Autonomous Stair Climbing

By

Kailas Narendran

B.S. Electrical Engineering & Computer Science
Massachusetts Institute of Technology, 2001

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTERS OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 2003

Signature of Author ________
Department of Electrical Engineering and Computer Science

Certified by ________
Jamie Anderson
Charles Stark Draper Laboratory
Thesis Supervisor

Certified by ________
Prof. Kaelbling
Professor of Computer Science and Engineering
Thesis Advisor

Accepted by _
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

[This Page Left Blank Intentionally]

Autonomous Stair Climbing

By

Kailas Narendran

Submitted to the Department of Electrical Engineering and Computer Science on January 15, 2003, in partial fulfillment of the requirements for the Degree of Masters of Engineering in Electrical Engineering and Computer Science

Abstract

As the face of warfare changes, the military has started to explore the application of robotics on the battlefield. Robots give soldiers a flexible, technologically advanced, disposable set of eyes and ears to assist them with their goal. This thesis deals with the design and implementation of a system to allow a small highly mobile tactical robot to climb stairs autonomously. A subsumption architecture is used to coordinate and control the maneuver. Various approaches to the problem including evolved architectures and use of contraction analysis are explored. Code was written and tested for functionality with basic test software. The functionality of parts of the system and control architecture was tested on the robot in a simulated operational environment.

Technical Supervisor: Dr. Jamie Anderson
Title: Principal Member of Technical Staff & Group Leader GBB3

Thesis Advisor: Professor Leslie P. Kaelbling
Title: Professor of Computer Science and Engineering

[This Page Left Blank Intentionally]

ACKNOWLEDGEMENT
January 15, 2003

This thesis was prepared at The Charles Stark Draper Laboratory, Inc. based on work supported by the Naval Air Warfare Center, Aircraft Division and The Defense Advanced Research Projects Agency (DARPA) under Contract N00421-01-C-0297.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the Naval Air Warfare Center, Aircraft Division and DARPA. It is published for the exchange and stimulation of ideas.

(Author's signature)

[This Page Left Blank Intentionally]

Table of Contents

Table of Figures

Table of Tables

Chapter 0

Acknowledgements

I'd like to thank the Draper education office, the Draper Fellows Program, the Naval Air Warfare Center, Aircraft Division and DARPA for giving me the incredible opportunity to conduct my MEng thesis work at Draper Labs. I learned a lot and had the pleasant opportunity to work with many talented individuals. Especially, I would like to extend my heartfelt gratitude to Jamie, George, Dave, and Sean who helped out as much as they could even when the work on my thesis was in its darkest hour. I would also like to thank Professor Kaelbling for her understanding and flexibility in helping me sort stuff out and agreeing to be my on campus reader at the last minute.

Chapter 1

Introduction

There has been a recent push to construct robots that fit the role of modern warfare. Rather than dealing with a uniformed enemy on a distinct battlefield, the current era of war takes the battle into urban environments. This juxtaposition of combatants and civilians results in much of the equipment and strategies used for conventional warfare (heavy munitions, blind fire strategies, etc.) having to be replaced by more precise and calculated measures. Where in the past one could simply drop bombs on a foxhole, knowing all the occupants were enemies, the possibility of invading the capital of a foreign city would preclude such methods as civilians and enemy combatants occupy the same structures.

Many recent innovations in surveillance technology have facilitated this shift in goal of modern warfare. The US military currently has unmanned aerial vehicles that are able to track and target with the accuracy of feet, giving the fighting force unprecedented precision in their attacks. Use of satellite imagery provides covert views of large expanses of earth that would be impossible to survey otherwise. In all of these cases, however, the target is out in the open. These technologies could help the military target a specific building, but what about a room in the building? What happens when the special ops make their landing under the cover of night and need to determine whom the occupant of that room is, around the corner, without waking the entire neighborhood?

The next step in modern warfare is the introduction of robotics to the playing field. Robots give soldiers a mobile, and most importantly, disposable, set of eyes, ears and more sophisticated surveillance equipment. They can generally be packages in a form factor that is more robust and easier to hide than an adult. Recently, DARPA has made a push for tactical mobile robotics (TMR). This involves creating robots that can be used in the aforementioned ways, in a hostile environment. The creation of small, maneuverable, robust, high performance, and low lifetime vehicles for indoor use is the next step to realize the dream of robots on the modern battlefield.

Motivation

The motivation for this project stems from the target mission for the Distributed Robotics (DR) program vehicle, designed for DARPA.

DARPA Grant

The proposal to DARPA from Draper Laboratory for this project outlined the creation of a collection of small, reconfigurable, robots with operator control units (OCU) and a communications relay node (CRN) that can provide higher echelon vehicle data, image distribution and vehicle tasking.

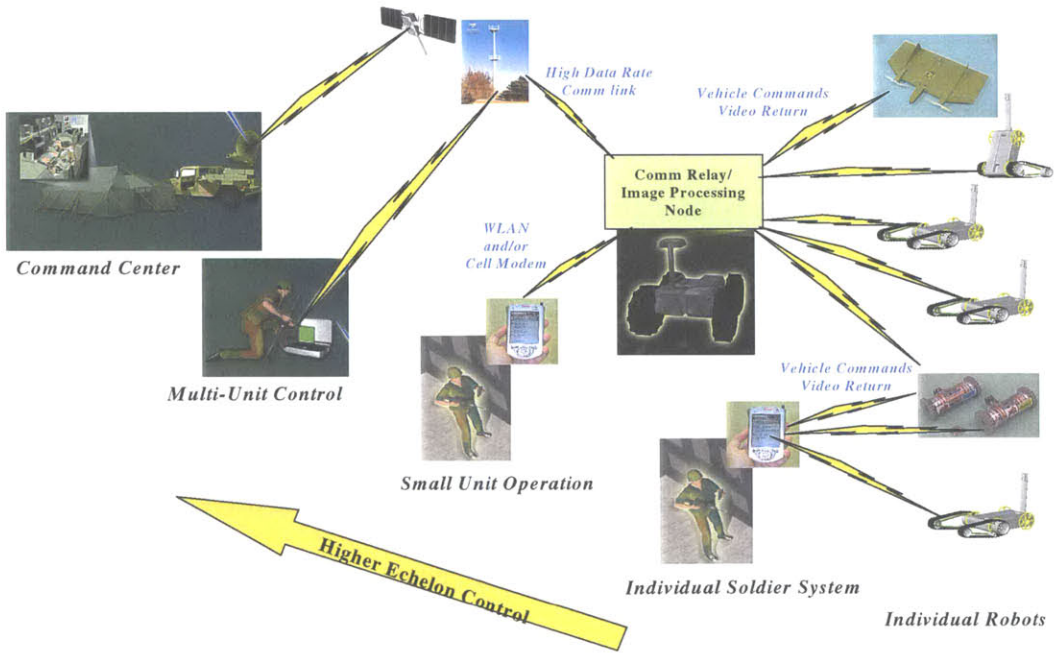


Figure 1 - High level system diagram for use of high mobility tactical robots (HMTR) in warfare

The distributed nature of this system makes it very flexible and open to future development. The vehicle design is centered on an indoor operating environment. The OCU is used by a solider, in the field to provide direct control over all aspects of the robot, while providing feedback from onboard sensors (imagery, thermal, etc.).

Vehicle Hardware Design

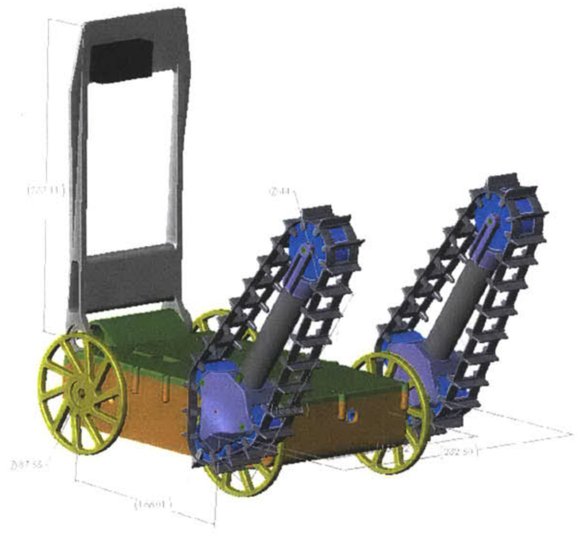


Figure 2 - DR Vehicle Solid Model

The final design of the DR vehicle is shown in Figure 2. The vehicle has a set of wheels, tracks and an actuated mast in the rear. Both of the tracks rotate in unison through 360° about an axis through the centers of the front two wheels. The rear mast can rotate through a little over 270°, from being flush with the top of the vehicle, to past parallel with the back face. Each side of the vehicle is driven with a brushless astroflight motor. Maxon brushed DC motors actuate the rotation of the track arms and the rear mast. There is a simple clutching mechanism that engages the drive train for each side to transfer power to the treads. The design of the vehicle was inspired by the PackBot from iRobot, shown in Figure 3. On account of its robust and flexible design, the PackBot has already received limited deployment in Afghanistan [1.1].

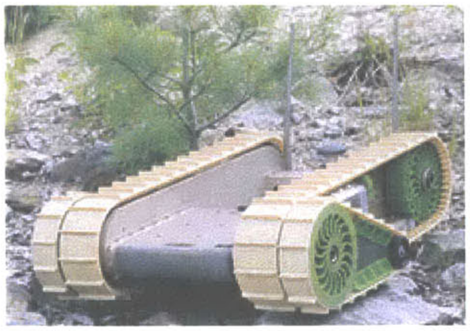
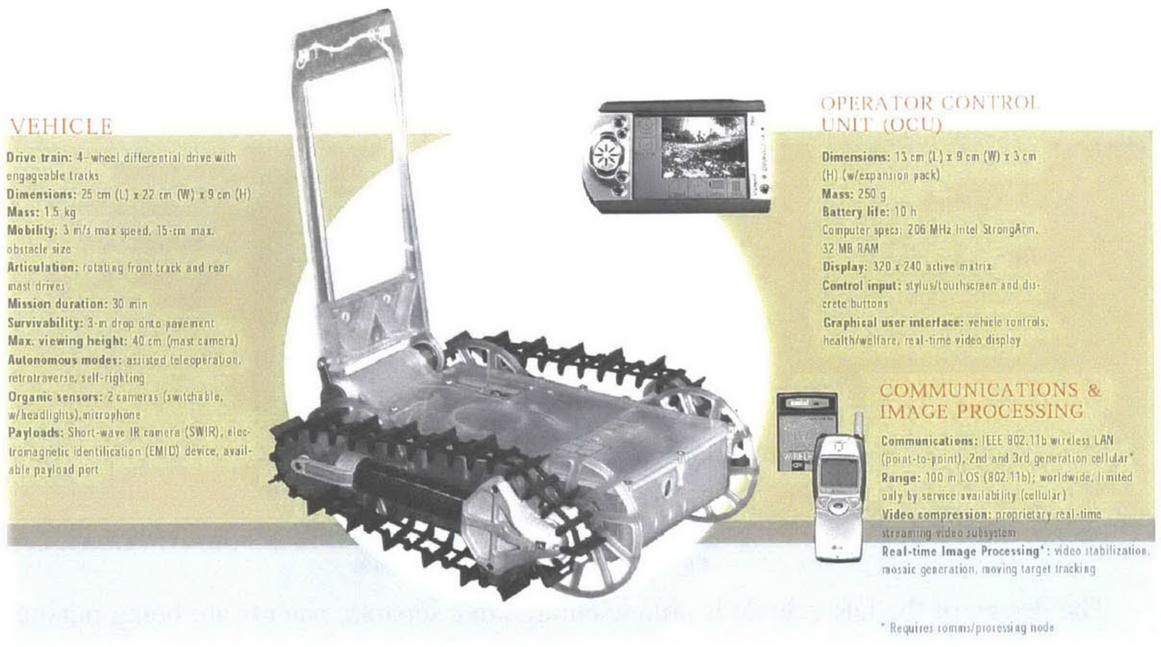


Figure 3 - PackBot from iRobot

The design of the DR vehicle is unique enough that separate patents are being pursued.

Design specs

This vehicle was created with aggressive drop-test and robustness goals and functional flexibility in mind. The material is a soft, rubber like compound. Most elements of the structural body were cast. Almost all components were custom made from aluminum to reduce weight. The spiral patterns in the wheels allow the robot to sustain considerable impact loads from being dropped [1.2]. The appendages to the main body give it the ability to adapt to a plurality of operational circumstances.



VEHICLE

Drive train: 4-wheel differential drive with engageable tracks
Dimensions: 25 cm (L) x 22 cm (W) x 9 cm (H)
Mass: 1.5 kg
Mobility: 3 m/s max speed, 15-cm max. obstacle size
Articulation: rotating front track and rear mast drives
Mission duration: 30 min
Survivability: 3-m drop onto pavement
Max. viewing height: 40 cm (mast camera)
Autonomous modes: assisted teleoperation, retrotraverse, self-righting
Organic sensors: 2 cameras (switchable, w/headlights), microphone
Payloads: Short-wave IR camera (SWIR), electromagnetic identification (EMID) device, available payload port

OPERATOR CONTROL UNIT (OCU)

Dimensions: 13 cm (L) x 9 cm (W) x 3 cm (H) (w/expansion pack)
Mass: 250 g
Battery life: 10 h
Computer specs: 206 MHz Intel StrongArm, 32 MB RAM
Display: 320 x 240 active matrix
Control input: stylus/touchscreen and discrete buttons
Graphical user interface: vehicle controls, health/welfare, real-time video display

COMMUNICATIONS & IMAGE PROCESSING

Communications: IEEE 802.11b wireless LAN (point-to-point), 2nd and 3rd generation cellular*
Range: 100 m LOS (802.11b); worldwide, limited only by service availability (cellular)
Video compression: proprietary real-time streaming video subsystem
Real-time Image Processing*: video stabilization, mosaic generation, moving target tracking

* Requires comms/processing node

Figure 4 - Specifications of vehicle

The physical design of the robot gives it the flexibility of fast, lower power operation on a smooth surface (when the wheels are driving); but the option of climbing over rough terrain is available by putting down the tracks. Figure 4 shows the physical dimensions and parameters of the vehicle. The rear mast was added to give the vehicle the ability to climb over large obstacles. The concept of a rear mast for climbing obstacles was proven in a previous robot, Throwbot B, shown in Figure 5.

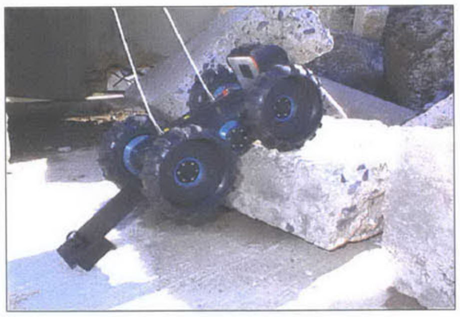


Figure 5 - Throwbot B

Throwbot B is of approximately the same size and weight as the DR vehicle. The added tracks in the DR vehicle replace the larger wheels in Throwbot B. Figure 6 shows an illustration of the robot using the mast for maneuvers such as climbing a large obstacle or righting itself. All of these operations are key to the robustness requirement of the vehicle.

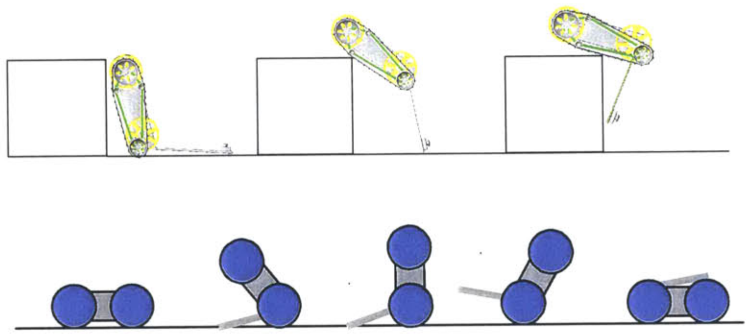


Figure 6 - Mast assisted climbing and self righting [1.2]

Degrees of freedom & Sensors

As described earlier, the robot has five degrees of freedom: two independent drives, independently actuated mast and track arms, and a clutching mechanism to engage the tracks. Using the mast and track arms, the robot can achieve various operational poses enumerated in Figure 7.

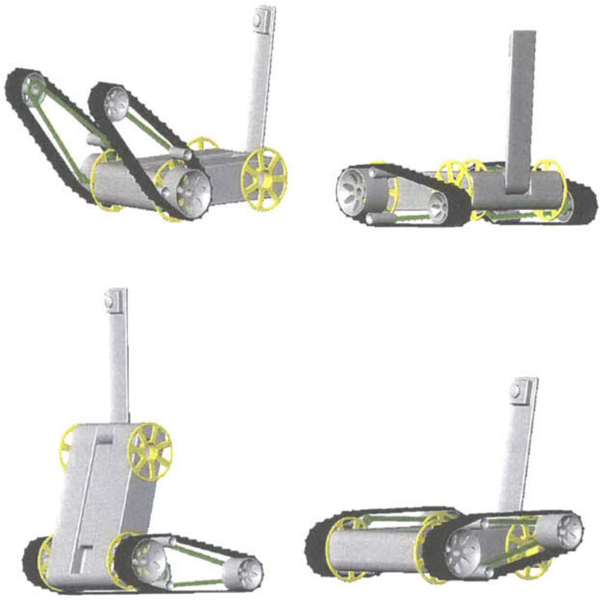


Figure 7 - Various robot poses

These poses allow varied functionality, from climbing and looking over large obstacles (left side of figure), to driving on either tracks or treads (right side of figure). Onboard, the vehicle has a 3-axis accelerometer, and a heading rate gyro. There are also current sensors on drive motors and rate encoders on the wheel shafts allowing precise closed-loop control of drive torque and speed. There are also electronics onboard to transmit information from onboard cameras or other configurable sensing electronics back to the operator.

Operator Interface

The operator interface (OCU) for the robot is provided through a Compaq iPaq

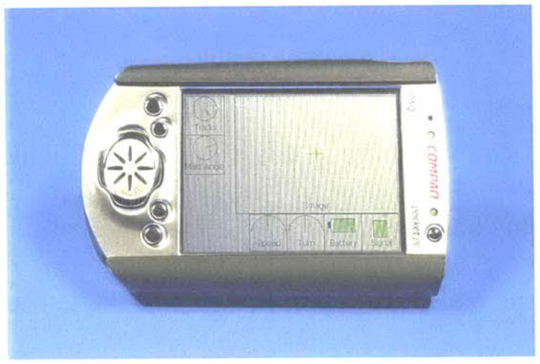


Figure 8 - Operator control interface (OCU)

handheld computer, shown in Figure 8. The user interface provides the user with control over the position of the mast and track arms, and turn rate. Speed is controlled by position of the stylus on the screen. Feedback is provided about vehicle state. The on screen display can provide the close to real-time video feed from cameras on board the robot. The OCU communicates with the robot over a wireless 802.11b network link.

Vehicle Software Design

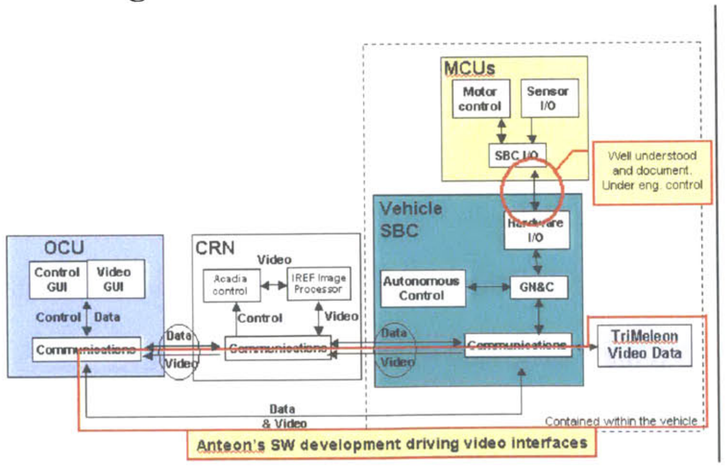


Figure 9 - System software design

The software design for the entire system is shown in Figure 9. All of the software that is applicable for this thesis resides in the "Autonomous Control" block. Other layers of software are used for communication with all hardware actuators and sensors. All of the on-board software is compiled for the OMNI1400 platform, running under Windows CE on the InHand single-board computer on the robot.

Thesis Motivation

The goal of stair climbing in this thesis was motivated by the fact this robot is an indoor vehicle. Stairs are very commonplace and will almost certainly be encountered by this vehicle during its real world operation. The vehicle certainly possesses the capability to climb stairs, but functional tradeoffs in the user interface make it difficult for a human to control the robot during the maneuver.

To climb stairs, the robot must use all of its actuators simultaneously. It needs to be able to control its heading, while actuating the track arms and mast appropriately to ensure the body is carried over the rise of the stair. While the OCU provides control over each of these degrees of freedom independently, using all of them at the same time is problematic due to the limitations of the physical interface. There are four degrees of freedom that need to be controlled (mast, tracks and each side of drive), and only two degrees of input (x and y position on screen) available to do so. In addition, the feedback provided to the operator from the vehicle is minimal. While a video image is great for surveillance, using it to guide the vehicle through complicated, three-dimensional maneuvers is very difficult. The conclusion to draw is that more intelligence is needed on board for the vehicle to be able to complete this maneuver.

Robust to various changes in physical constants

The control architecture that will do the stair climbing control needs to be very robust. There are no guarantees for the physical constants or geometries in a combat situation. The robot should be able to climb a variety of stair geometries (rather than just those that are up to code), and surmount the steps even if there are surface hazards (loose or uneven surface).

Easy to operate

The end user of this robot will be a special ops soldier in the field. Given that their ultimate goal is self-preservation and completion of the mission, they aren't going to tolerate a complicated and involved system to climb stairs. The control system to implement stair climbing should be as easy to use as possible, requiring minimal operator interface and input.

Feasible to implement

There must be a way to implement the system in a timely fashion that's compatible with my MEng appointment and DARPA contract timeframes. There is also a limited amount of computational resources onboard the robot, and available during development. Finally, the state feedback is only available through a real world, noisy, incomplete channel. The system should be able to work even given these real world constraints.

Chapter Preview

The following chapters describe the progression of this project from exploration of possible control architectures, through software implementation and testing on the final vehicle.

Chapter 2

Candidate Control Architectures

Over the course of this project, a few control architectures were evaluated before choosing the final candidate for implementation. The first architectures were chosen with some degree of rationale, but eliminated for problems that were unforeseen upon their initial selection.

The design of the control system of the DR robot progressed from attempting to evolve a control structure, to using contraction analysis to design a stable control law, to a subsumption architecture. The evolved architecture was attempted in part as a 6.836 (Embodied Intelligence) final project. Contraction analysis was suggested as an option by Professor Jean Jacques Slotine. Finally, subsumption was selected as the most appropriate architecture for this task.

Vehicle State

The state of the vehicle from the point of view of control software is encapsulated in a few variables, shown in Table 1.

Observable States	Controllable States
X Axis Acceleration	Mast Position
Y Axis Acceleration	Tracks Position
Z Axis Acceleration	Left Side Drive Velocity
Z Axis Rate Gyro	Right Side Drive Velocity

Table 1 - Observable and Controllable Robot State Variables

The observable states are all the variables of inertial navigation from the IMU onboard the robot. Sensors exist for other state variables on the robot (motor current, temp, etc.), but they are not available via the low level software interface. Controllable states are accessed by putting commands in a queue that is processed by lower level software. The commands relay desired limb position and the max velocity used to move it to that position. According to software interface rules, when the robot is in the autonomous climbing mode, no commands other than that of the autonomous climbing architecture should be entered into the queue.

Evolved Architecture

The use of an evolved architecture for this project was inspired by the work of Karl Sims. Sims' work on evolving creatures that could swim was a very compelling example presented in 6.836. This project seemed to parallel that one in a number of ways.

The system of the DR vehicle climbing stairs is quite complicated when a purely mechanical analysis is attempted. There are numerous physical parameters that come into play when one analyzes the frictions, moments of inertia, etc. that affect the motion of the robot while climbing stairs. More importantly, there are many possible solutions that will work for getting the vehicle to climb stairs. Since this problem has numerous solutions in a high dimensional space, a genetic algorithm was chosen as the method to search the space and arrive at a solution.

Attempted Setup

The use of a genetic algorithm for searching the parameter space for stair climbing required some standard elements, a model, a control and genetic structure, and a method of selection and breeding (evolution).

Model

The fitness of each genome was evaluated in simulation. Due to time and computational constraints, the vehicle model was a simplified version of Throwbot B, with some of the sensors from the DR vehicle (shown in figure 3). This model is much simpler than the actual DR vehicle. The idea was to get the system working on a simple model, then increase the complexity to get better results. The problems with that plan are discussed later.

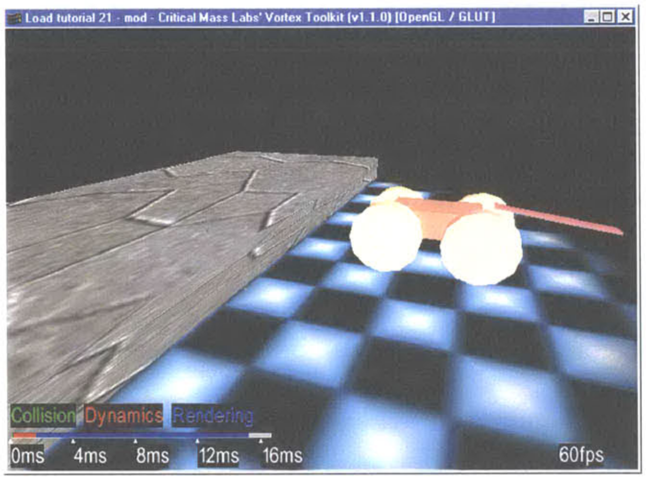

Load tutorial 21 - mod - Critical Mass Labs' Vortex Toolkit (v1.1.0) [OpenGL / GLUT]
Collision
Dynamics
Rendering
0ms
4ms
8ms
12ms
16ms
60fps

Figure 10 - Simulator model

The simulator used was Vortex from Critical Mass labs. It includes mass properties and friction models in the simulation. The world provided a “step” for the vehicle to climb. The sensors on the vehicle provided feedback for full body attitude (heading, pitch and roll).

Control Structure

The control for the vehicle uses an augmented neural network (ANN) shown in Figure 11. At each time step, an input vector of body attitude and state vectors are transformed through a matrix of weights to an output vector of mast velocity and new state variables. In the figure, "s" refers to the state vector. This vector evolves over time.

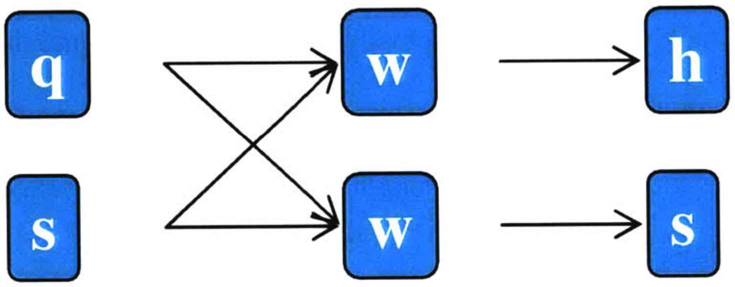


Figure 11 - Augmented Neural Network diagram, q- body attitude, s- state vector, w- weight vector, h-mast velocity

The state vector allows the transmission of information from one time step to another. The "q" vector encapsulates the body attitude as would be available from a 2 axis gyro, returning the plane of the body of the robot relative to a plane orthogonal to gravity. The "h" vector is the mast velocity, evolving over time as a function of the body attitude and the state.

Due to the complexity of the model and the issues involved in searching the parameter space (discussed later), a constant wheel velocity was chosen. The ANN and wheel velocity are encoded in a genome. Each individual is evaluated by decoding the genome to the above control structure, and simulating the controller on the model described above.

The actual range of values the weights, states, and initial velocity could have is an experimental parameter. In addition, the number of state variables is set at compile-time.

Genome Structure

The genome was simply a string of bits that encode all the weights for the transformation matrix from input space to output space (shown in Figure 11), and the velocity of the vehicle. Each of the numbers in the genome was encoded with a grey code to equalize the effect of one-bit mutations on the system. The number of bits encoding each value is a system parameter. In all experiments there were 50 bits encoding each value. I initially planned on experimenting with different encoding sizes. In the end, it took way too long to actually run the simulations and there was no time to experiment with different encoding sizes. The reason for choosing 50 bits for encoding each value was to find an encoding length that wasn't too long, yet not short enough that one bit flip makes a big difference.

Fitness and Simulation

Each genome was given approximately three seconds of simulated time to climb the step. The simulator was run till termination criteria were achieved. In most cases this amounted to time running out. If the vehicle was successful, termination implies that it climbed the step before time ran out. The metric for determining if the step had been climbed was the distance between the center of mass of the vehicle and some point on the top of the step, scaled to increase resolution after it is converted to an integer (shown in equation 1). The total fitness was the sum of that difference and the number of ticks the simulation ran for.

$$-fitness = ticks_{simulated} + 100 \cdot \left|(COM_{x,z} - GOAL_{x,z})\right|$$

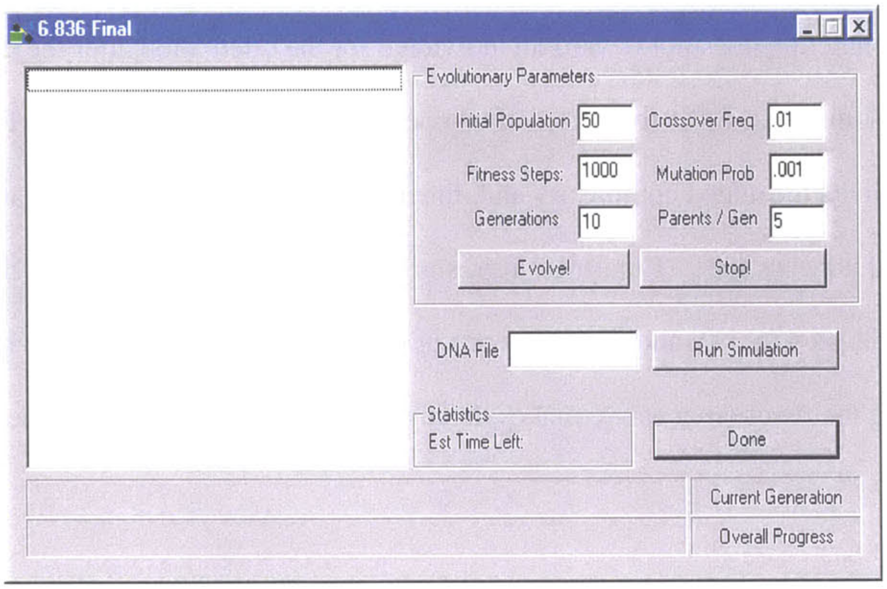
Equation 1 - Fitness function

This structure of the fitness function allows the robot to first learn how to climb the step, and continually improve its fitness as it climbs the step faster. It is useful to note that this fitness function does not reward an individual for the orientation after they climb the step. It is assumed that if the robot can climb one step, the operator can realign for the next one. Given the added complexity and dimensionality for including orientation, it was deemed unnecessary. The fitness function returns a negative value since the time to climb the step (ticks), and the difference in vertical displacement needs to be minimized. Making the function negative makes the highest fitness individual have the most positive value.

Evolution

The evolutionary algorithm implemented was very simple. User parameters included the population size, number of parents, number of generations, and crossover and mutation frequencies (shown in figure 12 in program front panel).

Initially random genomes were created and their fitness is evaluated in the simulator. After evaluating the fitness of all the individuals, parents were selected as the top few individuals, and created the next generation. Parents were kept in the population (eliteist strategy), so the best fitness never decreased.



Initial Population 50
Fitness Steps: 1000
Generations 10
Crossover Freq .01
Mutation Prob .001
Parents / Gen 5
DNA File
Est Time Left:

Figure 13 - Program front panel

Crossover was applied during reproduction. To create the offspring DNA, initially the genome of a random parent (probability ½ that it is either parent) was copied bit by bit to the child DNA. With a probability equal to the probability of crossover at each bit, the copying switched to the other parent's DNA. After the child had been created, there was a bit-flip that occurs with a probability equal to the probability of mutation.

Problems

The execution of this project was an exercise in patience and frustration. Due to the computational complexity of simulating this 3D world, evolution took an extremely long amount of time to run. Evaluating the fitness of an individual genome took from five to ten seconds each. This quickly adds up when decent population and generation sizes are chosen. As a result, most runs were only fifty generations with a population size of 100. The time that it took to run fit well with my schedule and allowed me to get in two or three runs in a 24-hour period.

This evolutionary system used as my 6.836 project on the Throwbot B model was essentially an experiment to see if it would be an appropriate system to use for the full blown thesis project on the actual DR vehicle. All experiments were done in a few weeks in the spring. The computational time constraints became significant as even this simple model took a long time to run.

Performance

The evolutionary system implemented was successful in that it did create a control system that was able to climb the step. The parameters that I initially intended to vary, however, generally did not function as desired.

One main goal of the experiment was to see the effect of evolutionary variables on performance. Due to the size of the genome, it turned out that that fifty generations generally wasn't enough for performance to change radically (see figure 14).

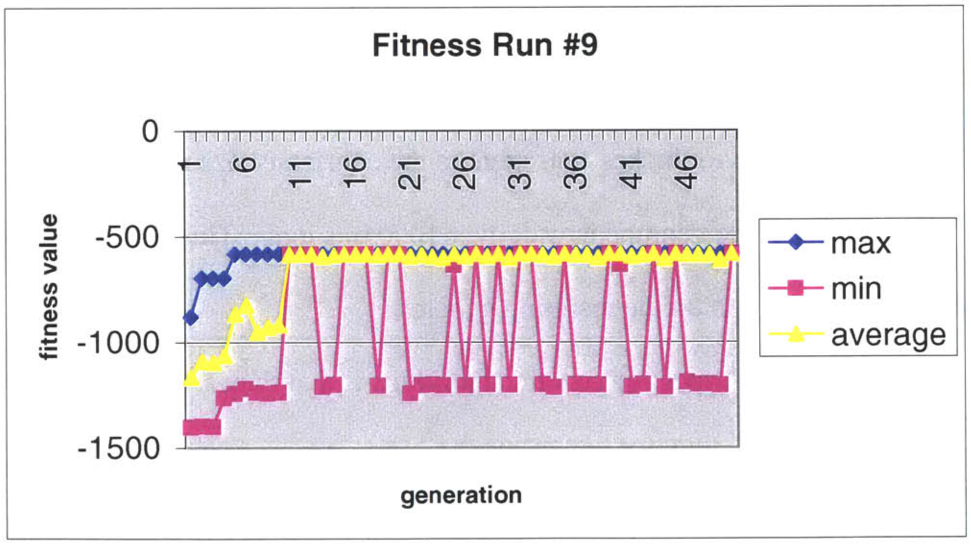


Figure 15 - Graph of fitness as function of generation. $P_{mutation}$=.1 $P_{crossover}$=.05. 4 state variables

Since the algorithm couldn't run very long, the initial jump in performance is due to some random genome that did pretty well, and only needed slight modifications to increase its performance. The plateau is characteristic of the behavior between big jumps in fitness caused by mutation and recombination. Unfortunately, it wasn't feasible to run the simulation for longer periods of time due to the lack of time to change other parameters. One interesting fact that appeared from this limited-run-time evolution is that genetic recombination is much more applicable than point mutations. Figure 6 shows the fitness increase with a $P_{crossover}$ that is 1/5 that of figure 16.

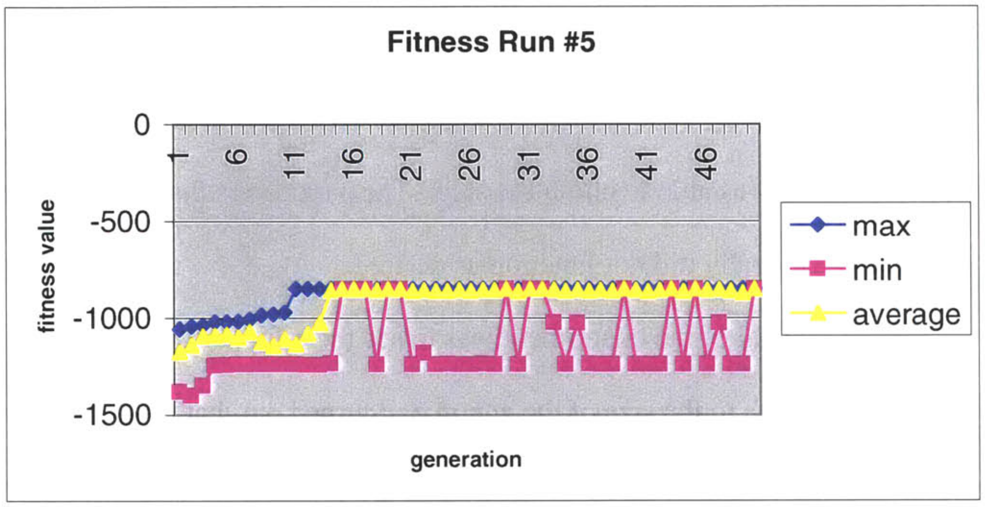


Figure 17 - Graph of fitness as function of generation. $P_{mutation}$=.1 $P_{crossover}$=.01. 4 state variables

The rate of increase in fitness is much sharper in the run that has a higher probability of crossover. This behavior makes sense due to the enormous size of the genome. With four state variables, the genome is well over 1000 bits long. This results in minimal effect of point mutation on the resulting genome.

One annoying aspect of genetic algorithms is that they seem to find the bug in software faster than the solution you're looking for. There were some solutions that simply exploited holes in the fitness function (which was revised a few times during the progress of experiments). Initially the stopping criteria for the simulator just checked the distance from the z coordinate of center of mass to the z coordinate of the goal.

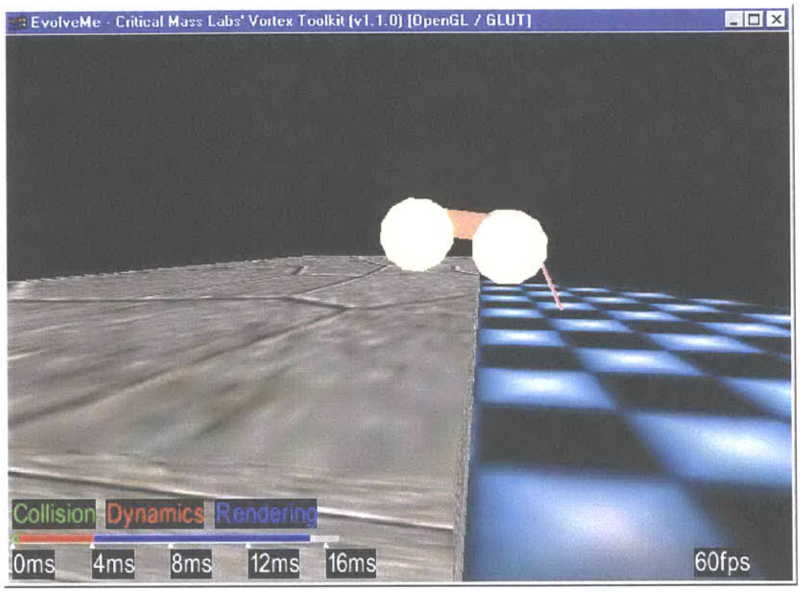

EvolveMe - Critical Mass Labs' Vortex Toolkit (v1.1.0) [OpenGL / GLUT]
Collision
Dynamics
Rendering
0ms
4ms
8ms
12ms
16ms
60fps

Figure 18 - Bounce fools fitness function

Figure 19 shows the vehicle, as it is about to get on the step. It's hard to see, but at this point the vehicle jumps up a bit due to the speed the wheels are spinning and it registers as if it has cleared the top of the step. To overcome this bug, the stopping criteria were changed to make sure the center of mass passed the edge of the step. This would ensure that no slight bumps or awkward movement of the mast would make it look like the vehicle had climbed the step. After fifty generations, the winner didn't make it on the step in less than three seconds, but executed the maneuver shown in figure 20.

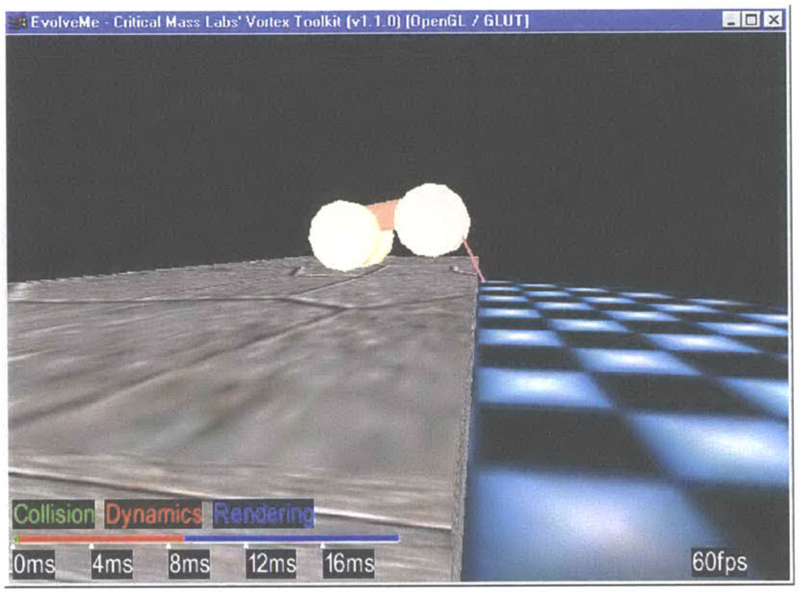

EvolveMe - Critical Mass Labs' Vortex Toolkit (v1.1.0) [OpenGL / GLUT]
Collision
Dynamics
Rendering
0ms
4ms
8ms
12ms
16ms
60fps

Figure 21 - Vehicle fools second fitness function

Instead of making it over the step, the vehicle was traveling fast enough that its momentum carried it forward along the angle of the mast. To beat this strategy, the finish line was moved farther back on the step.

The vehicles that eventually evolved after fifty generations, however, were able to climb the step. They didn't do it as fast as desired, but they did make it up.

Variables

The parameter space I wanted to explore primarily dealt with the effect of state variables and evolutionary parameters on the quality of the results. I was really unable to achieve this goal due to the inordinate amount of time it took to evolve anything useful. In addition, velocities that were not very high caused the simulator to break and the robot would go flying off into outer space. Since the mast velocity is a weighted sum of the state variables and body attitude, too many state variables would cause the mast to move too fast. To prevent that from happening, the value of the weights was decreased. This, in turn, resulted in more time being required to increase performance.

For this short evolutionary time, the search really just seemed to be random. Higher mutation and crossover values yielded better results. This is probably only partially due to the fact that useful information was transferred from parent to child, but more likely due to the fact that the recombination randomly resulted in better individuals (there was usually only one big jump in fitness, then a plateau).

At the end of the spring term, I determined an evolutionary strategy was not the optimal plan of attack mainly for computational and logistical reasons. I was very limited by the amount of computational resources available to me at Draper. Due to the nature of three dimensional physics, the simulation of a very simplified model of the robot took a very long time. In addition, the vortex package didn't provide any interface to import models from SolidWorks (the CAD package used for mechanical design at Draper), requiring that I would build a very simplified model in their scripting language by hand (not a simple, or accurate task). Finally, the documentation and support for the modeling and simulation package were very poor. For the plurality of aforementioned reasons, and suggestions from my on campus advisor, I decided to explore Contraction Analysis.

Contraction

Contraction Analysis is a novel form of analysis of stability of non-linear systems developed by Jean Jaques Slotine and Winfried Lohmiller [2.1]. Contraction analysis was devised as a general way to look at stability of non-linear systems. Its structure is based on the notion of stability being nominal motion about some equilibrium point. If something is called stable, you don't need to actually know what the motion is, just that there is one that exists. In the end, all initial trajectories tend to the same one and initial conditions or disturbances are forgotten.

Contraction analysis guarantees global exponential convergence of trajectories. The summarized results from the derivation of the theory [2.2] are presented in this section. Assume we have a general deterministic system of the form

$$\dot{x} = f(x,t) \qquad \textbf{Equation 2}$$

where f is a n x 1 nonlinear vector fuction and x is the n x 1 state vector. We assume that all functions are real and smooth.

If there exists a uniformly positive definite metric

$$M(x,t) = \theta(x,t)^T \theta(x,t) \qquad \textbf{Equation 3}$$

such that the associated generalized Jacobian

$$F = (\dot{\theta} + \theta \frac{\partial f}{\partial x})\theta^{-1} \qquad \textbf{Equation 4}$$

is uniformly negative definite, the all system trajectories converge exponentially to a single trajectory with the rate of $|\lambda_{max}|$, the largest eigenvalue of the symmetric part of F.

Using the definitions above, it is possible to show that combinations of contracting systems are also contracting [2.1, 2.2]. Interesting work in neural research has suggested that the computations executed by the brain for motion are contracting in nature. Rather than the brain learning entire motions, it's possible that the control is composed of smaller "primitives", which are contracting in and of themselves. A common example is that of a frog's spine [2.2]. Stimulating separate parts of the spine causes predictable joint torques. When stimulations are combined, the resulting forces are vector sums of the individual stimulations [2.3]. If these input stimulations come from some contracting system, the overall result could be the stable motion seen in the frog's movement [2.2]. The notion of these motion primitives inspired the thought of possibly using contraction analysis as a method of designing the control law for the stair climbing behavior of the robot.

The idea was to come up with contracting primitives of motion for stair climbing (for example, mast and tread motion). Next, a genetic algorithm would search a smaller space of combinations of these motions to determine a maneuver to climb stairs that was contracting (search for M in Equation 3). When I started working on the problem, it rapidly became very intractable.

Problems

The primary issue was the mathematical model of the entire system was required to prove any type of contraction. From the aforementioned equations involved in proving contraction, it's evident that to prove stability using contraction analysis, a comprehensive state space model is required. This task is not trivial at all. I began to work on the three-dimensional state space model, and quickly fell back to a two dimensional simplification. Finally, it quickly became evident that physical constants and physical parameters of the robot would play a big part in the robustness of this solution (affecting the eigenvalues of important matrices). Since the physical model of the robot wasn't fully nailed down till some point in the summer of 2002, and the operational environment isn't certain at all, those constants are sure to be variables. On top of all these problems, contraction analysis is a very new theory. It was impossible to find anyone familiar with the theory other than Professor Slotine, let alone many examples of application of contraction theory to multidimensional problems. For all these reasons, a more conceptual notion of contraction, subsumption, was explored.

Subsumption

The notion of a subsumption architecture was recently made popular by Brooks as a robust method of control for an autonomous robot in a complex, dynamic world [2.4]. The design rationale behind a subsumption architecture is to create a complex overall behavior by layers of simpler behaviors. The robot is usually tying to accomplish a lot of tasks at the same time, but some tasks take priority from time to time allowing a complex range of behaviors.

Brook's general design rationale is very centered in the real world, assuming the control should be as simple as possible, and the world will be complex. The control system should be resilient to errors and failure. He creates the subsumption architecture specifically for a robot that has multiple goals, sensors and requires robustness and extensibility, a perfect match to the DR vehicle [2.4].

The subsumption architecture design rationale creates a control system from the bottom up. The control systems are layered, with the bottom layers performing the lowest level, mundane tasks required for the robot to move its actuators. The higher level layers perform more intelligent tasks like path planning or obstacle avoidance. The feedback loop of control is closed through the environment as all the input for the robot comes through noisy sensor channels.

These subsumption layers are created by a collection of augmented finite state machines (AFSMs). The AFSMs run completely independently, and asynchronously of each other, with no global knowledge. They only communicate with each other through what are conceptually called "wires". These wires can carry some value or nothing at all. The AFSMs are generally Mealy machines that can have multiple inputs and outputs [2.4]. Wires are connected together by "suppression" and "inhibition" blocks. These blocks have an *input*, *output* and a *sidetap*. In the case of the suppression block, the value of the *output* is equal to that of the *input* when the *sidetap* is empty. When there is a value on the *sidetap* line, the *output* gets the value of the *sidetap*. In the case of the inhibition block, the *output* is equal to the *input* as long as the *sidetap* line is empty. Whenever the *sidetap* gets a value, the *output* becomes empty [2.4].

These connection blocks, in conjunction with computational blocks, can allow for very complex and intricate behavior.

Rationale behind decision

The decision to use a subsumption architecture in this robot was based primarily on robustness, flexibility and feasibility. Much of the prior work done with subsumption based control was on a higher level, where subsumption usually coordinated some searching or exploring algorithm. In a few cases subsumption controlled and coordinated physical motion during some complex maneuver [2.4, 2.5].

Chapter 3

Related and Prior Work

Subsumption architectures and the goal of stair climbing are not new and there has been a good deal of work done in the past in both of these areas. This chapter describes some previous work that has been done with subsumption architectures, highlighting the nature of most applications of the subsumption architecture. Finally, the design and strategies of some other robots that are able to successfully navigate stairs are discussed.

Subsumption Architecture

Much of the work dealing with subsumption architectures has come from the AI Lab at MIT in the late eighties and early nineties. Brooks presented the architecture in a paper in 1986, suggesting a decentralized control system that acted in a goal driven, task-achieving manner [3.19]. Subsumption architectures have been used in everything from low level movement control to high level path planning.

The Ghengis robot is one of the six-legged walking robots from the AI lab. The robot has six independent legs that communicate with each other over a token ring network. With little central coordination and low amounts of computational power, the robot is able to successfully walk. The layers that provide robust leg movement exist in very small, robust, local asynchronous units. Little information from the higher levels make it down to the lower levels that were actually involved in moving the legs around. The Ghengis robot clearly demonstrates the use of a subsumption architecture to accomplish a complex control and movement task in a robust fashion [3.1].

The robot Toto, also from the AI lab at MIT, is controlled using a subsumption based architecture. The goal of Toto is general navigation including obstacle avoidance, path planning and map creation [3.2]. Toto is an upright robot outfitted with numerous sonar and collision detection sensors. At the heart of Toto's control system is a subsumption architecture that ties together all the goals of navigation. Toto's lowest levels of competence allow it to wander aimlessly, as the higher levels give it the ability to accomplish map navigation.

There are a lot of other robots various individuals have built that use subsumption architectures for control. There are commercial robot programming packages (such as Robolab) that have subsumption architectures built into them. In all cases of robots controlled with subsumption architectures I was able to find, the subsumption architecture is used to control path planning. This is probably due to the fact that using a subsumption architecture is not very analytical. Most problems of complicated motion are approached in a more traditional sense, devising equations and simulations to prove an outcome. This was generally the case when I searched for robots that accomplished stair climbing.

Stair Climbing

It is a reasonable generalization to say that robots fall into two categories, legged and not legged. I found many robots that climbed stairs, in both categories. The majority of the robots in the legged category are able to climb stairs, as one of the motivations for legged locomotion is to deal with stairs, which are an architectural artifact that descends directly from man's evolution to walk to two legs.

I found robots that could climb stairs with as few as one or as many as six legs. The simplest legged robot that could climb stairs was the planar hopper from the MIT Leg Lab [3.6]. The robot has one leg that is actuated along an axis that can be directed in a plane. The robot, with the body constrained to a plane, can hop around and bounce over small objects, conceivably giving it the ability to climb stairs.

A four legged robot that can climb stairs is the SCOUT, from McGill University [3.5, 3.7]. Each of the legs can be actuated along an axis that can be tilted forward and back. By using a sequence of motions, the robot is able to bounce up steps. The six legged robot RHEX (aka the robotic cockroach) from McGill, is able to climb stairs by simply running at them till it climbs [3.7]. This is due to the innovative and very simple mechanical design of the robot [3.8]. The robot has a small, rectangular body with six legs. Each leg is made of a small, curved piece of metal. The legs rotate about the point where they connect to the body, through an axis in the plane of the body and orthogonal to the direction of motion. The four outer legs rotate together, and are out of phase with the inner ones by 180 degrees.

The series of TITAN quadruped walking robots from Hirose & Yoneda Robotics Lab accomplish stair climbing by maintaining static equilibrium during coordinated motion with four legs while climbing [3.10, 3.11]. The goes-over-all-terrain (GOAT) robot from CMU uses a unique design of wheels on four actuated arms to allow itself to climb stairs and accomplish self righting [3.3]. A robot that is similar in appearance to the GOAT is the SHRIMP [3.22]. This vehicle is unique in that its clever mechanical design allows it to climb large obstacles (including stairs), completely passively. The connection of a number of mechanical members passively moves to surmount the stair while the only electronics necessary drive the wheels.

One of the newest additions to the realm of legged stair-climbing robots is the walking robot from Honda, ASIMO [3.16]. The robot walks upright on two legs, using a variety of algorithms. I was unable to find details about its stair climbing algorithms, although there are movies available of the robot climbing stairs [3.16, 3.17].

As early as 1968 General Electric had built a large (11 feet tall, 3 ton) vehicle that was legged, and could climb over stair like obstacles [3.18]. A human was in full control of the vehicle, actuating all of its degrees of freedom using handles and pedals in the cockpit. OSU created a hexapod walker in 1989 that was able to traverse most terrain [3.18]. While it was human controlled, the leg positioning was autonomous.

I found numerous other small robots designed by individuals that could accomplish stair climbing tasks (made for various school competitions and such). In all cases, these legged vehicles are large, usually not self contained, and generally slow. For most practical, all terrain use, non-legged robots are used.

In the area of non-legged robots for stair climbing, the main contenders were large tracked vehicles. The most popular one is the Urbie, a tracked vehicle from iRobot also developed under DARPA initiatives [3.13]. Urbie is a large vehicle with a design similar to that of the packbot from iRobot (described in chapter 1). The robot is able to cover most terrain and tackle a good range of obstacles while driving. When it climbs stairs, it simply dominates the geometry of the stair and drives over it. To maintain stability while climbing, it is crucial that it tracks straight up the stairs. Urbie uses a vision based algorithm for maintaining stability while climbing stairs, using information from the straight lines visible while climbing (edge of steps, rails, etc.) to determine heading information [3.12]. Work done at UPenn with Urbie has used multiple sensor inputs (accelerometer, vision and sonar), with confidence estimates into an arbitrator to choose the best one for heading estimates [3.20]. This strategy provides a much more robust heading estimate, as it can work when conditions aren't ideal for any one or two of the sensors. For example, the vision system fails in extreme lighting conditions. When that happens, the robot relies more on the sonar or body attitude from inertial measurements (accelerometer). If there's no wall next to the staircase, the sonar fails and the vision and accelerometer take over. If there's a sudden jarring motion that throws off the accelerometer, the other two are used. The control system uses an arbitrating entity to decide which sensor to use as input for the control system.

Another robot that is being used popularly for industrial applications that can climb stairs is the Andros Mark VI. The robot is large with two sets of treads that can be rotated in our out [3.14]. Similar to Urbie, it climbs stairs by dominating the geometry and driving straight up. Rather than using vision to ensure straight tracking, the robot has accelerometers onboard that relay its bank angle (relative to a vertical gravity vector) to the control software. A simple proportional control algorithm that increases the velocity of the side that's lagging behind based on the turn angle is enough to make the robot climb stairs in a stable fashion [3.15]. Just approved by the FDA for prescription use, is the iBot, the stair climbing wheelchair developed at Deka and marketed by Independence Technology [3.21]. This robotic wheelchair climbs stairs by rotating the member that connects its two driven wheels to lift it over the edge of a step. This approach to stair climbing is based on complicated control theory and uses many inertial sensors to ensure stability.

Discussion

Its clear there has been a good deal of work done regarding stair climbing with robots. In the case of legged robots doing the climbing, the size of the robot is generally very large (including power source). In addition, they aren't very fast or cheap. The class of non-legged robots able to climb stairs is predominantly robots that geometrically dominate the stairs. This makes for a robot that is still relatively large.

The design tradeoff that seems to occur is between four elements, mechanical complexity, control complexity, size, and capabilities. All of the legged robots have a high degree of mechanical and control complexity and a large form factor. Consequently, they have a good degree of functionality and can generally make it over reasonable terrain.

The non-legged robots that accomplish stair climbing have a simpler degree of control (since there is no legged locomotion involved), but they are lack the operational flexibility and have an intermediate size. To be able to climb stairs, all non-legged robots other than the GOAT are large and climb by dominating geometry of the stair [3.3]. If the robot doesn't simply climb by being larger than the steps, it has a very specific design that doesn't really lend itself to much else [3.9].

Due to the specific size requirements on the DR vehicle, a coordinated maneuver was chosen as the method of climbing stairs. Stair climbing isn't in the mission goal of the robot, resulting in many other requirements driving the physical design of the robot.

From my research, it looks like this approach to stair climbing is unique, probably due to its complexity.

Chapter 4

Control Architecture Design

A subsumption architecture was implemented to control the DR vehicle. The system was designed in a similar fashion to its predecessors.

The first step in the design of the control architecture involved identifying a set of goals for the system to accomplish [4.1]. After the goals had been established, a set of behaviors and precedence must be created. After having accomplished the first two steps, it was possible to start creating the necessary blocks to implement the said system with a subsumption architecture.

On a high level, this system was split into control of the tracks, mast, and drive train. The mast and track controls were responsible for raising the center of mass of the vehicle over the lip of the stairs. The drive train layer was responsible for keeping the robot moving up the stairs in a reasonably straight trajectory. There was some interaction between the mast and tracks layers (described later), but the drive layer remained isolated from the other two.

Unlike other subsumption systems that controlled movement, this one lacked a central coordinating entity, like the central pattern generator of The Ghengis robot [4.2]. The reason for not having a centrally coordinating structure was to keep all blocks as simple as possible, as per Brook's rules of subsumption [4.1]. In a sense, the design of the architecture causes a pattern generator to emerge. This is discussed more in depth later.

Pose Sequence

The robot climbs stairs by moving itself through the sequence of poses shown below.

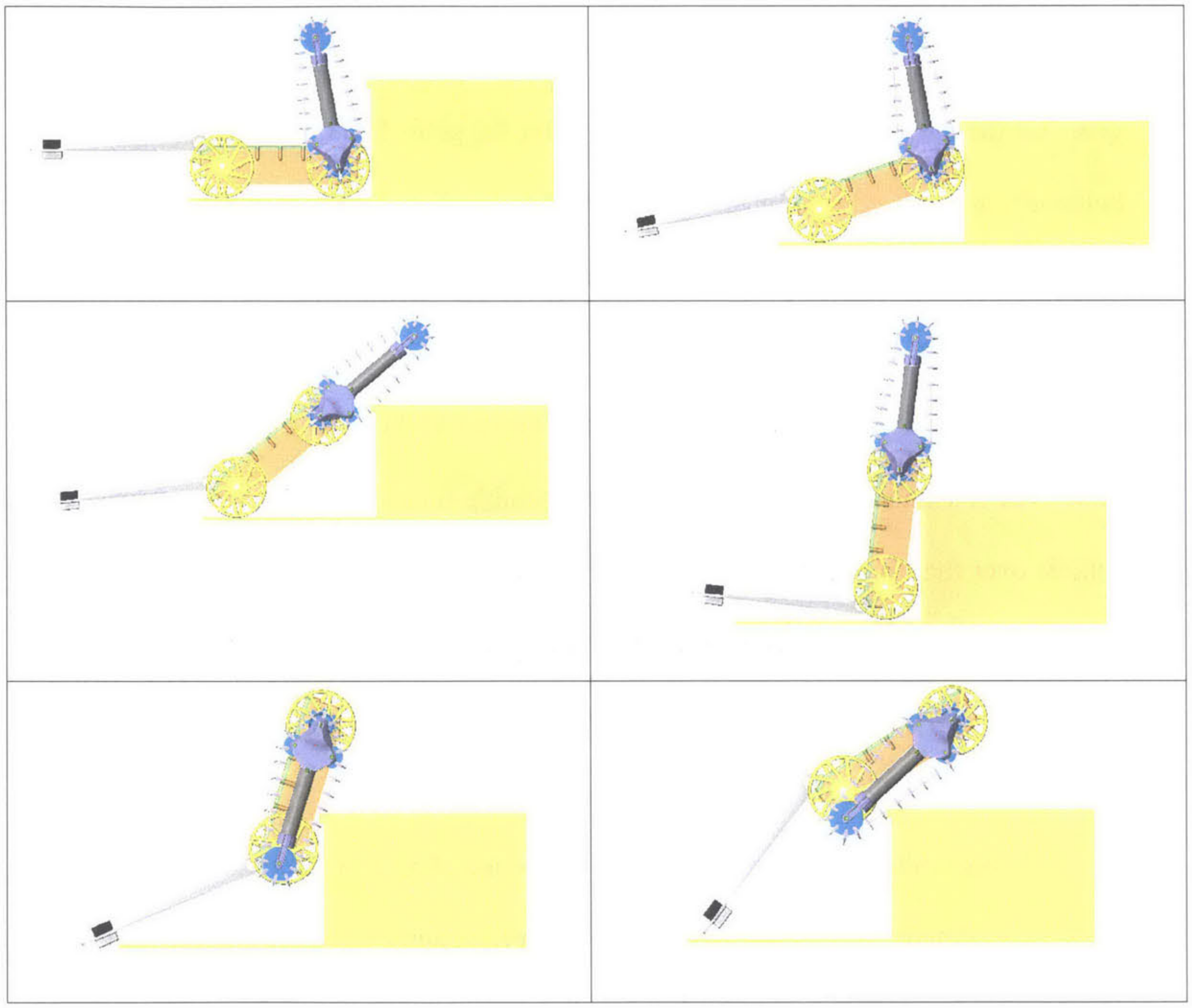


Figure 22 - Stair climbing maneuver. Left to right, top to bottom steps 0 through 5

This sequence of poses utilizes the knobby treads to grab the corner of the stairs and pull the robot up. During that process, the mast pushes the back end of the robot off the ground. To keep the middle of the robot from being stuck on the corner of the step, the tracks rotate back around to help push over again, using the knobby tracks.

To accomplish this sequence, a set of behaviors is established for each component of the robot. As the robot moves, the simple behaviors for the individual limbs interact to produce complex motion.

High-Level Goals

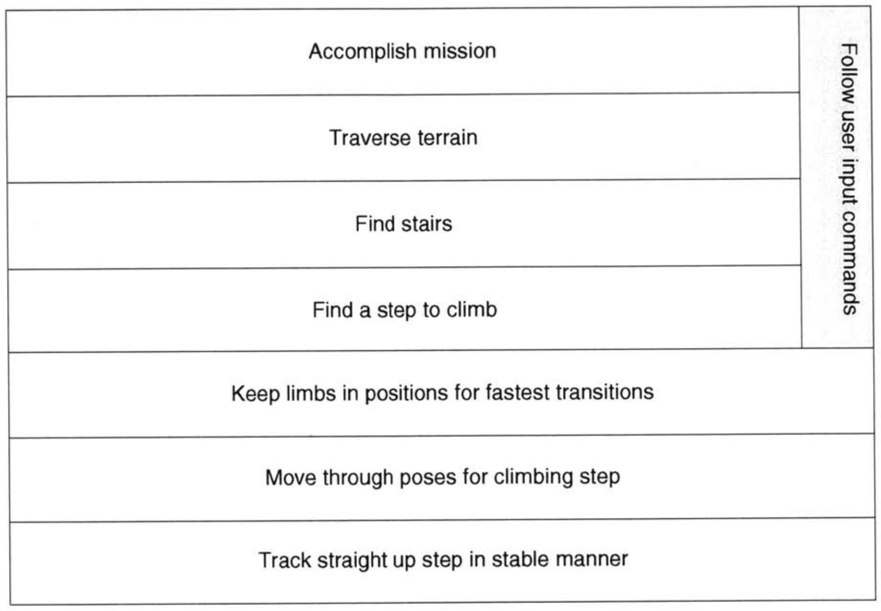

Accomplish mission
Traverse terrain
Find stairs
Find a step to climb
Follow user input commands
Keep limbs in positions for fastest transitions
Move through poses for climbing step
Track straight up step in stable manner

Figure 23 - High-level goals for climbing

The high-level goals for stair climbing are shown in Figure 23. These goals describe the most general behaviors the robot needs to exhibit. It is useful to note that the highest-level goals are all user controlled. This arises from the fact the robot is teleoperated and the human operator controls most functionality of the vehicle. The autonomous climbing is only meant to be an operational crutch for maneuvers where the user interface lacks functionality.

The layering of the high-level goals implies precedence for their execution. At all times stability is most important. As the levels get higher, they become less critical to the survival of the robot or execution of the desired action. Functional blocks for the tracks, mast and drive train implement the non-operator goal layers. The following sections describe the goals for each functional block, the building blocks to create these functional blocks, and finally the actual blocks themselves.

Tracks

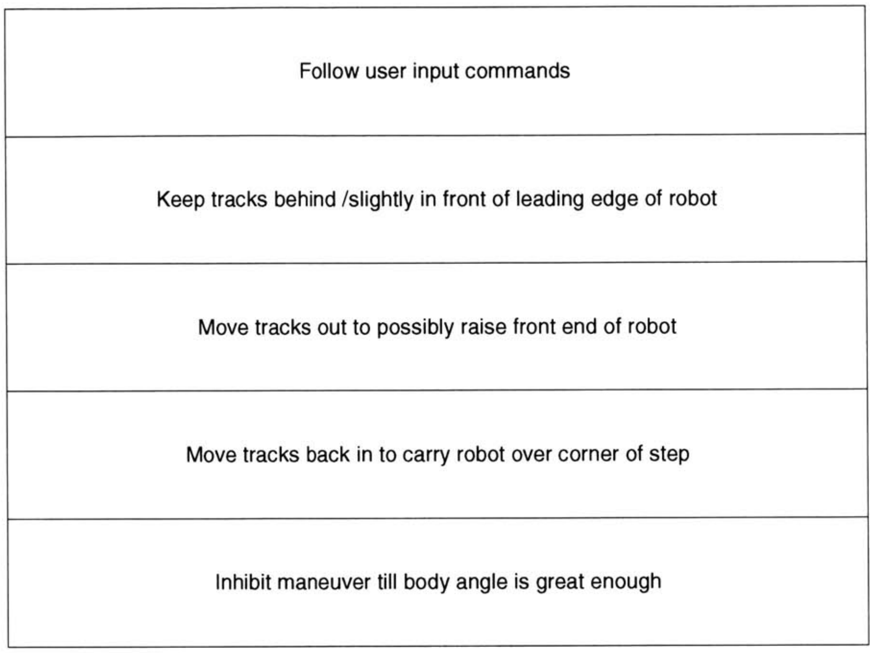

Follow user input commands
Keep tracks behind /slightly in front of leading edge of robot
Move tracks out to possibly raise front end of robot
Move tracks back in to carry robot over corner of step
Inhibit maneuver till body angle is great enough

Figure 24 - Track arm control goals.

The goals for the track arm control layer are shown in Figure 24. The lowest layer functions as a goal to keep the tracks in the best position for the next transition. All of the other layers move the track arms throughout the maneuver. These layers are consistent with the general layering of the overall goals for the autonomous climbing.

Mast

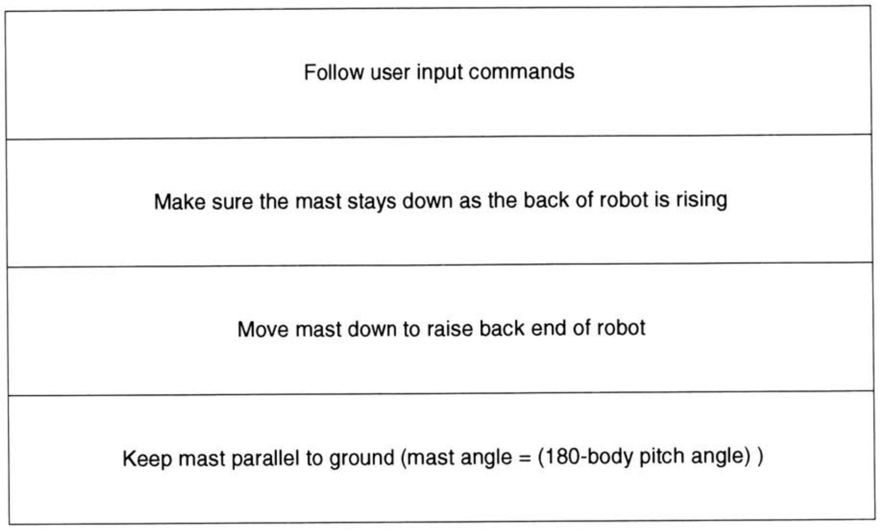

Follow user input commands
Make sure the mast stays down as the back of robot is rising
Move mast down to raise back end of robot
Keep mast parallel to ground (mast angle = (180-body pitch angle))

Figure 25 - Mast control goals

The goals for mast control are shown in Figure 25. At the beginning of the maneuver (shown in Figure 22), the mast is kept parallel to the ground to keep the robot from falling over backwards. In addition, the time required to start raising the rear section of the robot (the second layer from the bottom), is minimized when the mast is almost in position. The 3rd layer is present to make sure that the mast stays down as the body angle moves past the point where the robot thinks it is horizontal.

Drive

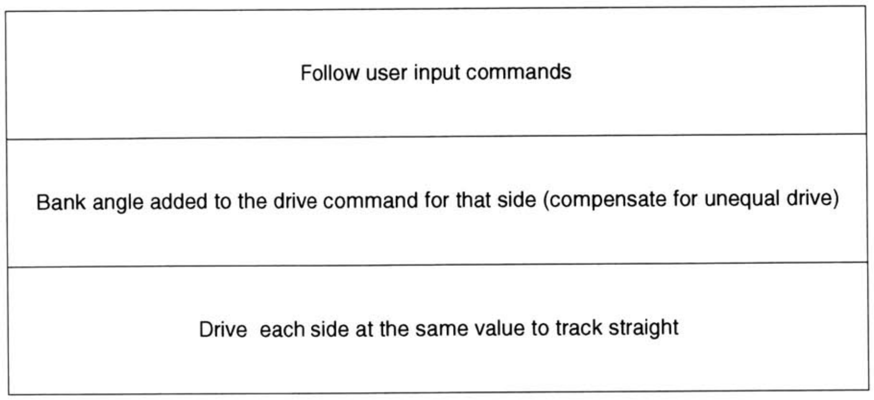

Follow user input commands
Bank angle added to the drive command for that side (compensate for unequal drive)
Drive each side at the same value to track straight

Figure 26 - Drive control goals

The goals for the drive functional block are shown in Figure 26. These goals are a bit different from the rest, as they all deal with stability. Through the entire climbing maneuver, the goal is for the robot to track straight. To accomplish this, a proportional drive based on the banking angle (angle from left to right from the robot's point of view) is used. This is a method similar to that used in other stair climbing vehicles [4.3].

At the onset of this project, I thought there would be some closed-loop velocity control on the drive motors. Due to the dynamic range required from the drive train, the motors are brushless motors with sensors for stator position. The controller runs the motors in a sensored mode for low velocities, and moves to sensorless control when the speed increases. The controller was not able to do velocity control in the sensored region. As a result, this layer is critical to function as the commands to the drive motors result in a torque, rather than a velocity. Since the mechanics and physical constants are not symmetric, there is a good deal of fluctuation in the drive speeds.

Behavioral Blocks

After a set of goals for each functional block have been established, they are implemented with a network of behavioral blocks. These blocks act as the control architecture's interface to the robot and the building blocks of functionality. The blocks function as elements that process information in close to real time, and provide modular, independent units to create the overall subsumption architecture.

Wires connect each of these blocks together, carrying the output of one to the input of another. Wires can have a value (the output of the box connected to the inputs), or be empty (if the block connected to its input has no output). Each of these blocks can be thought of as an FSM, transitioning on its own local clock. At each clock cycle, the state transition is dictated by the input to the FSM. The state the machine is in dictates the output.

All of the blocks used to construct the control architecture are shown in Figure 27.

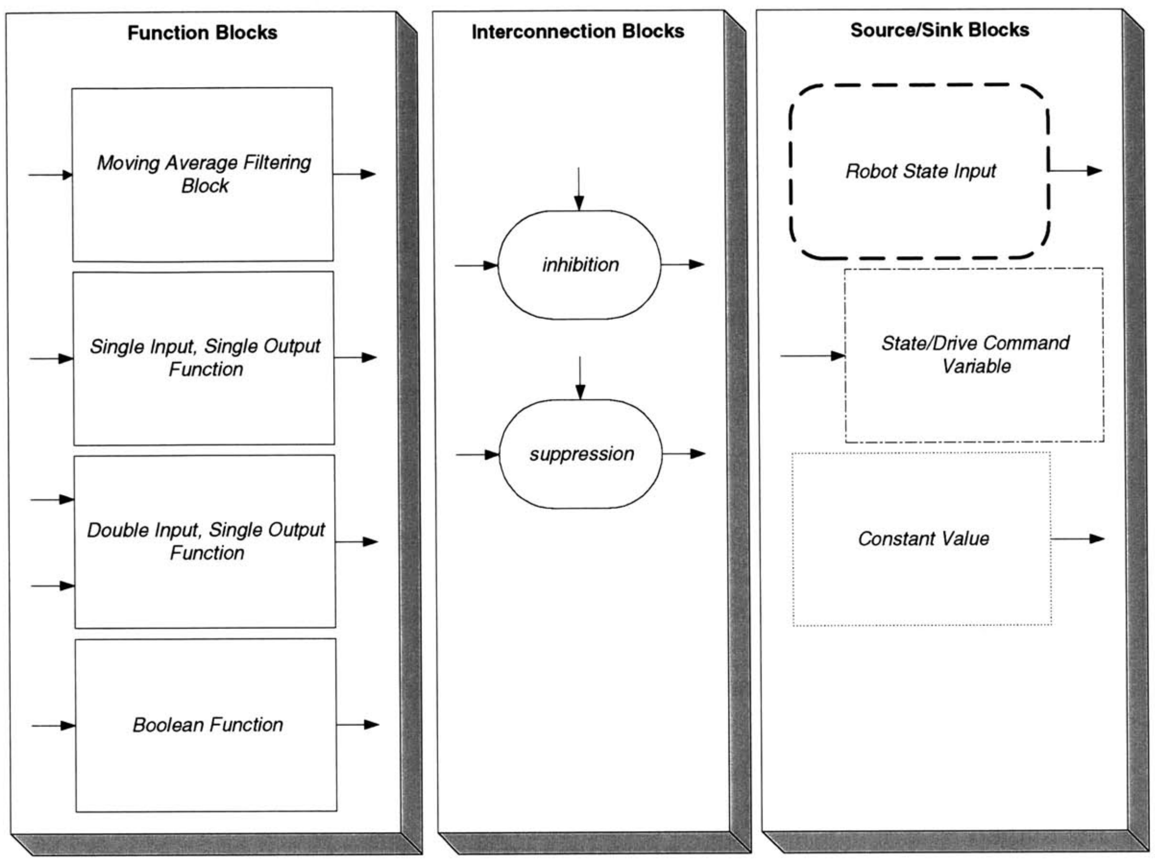

Function Blocks
Moving Average Filtering Block
Single Input, Single Output Function
Double Input, Single Output Function
Boolean Function
Interconnection Blocks
inhibition
suppression
Source/Sink Blocks
Robot State Input
State/Drive Command Variable
Constant Value

Figure 27 - Behavioral blocks

The blocks are organized in to three categories, function, interconnect and source/sink.

The function blocks take care of computational tasks that may be unique to this application of a subsumption architecture. All of the function blocks except for the moving average are implemented separately in each instance (ie – the boolean function isn't always the same in the "Boolean Function" blocks). In the case of the moving average filtering block, the only parameter is the size of the window. This block provides rudimentary first-order filtering on sensor input. The functional blocks get their input from the source blocks.

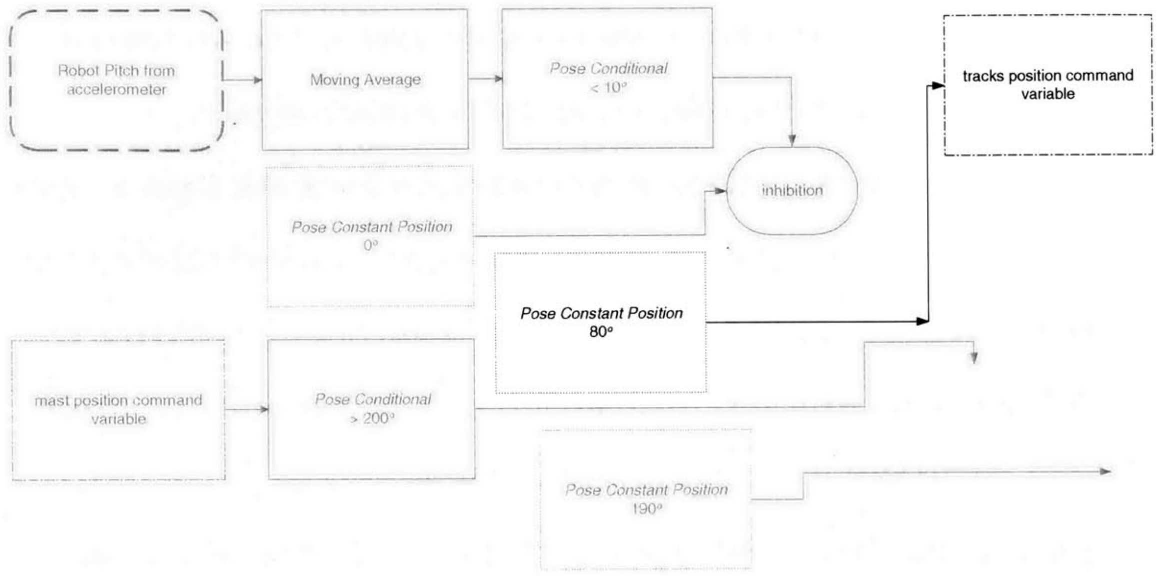
The source/sink blocks take care of the interface of the control architecture to the physical robot. They act as an abstraction barrier that takes care of running actuators and pulling data from sensors. They run at a high sampling rate to work as close to real time as possible. Background machinery takes care of writing appropriate values from the physical system into source blocks, and sending the values from the sink blocks to actuators. The interconnection blocks connect all of the blocks together.

The interconnection blocks are the standard ones described in Brook's original subsumption architecture [4.1]. The inhibition and suppression blocks have input, output and sidetap ports. In the illustration, the input and output ports are parallel and the sidetap port is orthogonal to the other two. In input on the sidetap dictates the behavior of both of these blocks. If there is no input available on the sidetap, the output equals the input in both the inhibition and suppression blocks. In the inhibition block, if there is data on the sidetap the output becomes empty. In the suppression block, the output gets the data from sidetap if its available. The use of these blocks allows layers of control to be connected together in a hierarchical fashion.

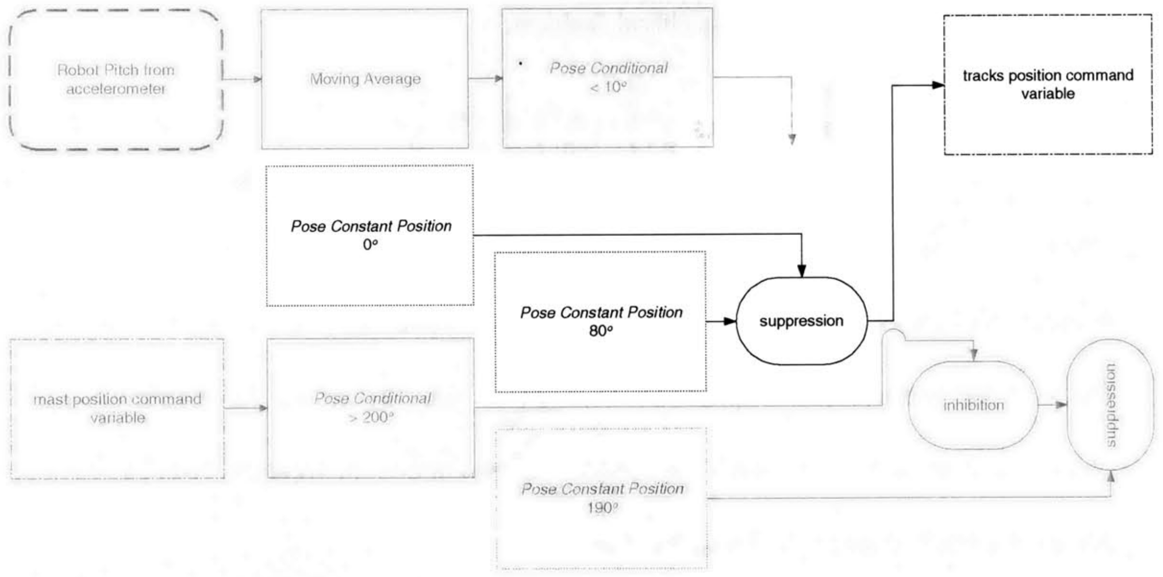
Interconnection of Blocks

Each functional block is composed of smaller functional blocks, tied together with interconnection blocks. Each smaller functional block is composed of function and source/sink blocks connected to create a datapath. Each of these smaller functional blocks embodies the earlier specified layers of control. In this section the functional block that implements the goals for control of each degree of freedom (tracks, mast and drive), are built in an incremental fashion.

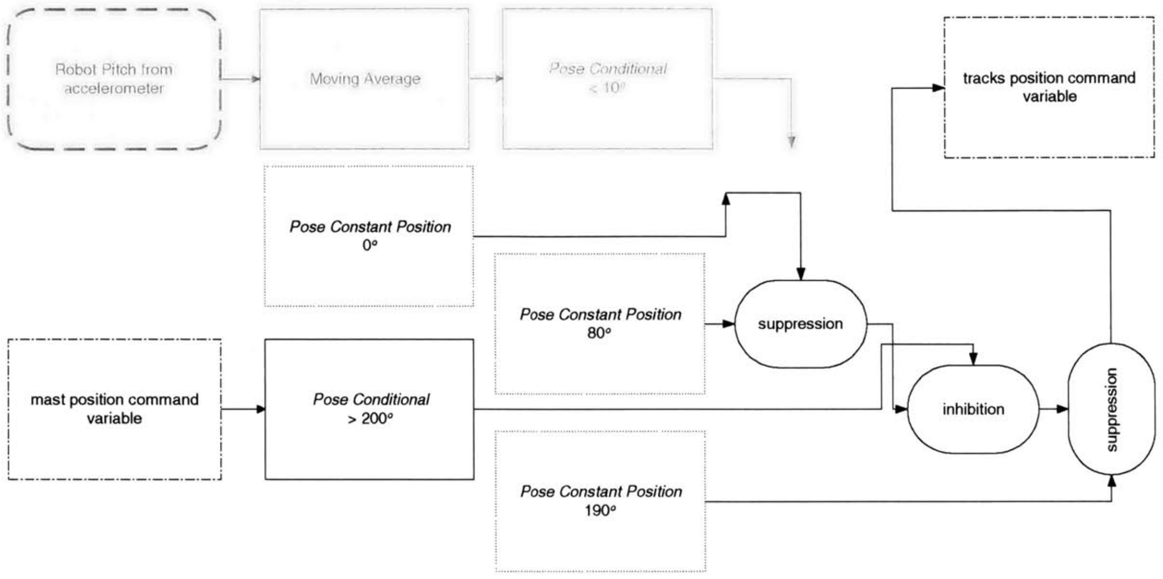
Tracks


Robot Pitch from accelerometer
Moving Average
Pose Conditional < 10°
tracks position command variable
inhibition
Pose Constant Position 0°
Pose Constant Position 80°
mast position command variable
Pose Conditional > 200°
Pose Constant Position 190°

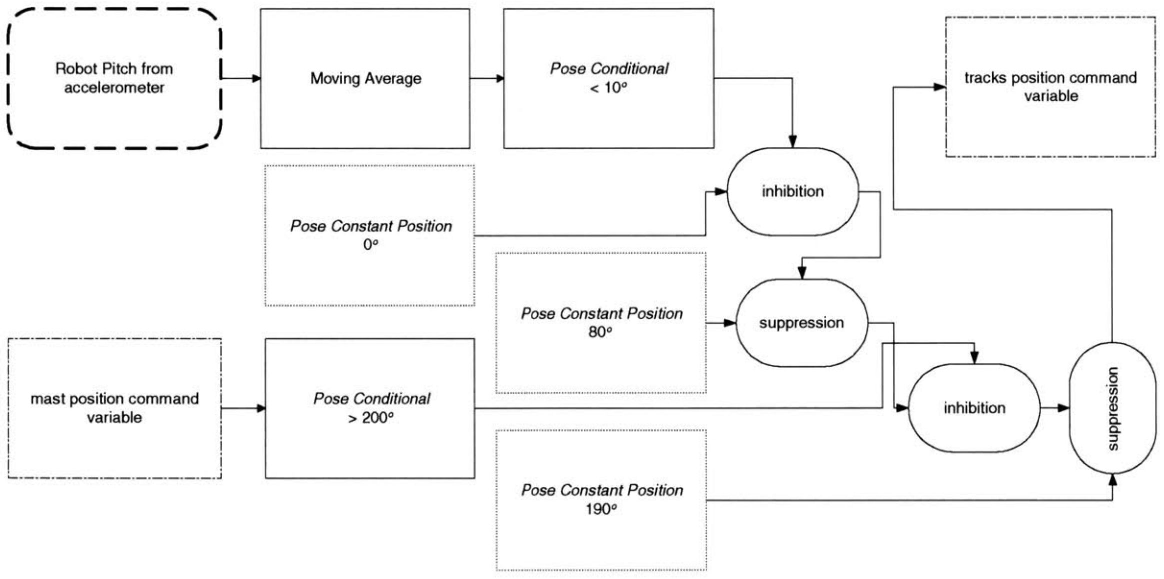
The highest layer keeps the track arms in a position behind the leading edge of the vehicle.


Robot Pitch from accelerometer
Moving Average
Pose Conditional < 10°
tracks position command variable
Pose Constant Position 0°
Pose Constant Position 80°
suppression
mast position command variable
Pose Conditional > 200°
inhibition
suppression
Pose Constant Position 190°

The next layer down tries to move the tracks out to lift the front end of the vehicle. At this stage, it continuously suppresses the lowest layer.

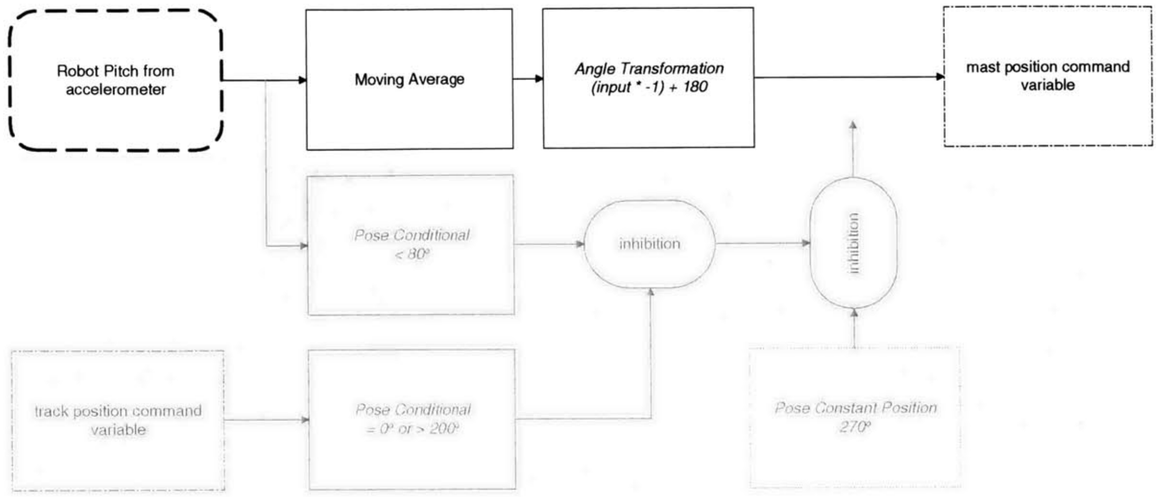

Robot Pitch from accelerometer
Moving Average
Pose Conditional < 10°
tracks position command variable
Pose Constant Position 0°
Pose Constant Position 80°
suppression
mast position command variable
Pose Conditional > 200°
inhibition
suppression
Pose Constant Position 190°

The next layer down moves the tracks back to climb over the edge of the step. There is a conditional to ensure that the mast is positioned down to lift the back end of the robot before this layer is activated.


Robot Pitch from accelerometer
Moving Average
Pose Conditional < 10°
tracks position command variable
inhibition
Pose Constant Position 0°
Pose Constant Position 80°
suppression
mast position command variable
Pose Conditional > 200°
inhibition
suppression
Pose Constant Position 190°

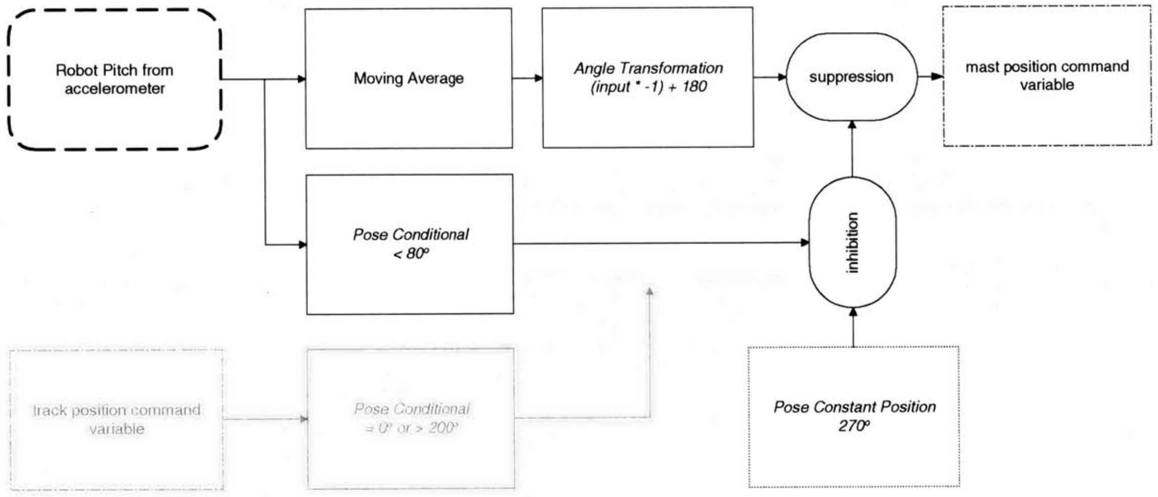
The final, lowest layer is the most important, inhibiting action until the body angle is high enough. Its positioning allows it to control inhibition and suppression of higher layers.

Mast



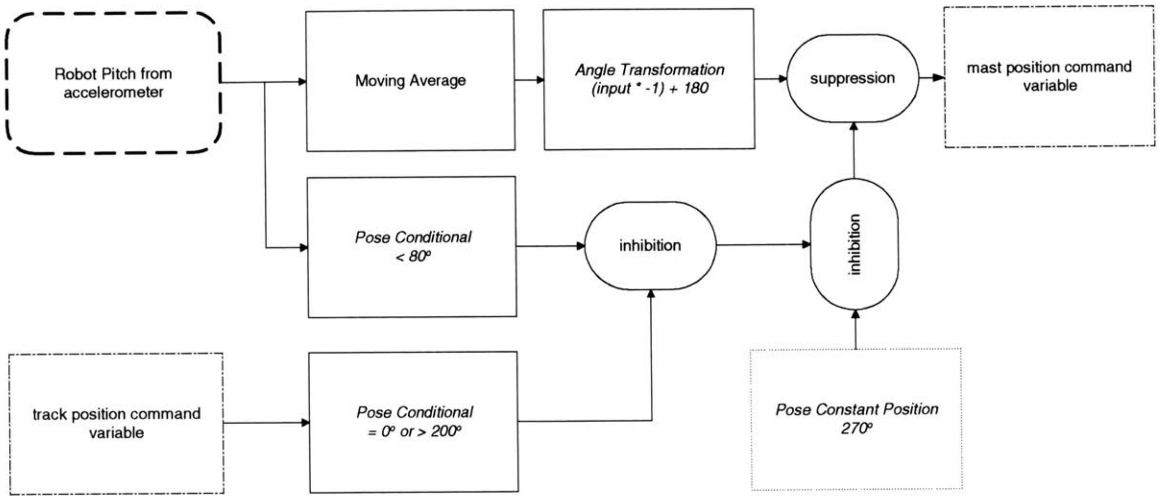
Robot Pitch from accelerometer
Moving Average
Angle Transformation (input * -1) + 180
mast position command variable
Pose Conditional < 80º
inhibition
inhibition
track position command variable
Pose Conditional = 0º or > 200º
Pose Constant Position 270º

In the lowest level, the goal is to keep the mast parallel to the ground at all times



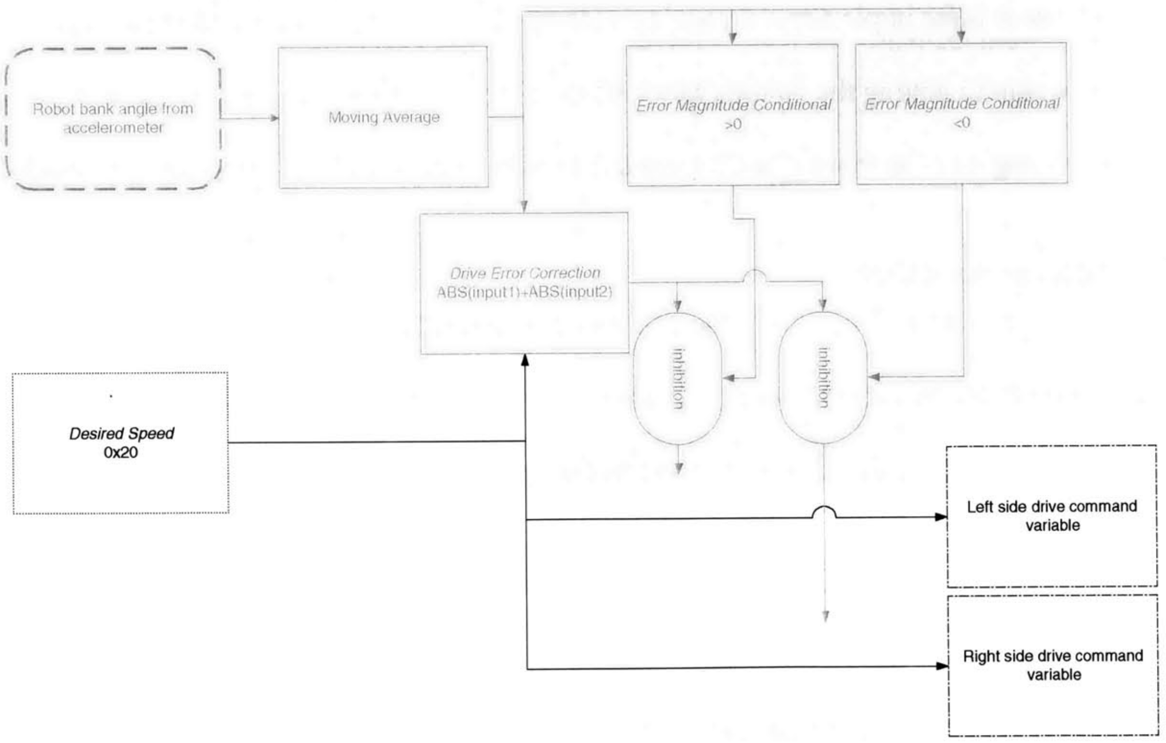
Robot Pitch from accelerometer
Moving Average
Angle Transformation (input * -1) + 180
suppression
mast position command variable
Pose Conditional < 80º
inhibition
track position command variable
Pose Conditional = 0º or > 200º
Pose Constant Position 270º

The next level adds the ability to move the mast down to raise the back end of the vehicle when the body pitch is at the right level.

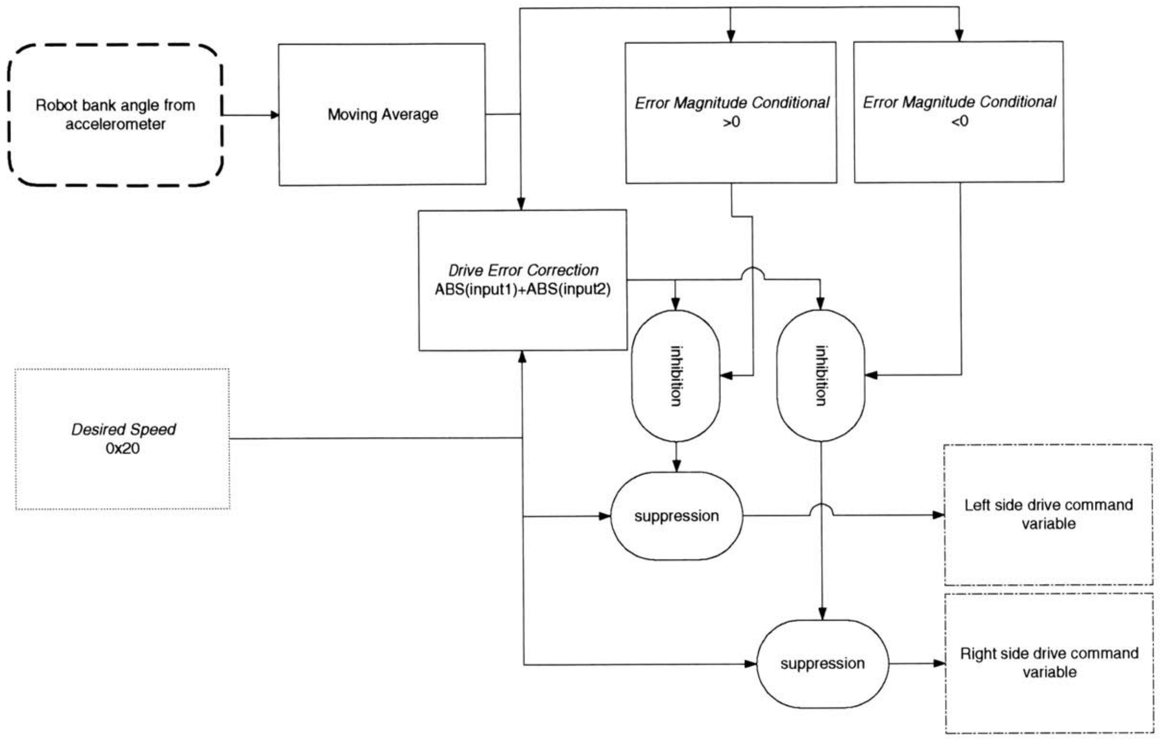

Robot Pitch from accelerometer
Moving Average
Angle Transformation (input * -1) + 180
suppression
mast position command variable
Pose Conditional < 80°
inhibition
inhibition
track position command variable
Pose Conditional = 0° or > 200°
Pose Constant Position 270°

The final layer adds a conditional to ensure that the mast stays down even after the robot has passed the horizontal position when it is entering the run of the top of the step.

Drive


Robot bank angle from accelerometer
Moving Average
Error Magnitude Conditional >0
Error Magnitude Conditional <0
Drive Error Correction ABS(input1)+ABS(input2)
inhibition
inhibition
Desired Speed 0x20
Left side drive command variable
Right side drive command variable

The first layer simply sends the same drive command to both motors. Since there is no closed loop control for velocity around the motors (and frictional constants are not known), another layer is necessary to ensure the robot tracks straight up the stairs.



Robot bank angle from accelerometer
Moving Average
Error Magnitude Conditional >0
Error Magnitude Conditional <0
Drive Error Correction ABS(input1)+ABS(input2)
inhibition
inhibition
Desired Speed 0x20
suppression
suppression
Left side drive command variable
Right side drive command variable

The next layer implements a tracking strategy similar to that used by Martens and Newman in getting the Andros Mark VI to climb stairs [4.3]. The algorithm looks at the bank angle of the robot (the tilt from left to right) and adjusts the drive proportionally.

Interconnection

From the diagrams of the control networks, it is evident that the topology of the connections dictates the behavior. There are many conflicting goals at any point, whose priorities are decided by the interconnection blocks.

Desired Emergent Behavior

While each layer or block may exhibit some simple functionality, the total emergent behavior is the interesting element of using a subsumption architecture. When all the layers interact, the input to output mapping is different from what might appear by using just one of the layers. Even between the three distinct functional blocks (tracks, mast and drive train), there is interconnection through the environment.

While there is no distinct central pattern generator for this system, the behavior of one emerges from the interaction of the blocks. Using the pose conditionals and interconnection blocks, the system controls itself, making the transition from one pose to another. Rather than containing one entity that coordinates these transitions, many blocks work in unison to achieve the same effect. This is one example of an emergent behavior from the system. Rather than being a behavior that is observable externally, the emergent behavior is a structural one that plays and integral part in the control of the robot.

Feedback loops

The use of subsumption architecture is essentially design of a feedback system. In this case, the outside environment and inertial sensors for body attitude complete the loop. Between the main functional blocks, there are connections such as the commanded mast and track position.

Analysis

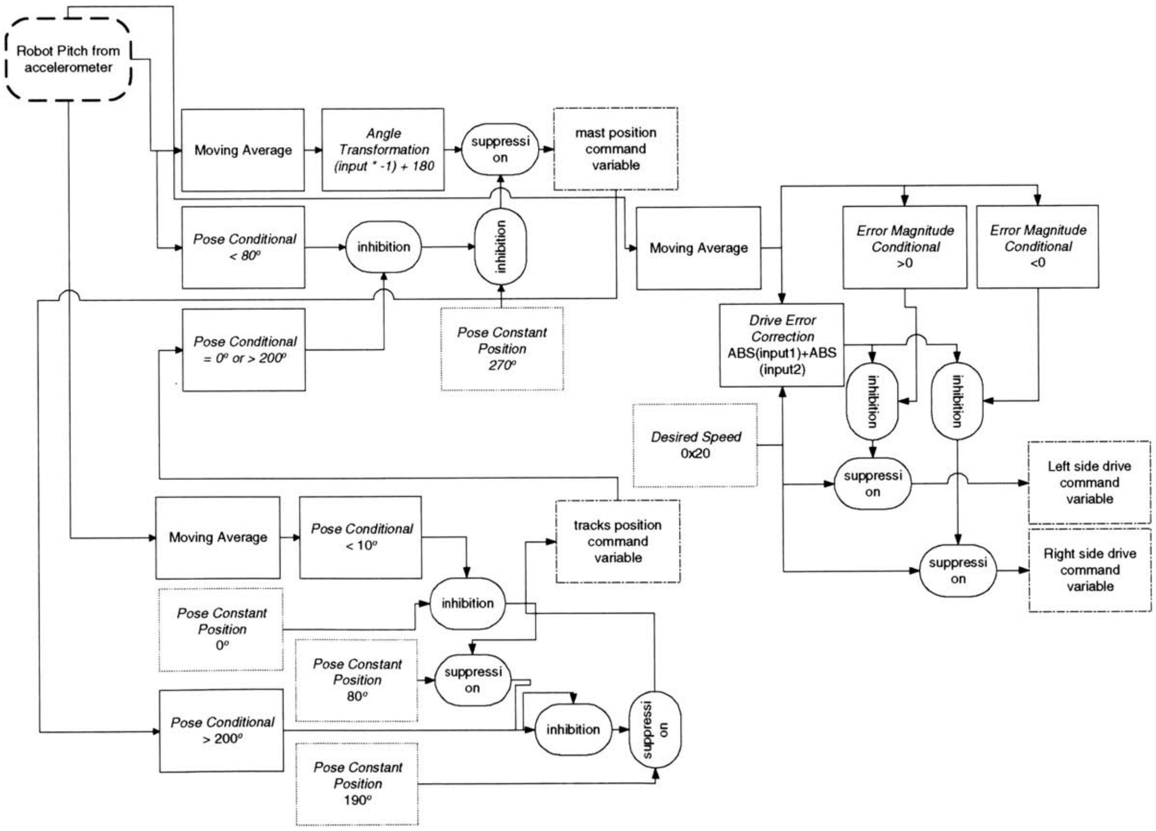
Analysis of this system is very difficult. A wise man once said, "All simulations are doomed to succeed". That statement is very true and very applicable to this project. One of the goals of this project is feasibility and applicability. There is no point in having a control system that works in a simplified simulation and has to be redone for the actual application. For these reasons, no simulation was used in the development of this control system.

All evaluations and analysis of the performance of this control system had to happen on the actual vehicle. This ensures that the algorithm works with all the real life variables and non-idealities.

Degrees of Freedom in Design

Even with a general control topology, there are still degrees of freedom in the design of control architecture. Most of the variables lie in constants that dictate the intermediate poses during stair climbing. The pose conditionals all look at body attitude to decide if its time to inhibit certain data lines. These actions cause different layers to take control and change the pose of the robot.

Getting the robot to climb stairs robustly will require tweaking these values. Since there is no automated simulation, there is no quick way to choose the optimal values of these constants. Their starting values (shown in Figure 28) were chosen through careful trials, manually controlling the robot.



Robot Pitch from accelerometer
Moving Average
Angle Transformation (input * -1) + 180
suppression
mast position command variable
Pose Conditional < 80°
inhibition
inhibition
Pose Conditional = 0° or > 200°
Pose Constant Position 270°
Moving Average
Error Magnitude Conditional >0
Error Magnitude Conditional <0
Drive Error Correction ABS(input1)+ABS (input2)
inhibition
inhibition
Desired Speed 0x20
suppression
Left side drive command variable
suppression
Right side drive command variable
Moving Average
Pose Conditional < 10°
tracks position command variable
inhibition
Pose Constant Position 0°
Pose Constant Position 80°
suppression
Pose Conditional > 200°
inhibition
suppression
Pose Constant Position 190°

Figure 28 - Diagram of full subsumption control architecture

These careful trials generally involved moving the robot by hand into the intermediate poses, shown in Figure 22. *Pose conditional* and *pose constant* blocks were given the values of the accelerometers at each of the poses. The desired speed was set to the lowest value where motion still occurred. If the vehicle moves too fast, the update rates in the system and time constants for actuators would not be able to keep up.

Chapter 5

Implementation

This entire system was implemented in software. I used Microsoft Visual C++ to write and compile code for the Omni 1400 Single Board Computer. The software to implement the subsumption control architecture was layered on top of the standard control software running onboard the robot.

Software Architecture

There are many layers to the software used to control the DR vehicle. As mentioned before, all of the software for this project resides onboard the vehicle in the "Autonomous Control" block of Figure 29.

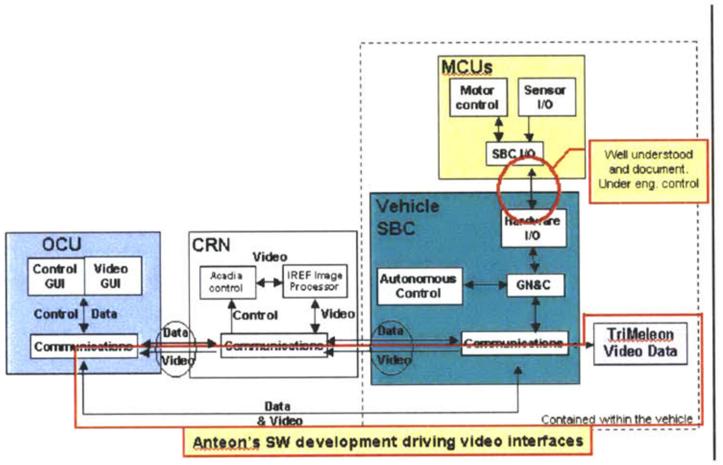


Figure 29 - Software system block diagram

Also in the "Autonomous Control" block resides software for retrotraverse, and other software assisted maneuvers executed by the robot. The entire interface to the rest of the vehicle occurs through lower level software written by other developers on this project. In Figure 29, the OCU is the operator control unit (described in chapter 1). The CRN is the communication relay node. The CRN acts as a repeater or a higher level coordinating entity (it can talk to many robots at once). The MCU is the master control unit, taking care of low level control of motors and sensors. The interface provided was very simple, and consisted of only three or four functions that allowed insertion of pose commands into an execution queue, and retrieval of state information from the robot.

Operating System

The entire system of the vehicle SBC (single board computer) on which the autonomous climbing (hereafter referred to as autoclimb) software resided ran Windows CE. The subsumption architecture was implemented by creating multiple threads (described in detail in the following sections). Due to the nature of windows task handling, there were no real-time guarantees. This drawback strengthens the argument for using the subsumption architecture to control the robot. Brook's original design of the subsumption architecture was to control a robot running in an asynchronous, uncertain world. The lack of timing guarantees is overcome by running loops at higher than necessary rates inside each of the independent threads.

Object Organization and Implementation

Completely independent threads in memory realize each subsumption block. These threads are created by descendents of a *CSubsumptionBaseBlock* class, shown in Figure 30.

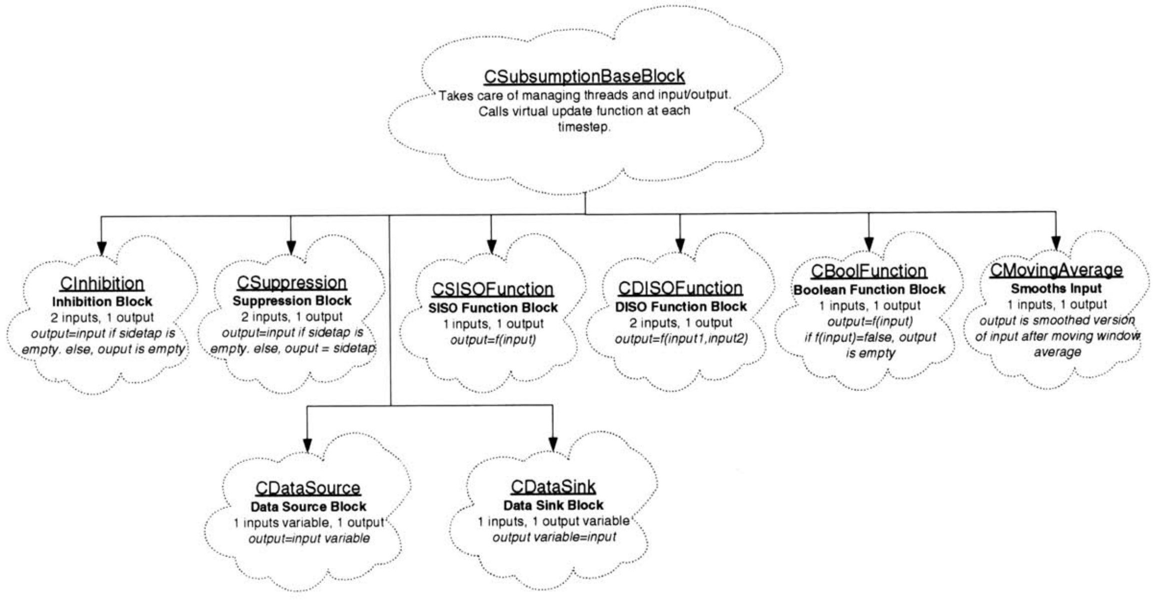


Figure 30 - Object diagram

CSubsumptionBaseBlock is an abstract class that takes care of thread management. Upon its start, it runs in a periodic loop (no timing guarantees), updating internal state and outputs at each execution with a pure virtual update function. The descendents of *CSubsumptionBaseBlock* implement the update function and an initialization routine to define inputs and outputs to create a class that can be instantiated. This generic structure allows simple creation of a vast number of useful blocks. *CWire* objects connect subsumption blocks together.

The *CWire* interface is very simple, providing access to shared memory. Each object can have only one source and any number of recipients. Source and recipient blocks receive a pointer to the *CWire* object that connects them. A set of three simple functions provide the interface to put data into the wire, remove it, and check if the wire is empty.

The function blocks (*CSISOFunction, CDISOFunction,* and *CBoolFunction*) are generic objects that take a pointer to a function they evaluate during their update. They implement blocks that do simple math, or conditional blocks.

The sources and sink blocks (*CDataSource* and *CDataSink*) act as part of the abstraction barrier between the subsumption architecture and the interface of the robot. Both of these objects connect a *CWire* object to a variable. In the case of the *CDataSource*, the value of the variable is pushed into the *CWire*. In the *CDataSink*, the variable receives the value from the *CWire*. There are threads running outside of the subsumption architecture code that interface the variables to the command queue for actuators and input from robot sensors.

The *CMovingAverage* block allows conditioning of input data. It is reasonable to assume there will be a fair amount of noise on the inputs. This block provides a real time moving window average on the data at its input port. The filter is very simple and does decrease overall bandwidth. The parameters on the block allow configuration of the width of the window.

Notes

It is important to note that this subsumption architecture is abstracted to behave as if it is continuous, although it is discrete [5.1]. Due to timing issues inherent in windows programming, there are no real-time guarantees for any processes that are running. In addition, the timing for all of the processes was arbitrarily set at 30ms to begin with (update function is called every ~30ms). It is likely that the processes must run much faster to be able to do their job, but that is to be determined through the empirical tests. The time constants for the actuators are actually quite slow, and could affect the configuration of many components of this system.

Chapter 6

Results

A robot was not available to test the entire system. In the end, I wrote code, compiled and ran limited sections of the subsumption diagram presented in the previous chapter. I was able to test the layer that ensured the mast stayed parallel to the ground (layer 0 of the mast functional block). When the layer was run, the mast did indeed stay parallel to the ground. I was able to hold the robot in my hands and change the pitch angle of the robot, and the mast stayed parallel to the ground. There was a bit of phase delay resulting from the moving average filter, but not really significant compared to the delay from the time it took to actually move the mast (due to limits on the speed of the motors). Due to lack of software interface available, I wasn't able to get graphs and quantifiable results of mast movement and the actual performance of the control architecture.

While working on the drive layer, I discovered some bugs in the programming of the speed controllers. I wasn't able to gain access to a robot to do further tests in time to retest the drive train with the full architecture. From the behavior expressed during the tests with the faulty controller, it did look like it seemed to work as the drive for a side would start to speed up as the robot tilted to that side. This isn't very representative since the bug in the controllers would cause the motor to start jittering and it was hard to get a sense of how fast it was actually trying to go. In addition, it wasn't run on the ground, making it subject to the real world artifacts of slippage, etc.

Chapter 7

Discussion

Since I wasn't able to perform extensive tests on the performance of the designed subsumption architecture, this project provided more insights about the benefits of various approaches to control of the robot. Each of the approaches, subsumption, contraction analysis and evolved architectures had their own merits and would appropriate in different situations.

Evolved architectures have been used in very compelling ways to solve various engineering problems. They work well in searches of a relatively low dimensional space with a suitable fitness function. The problem with using an evolved architecture lies in the fitness function. In a robotic application such as the one presented in this paper, the goal is some complicated motion. As a result, the fitness function is very complicated and computationally intensive. These drawbacks are overcome if one possesses a great deal of computational resources or a lot of time. Another approach is to simplify the simulation/fitness function to decrease the computational complexity (in the case of this project, perhaps constrain simulated motion to two dimensions). While making the problem simpler, you most likely remove the elements that made it difficult in the first place (sensor noise, slippage, etc.). In the end, you often find a solution that is "doomed to succeed". While the solution may work well in simulation, it doesn't necessarily transfer to the real world. A similar problem arises with contraction analysis.

To apply contraction analysis to a problem, a comprehensive mathematical model must exist for the system. The benefit to using contraction analysis is that the solution you devise is guaranteed to work (assuming the model is correct or very close to it). The drawback lies in the fact a full mathematical model of the system is required. In the case of a highly non-linear system (such as this one), the state space model of motion is very difficult to devise. Some simplifications are definitely required. After the simplifications, you run into the same problems as the evolved architecture. If you don't have any simplifications, all you can prove is that the solution will work for the exact system you have modeled in your state space equations.

In the engineering task of mobile robotics, one fundamental problem is uncertainty. Since the robots live and act in the real, high dimensional world, it's hard to account for and simulate everything that they will experience. From the perspective of time effectiveness, it often takes longer to create a consummate simulation than to just test something in the real environment. For those reasons, in the case of mobile robotics, a subsumption control architecture is most appropriate.

The subsumption architecture has nice aspects in both design and application. Due to its decentralized nature, the various components are designed and implemented with no knowledge of any others. This makes the subsumption architecture more fault tolerant as each unit block and layer is very robust. The architecture is designed to give good results from an intuitive point of view, rather than a mathematical one. The design of a subsumption architecture control system utilizes higher level design on the part of the human creating it, rather than mathematical methods. As a result, the solution is often more complicated, but not quantifiable and provable. If one operates under the assumption that simulations are not a realistic measure of functionality, this is not a problem.

The drawback to a subsumption architecture is that you must have a platform to evaluate the performance of the control architecture. That was not the case in this project. As a result, only partial results could be concluded. For the layer that was tested, keeping the mast parallel to the ground, the control architecture worked pretty well. The functionality of that layer proves that the software architecture for implementing the subsumption control is functional. It also demonstrates a functionality that is independent of all the uncertainties inherent in the system (sensor noise, no real-time guarantees, etc.). There were a few modifications necessary while testing on the actual robot (as expected), to account for offsets in potentiometers and variations in motors.

While concluding the entire system would robustly climb stairs successfully is a matter of future work, it is reasonable to say that it looks like a subsumption architecture could accomplish the goals of this project given enough time to test and tweak on a functional robotic platform.

Works Cited

[1.1] http://www.cnn.com/2002/TECH/science/08/01/packbot/ . Accessed 11-24-02.

[1.2] Weagle, David and Pete Kerrebrock. *High Mobility Impact Survivable Small Robotic Vehicle.* AUVSI 2002.

[2.1] Lohmiller, Winfried and Jean-Jacques Slotine. *On Contraction Analysis for Non-Linear Systems.* Automatica, Vol. 34, No. 6, pp. 683-696, 1998.

[2.2] Slotine, Jean-Jacques and Winfried Lohmiller. *Modularity, Evolution and the Binding Problem: A View from Stability Theory.* Neural Networks 14, 2001.

[2.3] Bizzi E., Giszter S.F., Loeb E., Mussa-Ivaldi F.A., Saltil P. *Trends in Neurosciences. Review 18:442.* 1995.

[2.4] Brooks, Rodney. *A Robust Layered Control System for a Mobile Robot.* IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, March 1986.

[2.5] Brooks, Rodney and Cynthia Breazeal. *Draft of book distributed in 6.836 Spring 2002.* Draft of July 24, 1998. pp 33-44.

[3.1] Brooks, Rodney and Cynthia Brazel. *Draft of book distributed in 6.836 Spring 2002.* Draft of July 24, 1998. pp 33-44.

[3.2] Mataric, Maja J. *Integration of Representation into Goal-Driven Behavior-Based Robots.* IEEE Transactions on Robotics and Automation, Vol. 8, No. 3, June 1992.

[3.3] *GOAT Model Photos.* http://www-2.cs.cmu.edu/~trb/goat/. Last accessed December 11, 2002.

[3.5] Buehler, M., R. Battaglia, A. Cocosco, G. Hawker, J. Sarkis, K. Yamazaki. *SCOUT: A simple quadruped that walks, climbs and runs.*

[3.6] *MIT Leg Lab 2D Hopper.* http://www.ai.mit.edu/projects/leglab/robots/2D_hopper/2D_hopper.html. Last accessed December 11, 2002.

[3.7] M. Buehler, " Dynamic Locomotion with One, Four and Six-Legged Robots ," Invited Paper, Journal of the Robotics Society of Japan, 20(3):15-20, April 2002.

[3.8] E.Z. Moore and M. Buehler, "*Stable Stair Climbing in a Simple Hexapod*", *4th Int. Conf. on Climbing and Walking Robots*, Karlsruhe, Germany, September 24 - 26 , 2001.

[3.9] Shorya Awtar, Saket Kumar Singh, Praveen Tewari. *Stair Climbing Mechanism: Senior Year Project.* http://web.mit.edu/~shorya/www/btp.htm. Last accessed December 12, 2002.

[3.10] *Research on Walking Machines.* http://mozu.mes.titech.ac.jp/research/walk/walk.html. Last accessed December 12, 2002.

[3.11] Shigeo Hirose, Kan Yoneda, Kazuhiro Arai, Tomoyoshi Ibe; *Design of Prismatic Quadruped Walking Vehicle TITAN VI,* Proc. 5th INt. Conf. Advanced Robotics, Pisa, Italy, pp.723-728 (1991)

[3.12] Yalin Xiong, Larry Matthies. *Vision-Guided Autonomous Stair Climbing.* Proceedings of the 200 IEEE International Conference on Robtics and Automation. San Francisco, CA. April 2000.

[3.13] L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. Maclachlan, C. Won, T. Frost, G. Sukhatme, M. McHenry, S. Goldberg. *A Portable, Autonomous, Urban Reconnaissance Robot.* The 6th international Conference on Intelligent Autonomous Systems.

[3.14] *CESAR Experimental Facilities.* http://avalon.epm.ornl.gov/IS/cesar/cesar_facilities.html#andros. Last accessed December 12, 2002.

[3.15] John D. Martens, Wyatt S. Newman. *Stabilization of a Mobile Robot Climbing Stairs.* Center for Automation and Intelligent System Research. Case Western Reserve University, Cleveland, OH.

[3.16] *Honda Robot Top Page.* http://world.honda.com/ASIMO/. Last accessed December 12, 2002.

[3.17] http://www.plyojump.com/asimo.html. Last accessed December 12, 2002.

[3.18] Boone, Gary, and Hodgins, Jessica. "Walking and Running Machines". The MIT Encyclopedia of the Cognitive Sciences, MIT Press, 1998.

[3.19] Brooks, Rodney. *A robust layerd control system for a mobile robot.* IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, March 1986.

[3.20] Solomon Steplight, Geoffrey Egnal, Sank-Hack Jung, Daniel B. Walker, CAmillo J. Taylor and James P. Ostrowski. *A Mode-Based Sensor Fusion Approach to Robotic Stair-Climbing.* GRASP laboratory, University of Pennsylvania.

[3.21] *Independence Technology a Johnson & Johnson Company.* http://www.indetech.com/ibot/. Last accessed December 12, 2002.

[3.22] *Bluebotics SHRIMP Robot.* http://www.bluebotics.com/products/shrimp/. Last accessed January 14, 2003.

[4.1] Brooks, Rodney. *A robust layerd control system for a mobile robot.* IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, March 1986.

[4.2] Brooks, Rodney and Cynthia Breazeal. *Draft of book distributed in 6.836 Spring 2002.* Draft of July 24, 1998. pp 33-44.

[4.3] John D. Martens, Wyatt S. Newman. *Stabilization of a Mobile Robot Climbing Stairs.* Center for Automation and Intelligent System Research. Case Western Reserve University, Cleveland, OH.

[5.1] Brooks, Rodney. *A robust layered control system for a mobile robot.* IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, March 1986.