Proceedings of the Southern Conference on Computing
The University of Southern Mississippi, October 26-28, 2000

Control/Learning Architectures for use in Robots Operating in Unstructured Environments

P. McDowell[*], S. Iyengar[+], M. Gendron[*], B. Bourgeois[*], and J. Sample[*]

Abstract

This paper describes work being conducted at the Louisiana State University Robotics Research Laboratory towards the development of legged robot walking algorithms for unstructured environments. This work is associated with the Robo-Tiger project whose goal is the construction of a life-size robotic tiger mascot with the physical attributes of a real tiger. The focus of this research is on robots with multi-jointed legs, and the goal is to develop algorithms and architectures capable of learning new walking patterns to allow autonomous compensation for varying and unknown terrains. This paper describes the project developments to date and the details of the approaches being proposed to enable a legged robot to cope with uncertain terrains.

1. Introduction

This paper discusses current efforts at the Louisiana State University (LSU) Robotics Research Laboratory towards the development of control/learning architectures for use with robots that will operate in unstructured environments. Our primary focus is on the development of a distributed system for controlling the leg movements of a mobile robot. An architecture is desired that will allow the robot to autonomously adapt its leg and gait control to variable terrains. The targeted application for this research is the 4-legged LSU Robo-Tiger.

The LSU Robo-Tiger project is a joint effort involving LSU's Computer Science, Electrical Engineering, and Athletic departments. The idea was conceived by Dr. S.S. Iyengar and Dr. Lynn Jelinski late in the fall semester of 1999. The goal was to build a cybernetic tiger to be used to help the school mascot, Mike the Tiger, with his duties at the school's various athletic functions. In doing so, it was hoped the project would not only catch the imagination of the students and faculty, but also provide a viable research platform for studying various issues pertaining to mobile robots in unstructured environments. Under the direction of Dr. S.S. Iyengar, the LSU Computer Science department's Robot Research Lab has conducted preliminary studies to determine the feasibility of the project.

The first phase of the project consisted of constructing, modifying and programming a small prototype of the tiger. A series of three prototypes were used to identify the various issues involved in developing a full sized robotic tiger. Although the resulting prototypes are much-simplified versions of the final product, many of the ideas, concepts, algorithms and construction methods are applicable.

In the next section we present a brief overview of ongoing work and concepts in the field of mobile robotics and compare it with classical Artificial Intelligence (AI) methods. In section 3 we describe what has been done at LSU and some similarities between this work and that done at other institutions. In section 4 we describe the different gait level control algorithms implemented and some of the concepts under development at LSU that are being used to tackle the challenges posed by changing environments. Finally, goals for continuing investigation are discussed and a summary of the work completed to date is given.

2. Background

Several researchers have tackled the problems associated with walking and sensor perception (vision and hearing). There are excellent examples of four legged walking machines, including those created by MIT and McGill University. The MIT AI Lab has focused its talents primarily on the "brain" of the robot. They have created a series of robots, both wheeled and legged, to test their theories. MIT's Leg Lab has

[*] Naval Research Laboratory, Stennis Space Center, MS 39592, patrick.mcdowell@nrlssc.navy.mil
[+] Chairman, Dept. of Computer Science, Louisiana State University, Baton Rouge, LA 70803

concentrated on the dynamics of walking and gait development in one, two and four legged robots, and they have built several robot systems for studying the different aspects of legged locomotion. The work at McGill University is somewhat similar in that they have concentrated their efforts on learning the dynamics of legged locomotion. However, they are using small self-contained four and six legged robots. MIT's AI Lab has done extensive work in the field of "Behavior Based" robotics [1]. Under the direction of Dr. Rodney Brooks, the lab is credited with creating the Subsumption Architecture, Behavior Language (BL) and a cast of mobile robots that operate successfully in unstructured dynamic environments. Among projects that the lab is currently involved with is a "humanoid" robot, called COG.

The Subsumption Architecture is a method for controlling mobile robots that differs markedly from centrally controlled, top down approach, classical AI methods. It can be described as a hierarchical, asynchronous, distributed system whose components produce behaviors. The interactions of the behavioral components in a subsumption architecture produce the system functionality. In general, sensor inputs are available throughout the system and thus sensor input and actuator actions are closely coupled. The behavioral components are set up in a hierarchy of layers, in which higher layers can inhibit or subsume lower layers, thus the name Subsumption Architecture. Figure 1 illustrates the differing approaches of classical symbolic AI and the Subsumption Architecture.

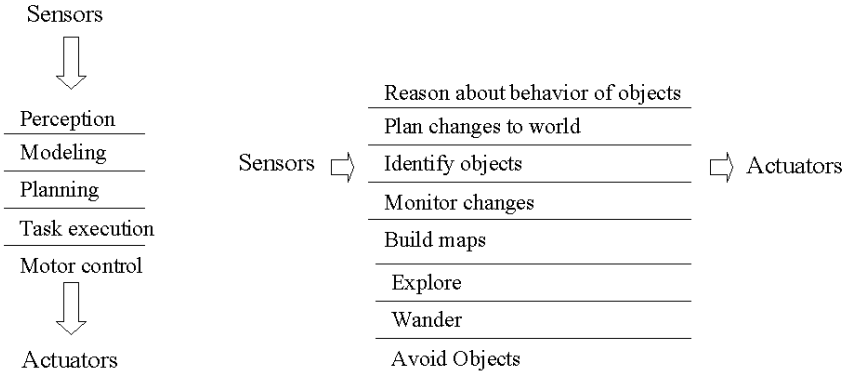


Figure 1. The differences between the Classical AI approach and that of Subsumption are illustrated in this figure. Classical AI is much more of a top down, centrally controlled approach while Subsumption takes a bottom up distributed approach.

Learning under the Subsumption Architecture is primarily a matter of conflict resolution among the various behaviors. That is, deciding which behaviors should be active at any particular instance in time. Historically the developers resolve these conflicts at compile time or by using a description of the desired behavior selection. In both cases, the resulting behavior hierarchy is static [2].

In 1990 P. Maes and R. Brooks [2] described an algorithm which allows a behavior based robot to learn based on positive and negative feedback, thus eliminating the need for the programmer to solve all the potential walking problems ahead of time. The end goal was to control overall robot functionality by selecting the appropriate behaviors from a "library" of behaviors, defining the connections to the sensors and actuators, defining the positive and negative feedback functions, and then letting the behaviors learn from experience when to become active.

As with all behavior based robots the algorithm is mostly distributed. Each behavior tries to find out, based on feedback, when the appropriate times are for it to be active. That is, in what situations is a particular behavior's operation relevant? Also, what are the conditions in which it's operation becomes reliable? That is, when is the behavior's operation consistently associated with positive feedback? Based on feedback, each behavior is able to determine its own relevance and reliability in the context of a given situation.

Using this algorithm the six legged robot "Genghis" successfully learned to walk using swing forward behaviors for each leg, avoiding negative feedback produced when it fell on its belly, and maximizing positive feedback generated when it moved forward. Genghis learned the tripod gait, that is, keeping three legs on the ground at any one time: the middle leg on one side and the front and back leg on the other side. This is not a trivial task, neither the negative nor the positive feedback is associated with any single behavior, but with the coordination of the behaviors in a distributed system.

The MIT Leg Lab has concentrated on the locomotion aspect of robotics. They have built a series of one leg, two leg, and four leg robots. One of their more impressive accomplishments is their Quadraped system [3], which can trot and even gallop at speeds up to 13mph. It is the size of a medium sized table and uses hydraulic actuators and air springs. The Leg Lab has been able to generalize multi-leg walking into a "virtual biped" system. That is, groups of legs are collected together and carry out the function of the right leg or left leg of a biped. For example, a trotting gait for a four-legged robot pairs the front left leg with the rear right and the front right with the rear left. For galloping, the front legs are a pair and the rear legs are the other pair.

McGill University stands out because of their research in small, walking robots. They have created robots that are completely stand alone and can effectively traverse varying terrains without the use of control tethers or outside power sources (SCOUT I, SCOUT II and RHex projects) [4]. McGill's use of radio control (RC) components and other cost effective hardware is inspiring, as is their belief that biological locomotion is effective and robust because there are simple principles embedded in complex biological systems.

2. The LSU Robo-Tiger Development

a. Hardware

Through a series of three prototypes, we have developed a basic four-legged walking robot. Our current robot, "Stubby", has two degrees of freedom per leg and non-actuated feet. Stubby has about the same dimensions and weight of an average-sized house cat, and borrowed much of its technical points from Mickey, the first powered prototype. Because Mickey's servos did not generate adequate torque, we had to simplify the leg structure by lightening and shortening. Hence development went from the anatomically correct, but under powered Mickey, to the working prototype Stubby. Figures 2 and 3 are photos of Stubby.

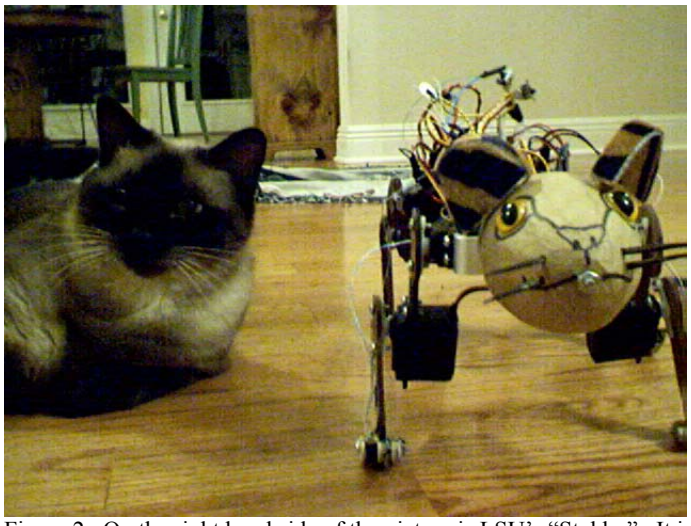


Figure 2. On the right hand side of the picture is LSU's "Stubby". It is the third of three small prototypes developed during the LSU Robotic Tiger feasibility study. Note that Stubby has the same general size as its biological counterpart, Mitzy.

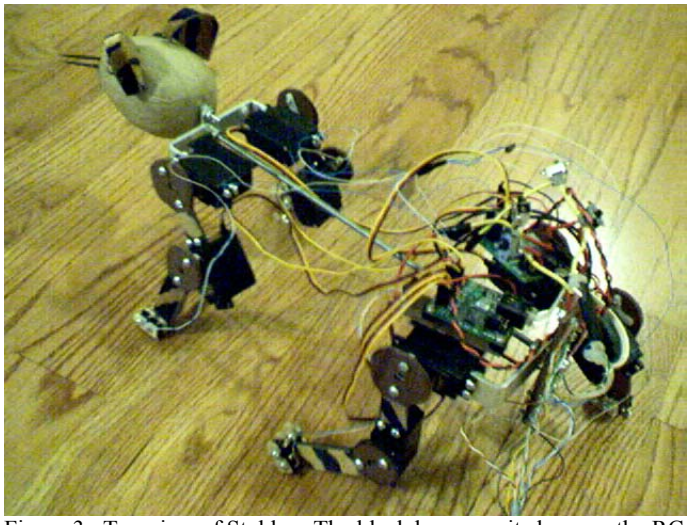


Figure 3. Top view of Stubby. The black boxes on its legs are the RC servo-motors. Also note the two controller boards above its rear legs.

From figures 2 and 3 the general attributes of Stubby can be seen. Like the robots built at McGill and by other researchers and hobbyists, Stubby is small and makes use of off the shelf components such as RC servos.

One of the major mechanical differences in our series of prototypes and those developed at MIT and McGill is the joint structure. Their joints usually are telescoping or pantograph, while ours are more similar to those of a cat, in that they are jointed and move in one plane. Stubby uses folding joints because they are more animal like. Pantograph joints work well but they are used mostly by insects and crabs, and not mammals. It is hoped that through the use of jointed legs, the Robo-Tiger will more closely mimic actual feline walking patterns.

Feedback for Stubby consists of touch sensors on each foot. Standard RC servos are used for locomotion. They are controlled by two Scott Edwards Electronics, Inc. Mini SSC II boards linked to the serial port of a laptop computer. Inputs from the foot sensors are collected by a National Instruments DAQCard-1200 card. A full description is available in [3].

b. Software

The software that controls Stubby is arranged in three layers, the gait level software, the joint level software and the software to hardware interface. The gait level software is responsible for coordinating the leg movements in a manner that produces a walk. The mid-level software keeps the joints moving at a constant rate and tracks all joint positions. The device level software transmits the coordinates generated by the mid-level software to the robot over a RS232 line and acquires sensor inputs from the micro switches located on the robot.

In figure 4 below, the flow of data and control is illustrated. The figure shows the different layers of the software. The gait level software is shaded green, the joint level is light gray, the software to hardware layer is dark gray, and the hardware is blue. The gait level software in the figure is that of the last of the three control methods, the Virtual Controller method.

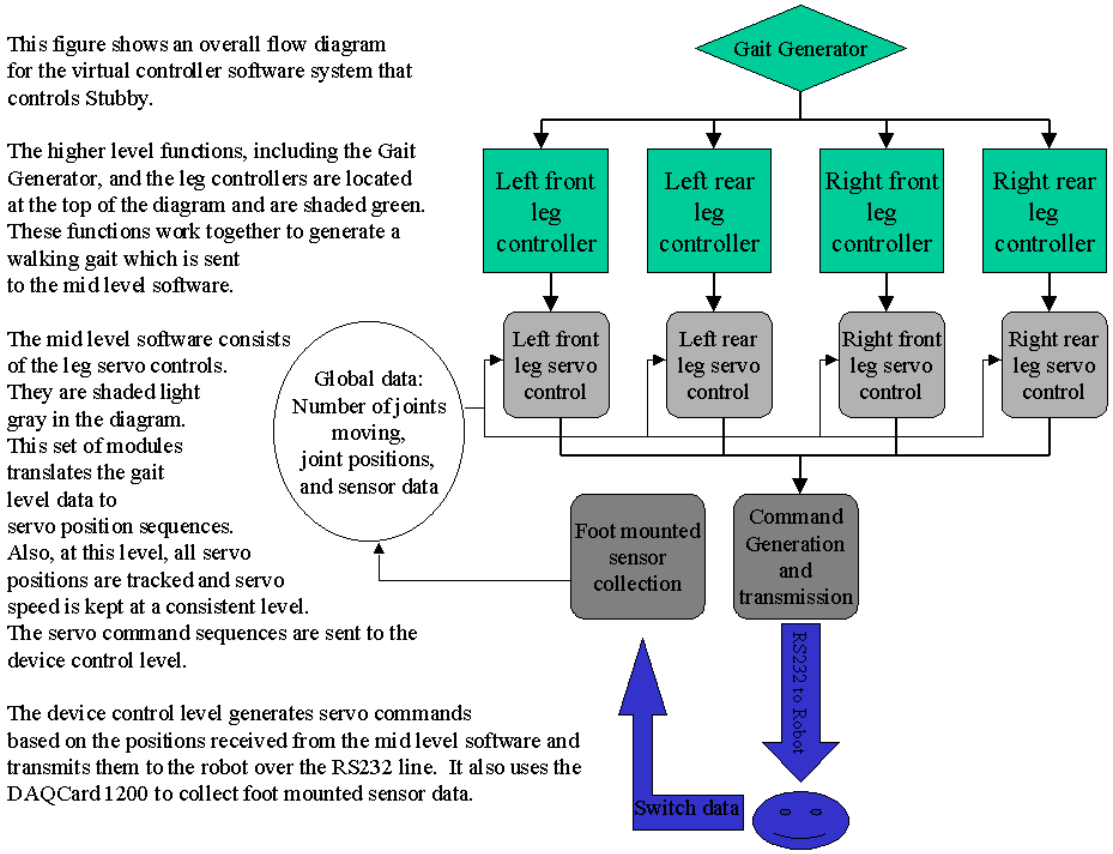


Figure 4. This figure shows the general flow of control and data for the hardware and software of the third prototype, Stubby.

i. Device level

The job of the device level software is to create and transmit command sequences to the robot over the RS232 line and collect and make available sensor data from the robot. The device level software takes input from the joint level software and builds and transmits servo commands to robot's control cards. Data from foot located micro switches is acquired and is made available to the joint layer and gait level software.

ii. Joint level

The job of the joint level software is to generate positions of the servos that will keep leg speed constant regardless of how many joints are moving. This is an important layer because it helps to smooth the overall motion of the robot. Without this level of control, each time a joint arrives at its destination the remaining moving joints would speed up. Without this speed regulation the robots limb motions would be constantly changing speed resulting in overall jerky motion. For example, if the right forearm of Stubby were to move from servo position 100 to 120, the command stream going to the control board would contain a sequence of positions as follows:

Right forearm position sequence: {100, 101, 102, ……119, 120}

If we wanted to move both the left and right forearms from position 100 to 120, the sequences would be as follows:

Right forearm position sequence: {100, 102, 104, ……118, 120}
Left forearm position sequence: {100, 102, 104, ……118, 120}

Notice the sequences are incremented by 2 instead of 1. Because of the serial nature of the Mini SSC II card, if the increment remained at 1, moving two joints would take twice as long as moving one joint. The

algorithm below describes the method used to control joints in order to keep their motions at a constant speed.

```
Input:    njoints:   int; Number of joints to be moved
          Jstart:    int array: Array of starting positions
          Jend:              int array: Array of joint stop positions

Output: Sequence of commands to RS232 port

Function MoveJoints(njoints, Jstart, Jend)
{
        For each joint to be moved, determine the direction to be moved (+, or -)

        While (there are still joints to be moved){
                For(each joint, MJOINT, that has not yet reached Jend) {
                        Move the joint by using the equation :
                        Joint inc = direction[MJOINT] * number of joints still moving;
                        Send sequence of commands to robot over RS232 line.
                }/* End for. */
        }/* End While. */
}/* End MoveJoints */
```

iii. Gait Level

Most of the development for the Robo-Tiger has concentrated on the gait level software. Three methods of gait level control have been implemented: movie frame method, genetic algorithm and the virtual controller method. These different methods will be discussed in more detail in the next section.

In order to test the basic functionality of the robot hardware and software, a GUI was developed that allows individual control of each leg servo. With this GUI the user can move slider controls on the control screen to place the servos of each leg at specific locations. When the servos are where the user wants them to be, the positions of all servos at that time can be recorded. An instance of the positions of the servos is called a frame, like a frame of a movie film. Each time a new frame is recorded it is added to the sequence of frames. These frames can then be stored, retrieved and played like a movie, thus the name 'Movie Frame' method.

This GUI proved to be an excellent servo-testing tool. The first attempts at making the robot walk were made simply by using the movie frame software to build a sequence of leg positions that would allow the robot to walk. Although it was very crude, the robot was able to stumble forward. In order to improve the results a genetic algorithm was used to optimize the servo positions in the walking frame sequence. Although improvements were obtained with the genetic algorithm, the results were still not satisfactory.

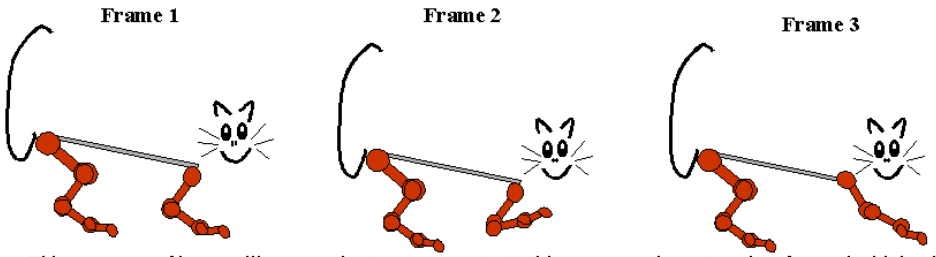
Ultimately, the frame-based method was abandoned in favor of a subsumption like method in which each leg was considered a separate entity that had its own control system, thus the name Virtual Controller method. A simple gait generator was used to coordinate the actions of the separate leg controllers in order to produce a walk.

3. Gait level control algorithm development

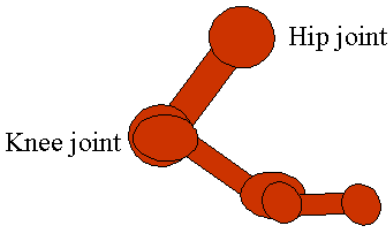
The purpose of the gait level software is to coordinate the timing and movements of the limbs of the robot in order to make it walk. Three basic strategies of control have been used during the development cycle. The Movie Frame method of control was the first method. It was a natural extension of the servo testing software developed to test the hardware. After it appeared the robot had the physical ability to walk, the output of the Movie Frame method was optimized using a Genetic Algorithm. Although the distance walked improved using this optimization, the clumsy nature of the walk was undesirable. A method that considered each leg as a separate entity was then developed. With this new method, each leg had its own software controller. Using the Virtual Controller method the distance and coordination of the robot's walk improved markedly. Below we describe each of the three methods in detail.

a. Movie Frame method

The first and most basic approach is the *Movie Frame* method. It operates on the premise that if we could take rapid snapshots of the robot and determine the positions of its joints as it walked, we could then play it back into the robots controllers whenever we wanted it walk. But here is where the paradox lies, because in order to take the snapshots, the robot first has to walk. By observing house cats and using the servo control program the frames were estimated and recorded. Then they were played back into the robot in a loop. The typical number of frames for the loop was 8 frames. The results were less than stellar. At its best the robot would take a few staggering steps forward. Figure 5 below illustrates the frame concept which is key to the Movie frame method.



This sequence of images illustrates the ***frame*** concept. In this sequence the cat reaches forward with its right front leg. Frame 1 shows the cat standing. Frame 2 shows the cat lifting its right front leg. Frame 3 shows the cat setting its foot on the ground. This sequence of three frames is a **reach sequence**. During walking, each leg of the cat reaches, as shown in the sequence, and once the leg is in position it pulls with a **pull sequence**.



For each frame in the sequence the hip and knee joints have coordinates. The table below shows a possible set of coordinates for the hip and knee joints for the **reach sequence**.

Frame #	Hip joint coordinate	Knee joint coordinate
1	10	20
2	10	40
3	20	10

Command sequences are formed by generating the joint positions between the frames. For example, the knee joint in frame 1 is at position 20, and in frame 2 it supposed to be at position 40. The command sequence generator will send positions 21 to 40 to the robot over the RS232 line.

Figure 5. This figure shows the frame concept. The 3-frame sequence shows a cat reaching forward with its right front leg. The table shows a possible set of coordinates for the joints in the frame sequence. From these coordinates, commands sequences will be generated and sent to the robot by the joint and device layers.

b. Optimization using the Genetic Algorithm

Once it appeared that the robot did have the mechanical ability to walk, a genetic algorithm was used to refine the robots best walking sequence developed using the Movie frame method. Since the robot is a mechanical entity, which will break or wear with use, the population size and number of generations was kept low by genetic algorithm standards. A typical population size would be 5, and during a run, 50 generations would be the upper limit. The fitness function used was distance walked by the robot in 16 steps, measured by a yardstick.

Although this method was crude, after about 200 generations the distance walked in 16 steps was nearly doubled. The overall distance that the robot walked greatly improved, but it still "walked ugly". The most noticeable failing was that when a leg movement was being made it would often be incomplete, or reverse direction, making the robot consistently drag its feet, stumble, and jerk. These incomplete leg movements were caused by the mutations of the leg positions in the sequence of frames. The mutation factor in the genetic algorithm was doing its job, but it would require several more generations (which could easily wear the robot out) and a better ranking function in order to get the desired results.

c. Virtual Controller method

The Virtual Controller method relies on the premise that given a relatively stable robot and acceptable leg reach and leg pull movements, the walking problem boils down to one of timing. That is, for the robot to walk, each leg has to reach and pull at the appropriate time. This idea is fundamentally different from the frame approach in that each leg has its own state, rather than having a single state for the whole robot.

To implement this idea, the control software consists of four virtual controllers, one per leg, which are coordinated by a gait pattern generator. Each virtual controller has a "reach" and "pull" subsection, shown in figure 6. A leg can never be doing both reach and pull functions at once. All leg controllers run asynchronously communicating to the joint layer simultaneously. The reach and pull movements were developed using a separate utility program that provides direct control over all servos and allows recording and editing of movements, i.e. the Movie Frame program. Figure 6 shows a block diagram of the virtual controller software.

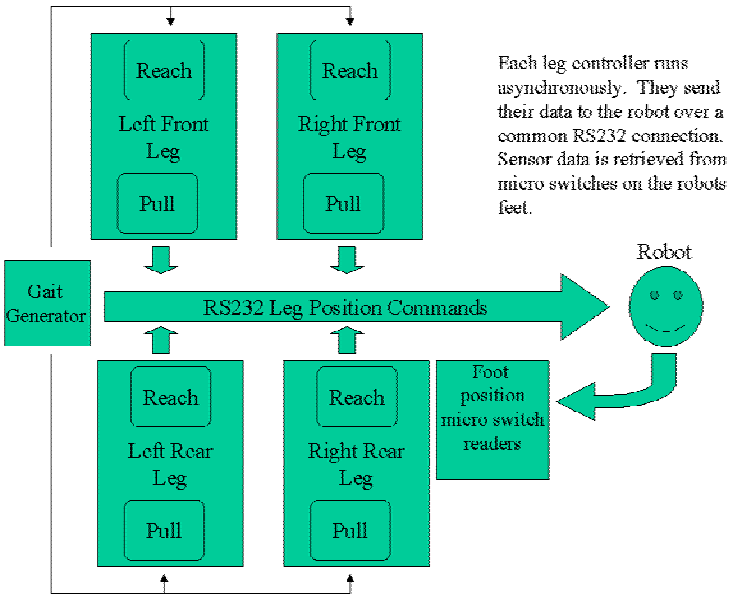


Figure 6. This figure shows a block diagram of the virtual controller system used to control Stubby. Since it is an asynchronous system, each of the processes (represented by the square boxes) is running independent of the others. The gait generator signals when each of the leg processes start. The leg processes signal the gait generator when they have completed their process. The sensor reading process makes the current micro switch data available to any processes in which it is required.

The virtual controller method dramatically improved both the smoothness and efficiency of Stubby's walking. Several factors were observed to influence the effectiveness of the gaits. The most influential are the times between steps, times between the actions that make up the steps, and the surface that the robot is walking on. Additionally, several different gaits were implemented on Stubby based on MIT's "virtual biped" approach. Stubby was able to gallop by pairing the two front legs and the two rear legs, and was able to trot by pairing front and rear legs on opposite sides. The entire system was implemented in LabView on a lap-top computer.

4. Continuing Development

Stubby's stage of development could be compared to that of an animal that has just been born in that it has a workable control system that has not yet adapted to its body or trained in its environment. The big difference is that animals have excellent sensors and brains that are pre-wired to immediately start the processes of adaptation and learning.

Using subsumption like extensions to the current virtual controller system, we plan to add "reflexes" that help the robot avoid toe stubbing and wasted motion caused by pushing with a leg before it is in contact with the ground. We also plan to develop a method to optimize the walking gait if the robot detects that its current gait is inadequate. If the robot were to remain on a constant surface, toe stubbing and other walking problems could be attributed to the walking gait having either timing errors or the leg motions themselves being wrong, or both. An optimal walking gait can be found by altering the timing and/or leg movements in a way that minimizes the number of times that the reflexive actions are used.

For example, if the robot made the transition from a known smooth surface, to a known rough surface, the type and timing of the reflexive actions that would occur to correct the robot should have a somewhat distinct pattern. Using this supposition, we plan to create a system that matches the pattern generated by the reflexive actions firing, and a gait that will minimize those actions. This is akin to walking blindfolded through a parking lot into a field of thick tall grass. A person would naturally start picking up their feet a little higher. The first few steps may surprise them, but after that it would become routine and they would not be surprised again, until the walking surface changed again.

So, when the reflexive actions increase above some threshold, the pattern will be used to key an associative memory that holds the closest gait. If the reflexive actions decrease below the threshold, the new gait is adequate, and the surface is known; if not, the gait optimization process will begin again using the new gait as a starting point. Figure 7 below shows a block diagram describing the different components of the proposed system and their hierarchies. Notice that the inner part of the diagram, from the leg motion control blocks inward is functionally identical to the Virtual Controller system shown in figure 6. The reflexive actions and gait modification is layered onto the existing system.

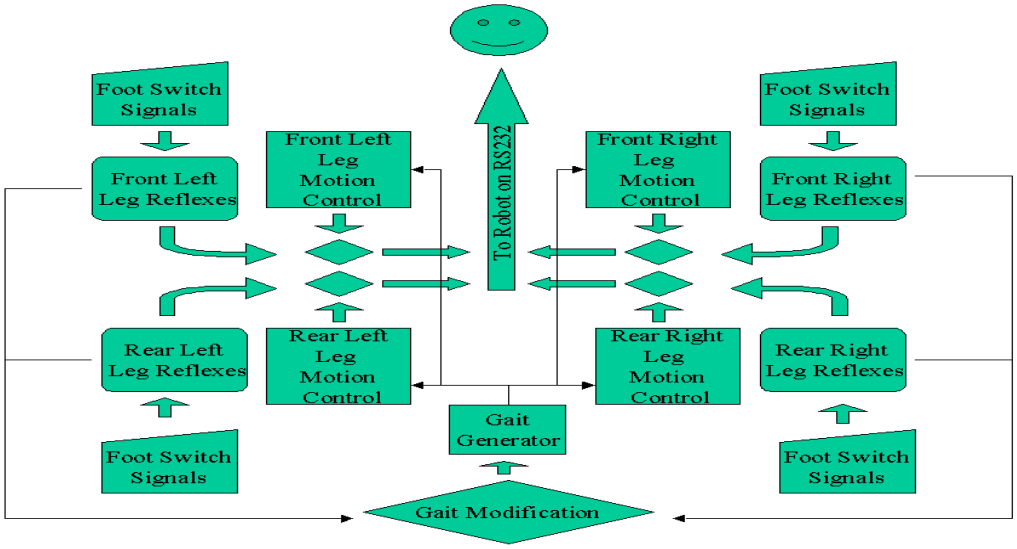


Figure 7. This diagram illustrates the proposed interaction of the reflexive actions and the leg motion controls. Reflexive actions will take priority over the leg motion controls. Initially, gait adjustment will be implemented at the Gait Generator Level, but will later be at the leg motion control level. This diagram does not show the interaction of the associative memories discussed in the text.

The foot-mounted micro-switches that are currently used for detecting a foot down situation are not ideal for adaptation purposes because they cannot differentiate between when the robot is stubbing its toe or when its foot is down. Toe stubbing caused by feet reaching forward but not being raised high enough is one of the major impediments to smoother and more efficient walking. A new leg has been designed and prototyped that uses two switches, one for detection of a foot down situation and the other for detection of a toe stub.

5. Summary

The LSU Robo-Tiger project has provided a platform from which to research, explore and integrate the various technologies that will be used in creating truly useful machines that can operate autonomously in unstructured environments. Experience with earlier prototypes has shown that propulsion, balance and control of a four-legged robot is not trivial. The first approach to making the Robo-Tiger walk was through an ah-hoc 'Movie Frame' method. This initial technique proved that the robot could "walk ugly", and its performance was improved using a genetic algorithm. By switching to the subsumption-like Virtual Controller method the robot's performance and walking manner improved tremendously. Using the Virtual Controller method as a base to build on, the near term focus will be on improving performance by adding reflexive behaviors, gait optimization, and situational memories based on reflexive action driven associative memories.

Acknowledgements

The mention of commercial products or the use of company names does not in any way imply endorsement by the U.S. Navy. Approved for public release; distribution is unlimited. NRL contribution number NRL/PP/7440-00-0008.

References

[1] Brooks, Rodney A. "A Robust Layered Control System for a Mobile Robot", MIT AI Lab Memo 864, September 1985.
[2] Maes, P. and R. A. Brooks, "Learning to Coordinate Behaviors", AAAI, Boston, MA, August 1990, pp. 796--802.
[3] "Quadraped (1984 –1987)", http//www.mit.edu/projects/leglab/robots/robot.html
[4] "Ambulatory Robotics Lab", http//www.cmi.mcgill.ca/~arlweb/
[5] "LSU Robotic Tiger: Results of Small Prototype Reasearch and Development Phase", May 1, 2000; Patrick McDowell