hannibal has redundant walking behaviors - each behavior implements a different gait. If a leg fails, some of these gaits are unstable. The redundant strategy approach requires that Hannibal undergo unstable locomotion until the controller finds a gait that is stable with the loss of that leg. It is more desirable if Hannibal could recognize which leg failed and adapt its gait to specifically address the failure. Second, the redundant strategy approach inherently requires that the hardware errors manifest themselves in the robot's behavior before the control system can detect something is wrong. Ergo, the performance of the infected behaviors must degenerate to an unacceptable level before the controller takes corrective action. This could be detrimental to a robot that must function in a hazardous environment. Take for example, the case where a sensor used by the step-in-hole virtual sensor is broken. The performance of the step-in-hole virtual sensor would have to sufficiently degrade before the control system would take notice of the failure. It would be unfortunate if Hannibal's control system had to wait until Hannibal walked off a cliff before it could determine the step-in-hole virtual sensor wasn't working well. Hannibal's survival could depend on detecting, masking, and recovering from errors in the low level of the control system – before the errors infect high level behaviors.

6.3.3 Robust Virtual Sensors

Hannibal's controller uses robust virtual sensors to confine hardware failures to low level control. Robust virtual sensors are virtual sensors which remain reliable despite sensor failures (figure 6.4). Recall that the virtual sensors are responsible for characterizing the robot's interaction with its environment using sensor information, and for activating the robot's behaviors. If the virtual sensors give the correct output despite sensor failures, then the robot will continue to do the right thing at the right time despite these failures. For example, if the ground-contact virtual sensor can correctly determine ground contact despite failure of the force sensor, the behaviors that use ground contact information will not be affected by this failure.

Robust virtual sensors are appealing because they confine the effect of faults to low level control and prevent errors from infecting high level control. This approach effectively compensates for *local failures.* Local failures (also

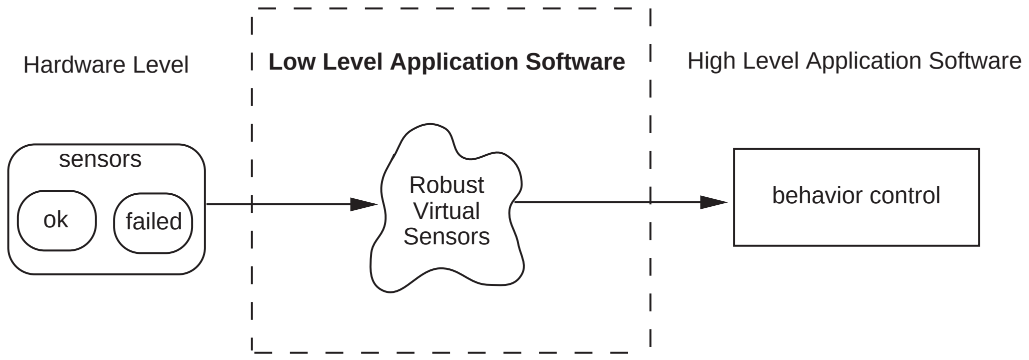


Figure 6.4: Robust virtual sensors remain reliable despite sensor failures. We use them to make Hannibal tolerant of sensor failures.

called non-catastrophic failures) are failures whose effect is confined to the leg on which it occurs. For example, the failure of a leg's ankle sensor is a local failure because it affects that leg's ability to sense ground contact, but it does not affect the ability of the other legs to sense ground contact.

Unfortunately, it is not possible to confine the effects of all sensor-actuator faults to low level control. Some failures affect the behavior of the overall system - we call these failures *global failures*. Global failures (also called catastrophic failures) must be compensated for within high level control. For example, if a leg's shoulder actuator fails then the leg cannot support the body. Consequently, this failure affects the global stability of the robot. The high level control must compensate for this failure by changing the robot's gait so the robot can walk in a stable manner with one less leg.

6.4 Adaptivity vs Redundancy

For the reasons presented above, we have chosen to compensate for local failures within low level control and to compensate for global failures in high level control. How should we implement the robust virtual sensors to address local failures? How should we implement the high level behaviors to address global failures? A common approach is to exploit redundancy to achieve robustness. To handle local failures, we could create redundant virtual sensors for each leg-terrain interaction of interest to the controller. For example, we could design three redundant ground contact virtual sensors that use different sensors to sense ground contact. Each redundant ground-contact virtual sensor would vote for either `ground-contact = true` or `ground-contact = false`, and the verdict would be determined by majority vote. To handle global failures, we could write redundant walking agents where each agent implements a different gait. The controller could activate a given gait agent depending on which legs have failed. However, we found that implementing our fault tolerance capabilities using redundant agents was inappropriate for our application (for reasons we present below). Instead, Hannibal's fault tolerance capabilities are implemented using *adaptive* agents. The use of adaptive agents distinguishes Hannibal's implementation of robustness from other implementations.

We initially tried using redundancy to achieve robust virtual sensors on Hannibal, but abandoned it for the following reasons. First, there is no way the controller can obtain a reliable verdict once the majority of the sensors fail. Second, code size grows significantly with this approach because multiple virtual sensor agents are written for each leg-terrain interaction we want to determine. Given Hannibal's small program memory this is an important consideration. Third, once a sensor fails, all virtual sensors that use information from that sensor become less reliable. Hence, a minority of sensor failures could adversely affect the majority of redundant virtual sensors. Payton et al. (1992) presents an example where all the virtual sensors that activate avoid bottom behaviors use an altitude sensor. If the altitude sensor fails, then the reliability of all these virtual sensors degenerates. Consequently, the `avoid bottom` performance of the submarine degrades with the failure of one sensor. Finally, the inherent reliability of redundant virtual sensors is not uniform. Virtual sensors that use more sensory information generally produce a more reliable result than virtual sensors that use less sensor information. For example, say we designed three redundant ground-contact virtual sensors: GC_{PFA} which uses position, force, and ankle information, GC_A which uses ankle information, and GC_F which uses force information. Hence, GC_{PFA} is more reliable than both GC_A and GC_F. We may want to reflect the relative believability of virtual sensors by weighting their votes so that two less reliable virtual sensors cannot override the vote of a significantly more reliable virtual sensor. However, if the position sensor fails then GC_{PFA} becomes less reliable than GC_A and GC_F. Suddenly we what GC_A and GC_F to override GC_{PFA}. In essence, the differences between the inherent reliability of redundant virtual sensors, and the effect of failed sensors on their reliability significantly complicate the arbitration process.

The approach Hannibal uses to implement robust virtual sensors is substantially different from the approach described above because it exploits adaptivity instead of redundancy. One adaptive virtual sensor (instead of several redundant virtual sensors) exists per leg for each leg-terrain interaction of interest to the controller. As sensors fail, the adaptive virtual sensors maintain reliable performance by changing how they use their sensor information. For example, when a failure is detected, the appropriate virtual sensors are alerted of the failure and respond by reconfiguring the way they uses their sensory information. This entails ignoring the input from the broken sensor and changing the way they use information from the reliable sensors. In this manner, the virtual sensors use their most reliable sensors to produce the most reliable result. If the failed sensor starts working again, the virtual sensor reintegrates the previously failed component.

The approach Hannibal uses to tolerate catastrophic failures also exploits adaptivity instead of redundancy. When a leg suffers a catastrophic failure it is not usable. High level control must change the gait so locomotion remains stable with one less leg. A redundant approach might involve implementing redundant walking behaviors where each behavior exhibits a different gait. This is undesirable because of the extra code space required to implement each gait behavior plus the gait switching mechanism. In contrast, the adaptive approach implements one walking behavior which can alter its gait by changing a parameter. Low level control is responsible for detecting catastrophic failures and alerting high level control. High level control is responsible for adapting the robots behavior so that locomotion remains stable.

6.5 Fault Tolerance Network

The following sections describe the distributed network that implements fault tolerance on Hannibal. As with the rest of Hannibal's control system, fault tolerance is implemented with concurrently running processes. Fault tolerance consists of four phases: error detection, masking, recovery, and reintegration. Non-catastrophic faults affect local control. They are detected within the low level network and compensated for within the virtual sensors. In this way, these faults do not affect the high level performance of the system. Catastrophic faults unavoidably affect global system performance. They are detected within the low level control and compensated for within the high level control.

In the following sections we illustrate Hannibal's fault tolerance processes through an example. The example we use to illustrate tolerance to local failures is a robust ground-contact virtual sensor. Keep in mind that this example is of only one virtual sensor on one leg which uses only a few of the sensors on that leg. Similar processes run concurrently on the same leg to make its other virtual sensors robust as well. The example we use to illustrate tolerance to global failures is a broken shoulder actuator. Similar processes run concurrently on the same leg to tolerate other global failures as well. All these processes are implemented on each leg and run simultaneously.

6.5.1 Detection

The detection processes are responsible for recognizing sensor and actuator failures. Detection is the hard part of the fault tolerance problem because the robot doesn't know a priori the correct behavior of the sensor. For example, if the robot had an accurate terrain map, then it could compare its leg's sensor readings with the terrain map. The robot could essentially go through a process like "I know the ground is flat ahead, so when I step down my sensors should eventually tell me I have ground contact", or "I know there's a hole ahead, so when I step down my sensors should tell me I'm stepping in a hole". If a sensor does not behave as expected, then the robot can conclude that sensor is broken.

However, Hannibal *does not* know what its sensory outputs should look like for any given step cycle. This is because Hannibal does not know what the terrain looks like before hand. Furthermore, Hannibal cannot predict what the terrain looks like because the terrain can change dramatically. For example, Hannibal's leg sensors could tell it that the leg is stepping on the ground one step cycle and stepping in a hole the next. In this case the sensor outputs for both step cycles look different from each other, yet both are correct. But there's also the case where the robot's leg sensors say the leg is stepping on the ground when in reality the leg is stepping in a hole. The robot does not know whether its sensors are reflecting reality or not!

When should the robot believe its sensors? Hannibal determines the reliability of its leg sensors by evaluating the output of its leg sensors in the context provided by both the time history of the leg motion and the output of complementary leg sensors. To illustrate this idea more clearly, imagine the following: you wake up in the morning and you want to know what the temperature is outside. You look at the thermometer outside your window and it reads 90 degrees Fahrenheit. However, you feel the window and find it's ice cold. The temperature sensors in your hand complement the thermometer reading. From evaluating the reliability of the thermometer in the context provided by the temperature time history and complementary temperature sensors in your hand, you conclude the thermometer is broken and ignore it. Hannibal essentially goes through the same process to determine the reliability of its sensors.

Sensor failure recognition is performed using two methods. The first method exploits the context provided by the time history of the leg motion. Remember, the robot does not know what the correct sensor behavior is for a given step cycle. However, the robot does know the *plausible* leg motions because the plausible leg motions are the behaviors that have been programmed for the leg. We call the set of plausible leg motions the *model.* If the leg sensors reflect a plausible leg motion, i.e. they agree with the model, then the robot has some confidence that the sensor is working. However, we could still have the case where the robot's sensors do not reflect reality although they reflect a plausible reality. For instance, a sensors could say the robot is stepping on the ground when it is really stepping in a hole. To overcome this problem, the second method exploits the context provided by complementary sensors. If reliable complementary sensors agree with the sensor in question, i.e. they confirm the robot is stepping on the ground, then the system has more confidence that sensor. The confidence level in a sensor is reflected by a *pain parameter* affiliated with that sensor. The pain level is the inverse function of the confidence level.

Sensor Model

It is possible to model sensor behavior if the behavior of the leg is known. Rotational potentiometers measure leg joint angle, strain gauges measure leg loading and linear potentiometers measure foot loading. Hence the motion of the leg and the leg's interaction with the environment directly affect sensor output. To model plausible sensor behavior, we first classify the leg behavior in terms of states. Each phase of the step cycle is divided into four possible leg

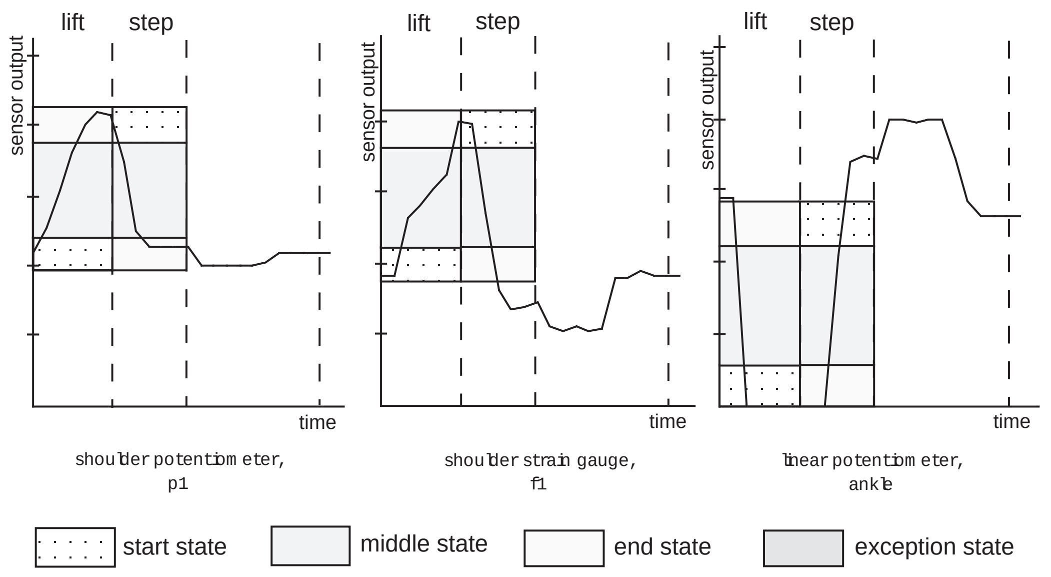


Figure 6.5: The relationship of sensor output to step cycle phase and leg state for the sensors used by the ground-contact virtual sensor.

states: S_{start}, S_{middle}, S_{end}, and $S_{exception}$. The S_{start}, S_{middle}, and S_{end} states account for the typical behavior of the leg. The $S_{exception}$ state accounts for leg-terrain interactions that may occur during that phase but typically don't. For example, during the step phase S_{end} corresponds to ground contact and $S_{exception}$ corresponds to stepping in a hole. We assume the leg behaves as programmed (unless a catastrophic failure occurs), so the constraints on the transitions between leg states for each phase of the step cycle are known. A set of sensor values corresponds to each leg state. From this sensor-state relationship, we derive a model for plausible sensor behavior. The sensor model consists of the expected transitions between sets of sensor values given the plausible transitions between leg states. The transformation from leg states to sensor values were derived experimentally by observing sensor values as the robot walked through its environment. Figure 6.5 shows how sensor output varies with step cycle phase and leg state for various sensors.

A process is written for each sensor that classifies the sensor's values into leg states. These processes are called *sensor-state processes*. There can be a several to one mapping between sensor values and leg states, so there can be ambiguity as to which state the leg is in given a sensor value. During the step phase, for example, the same ankle displacement value could indicate S_{start}, S_{middle} or $S_{exception}$ since the ankle is not compressed for all these states. However, the constraints on the transitions between leg states reduces this ambiguity. A properly functioning sensor's output indicates a sequential and timely transition between leg states such that $S_{start} \rightarrow S_{middle} \rightarrow S_{end}$. The constraint on when $S_{exception}$ occurs depends on the phase. For example, $S_{exception}$ of the step phase indicates the leg has stepped in a hole, so it occurs after S_{start} and S_{middle}. If a sensor's output deviates from these leg state transitions, the sensor is likely to be broken.

This approach to modeling the sensor behavior makes several assumptions. As mentioned earlier, it assumes the leg moves as programmed. This is a reasonable assumption since the controller treats a leg as broken if it fails to move as programmed. Consequently, further monitoring of the sensors is not necessary. It assumes the sensor models are tuned to the robot's environment (i.e. if Hannibal were to walk in mud the sensor behavior may not look plausible although the sensors are ok). This is a reasonable assumption since robots are typically designed to perform in a given environment. It also assumes that failed sensor behavior and reliable sensor behavior do not look alike (failed sensors aren't trying to fool you). It also assumes the reference values used to compute the ranges are correct. Therefore the sensor reference values used by the models must be updated to compensate for uncalibration.

Monitoring Individual Sensors

Each sensor has a process that monitors the sensor performance. These processes are called *sensor monitor processes* and they exploit the context provided by the time history of the leg motion. Figure 6.6 illustrates these processes. Each sensor's monitoring process uses the corresponding model of acceptable sensor-state transitions to detect sensor failures. Essentially, if a sensor's behavior does not reflect plausible behavior, then the confidence in that sensor decreases. Each process monitors the transitions between states by recording when each state occurs. A clock variable is incremented every 0.1 second during each step cycle phase, and it is initialized to zero every transition between step cycle phases. Each sensor monitoring process records this clock value in the following manner: when the sensor value indicates S_{start} the clock value is written in T_{start}, when the sensor value indicates S_{middle} the clock value is written in T_{middle}, and when the sensor indicates

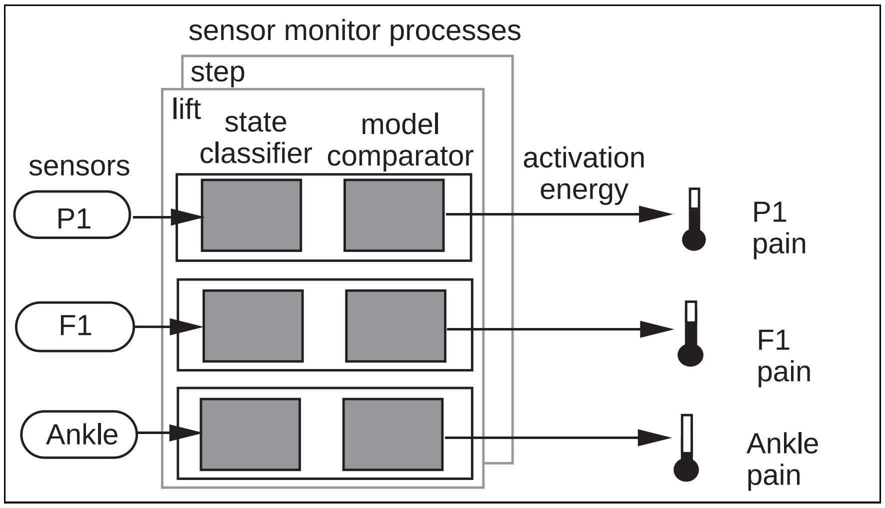


Figure 6.6: The sensor monitor processes for the ground-contact virtual sensor and step-in-hole virtual sensor.

S_{end} the clock value is written in T_{end}. If the sensor value indicates $S_{exception}$, the appropriate time variable is updated according to when the exception is expected to occur. For example, during the step phase the process updates T_{end} when $S_{exception}$ occurs because the possibility of stepping in a hole occurs at the end of the step phase. Whenever a time variable is incremented, the monitor process examines the contents of the time variables and checks that a timely and sequential transition of leg states is upheld. If the sequential and time constraints of state transitions are upheld, the process inhibits that sensor's pain parameter, otherwise it excites it. Figure 6.7 presents run-time data for the sensor monitor processes of the vertical force sensor.

Monitoring the Consensus of Sensors

Each step cycle phase has a *consensus monitor process* that monitors agreement between complementary sensors. Figure 6.8 illustrates the consensus monitor processes. The sensor-state processes categorize sensor values into leg states; this provides a common measure for comparing sensor behavior. The consensus monitor processes use discrepancies between sensor-state values of complementary sensors to detect sensor failures. As each leg moves though its step cycle, each sensor casts one vote for each leg state that corre-

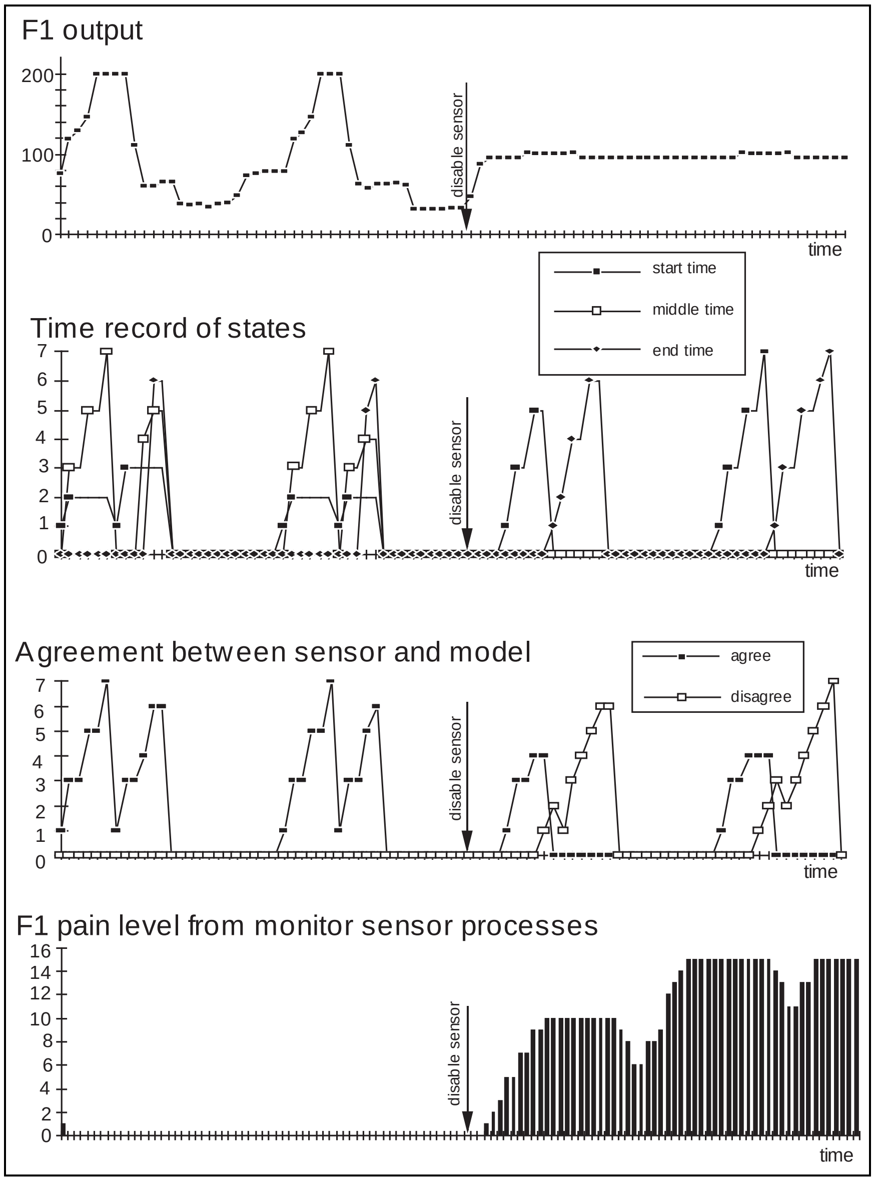


Figure 6.7: Response of the sensor monitor processes to vertical force sensor failure. Once the sensor fails, the actual sensor behavior frequently disagrees with the modeled sensor behavior. Consequently, the sensor's pain level increases.

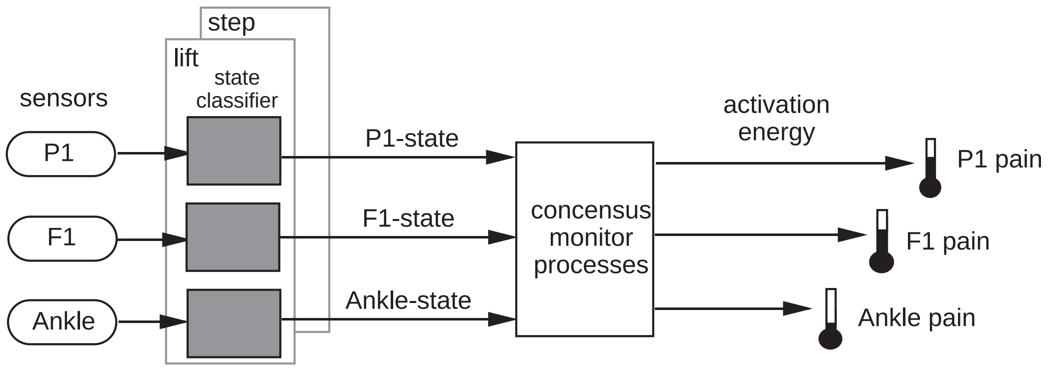


Figure 6.8: The monitor consensus processes for the ground-contact virtual sensor and the step-in-hole virtual sensor.

sponds to its value. For example, `ankle sensor = 0` corresponds to S_{start}, S_{middle}, or $S_{exception}$, so the ankle sensor votes for each of these states. Every 0.1 second, the consensus monitor processes tally the votes for each state and filter out votes for implausible states. Constraints on the motion of the leg determines the plausible states of the leg. For example, given the last state was S_{start}, the current state could either be S_{start} (provided the leg has not been in this state too long), or it could be in S_{middle}. Hence S_{start} and S_{middle} are plausible states whereas S_{end} is an implausible state. The actual state is taken to be the plausible state with the largest number of votes. If there is a tie between plausible states, the actual leg state is taken to be the sequentially higher state. Each time a leg state is elected, each sensor's votes are compared with the newly elected leg state. If a sensor voted for the elected state the consensus monitor process inhibits that sensor's pain parameter, otherwise it excites it. Figure 6.9 presents run-time data for the consensus monitor processes for the ground-contact virtual sensor.

Injury Agents

An *injury agent* for each sensor determines whether the sensor is working or broken by monitoring the sensor's pain level (figure 6.10). As described above, each sensor's sensor-monitor process and consensus monitor process excite its pain parameter when a discrepancy occurs, and inhibit the pain parameter when no discrepancy occurs. The level of the pain parameter

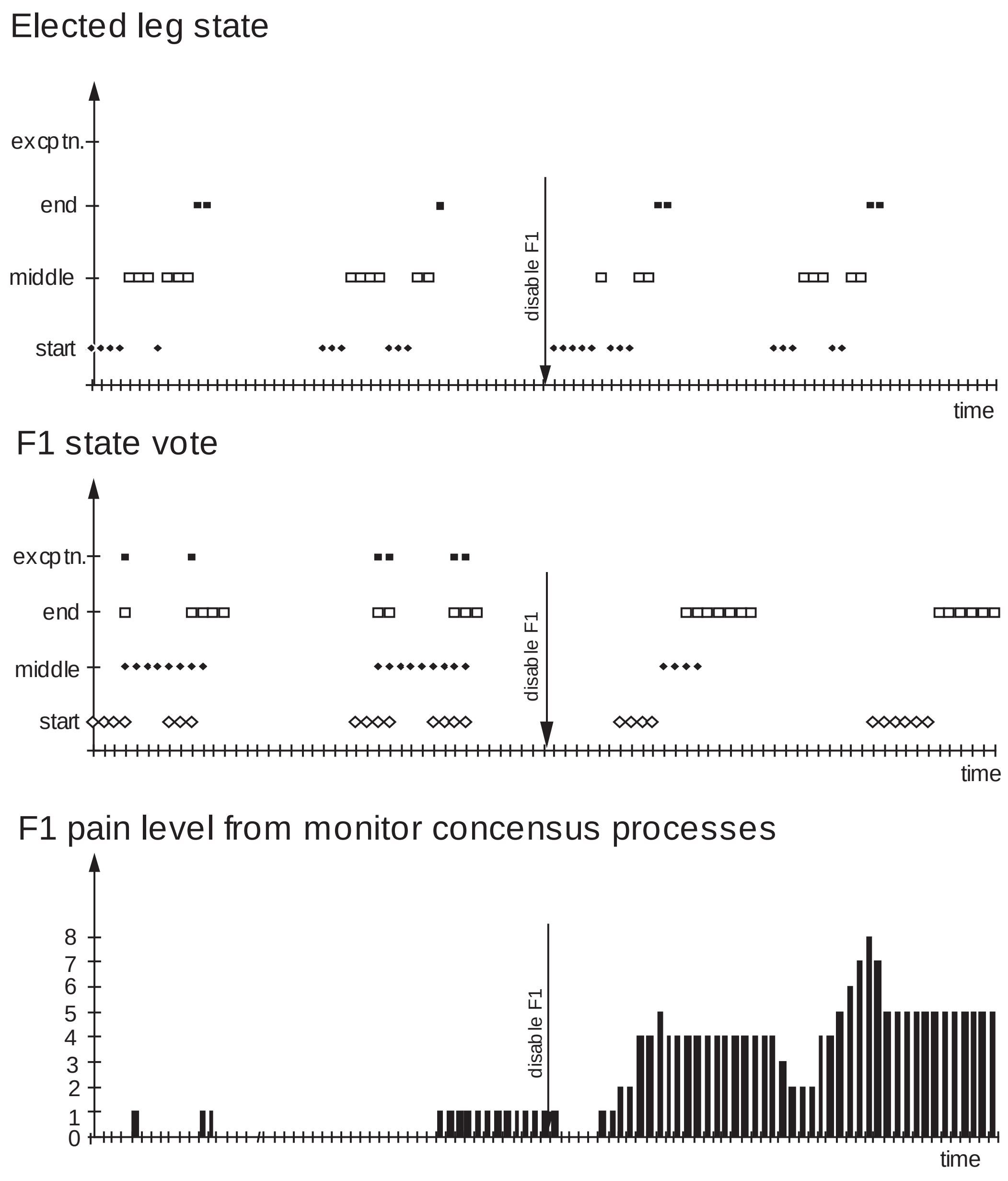


Figure 6.9: Response of the monitor consensus processes to vertical force sensor failure. Once the sensor fails, it disagrees frequently with the complementary sensors. Consequently, the sensor's pain level increases .

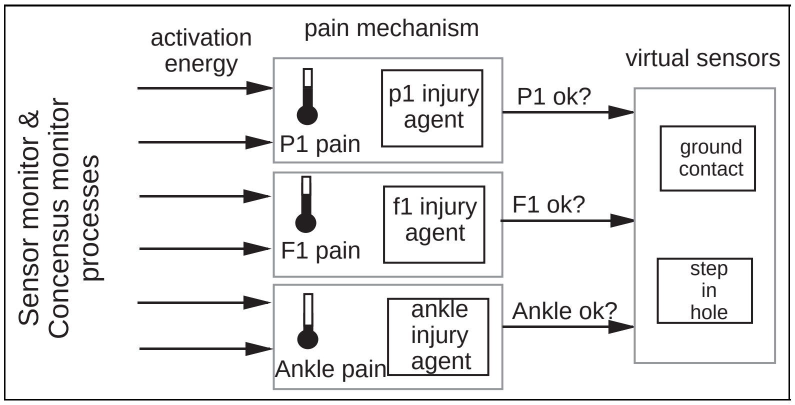


Figure 6.10: The injury agents inform the virtual sensors about which sensors are working and which sensors are broken.

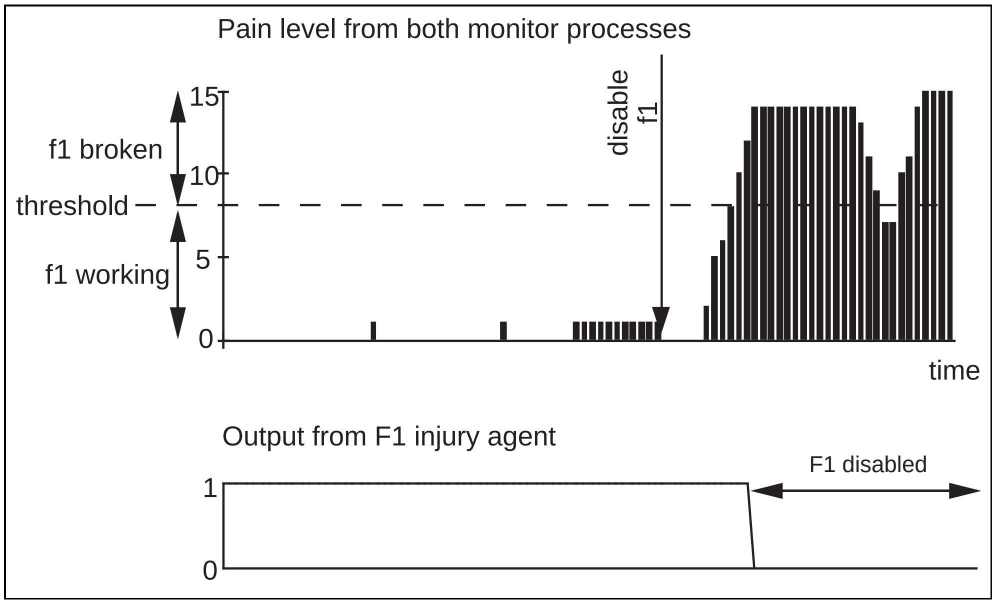


Figure 6.11: The injury agent of the vertical force sensor determines when the sensor has failed. The output of the injury agent is “1” when the sensor is working and “0” when the sensor is broken. It classifies the sensor as broken once the pain level exceeds the threshold value (the maximum pain level is 15).

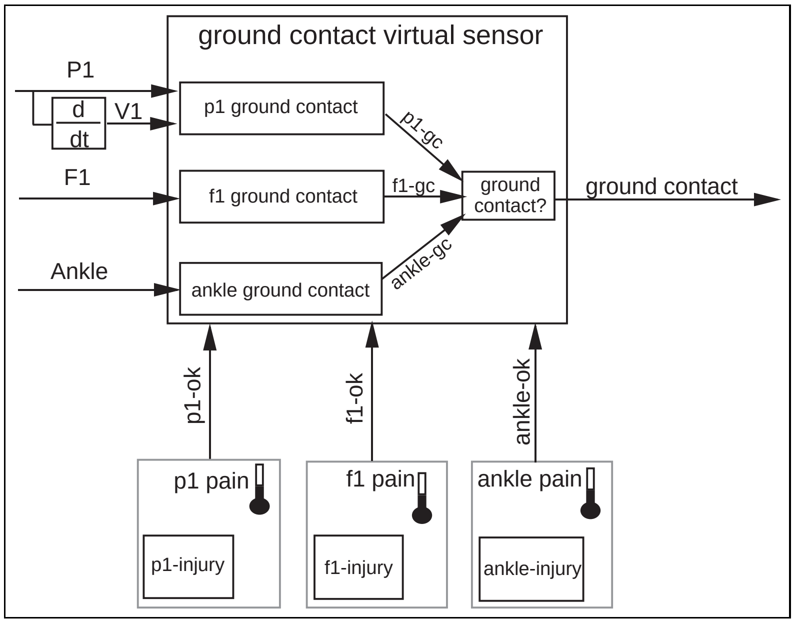


Figure 6.12: Illustration of a robust ground-contact virtual sensor.

increases upon excitation and decreases upon inhibition. The injury agent compares the sensor's pain level with a threshold value. If the pain level exceeds the threshold, the injury agent declares the sensor is broken, and when the pain level is below threshold, the injury agent declares the sensor is working. See figure 6.11.

6.5.2 Masking

The masking processes are responsible for removing the effects of local faults so that these faults do not affect high level behaviors. Masking is performed by the virtual sensors. A minor form of masking is also performed by the consensus monitoring processes. The injury agents continually inform the virtual sensors and consensus monitor processes about which sensors are functional or broken. Once informed of a broken sensor, the masking processes within the virtual sensors and consensus monitor processes remove the effects of the broken sensor.

To uphold the integrity of the elected leg state, the consensus monitoring processes disregard information from broken sensors. Provided the elected

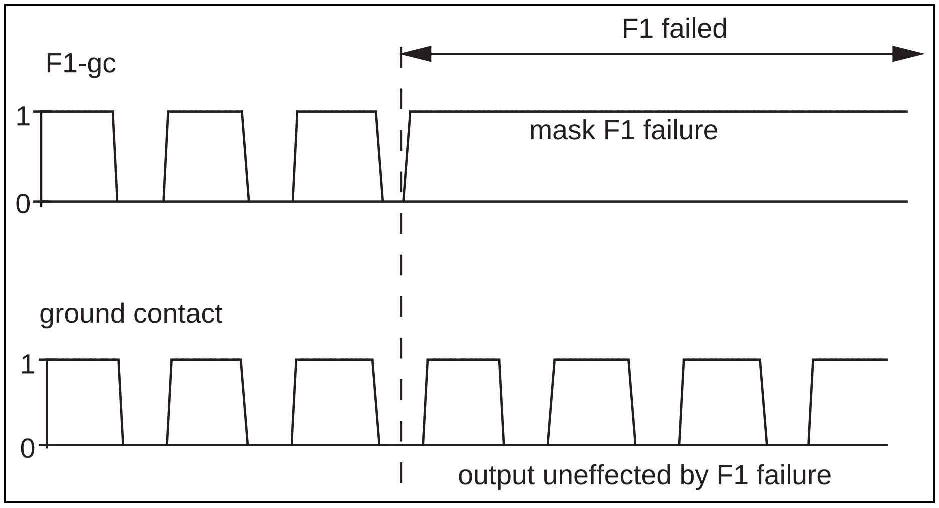


Figure 6.13: The virtual sensor continues to sense ground contact reliably despite the failure of the vertical force sensor (ground-contact = 1 corresponds to "true" and ground-contact = 0 corresponds to "false").

leg state is correct, the consensus monitor processes can detect valid disagreements between a sensor's leg state vote and the actual leg state. However, the wrong leg state could be elected if votes from broken sensors are honored. Therefore, if a sensor is faulty, the sensor consensus process removes the broken sensor's votes in the leg state election process. In this way, the leg state is determined only by functional sensors, and the result remains reliable despite failures.

The virtual sensors mask sensor failures by disregarding information from failed sensors. To show how this is implemented, we use the ground-contact virtual sensor as an example (see figure 6.12). The ground-contact virtual sensor uses information from the shoulder potentiometer, the ankle displacement potentiometer, and the vertical load strain gauge. Each sensor value is passed through a filter. The filter outputs `true` if the value satisfies its condition for ground contact, otherwise it outputs `false`. The filtered results are sent to a decision process that combines these results to determine whether the leg is contacting the ground or not. This process also receives inputs from the injury agent of each sensor. If an injury agent declares its sensor is broken, the decision process ignores the broken sensor in computing the output. Consequently the final ground-contact decision is made only by

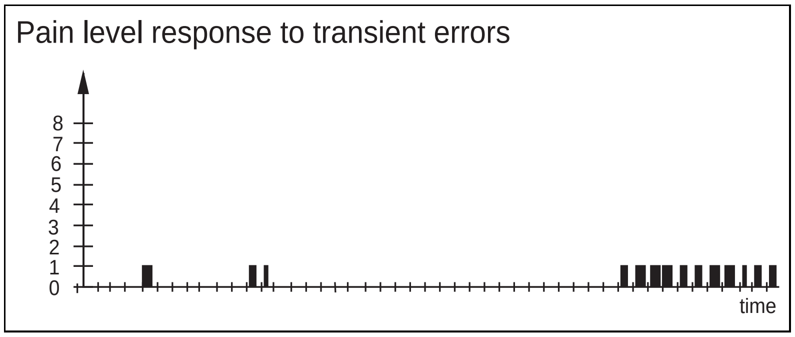


Figure 6.14: The effect of transient errors are filtered out by the pain mechanism.

functional sensors. Figure 6.13 presents run-time data of the masking process performed by a ground contact virtual sensor.

6.5.3 Recovery

The purpose of recovery is to return the system to an operational state once components fail. The new operational state should have as many of the original resources available as possible, and the transition to this new state should have minimal impact on normal system operation. We want the system to recover from transient errors as well as permanent errors. Transient errors result from occasional erroneous sensor values or sensor drift. Permanent errors result from sensor failure. Recovery takes three forms: retry addresses transient errors, dynamic recalibration addresses sensor drift, and reconfiguration addresses permanent failures.

Retry

Erroneous sensor values are filtered out by the pain mechanism. In essence, the pain mechanism provides a means for "retrying" a sensor if it produces a bad value. Instead of calling a sensor broken after it produces an erroneous sensor reading, the pain mechanism continually adjusts the pain level of the sensor. The pain threshold of each sensor is set such that a series of errors must occur before the pain level rises above threshold. However, since the error is transient, the sensor displays normal behavior during subsequent cycles, and the pain level diminishes. In effect, occasional errors are averaged

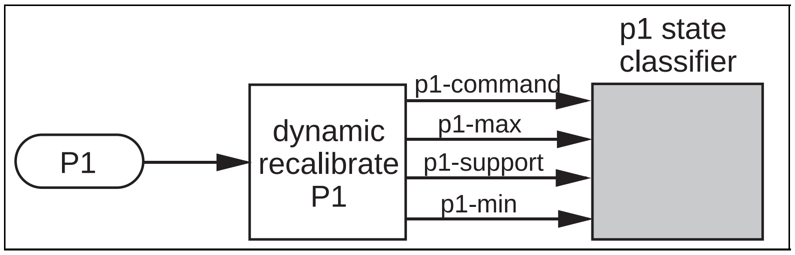


Figure 6.15: The updated reference values are sent to the sensor-state processes.

out and carry little weight for determining whether a sensor is functional or broken. Figure 6.14 illustrates the pain level's response to spurious sensor values.

Dynamic Recalibration

Dynamic recalibration processes are written for each sensor. These processes update the reference values used by the sensor model processes. Examples of reference values are `p1-max`, `p1-min`, `f1-max`, `f1-sNh`, `ankle-rest`, etc.. These values are used to compute the state transition values for the sensors. It is important to deal with sensor uncalibration because the sensor models become less accurate as the reference values become less accurate. The dynamic recalibration behaviors assume sensor gain remains constant; hence any uncalibration is attributed to DC offset only. This is a reasonable assumption given Hannibal's leg sensors.

To illustrate how the dynamic recalibration agents update the reference values, let's look at the *p1-recalibrate agent* as an example. The reference values for the shoulder potentiometer are `p1-max`, `p1-min`, `p1-support`, and `p1-command`. `P1-max` corresponds to the largest potentiometer value when the leg is lifted, `p1-min` corresponds to the smallest potentiometer value when the leg has stepped down (i.e. stepping into a hole), `p1-support` is the initial position the leg moves toward to find ground contact, and `p1-command` is the commanded lift position. Updating `p1-max` involves monitoring the potentiometer during the recover phase and finding the value at its highest point. The dynamic recalibration process sets this value to the updated `p1-max` value. Updating `p1-min` is somewhat problematic since the robot doesn't step into holes frequently. Consequently, the `p1-min` value could sig-

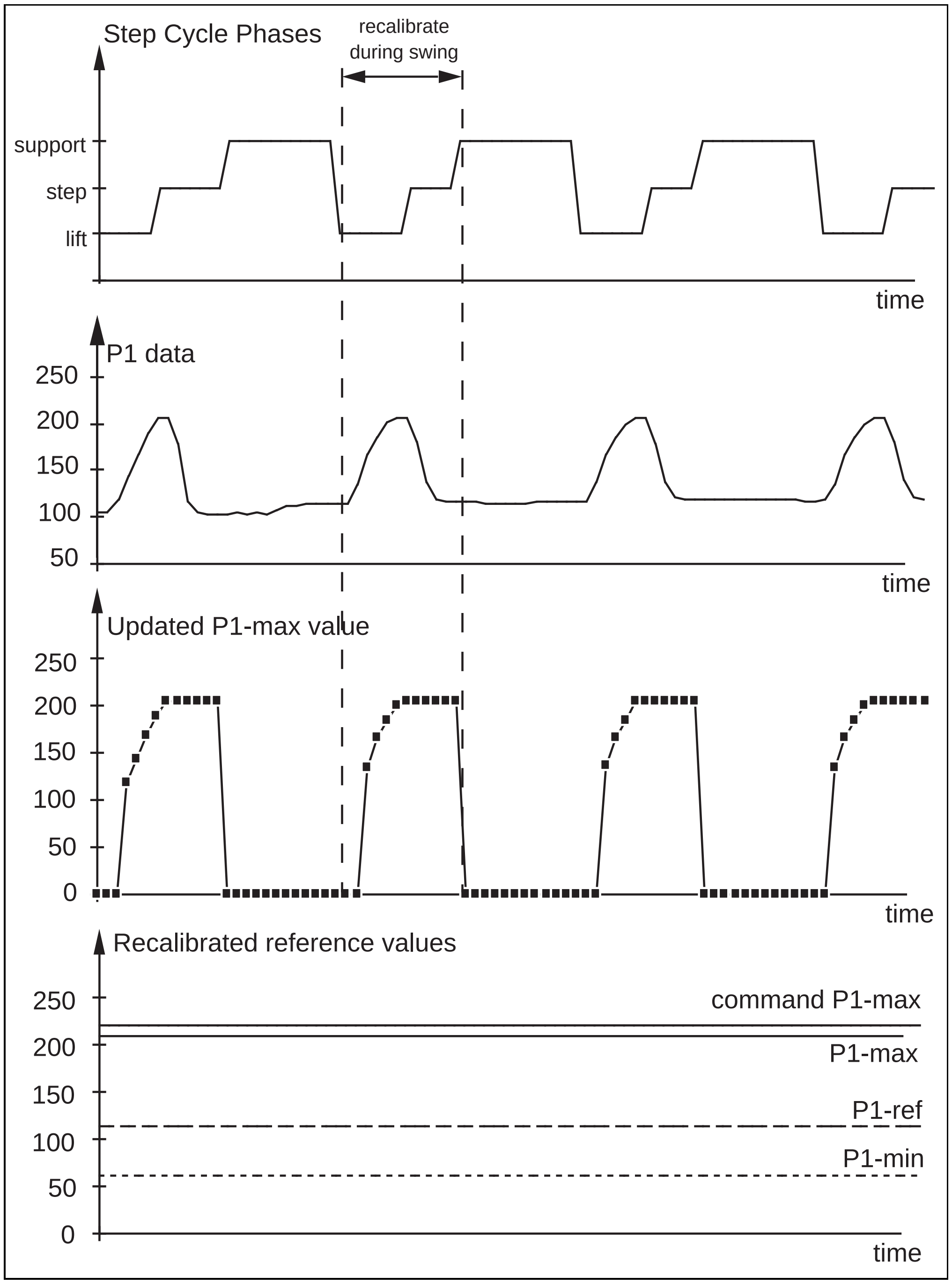
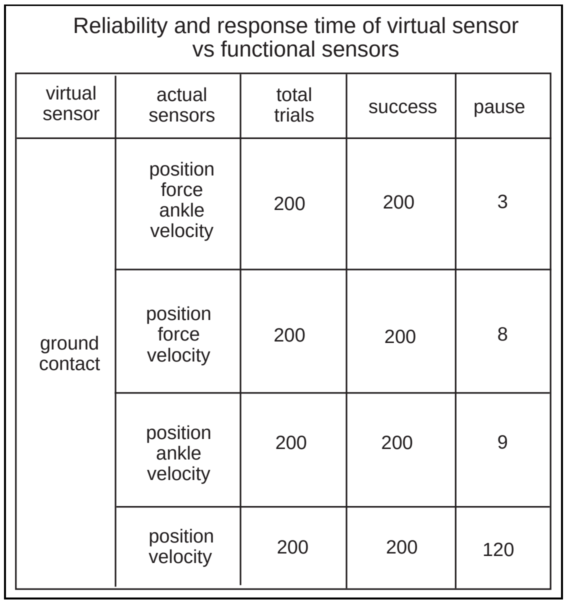


Figure 6.16: Dynamic recalibration takes place during the recovery phase.



Reliability and response time of virtual sensor vs functional sensors

virtual sensor	actual sensors	total trials	success	pause
ground contact	position force ankle velocity	200	200	3
	position force velocity	200	200	8
	position ankle velocity	200	200	9
	position velocity	200	200	120

Figure 6.17: Performance of the ground-contact virtual sensor as a function of sensors used. The reliability is maintained at the expense of response time as fewer sensors are used. The slower response time is reflected by the 'pause' column. The robot does not proceed to the next step cycle unless all stepping legs attain ground contact. Thus, if the ground contact decision takes longer, the robot waits an additional time interval between steps. This reduces the walking speed of the robot.

nificantly drift before the p1-recalibrate agent has the opportunity to update the `p1-min` value. To overcome this difficulty, the p1-recalibrate agent computes the updated `p1-min` value from the updated `p1-max` value and sample `p1-max` and `p1-min` values stored in memory (these sample values are determined experimentally). Once updated, the p1-recalibrate agent sends these values to the shoulder potentiometer model process (figure 6.15). Figure 6.16 presents dynamic recalibration data for the shoulder potentiometer.

Reconfiguration

As sensors fail we want system performance to degrade gracefully. In fact, we want to maintain the highest level of performance given the functional state of the robot. To accomplish this, the virtual sensors specifically tailor their use of sensor information to minimize the impact of sensor failures on virtual sensor performance. Virtual sensor performance consists of reliability and response time. We have found, by experimental means, that virtual sensors perform better as more sensors are used.

The virtual sensors can achieve faster response times without compromising reliability, provided sufficient quantity and type of sensors are used. To illustrate this, we look at the ground contact virtual sensor as an example. The ground-contact virtual sensor uses information from three types of sensors to ascertain leg loading: shoulder position, ankle displacement, and vertical force. Velocity and position are the only types of information the position sensor offers to determine loading. Loading can be inferred from the position sensor when the leg is pre maturely stopped above its lowest possible position. The force and ankle sensors directly measure loading and do it faster than the position sensor - the leg does not have to stop moving before they signal loading. Thus, if only the position sensor is working, the leg must come to a complete stop in the vertical axis to satisfy the condition for ground contact. However, if a force or ankle sensor is working, the condition for the position sensor can be relaxed so that it is satisfied sooner. In this case, the position sensor satisfies the condition for ground contact if either the downward velocity is sufficiently small or zero. Similarly, if all sensors are working, the conditions for the force and ankle sensors can be relaxed as well. Hence, the response time of the ground-contact virtual sensor can be sped up if more sensors are used without compromising reliability. Figure 6.17 presents our results for how performance relates to the number of sensors used for a ground-contact virtual sensor.

6.5.4 Reintegration

The robot's performance is enhanced if more sensors are used. The purpose of reintegration is to reincorporate repaired sensors so the robot uses the maximum number of reliable sensors. Repaired sensors are sensors that were previously faulty but behave well again. Reintegration is useful if a sensor is broken and then fixed, or if the sensor was incorrectly classified as broken. In either case, we call a sensor "broken" if it's pain level is above threshold, and we call it "repaired" if the sensor's pain level rises above threshold and then lowers below threshold.

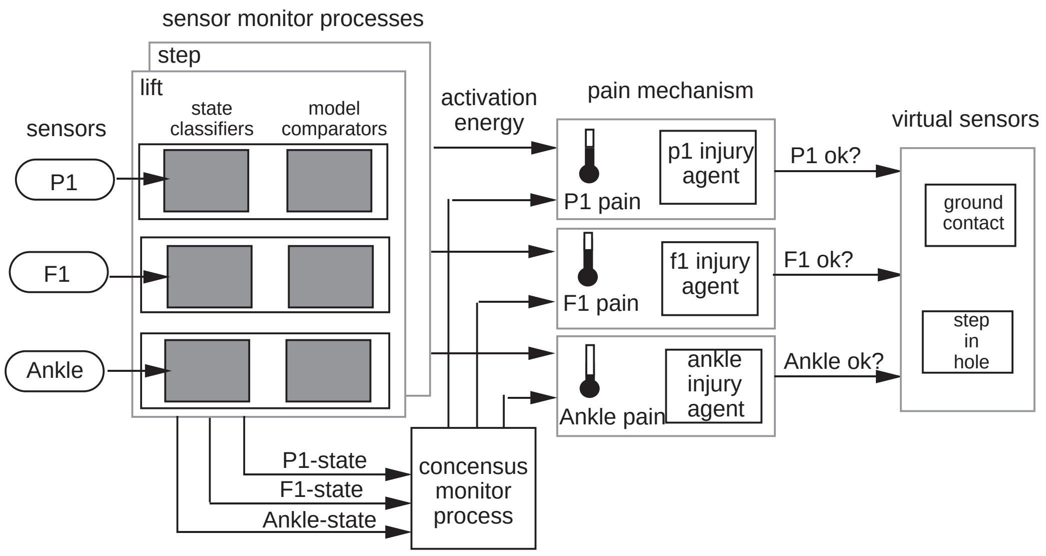


Figure 6.18: Both the sensor monitor processes and the consensus monitor processes assist in reintegrating repaired sensors.

Both the sensor monitor processes and sensor consensus processes induce reintegration of repaired sensors (figure 6.18). They accomplish this by inhibiting the pain parameter. If a failed sensor exhibits normal behavior, the sensor's behavior agrees with the modeled behavior again. Consequently, the sensor monitor process inhibits the pain parameter. Similarly, if a failed sensor exhibits normal behavior, the sensor behaves in consensus with other functional sensors again. Consequently, the sensor consensus process also inhibits the pain parameter. Eventually, the sensor monitor process and sensor consensus process lower the sensor's pain level below threshold. Once this occurs, the sensor's injury process tells the virtual sensors that the sensor is working. The virtual sensors respond by reincorporating the repaired sensor in computing their output. Hence, the influence of the repaired sensor is reintegrated into the control system. Figure 6.19 presents run-time results for reintegration of the ankle sensor.

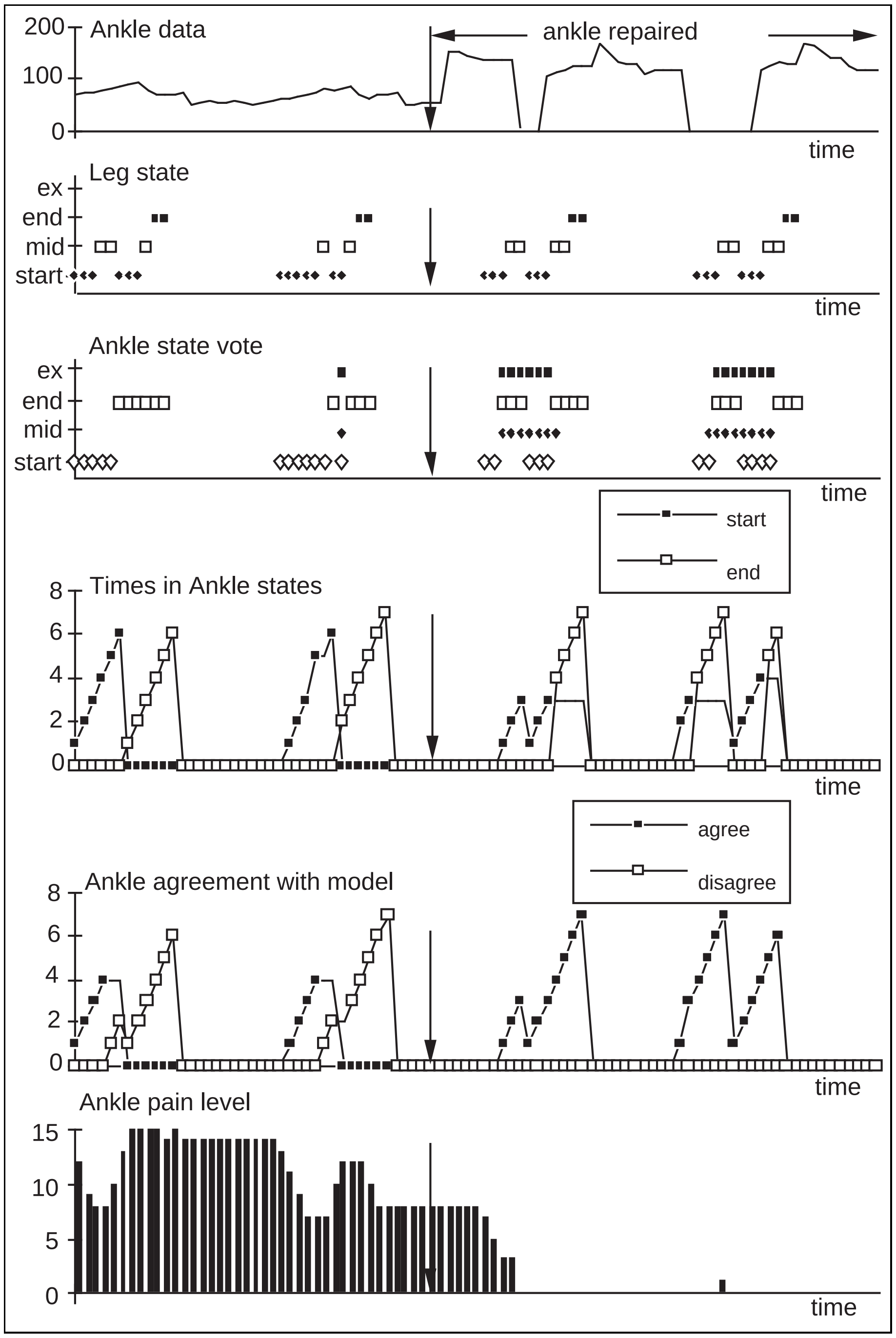


Figure 6.19: Once the ankle sensor is repaired, the sensor monitor processes and consensus monitor processes reduce the pain level of the sensor

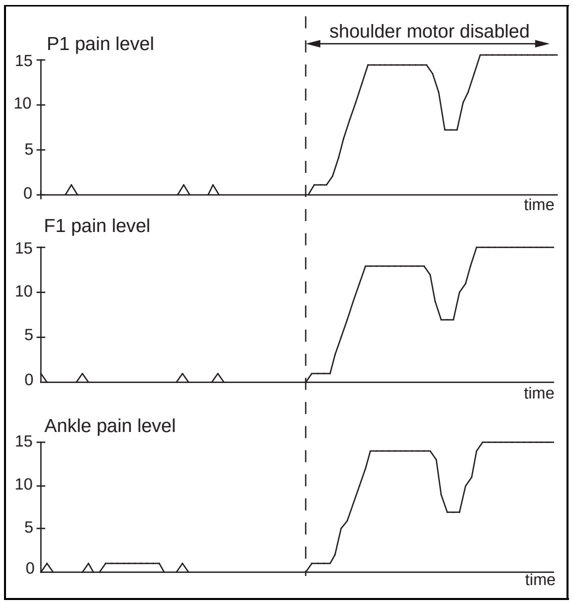


Figure 6.20: When the shoulder actuator fails, the shoulder position sensor, the ankle sensor, and the vertical force sensor look as if they have failed.

6.5.5 Catastrophic Failures

Global failures are detected in low level control, but must be compensated for in the high level control. Hip actuator failures, hip potentiometer failures, shoulder actuator failures, and shoulder potentiometer failures are global failures. These failures effectively prevent the leg from behaving as programmed. This is obviously the case if an actuator fails. If a joint angle potentiometer fails, the servo processors have no way of knowing the positional error, so they cannot servo the actuator. In the event of a global failure, the leg is rendered not usable, so the robot must modify its behavior to function with fewer legs.

Detection

Global failures are detected by the same processes used for detecting local failures. Potentiometer failures are found using their respective sensor monitor process and consensus monitor process. Actuator failures are inferred through concurrent failure of sensors whose behavior depends on that ac-

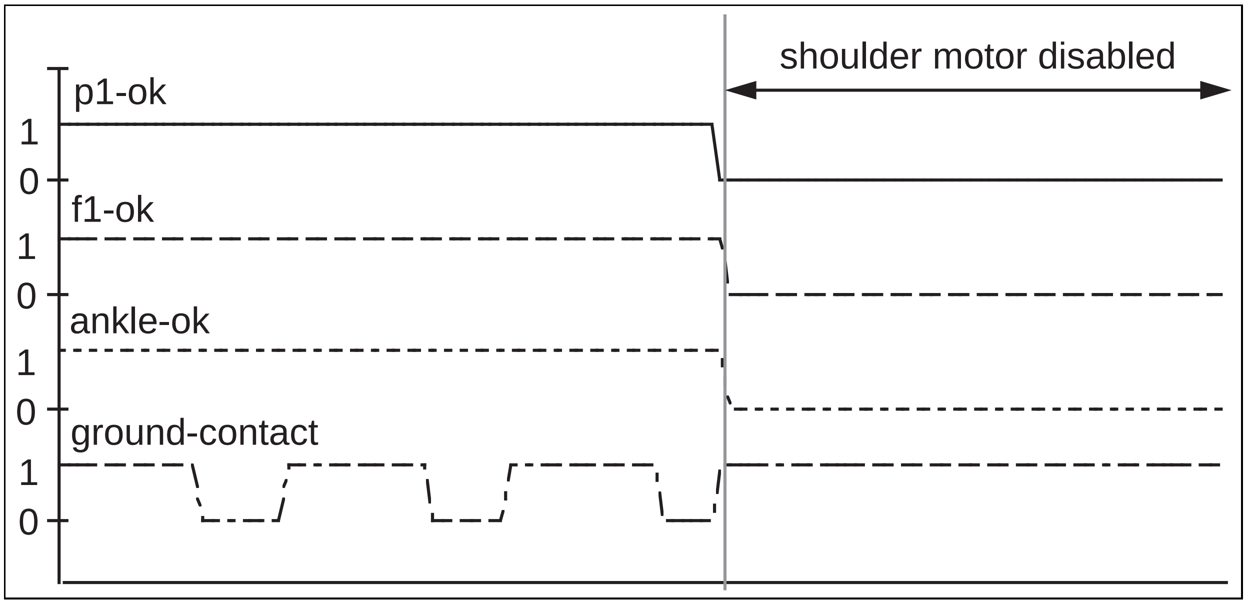


Figure 6.21: If the shoulder actuator fails, the ground-contact virtual sensor masks its output from the high level control.

tuator working. Once an actuator fails, all dependent sensor models are invalid. Consequently, the corresponding sensors look as if they have failed even though this may not be the case. If the shoulder actuator fails, for example, the ankle sensor, vertical loading sensor, and shoulder position sensor all appear broken to the monitoring processes (figure 6.20). Once a global failure occurs, it is irrelevant whether the local sensors appear broken because the leg is not usable anyway. The detection of global failures can be reduced to detecting potentiometer failures only. This is the case since the monitoring processes detect joint angle potentiometer failure when either type of global failure occurs.

Masking

Once a leg fails, the output of its ground-contact virtual sensor is not valid. The ground-contact virtual sensors of the stepping legs influence when the next step cycle occurs. Each recovering leg inhibits the supporting legs from proceeding to the next step cycle until it attains ground contact. Thus,

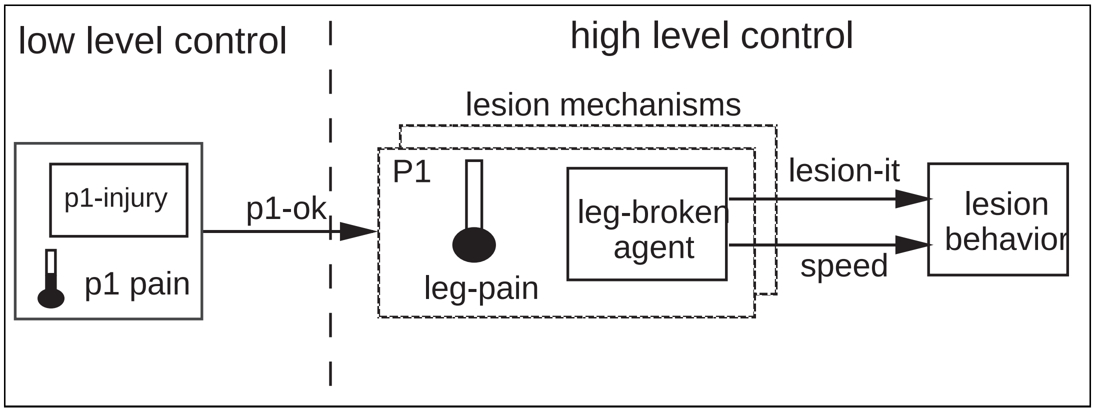


Figure 6.22: The leg-pain parameter and leg-injury agents are affiliated with high-level control.

it is important to mask the effect of non-valid ground-contact virtual sensors or else they may adversely affect the robot's gait. To prevent this from happening, the output of the ground-contact virtual sensor defaults to `ground-contact = true` for all broken legs (figure 6.21). By doing so, the effect of renegade ground-contact virtual sensors on the robot's gait is removed.

Recovery

Given a global failure, high level control agents compensate by lesioning the broken leg. Each leg has a lesion mechanism which is responsible for lesioning the leg once it suffers a global failure. Within high level control, each critical potentiometer has a leg-pain parameter and a leg-injury agent associated with it (figure 6.22). Each leg-injury agent receives messages from its corresponding low level potentiometer injury agent every 0.1 second. If a message indicates the potentiometer is broken, the corresponding leg-injury agent excites the appropriate leg-pain level. The leg-pain level automatically decays every three seconds. If a leg-pain energy level rises above the `lesion` threshold, the corresponding leg-injury agent activates the lesion behavior. The lesion behavior disables the leg and adopts a gait that is stable without the use of the damaged leg (figure 6.23). The lesion behavior is described in chapter 4.

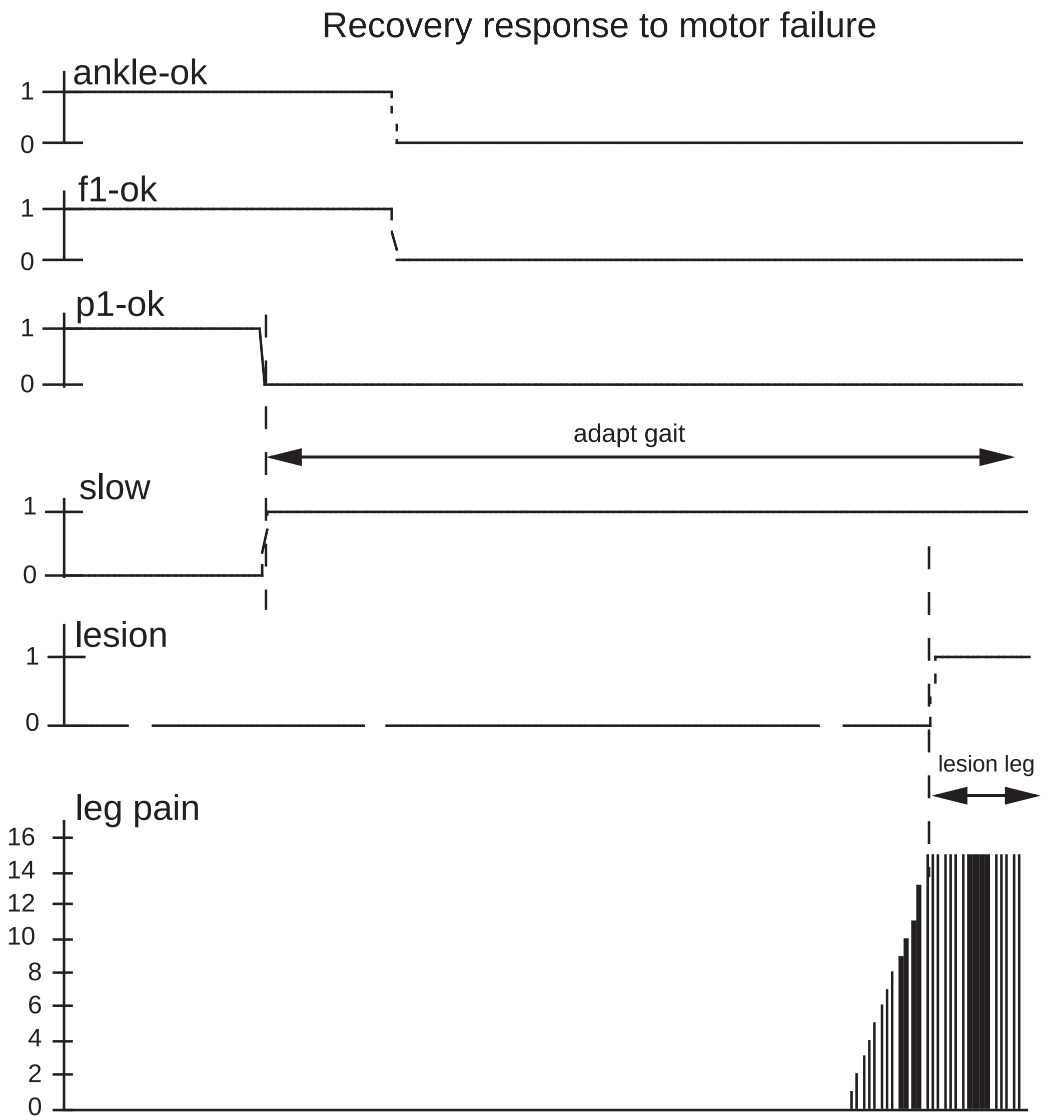


Figure 6.23: When the shoulder actuator fails, the system responds by lesioning the leg.

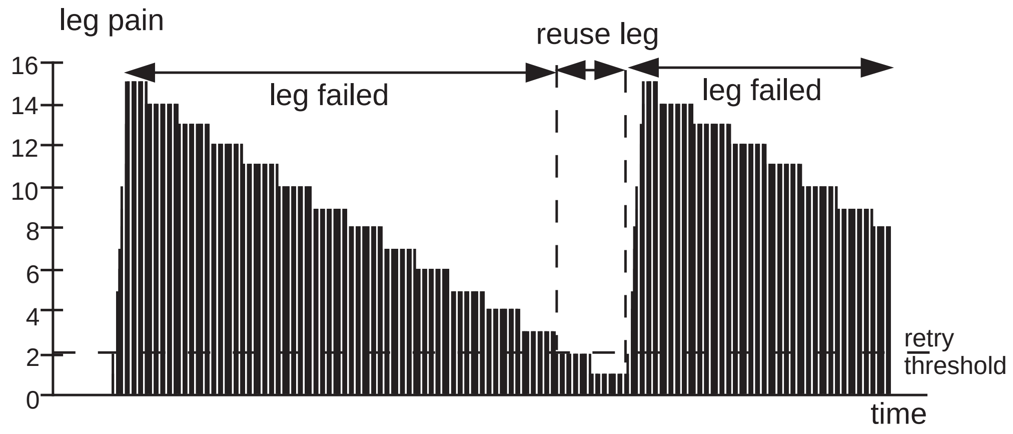


Figure 6.24: The system occasionally retries to use the leg. If it works again, the system reincorporates the use of that leg.

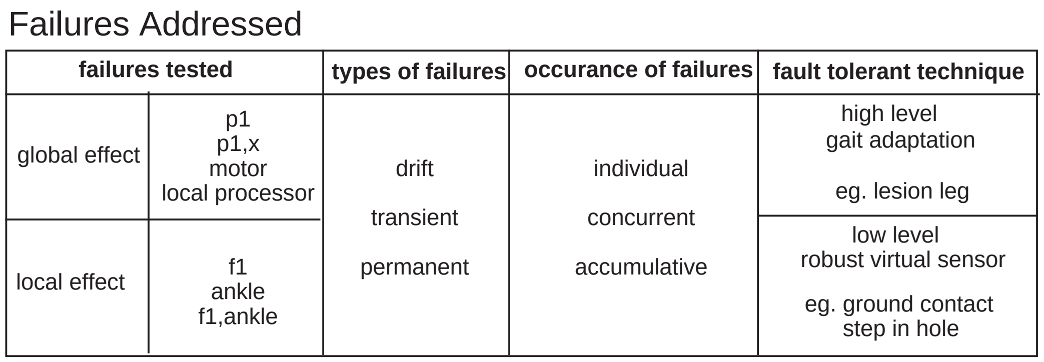
Reintegration

If the broken leg is repaired or the leg was wrongly determined to be broken, high level agents reintegrate the leg. This is accomplished by occasionally testing the leg to see if it is functional again. Leg reintegration is performed by the lesion mechanism. After the leg-pain level rises above the `lesion` threshold, the leg-injury agent is prevented from exciting the leg-pain level. Consequently, the leg-pain level decays slowly back towards zero. When the leg-pain level lowers to the `retry` threshold value, the system tries to use the leg and the leg-injury agent is allowed to increase the leg-pain level (figure 6.24). If the leg is still broken the leg-pain level raises above threshold, and the process repeats. If the leg works the next time it is tested, the leg-pain level decays to zero. Once this happens, the system acknowledges the leg is functional again, de-activates the lesion behavior, and resumes using the leg.

6.6 Performance

6.6.1 Tests

Several tests were conducted to determine the system's response to various types and combinations of failures. The tests involved inflicting the desired


Failures Addressed

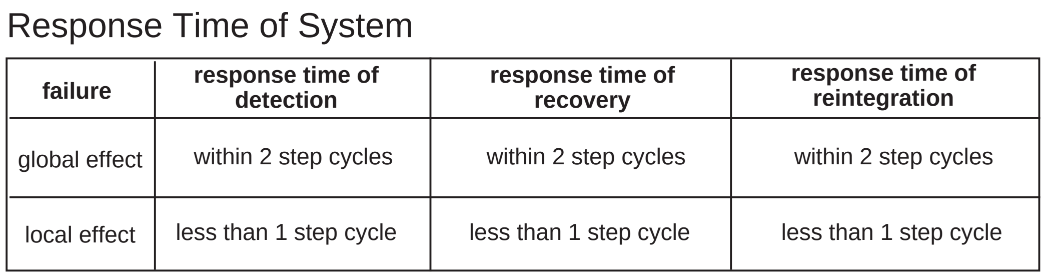
failures tested		types of failures	occurance of failures	fault tolerant technique
global effect	p1 p1,x motor local processor	drift transient permanent	individual concurrent accumulative	high level gait adaptation eg. lesion leg
local effect	f1 ankle f1,ankle			low level robust virtual sensor eg. ground contact step in hole

Figure 6.25: The system is tolerant of a variety of types and combinations of failures

fault and observing its effect on the robot's behavior while the robot walked through its environment. We tested sensors with a local effect and sensors with a global effect. Permanent sensor failures were inflicted by disconnecting the power, ground, or output wires to the sensors. Broken wires are the most common cause of permanent sensor failure on Hannibal. Sensor uncalibration was induced by adjusting the reference voltage offset of the sensor signals; uncalibration also occurred naturally over time. Transient errors are difficult to force but arose during the course of the tests. Different combinations of permanent errors were tested as well. We disabled sensors individually, concurrently, and sequentially over time. We repeated the concurrent and sequential tests by disabling the sensors in various permutations. Actuator failures were evoked by disconnecting the wires to the motor. On one occasion the shoulder actuator shaft sheared off, so we had the opportunity to test for motor shaft breakage as well.

6.6.2 Results

The results of the tests described above are presented in figure 6.25. As shown, the system is quite flexible and responds to a wide variety of types and combinations of failures. We conducted the tests by evoking various types and combinations of sensor faults on the left front leg (the same processes run on the rest of the legs). The tests were performed while Hannibal walked over flat terrain with holes and cliffs. Within this environment,


Response Time of System

failure	response time of detection	response time of recovery	response time of reintegration
global effect	within 2 step cycles	within 2 step cycles	within 2 step cycles
local effect	less than 1 step cycle	less than 1 step cycle	less than 1 step cycle

Figure 6.26: The fault tolerance processes have fast response times.

the interesting virtual sensors are the ground-contact virtual sensor and the step-in-hole virtual sensor. Consequently, we focused our tests on the sensors used by these virtual sensors and the actuators that affect the behavior of these sensors. Hence, the sensors and actuators involved in the tests are the shoulder potentiometer, ankle displacement sensor, vertical force sensor, and shoulder actuator. It is interesting to note that the system responds to the failure of a local leg processor as well. Once a local leg processor fails it cannot send the leg sensor signals to the main processor. Consequently all sensors look as if they have failed, and the system responds by lesioning the leg.

Figure 6.26 presents the response time of the system to recover from failures and to reintegrate repaired components. The system recovers from failures quickly enough such that the robot's behavior does not have to degrade to an unacceptable level before the failure is compensated for. It also reintegrates repaired components quickly so the system readily has access to its reliable resources. It is possible to tune the recovery-reintegration response time of the system by adjusting various parameters. The recovery-reintegration response time to local failures is tuned by adjusting the pain level threshold, the pain excitation gain, and the pain inhibition gain. For example, the response time is sped up by either lowering the pain threshold or increasing the excitation gain. Increasing the inhibition gain slows the response time and encourages the system to reintegrate components earlier. The recovery-reintegration response time to global failures is tuned by adjusting the `lesion` threshold of the leg-pain level, the `retry` threshold of the leg-pain level, the leg-pain excitation gain, and the leg-pain decay rate. For example, the recovery response time is sped up by increasing the leg-pain excitation gain or lowering the `lesion` threshold. The reintegration response time is increased by increasing the leg pain decay rate or increasing the `retry` threshold. Clearly a balance must be achieved between tuning the system for a fast response time while taking care not to make the system too sensitive to transient faults.

6.6.3 Evaluation

To evaluate the fault tolerant aspects of Hannibal's system, we discuss them in relation to the following topics:

Completeness of fault detection

The system successfully detects a wide assortment of common failures. It distinguishes between non-catastrophic failures and catastrophic failures. Regarding local failures, it recognizes which type of sensor has failed. Regarding catastrophic failures, the system recognizes potentiometer failures. Actuator failures appear as the massive failure of all sensors whose behavior depends on that actuator. Local processor failures appear as the massive failure of all sensors whose values are communicated to the main processor by that processor. When these types of catastrophic failures occur, the corresponding potentiometer(s) appear broken. Furthermore, all catastrophic failures evoke the same recovery procedure. Hence, the system only looks for potentiometer failures to detect catastrophic failures.

Fault coverage

The robot successfully recovers from a wide assortment of failures. The system recovers from sensor failures, actuator failures, and local processor failures. It addresses failures with a local effect as well as failures with a global effect. It handles transient errors, sensor uncalibration, and permanent failures. It compensates for various combinations of failures: failures that occur individually, concurrently with other failures, or accumulate over time. It tolerates these combinations of failures in various permutations.

Confinement of errors

The system effectively prevents sensor and actuator failures from adversely influencing robot behavior. To accomplish this, the detection processes monitor the sensor outputs and alert the system of failures as they arise. This nips the failure problem in the bud because the system can effectively compensate for failures once it knows when and what type of failures have occurred. For example, robust virtual sensors filter out the effects of non-catastrophic failures so the failures do not influence the robot's behavior. Potentiometer injury processes evoke the lesion behaviors when catastrophic failures occur, so the loss of the leg does not cause the robot to become unstable.

Response time to failures

The system successfully detects and recovers from failures before the robot's performance degrades to an unacceptable level. A fast response time is important for successfully implementing error confinement. After all, it does not do the system much good to implement the dection and recovery procedures in the low level control if it takes them a long time to respond to failures.

Extent of graceful degradation of performance

The recovery processes maintain the robot's performance at the highest level given the functional state of the hardware. Because the system purposefully recognizes failures, the recovery processes can specifically tailor the robot's use of sensors and actuators to minimize the effects of failures on the robot's behavior. At the virtual sensor level, the recovery processes tradeoff speed for reliability as the number of functional sensors decreases. At the locomotion level, the recovery processes trade off speed for stability as the number of usable legs decreases.

Availability of reliable resources

The reintegration processes reincorporate repaired components so the robot has access to all its reliable resources. This is important because the more sensors and actuators the system can use, the better its performance will be.

The reintegration response time is relatively fast so the system does not have to wait long before it can reuse repaired components.

Division of fault tolerance responsibilities among hardware and software

Our system implements all fault tolerance capabilities within software. We chose not to implement fault tolerance capabilities at the hardware level for weight and expense considerations. Consequently the code size was significantly increased. We estimate the fault tolerance processes to require the same amount of code space as the locomotion code and rough terrain code combined. In other systems it may make sense to make a more even tradeoff between code size and hardware weight and expense.

Survivability of the system

The survivability of the robot depends on many variables; hence, it is difficult to determine for our system without running extensive tests. Survivability is determined by how many and what kinds of failures the robot suffers. It also depends on the type of terrain the robot is walking over. However, the causes of these failures are diverse, and the rate at which these failures occur is not well understood at this time. Assuming the robot walks on flat terrain, the robot can locomote using only its shoulder and hip potentiometers. Hence, the *minimum* number of failures the robot can withstand before it can no longer walk in a stable fashion is two catastrophic failures on different legs provided the failures do not occur on the middle legs. This lower bound is determined by the physical constraints governing the stability of the robot. The *maximum* number of failures the robot can withstand before it can no longer walk in a stable fashion is 54 (48 non-catastrophic failures, 3 catastrophic failures on the left side middle leg, and three catastrophic failures on the right middle leg). Hence, it is possible for the robot to locomote with a small minority of its sensors working.

6.7 Comparison with other systems

Fault tolerance is relatively unexplored area of autonomous robot research. In this section, we compare our approach to related work in the field.

6.7.1 Robustness

The issue of robustness is commonly discussed in autonomous robot control architectures. Behavior based control architectures such as Brooks (1986) are robust because the loss of functionality of part of the controller (such as a layer of competence) does not result in a complete break-down of the controller. Chiel et al. (1992) argues the neural network locomotion controller implemented on the Case Western Hexapod is robust because the the loss of certain parts of the controller (such as sensor nodes) does not prevent the robot from walking. As another example, Chiel et al. (1992) argues that the controller is robust to the loss of a leg because the slowest gait the controller generates maintains stability with the loss of a leg. These arguments for robustness imply that robustness means that a system can still behave reasonably despite some loss of functionality of the controller.

Fault tolerance, in contrast, specifically deals with the detection and compensation of failures. Hannibal's controller purposefully recognizes failures and specifically compensates for them. In this way, the system maintains its best possible performance given the functional state of the system. For example, the controller presented in Chiel et al. (1992) does not tolerant of physical sensor failure. If a potentiometer physically failed, there would be no positional error, and leg could no longer be servo controlled. Their controller assumes the leg can still be servo controlled when a sensor node in their network fails. Similarly, their controller is not tolerant of leg failure because it does not detect when a leg has failed or automatically adapt the robot's gait to maintain stability. In contrast, Hannibal's controller is tolerant of these failures.

6.7.2 Replication of Hardware

Replication of hardware is commonly used to enhance hardware reliability. (Kabuka, Harjadi & Younis 1990) and (Lin & Lee 1991) present multiple processor architectures for robotic systems which can tolerate processor failures. Both papers parallelize the problem they want to solve (vision or inverse kinematics) and distribute the problem over the parallel network. If a processor fails the network reconfigures itself and redistributes the work load on the remaining processors. Fault tolerant computing systems presented in (Gray 1990) and (Siewiorek & Johnson 1981) exploit replication of VLSI hardware to achieve fault tolerance. The low cost of VLSI replication makes this approach feasible for this application. In contrast, complex systems such as the Boeing 757 and the Space Shuttle use replication of hardware to achieve fault tolerance at a significant expense.

Fault tolerance by replication is fairly straight forward. The system is made tolerant of a particular type of component failure by replicating it and ignoring the components that exhibit a minority behavior. Duplication of hardware/software increases the fault tolerance of the system, but it doesn't necessarily provide enhanced capabilities or performance over the non-replicated system. In contrast, the approach presented in this chapter has the advantage of using components with complementary yet different capabilities to achieve fault tolerance. Since the components have different capabilities, they can be used for different purposes. Hence the additional components not only enhance the fault tolerance of the system, but they also provide additional capabilities and performance beyond what the system would have were it to use fewer components.

6.7.3 Redundant Control Behaviors

Fault tolerance techniques using redundant sets of control strategies has been investigated in (Payton et al. 1992). We have previously discussed this approach in section 6.3.2, but we briefly repeat our discussion here for the reader's convenience. In this approach, the controller is designed with redundant strategies to perform a task. A performance model exists for each strategy, and a failure is detected if the behavior's actual performance is worse than the behavior's expected performance. If the first strategy tried doesn't suffice, the controller eventually tries another strategy. The controller goes through its repertoire of strategies until it finds a strategy with acceptable performance instead of unsuccessfully trying the same thing over and over.

This high level approach is not well suited for hardware failures. First, it does not specifically address the cause of the problem, it only addresses the symptoms. It may take several tries before the controller finds a strategy that works. In contrast, our approach purposefully recognizes the type of failure and compensates for the failure by specifically tailoring its use of reliable components to minimize the effect of the failure on the robot's performance. Second, the high level approach inherently requires that the hardware errors manifest themselves in the robot's behavior before the control system can detect something is wrong. Ergo, the performance of the infected behaviors must degenerate to an unacceptable level before the controller takes corrective action. In contrast, our approach detects and masks the effect of failures within the low level control. This prevents the hardware failures from manifesting themselves in high level control where they degrade the robot's performance to an unacceptable level.

6.8 Contributions

There are several contributions this work makes towards achieving fault tolerant autonomous robot systems. First, this work presents an autonomous robot which can purposefully recognize sensor failures quickly and reliably. Second, the robot specifically and dynamically tailors its use of sensors and actuators to minimize the impact of failures on its performance. Third, the robot dynamically reintegrates repaired components to enhance its performance. I have tested the capabilities of this system by physically disabling and enabling the robot's sensors and actuators. I have found the system recognizes and compensates for sensor and actuator failures with a fast response time. It tolerates a variety of sensor failures such as uncalibration, erroneous readings, and permanent failures. It also tolerates various combinations of failures such as individual failures, concurrent failures, and accumulative failures. It handles minor failures such as a broken ankle sensor as well as catastrophic failures such as a broken leg. This is the only autonomous robot we know of with this level of fault tolerant capabilities.

Chapter 7

Conclusion

7.1 Review of Significant Results

7.1.1 Control of Complex Robot

Expanding the capabilities of robots tends to make them more complicated. Equipping robots with more sensors increases the quantity and reliability of information the robot can extract from its environment. The robot can use this information to help it behave more intelligently. Providing robots with more actuators increases the physical capabilities of the robot (arms, hands, improved locomotion, active vision, etc.). Programming the robot with more behavioral capabilities makes it more flexible and versatile in the tasks it can perform. As the field of autonomous robot control advances, we will want to design robots with greater capabilities to perform more challenging and comprehensive tasks. Understanding how to manage complexity of autonomous robot control is a relevant issue which will be present in the future.

We demonstrated the subsumption approach (Brooks 1986) is effective for designing an intricate controller, and for controlling a robot of Hannibal's complexity. The current implementation consists of approximately 1500 concurrently running processes performing diverse tasks such as terrain sensing, rough terrain locomotion, and fault tolerance. In building a controller of this complexity from the bottom-up, we have demonstrated the modularity of the design. The controller scales well since Hannibal operates in real-time despite the large number of concurrently running processes.

We demonstrated the *local control with cooperation* paradigm is an effective means of controlling a robot of Hannibal's complexity. We tested this approach by running Hannibal on various test terrains. The control was quite flexible as the legs were able to handle obstacles local to themselves simultaneously with the other legs. Furthermore, through communication each leg behaves as a scout for the other legs and alerts them of common dangers. In this manner, a high level terrain view local to each leg is shared with the other legs. Communication between legs also enabled them to act as a team. By working together each leg accomplishes more than it could if it had to fend for itself. Finally, inter-leg communication enables the legs to maintain a unified effort, and the inter-leg priority scheme maintained the unified effort even when behaviors of different legs were in conflict.

Since all of Hannibal's control code runs on a single processor, the controller code can only be scaled until its requirements exceed what the master processor can provide. A logical step would be to implement behavior control on multiple processors. The scalability of this architecture is limited by communication bandwidth of the behavior code processors. This may not be a stumbling block for this implementation since the subsystems perform local functions the majority of the time and communicate with the other systems on a relatively limited basis.

7.1.2 Fault Tolerance

Fault tolerant behavior is important for autonomous mobile robots whether their task is grand or mundane. Building autonomous mobile robots capable of performing tasks in environments that are either too dangerous or unsuitable for humans has been a long term goal of the field. Planetary exploration with autonomous micro-rovers is one such example. These tasks require that the robot perform its task for long periods of time without the luxury of repair when components fail. It is vital to the success of the mission that the robot continue to function effectively despite component failures. For day to day goals, anyone who works with real autonomous robots is familiar with how frequently their hardware fails. Sensor and actuator failures are not surprising given how often the robots bump into things, rattle components, snag wires, and stress connectors as they move through their environment. It would be nice if we did not have to stop our research to repair the robot every time something fails. Given that a primary advantage of autonomous robots is their ability to perform tasks without human assistance or intervention, it is surprising the issue fault tolerance remains relatively unexplored. As it becomes more important for our autonomous robots to perform for extended periods of time without repair, fault tolerance must be investigated in depth.

We demonstrated that an autonomous robot which effectively tolerates component failures using a distributed network of concurrently running processes. A set of fault tolerance processes were created for each component. These processes are responsible for detecting the faults of their respective component, and for minimizing the impact of the failure of the robot's performance. By exploiting concurrency and distributedness, the robot monitors, detects, and compensates for component failures simultaneously. We tested the robot's fault tolerance capabilities by evoking a variety of failures in different combinations. We found the robot compensates for them before the robot's performance degrades to an unacceptable level.

We demonstrated that an autonomous robot can reliably and purposefully detect sensor failures by comparing the sensor's actual behavior with its expected behavior and with the behavior of complementary sensors. This has two important implications. First, if the system can recognize when components have failed, then it can purposefully tailor its use of its remaining sensors and actuators to minimize the impact of the failure on the robot's performance. Second, if the sensor can recognize when components are working, then it can purposefully integrate the use of all working components to enhance the performance of the system. This is useful for reintegrating repaired components. We demonstrated both these capabilities on Hannibal.

Fault tolerance for autonomous robots is a new area of research full of possibilities for future work. We have demonstrated initial fault tolerant capabilities on Hannibal, and there is plenty of room for improvement. The sensor behavior signatures (the models referred to in chapter 6) used by the sensor-state processes are hardwired. As a result, Hannibal can only reliably detect sensor failures in environments for which it has sensor signatures. For example, Hannibal's sensors may look different than what it expects if it were to walk in mud. It is a logical step to have Hannibal learn new sensor signatures over time so that it may recognize sensor failures in new environments. Furthermore, more work needs to be done in understanding how fault tolerance capabilities can be implemented on robots with different kinds of sensors and different numbers of sensors.

7.1.3 Robust Hexapod Locomotion

Legged biological systems are very effective in traversing terrain too rough for wheeled vehicles. Understanding how to build legged robots with locomotive capabilities comparable to those of biological systems has been a long standing goal in the field (Raibert & Hodgins 1993). Implementing models of insect locomotion on hexapod robots helps us to design robust locomotion controllers as well as gain insight into insect locomotion control (Quinn & Espenschied 1993).

We demonstrated how various insect locomotion tactics can be applied to hexapod robots. On Hannibal, we implemented several insect-based locomotion controllers as well as several rough terrain strategies used by insects. We tested the controllers by walking the robot on both flat and rugged terrain. The resulting locomotion was flexible and robust.

We have further confirmed that a distributed control scheme using simple, concurrently running processes is a viable approach to real-time locomotion control of hexapod robots with relatively little computational power ((Brooks 1989), (Donner 1987)). We have demonstrated this through implementing and testing a rough terrain locomotion controller with significantly more capabilities and on a much larger scale than past systems using a similar approach.

Future work for robust hexapod locomotion could take two different paths. On the biological path, it would be interesting to implement a rough terrain controller which is truer in spirit to real insect rough terrain locomotion. Other locomotion controllers are more strongly motivated by biology than ours, but they only address flat terrain locomotion. On the engineering path, it would be interesting to implement a similar controller on a different hexapod for comparison. We feel our controller pushed the physical capabilities of our hardware too soon. Furthermore, it would be interesting to implement rigorous climbing capabilities with a more capable robot.

7.1.4 Micro-Rover Missions

In regards to controller design for a planetary micro-rover, this thesis tackles relatively low level control issues. We have addressed several topics involved in wandering over challenging terrain for extended periods of time. However, higher levels of competence (such as navigation or mission scenario behaviors) are required for Hannibal to perform a complete mission. These additional layers could be added to the current controller. Our experience thus far indicates that the controller is modular and scales well, so adding these additional layers of competence is feasible and an area of future work.

7.2 Real Robots for Real Problems

In conclusion, we find this work of particular interest because the controller was implemented and tested on a physical robot of significant complexity. We argue it is important to study the behavior control systems on a real robots. As robots become more complex, the difficult issues in control are magnified to the point where they cannot be ignored or only partially addressed. It is important that all aspects of the control system be implemented and tested on real robots of sufficient complexity to make sure the *integrated* system works in the real world. By developing our controller on a physical system, we were forced to make sure all the pieces fit together at every stage. This also helped us maintain a common framework when implementing diverse capabilities. Finally, testing our controller on a physical system served as a strong reality check for our rough terrain locomotion and fault tolerance implementations.

Bibliography

Angle, C. (1991), 'Design of an Artificial Creature', *Masters Thesis, MIT.*

Angle, C. & Brooks, R. (1990), Small Planetary Rovers, *in* 'Proceedings of IEEE International Workshop on Intelligent Robots and Systems', Ibaraki, Japan, pp. 383–388.

Beer, R. & Chiel, H. (1993), Simulations of Cockroach Locomotion and Escape, *in* 'Biological Neural Networks in Invertebrate Neuroethology and Robotics', Academic Press, Inc., Boston, pp. 267–286.

Beer, R., Chiel, H., Quinn, R. & Espenschied, K. (1992), Leg Coordination Mechanisms in Stick Insects Applied to Hexapod Robot Locomotion, *in* 'IEEE International Conference on Robotics and Automation'.

Brooks, R. (1986), 'A Robust Layered Control System for a Mobile Robot', *IEEE Journal of Robotics and Automation* **RA-2**, 14–23.

Brooks, R. (1989), 'A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network', *Neural Computation* **1:2**, 365–382.

Brooks, R. (1990), 'The Behavior Language; User's Guide', *MIT A. I. Memo 1227.*

Brooks, R. & Flynn, A. (1989), 'Fast, Cheap, and Out of Control; A Robot Invasion of the Solar System', *Journal of the British Interplanetary Society* **42:10**, 478–485.

Chiel, H., Quinn, R., Espenschied, K., & Beer, R. (1992), 'Robustness of a Distributed Neural Network Controller for Locomotion in a Hexapod Robot', *IEEE Transactions on robotics and automation* **8**, 293–303.

Connell, J. (1989), 'A Colony Architecture for an Artificial Creature', *MIT Artificial Intelligence Lab Technical Report 1151.*

Cruse, H. (1976*a*), 'The Control of Body Position in the stick insect (*Carausius morosus*), when walking over uneven terrian', *Biological Cybernetics* **24**, 25–33.

Cruse, H. (1976*b*), 'The Function of the Legs in the Free Walking Stick INsect, *Carausius morosus*', *Journal of Comparative Physiology* **112**, 235–262.

Cruse, H. (1979), 'A New Model Describing the Coordination Pattern of the Legs of a Walking Stick Insect', *Biological Cybernetics* **32**, 107–113.

Cruse, H. (1980*a*), 'A Quantitative Model of Walking Incorporating Central and Peripheral Influences: II. The Connections between the Different Legs', *Biological Cybernetics* **37**, 137–144.

Cruse, H. (1980*b*), 'A Quantitative Model of Walking Incorporating Central and Peripheral Influences: I. The Control of the Individual Leg', *Biological Cybernetics* **37**, 131–136.

Cruse, H. (1990*a*), 'Coordination of Leg Movement in Walking Animals', *European Conference on Artifial Life* **13**, 105–119.

Cruse, H. (1990*b*), 'What mechanisms coordinate leg movement in walking arthropods', *Trends in Neurosciences* **13**, 15–21.

Dean, J. (1990), 'Coding Proprioceptive Information to Control Movement to a Target: Simulation with a Simple Neural Network', *Biological Cybernetics* **63**, 115–120.

Dean, J. (1991*a*), 'A model of leg coordination in the stick insect, *Carausius morosus*: I. A geometrical consideration of contralateral and ipsilateral coordination mechanisms between two adjacent legs', *Biological Cybernetics* **64**, 393–402.

Dean, J. (1991*b*), 'A model of leg coordination in the stick insect, *Carausius morosus*: II. Description of the kinematic model and simulation of normal step patterns', *Biological Cybernetics* **64**, 403–411.

Dean, J. (1992*a*), 'A model of leg coordination in the stick insect, *Carausius morosus*: III. Responses to perturbations of normal coordination', *Biological Cybernetics* **66**, 335–343.

Dean, J. (1992*b*), 'A model of leg coordination in the stick insect, *Carausius morosus*: IV. Comparisons of different forms of coordinating mechanisms', *Biological Cybernetics* **66**, 345–355.

Donner, M. (1987), *Real Time Control of Walking*, Birkauser, Boston.

Flynn, A., Brooks, R., Wells, W. & Barrett, D. (1989), 'Squirt: The Prototypical Mobile Robot for Autonomous Graduate Students', *MIT A. I. Memo 1220.*

Full, R. (1993), Integration of Individual Leg Dynamics with Whole Body Movement in Arthropod Locomotion, *in* 'Biological Neural Networks in Invertebrate Neuroethology and Robotics', Academic Press, Inc., Boston, pp. 267–286.

Gray, J. (1990), 'A Census of Tandem System Availability Between 1985 and 1990', *IEEE Transactions on Reliability* **39**, 409–417.

Hirose, S. (1984), 'A Study of Design and Control of a Quadruped Walking Vehicle', *International Journal of Robotics Research* **3**, 113–133.

Kabuka, M., Harjadi, S. & Younis, A. (1990), 'A Fault-Tolerant Architecture for an Automatic Vision-Guided Vehicle', *IEEE Transactions on systems, man, and cybernetics* **20**, 381–393.

Krotkov, E. & Simmons, R. (1992), Performance of a Six-Legged Planetary Rover: Power, Positioning, adn Autonomous Walking, *in* 'Proceedings of IEEE International Conference on Robotics and Automation', Nice, France, pp. 169–174.

Krotkov, E., Simmons, R. & Thorpe, C. (1990), Single Leg Walking with Integrated Perception, Planning, and Control, *in* 'Proceedings of IEEE International Workshop on Intelligent Robots and Systems', Ibaraki, Japan, pp. 97–102.

Lin, C. & Lee, C. (1991), 'Fault-Tolerant Reconfigurable Architecture for Robot Kinematics and Dynamics Computations', *IEEE Transactions on Systems, Man, and Cybernetics* **21**, 983–999.

Maes, P. (1990), 'How to Do the Right Thing', *A. I. Lab Memo 1180.*

Mataric, M. (1990), 'A Distributed Model for Mobile Robot Environment-Learning adn Navagation', *MIT Artificial Intelligence Lab Technical Report 1228.*

Mataric, M. (1992), Behavior-Based Control: Main Properties and Implications, *in* 'Proceedings of IEEE International Conference on Robotics and Automation, Workshop on Intelligent Control Systems', Nice, France, pp. 46–54.

McGhee, R. (1976), 'Robot Locomotion', *Neural Control of Locomotion* pp. 237–264.

McGhee, R. & Iswandi, G. (1979), 'Adaptive Locomotion of a Multilegged Robot over Rough Terrain', *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-9**, 176–182.

Nagy, P., Whittaker, W. & Desa, S. (1992), A Walking Prescription for Statically-Stable Walkers Based on Walker/Terrain Interaction, *in* 'Proceedings of IEEE International Conference on Robotics and Automation', Nice, France, pp. 149–156.

Payton, D., Keirsey, D., Kimple, D., Krozel, J. & Rosenblatt, K. (1992), 'Do Whatever Works: A Robust Approach to Fault-Tolerant Autonomous Control', *Journal of Applied Intelligence* **2**, 225–250.

Pearson, K. (1976), 'The Control of Walking', *Scientific American* **235**, 72–86.

Pearson, K. & Franklin, R. (1984), 'Characteristics of Leg Movements and Patterns of Coordination in Locusts Walking on Rough Terrain', *International Journal of Robotics Research* **3**, 101–112.

Quinn, R. & Espenschied, K. (1993), Control of a Hexapod Robot Using a Biologically Inspired Neural Network, *in* 'Biological Neural Networks in Invertebrate Neuroethology and Robotics', Academic Press, Inc., Boston, pp. 267–286.

Raibert, M. & Hodgins, J. (1993), Legged Robots, *in* 'Biological Neural Networks in Invertebrate Neuroethology and Robotics', Academic Press, Inc., Boston, pp. 267–286.

Rosenblatt, K. & Payton, D. (1989), A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control, *in* 'International Joint Conference on Neural Networks', Washington, D. C., pp. 317–323.

Siewiorek, D. P. & Johnson, D. (1981), 'A Design Methodology for High Reliability Systems: The Intel 432', *The Practice of Reliable System Design* pp. 621–636.

Song, S. & Waldron, K. (1989), *Machnies that Walk, The Adaptive Suspension Vehicle*, The MIT Press, Cambridge, MA.

Wilson, D. (1966), 'Insect Walking', *Annual Review of Entomology.*