NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Integrated System for Single Leg Walking

Reid Simmons
Eric Krotkov
Gerry Roston

CMU-RI-TR-90-15 2

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

July 1990

© 1990 Carnegie Mellon University

This research was sponsored by NASA under contract NAGW-1175. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA or the United States Government.

Contents

UNIVERSITY LIBRARIES
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA 15213

CAR
PITTSE

List of Figures

Abstract

The CMU Planetary Rover project is developing a six-legged walking robot capable of autonomously navigating, exploring, and acquiring samples in rugged, unknown environments. This report describes an integrated software system capable of navigating a single leg of the robot over rugged terrain. The leg, based on an early design of the Ambler Planetary Rover, is suspended below a carriage that slides along rails. To walk, the system creates an elevation map of the terrain from laser scanner images, plans an appropriate foothold based on terrain and geometric constraints, weaves the leg through the terrain to position it above the foothold, contacts the terrain with the foot, and applies force enough to advance the carriage along the rails. Walking both forward and backward, the system has traversed hundreds of meters of rugged terrain including obstacles too tall to step over, trenches too deep to step in, closely spaced obstacles, and sand hills. The implemented system consists of a number of task-specific processes (two for planning, two for perception, one for real-time control) and a central control process that directs the flow of communication between processes.

1 Introduction

The CMU Planetary Rover project is constructing the Ambler, an autonomous walking robot designed for planetary exploration [1]. The configuration is a six-legged vehicle with orthogonal legs and an overlapping gait [3, 17]. The Ambler is designed to meet particular mission objectives, including the ability to negotiate one meter boulders and to traverse one meter wide ditches, while on a 30° slope [25]. Power consumption is minimized by decoupling the motion of the horizontal axes from that of the vertical axis. In addition, the Ambler is designed to provide a stable platform for sensory instruments and scientific equipment.

To gain experience with legged robots, we built a single leg of the Ambler, and used it to experiment with both real-time control of the mechanism and walking over rough terrain. The rationale was that ideas would be easier to develop using just a single leg, and that many of the concepts would transfer to the full six-legged walker.

The leg, based on an early design of the Ambler, has three degrees of freedom, decoupled into vertical motion (a prismatic joint) and horizontal motion (two rotary joints). The leg can extend vertically about 4.5 meters, and can extend horizontally about 2.5 meters. To provide support, it is attached to a carriage that is free to roll along rails. Sensors include a potentiometer to measure the position and velocity of the carriage along the rails, incremental and absolute encoders to measure leg positions, and two inclinometers to measure the rotation of the carriage. A six-axis force/torque sensor is attached to the bottom of the leg to measure the forces experienced by the mechanism as it moves. In addition, a scanning laser rangefinder is attached to the carriage to provide perception of the terrain.

Below the rails is an 11×6 meter obstacle course. The obstacle course consists of a bed of sand, sculpted to form trenches, hills, and gently rolling terrain, and various obstacles, such as styrofoam rocks, pylons, and boxes.

We have developed an integrated system that can autonomously navigate the single leg through obstacle courses of rough terrain. This report describes the integrated system and its various components: perception, planning, and real-time control. The purpose of the report is to document the design decisions and rationale for the components and the methods used to integrate them. The report also describes the walking experiments performed, highlighting our progress in single-leg walking.

The walking system consists of five task-specific processes (called *modules*) that communicate with one another using the Task Control Architecture (TCA) [21, 26]. TCA routes messages through a centralized control module, which manages and synchronizes the sending and receiving of messages over the Ethernet. TCA is basically a high-level robot operating system that provides utilities for building and coordinating mobile robot systems. The utilities are meant to bridge the gap between task-level planners and real-time control systems. In particular, TCA supports 1) distributed processing, 2) resource management, 3) hierarchical task decomposition, 4) temporal synchronization of tasks, 5) execution monitoring, and 6) exception handling.

Of the five modules, one handles real-time control, two handle perception, and two do planning.

The *Controller* module, which runs under a real-time, multi-tasking operating system, handles all robot motions and responds to queries from other modules regarding leg position, carriage position and orientation, and force sensor readings. The *Image Sensing Manager (ISM)* acquires laser scanner images and determines the transformation from scanner to world coordinates. The *Local Terrain Map (LTM) Manager* processes scanner images to construct elevation maps of the terrain. The *Gait Planner* plans where to place the foot and how far to move the carriage in order to advance with minimal risk to the mechanism. The *Leg Recovery Planner (LRP)* determines a trajectory to the planned footfall location that is energy and time efficient and that avoids terrain collisions.

To walk the leg down the obstacle course, the user specifies a goal location for the carriage to reach along the rails. The walking system is totally autonomous from that point on. It uses terrain elevation maps and constraints imposed by the geometry of the leg to plan a series of moves that attempts to maximize advance while minimizing the possibility of collision with terrain features.

To take a step, the Gait Planner requests a footfall evaluation map from the LTM Manager. The LTM Manager requests an image from the ISM, which acquires one from the scanning laser rangefinder. The LTM Manager uses the *Locus Method* to construct an elevation map that indicates the height of the terrain at each grid point [11]. The elevation map is then processed to estimate the flatness of the terrain. The Gait Planner determines where to place the foot using the elevation and footfall-evaluation maps, and geometric constraints of the leg. The planner computes a weighted sum of the terrain and geometric constraints in an effort to find a spot that will avoid obstacles while allowing for reasonable progress.

The LRP uses this planned footfall, the current leg position, and the elevation map to plan a collision-free trajectory for the leg. Using the novel *Envelope Trajectory Finding Algorithm*, the planner is able to find paths through 3D space that are nearly optimal in terms of time and energy, while searching only a 2D grid.

The planned trajectory, consisting of a sequence of way-points, is sent to the Controller, which interpolates the way-points to get smooth motion in joint space. The Controller executes the trajectory, monitoring the force sensor in the leg to detect when contact with the terrain occurs. The Controller exerts enough force to compress the terrain, then relaxes to a force sufficient to provide traction. The horizontal (shoulder and elbow) joints are then actuated to drive the carriage forward. Finally, tension built up in the leg as a result of the body move is relieved, so that the leg does not slip when it is next lifted. This sense-plan-execute cycle is repeated until the commanded position on the rails is met.

To date, the single-leg integrated walking system has traversed several hundred meters of rough terrain. The system is able to walk both forwards and backwards: the LTM Manager caches elevation maps, so in walking backwards the planners can use the maps generated during the forward walking phase.

The effort has taught us much about legged robots in particular and mobile robot systems in general. Our experience with the single leg led to some significant changes in the configuration of the Ambler, especially with regard to compliance. The single leg was too flexible to permit the type of accurate control needed to negotiate very rough terrain. In addition, we found that the laser scanner did not have sufficient resolution and accuracy for our purposes. While this did not prevent us from successfully walking, it did limit the roughness of the terrains that we could traverse. For the six-legged Ambler we have procured another scanner that overcomes some of these problems.

The software components proved to be effective for single-leg walking, and seem to be transferable, with minor modifications, to the six-legged system. In particular, the perception subsystem and the Task Control Architecture can be ported almost as is. One surprise in the endeavor was the fine balance between geometric and terrain constraints for gait planning. Much of our effort in getting the leg to negotiate the obstacle courses was in fine-tuning the weighting function that combined constraints. Gait and footfall planning for the Ambler are areas of on-going research and will undoubtably consume much of our effort in getting the Ambler to walk on rough terrain [29, 31].

The level of detail in this report is appropriate for system designers and implementors. Users and programmers may find the description to be too abstract: users should consult the on-line manual pages, and programmers should see the on-line documentation in the code and in the `doc` directories reserved for documentation. We refer readers interested in the objectives and organization of the overall project to references [14, 15].

The next section describes the setup of the single-leg testbed, including the mechanism, sensors, obstacle course, and computing environment. Section 3 presents the Task Control Architecture, addressing both its principles and use in integrating robot systems. The next three sections describe the real-time control system, the perception system, and the planning software, respectively. Section 7 discusses single-leg walking experiments conducted during 1989, and presents their results. The report concludes with some general remarks about single-leg walking and system integration.

2 Single-Leg Testbed

The single-leg testbed is comprised of: (1) the mechanism itself (the leg and carriage), (2) sensors, (3) the control room, including the controller hardware and computing facilities, and (4) the obstacle course. The leg and carriage are described in Section 2.1 and the various sensors used are detailed in Section 2.2. The control room facilities are described in Section 2.3, and the obstacle course is described below.

To provide for a variety of "Mars-like" terrains, we constructed an experimental test pit measuring approximately 11 $\times$ 6 meters (Figure 1). It is made from steel I-beams clamped together at the corners. The test pit is filled with more than 40 tons of sand. Terrain features are introduced by resculpting the surface to form hills and trenches, and by placing objects on the sand. We have used styrofoam boulders, traffic cones, and large boxes to test the ability of the system to navigate over and around obstacles.

2.1 Mechanism

A single leg of the Ambler (based on an early design [1]) was built to test several concepts, including rough-terrain walking and the design of the leg itself, before committing to the fabrication of a six-legged vehicle.

The horizontal range of the leg is approximately 2.5 meters, and the vertical extension ranges from approximately 3 to 4.5 meters. This configuration enables the leg to meet its design objectives of crossing one meter wide ditches and stepping over one meter high obstacles on a 30° slope.

The shoulder and elbow axes contain brushless DC servo motors coupled to the joint axes by an 80:1 harmonic drive speed reducer and a 3:1 bevel gear. The vertical axis consists of a brushless DC servo motor connected to a 12:1 speed reducer and a lead screw. A drawback to this design is the slow speed of the vertical axis, approximately 3.6cm/sec maximum.

Walking requires the foot to rotate 360 degrees with respect to the leg. The foot is coupled to the leg by a system of bearings which allow this rotation. The force/torque sensor (see Section 2.2) is attached to the leg above the foot bearings to prevent the problem of cable wrap-up.

An area of continuing study is the design of the ankle. The existing leg uses a rigid ankle design (Figure 2). Since the normal force exerted on an object is independent of its surface area, this design allows the foot to dig in on slopes, perhaps getting better traction than if it conformed to the surface.

The leg is supported by a carriage mechanism that is mounted on a pair of rails. The central column of the leg is attached to the carriage, allowing the leg to rotate freely. The carriage is equipped with rollers, which provide for one degree of translational freedom. We desired a support system that would allow the leg to walk over the ground in a manner sufficiently similar to the Ambler, so that the knowledge gained from the single-leg test program could be used for the six-legged machine. Although the Ambler has a full six degrees of freedom, this approximation is sufficient to allow testing and algorithm production.

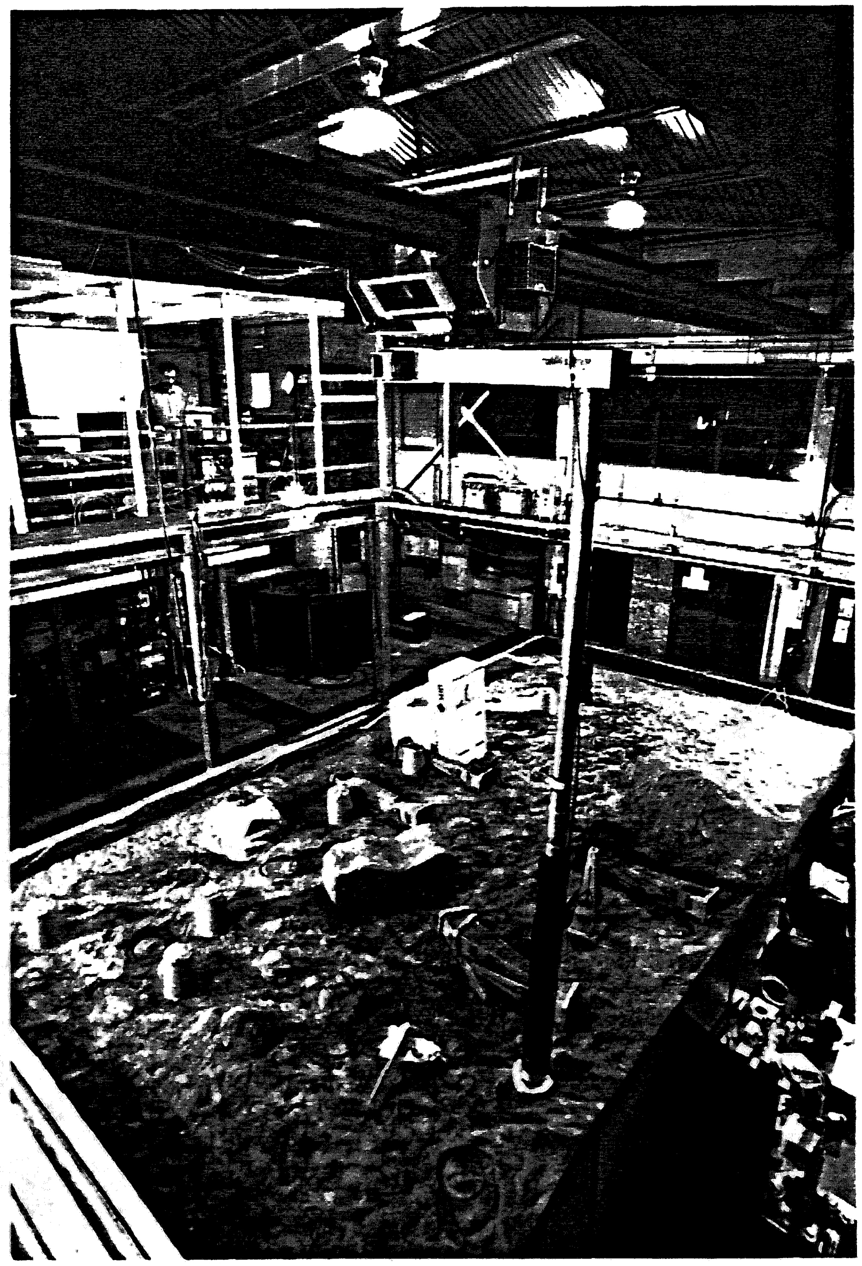

Figure 1: Leg and Obstacle Course

CAR
TTSE

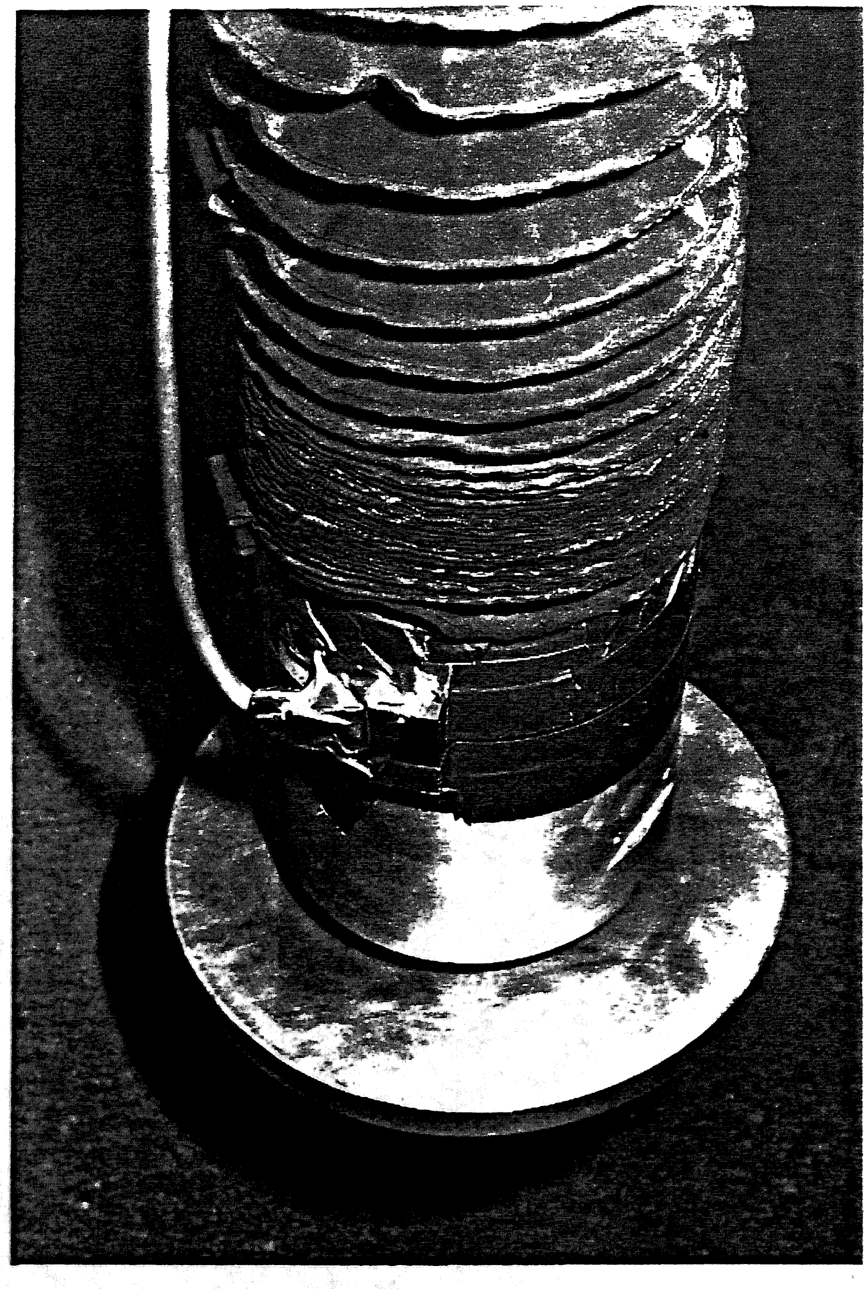

Figure 2: Ankle and Foot

The rails, although constructed of heavy duty steel I-beams, have a considerable amount of deflection due to their length, the applied loads, and the method by which they are attached to the building. This deflection is good in that it provides compliance for the system. The deflection changes as a function of the leg position, however, which makes accurate leg placement and planning difficult to achieve.

2.2 Sensors

Both imaging and non-imaging sensors are used at the testbed. The imaging sensor is a scanning laser rangefinder. Its physical characteristics are described below, and its usage in the integrated system is presented in Section 5. Non-imaging sensors include linear position measuring devices, force measuring devices, angle encoders, inclinometers, and limit switches. The non-imaging sensors are managed by the real-time controller (Section 4).

Imaging Sensor

We use a scanning laser rangefinder because it operates more rapidly and reliably than passive vision techniques (see Section 5). To date, we have been using a scanner manufactured by Erim. We have recently acquired, and will soon be using, a more accurate scanner manufactured by Perceptron. In this section, we summarize the operation of the laser scanners.

Both scanners are optical-wavelength radar systems that transmit laser signals generated by a laser diode operating in the near-infrared region (Erim 820 nm, Perceptron 810 nm). The devices modulate the amplitude of the output of the laser diode by varying its drive current. They scan the signal across the field of view using a nodding mirror and a rotating polygon mirror. The nodding mirror changes (tilts) the elevation and the polygon mirror changes (pans) the azimuth of the emitted signal.

The devices digitize two images: a *range image*, with pixel values proportional to phase difference, and a *reflectance image*, with pixel values proportional to reflected energy. To produce a range measurement, the infrared light is reflected off the target surface, gathered by the receiver optics, and focused onto a detector (a silicon avalanche photodiode). The scanners filter the optical signal to pass only the transmitted optical frequency, and filter the electronic detector signal to pass only the amplitude modulating frequency $f_{AM} = c/\lambda_{AM}$. An electronic phase detector then measures the phase difference between the transmitted and received signals, which is proportional to the transit time, and therefore the range. Since relative phase differences can be determined only modulo 2π, the range of a point is determined only to within a range ambiguity interval r_{ambig} where

$$r_{ambig} = \frac{c}{2f_{AM}} = \frac{\lambda_{AM}}{2}.$$

Although the two scanners follow a common principle of operation, they exhibit significant differences in implementation and performance.

The Erim acquires data in 64×256 pixel images at a rate of 2 frames per second [11, 33]. The scanner digitizes to 8 bits with a range ambiguity interval of approximately 20 meters. This provides a range resolution of approximately 7.62cm. The measurements cover 80° in the horizontal direction (azimuth) and 30° in the vertical direction (elevation).

The Perceptron acquires data in 256×256 pixel images, also at a rate of 2 frames per second. The scanner digitizes the range channel to 12 bits with an ambiguity interval of approximately 40 meters, which provides a range resolution of approximately 0.98cm. The scanner digitizes the reflectance channel to 12 bits. The measurements cover 60° in the horizontal direction (azimuth) and 60° in the vertical direction (elevation).

Non-Imaging Sensors

The sensors described in this section include linear position measuring devices, force measuring devices, angle encoders, inclinometers, and limit switches.

We have tried using two different sensors to measure the position of the carriage from a given reference point. This linear "body position" information is needed by our planning algorithms and by the perception subsystem to merge elevation maps. The two sensors that have been used are an optical ranging sensor similar to those used by land surveyors, and a position/velocity transducer connected to a take-up reel.

The optical ranging sensor is manufactured by Geodimeter. It was chosen because of its accuracy, ability to output data to an RS-232 line, and its relatively low cost. The device, however, was not successfully integrated into the system for two reasons. First, the serial line transmission of data was highly unreliable. Second, the rate at which the data could be queried was slow, about 1-2 Hz. In particular, it is too slow for closed loop positioning control of the carriage.

The position/velocity transducer is a potentiometer attached to a take-up reel. A cable is pulled from the housing, and the device effectively measures the length of the cable. As the tension on the cable relaxes, excess cable is drawn back into the housing. The device also incorporates a self-generating tachometer, which permits better control laws to be implemented (see Section 4). Analog data is transmitted from the device directly to an A/D card, therefore the reliability of the device is ensured. Furthermore, since the device is a potentiometer, the output can be read as quickly as an A/D card can sample it.

Attached to the bottom of each leg is a six-axis force/torque sensor that is used to measure the forces experienced by the leg as the vehicle moves. The sensor, which is manufactured by JR3 Inc., has the following load limits: 400 pounds force X and Y axes, 800 pounds force Z axis, and 3000 inch-pounds about all three axes. In addition, four to six times the rated load may be applied to a single axis without damaging the sensor. To achieve the necessary high speed data acquisition for closed loop control of the vertical actuator, the force/torque information is transmitted to the controller as an analog signal.

The leg uses two distinct types of angle encoders: incremental encoders and absolute encoders. The incremental encoders are coupled directly to the motor shafts, and are used by the motion control hardware for servo-control of the motors. The signals from the encoders are fed directly to the motion control cards; the controller software can access the signal values by issuing commands to the control cards. The incremental encoders have a resolution of 1000 counts per revolution of the motor, resulting in an accuracy of approximately 5 seconds of arc. Although the accuracy of the system is high, the inertia of the moving leg prevents the repeatability from being equally precise.

The absolute encoders, manufactured by Durapot, output an analog signal which varies from minimum to maximum output value with each complete revolution of the leg. The resolution of the sensor is about 5 minutes (0.087°). The absolute encoders are used to provide the position of the leg at system power-up. This is necessary because the incremental encoders attached to the motor shafts cannot be used to determine the absolute leg position.

Two inclinometers are being used to measure the rotation of the carriage about the world X and Y axes. The inclinometers are attached to the carriage at right angles to one another. The carriage rotation is needed to help transform the images acquired from the scanning laser rangefinder into the world reference frame.

Limit switches are installed on the leg to prevent over rotation of the shoulder and elbow joints as well as over travel of the vertical axis. The limit switches are Hall effect switches, which do not require physical contact to operate them. They are not very rugged, and several have ceased functioning for a variety of reasons.

2.3 Control Room and Computing Environment

The control room houses the requisite computer workstations and real-time controller hardware needed to operate the integrated system (Figure 3). Included are three Sun workstations connected by Ethernet, a VME cage containing a real-time control system and its associated hardware, specialized hardware for controlling the laser scanner, and specialized hardware for the leg control. The signal cables from the leg are routed to the control room by means of a tether. Although each cable in the tether is single source grounded, eliminating noise from the system has proven quite difficult. This is primarily due to the fact that a local radio station uses the steam pipes as a ground plane, thereby causing high-frequency noise to appear on all signal lines and ground.

The real-time controller hardware consists of a Motorola 68020 CPU single board computer (Motorola MV133XT), an Ethernet controller (Motorola MVME 330-1), an Analog to Digital (A/D) converter card (Datel 611), two Intel 80186 based motion control cards (Creonics MCC-VME), and an interface card to connect the Creonics cards to the motor amplifiers. Device drivers were developed that enable interrupts generated by the Datel and Creonics cards to be properly handled by the control software.

We have also designed a safety circuit card. This card is connected to numerous signal lines, such as the limit switches, the motor amplifier fault lines, manual "kill" switches, etc. Should a hardware error condition occur, the safety circuit is activated, shutting down the motion control cards and reporting an error to the control software.

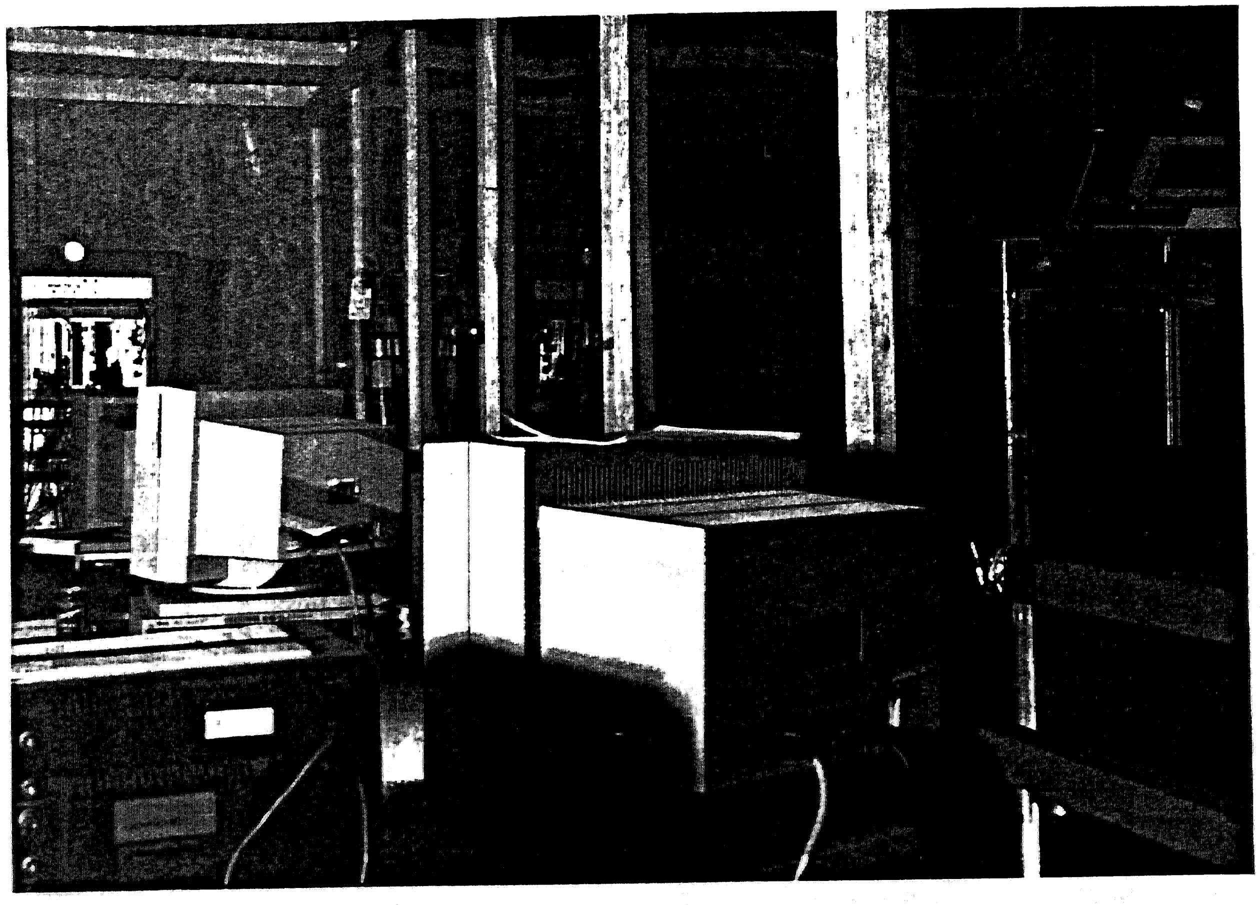

Figure 3: Control Room

The Erim scanning laser rangefinder is interfaced to one of the Sun workstations through a custom-built buffer board and an Ironics IV-1602 single board computer. A simple program running on the Ironics board reads images from the scanner and stores them into its onboard dual-ported memory. To the Sun, the Ironics board appears to be standard memory, so the images can be accessed via direct memory reads. A Matrox VIP-1024 display board in the Sun workstation is used to display the images on a Sony monitor.

Code development for the project is done in C under the UNIX operating system. This provides a level of commonality which enables different modules to communicate easily and allows common subroutine libraries to be used by all subsystems. In addition, system wide parameters, such as the dimensions of the leg, are accessed from a single common directory to ensure project wide uniformity.

While the code for the real-time controller is written in C, it uses the vxWorks operating system. Code development is done on a host Sun workstation, with the code being down-loaded to the real-time CPUs over the Ethernet. This provides a familiar development environment with all of the developer's accustomed tools, a high-speed kernel for real-time operation, and the ability to communicate with the non real-time systems over the Ethernet.

CAR
ITSE

3 Task Control Architecture

The Task Control Architecture (TCA) is used to integrate the software components of the walking system: real-time control, perception, and planning. TCA provides a number of useful facilities for building and operating complex robot systems. In particular, it provides mechanisms to support message passing between distributed processes, hierarchical planning, plan execution, monitoring, and exception handling.

A system built using TCA consists of a number of task-specific processes, called *modules*, and a general *central control* module that directs the flow of communication between modules. The single-leg system consists of six modules plus the central control, which we typically run on three separate machines (Figure 4). The individual modules will be described in detail in subsequent sections. This section describes how TCA is used to integrate the pieces.

3.1 Rationale

TCA was designed with two major goals in mind: to provide facilities for integrating software and hardware components into a complete robotic system, and to provide mechanisms for coordinating and executing robot tasks [21, 26].

To facilitate integration, TCA was designed to support coarse-grained message passing between arbitrary numbers of modules. The idea was that a message passing protocol would encourage the development of clean, well-defined interfaces between modules. In practice, this methodology has been quite successful: by the time the components are ready to be combined, the interfaces between modules are well worked out and compatible.

Another integration concern is to provide tools to aid in debugging a distributed system. To this end, the central control module can log all message traffic between modules. This log is valuable in localizing the cause of bugs, particularly those due to timing interactions between the distributed, asynchronous modules. Another debugging feature of TCA is that modules can connect and disconnect from the central control at will. Thus, if one module crashes it can be debugged and restarted while the others are still running. This has proven to be a great time saver, since initializing the controller and perception modules is fairly time-consuming.

Besides providing facilities for integrating components, TCA is designed to coordinate and execute tasks. In effect, TCA is a high-level robot operating system, providing utilities for building task-specific systems. Capabilities supported include building complex behaviors hierarchically out of more primitive ones, resource management, concurrent planning, perception and execution, execution monitoring and error recovery, and handling of multiple, conflicting goals. While not all of these capabilities are being used in the single-leg system, they are being investigated using an indoor mobile robot [20, 21].

A prominent aspect of TCA is its use of centralized control. Although decentralized control has recently been advocated for mobile robots [5], we believe that centralized control has many advantages for supporting the above capabilities. First, it can more easily control multiple tasks

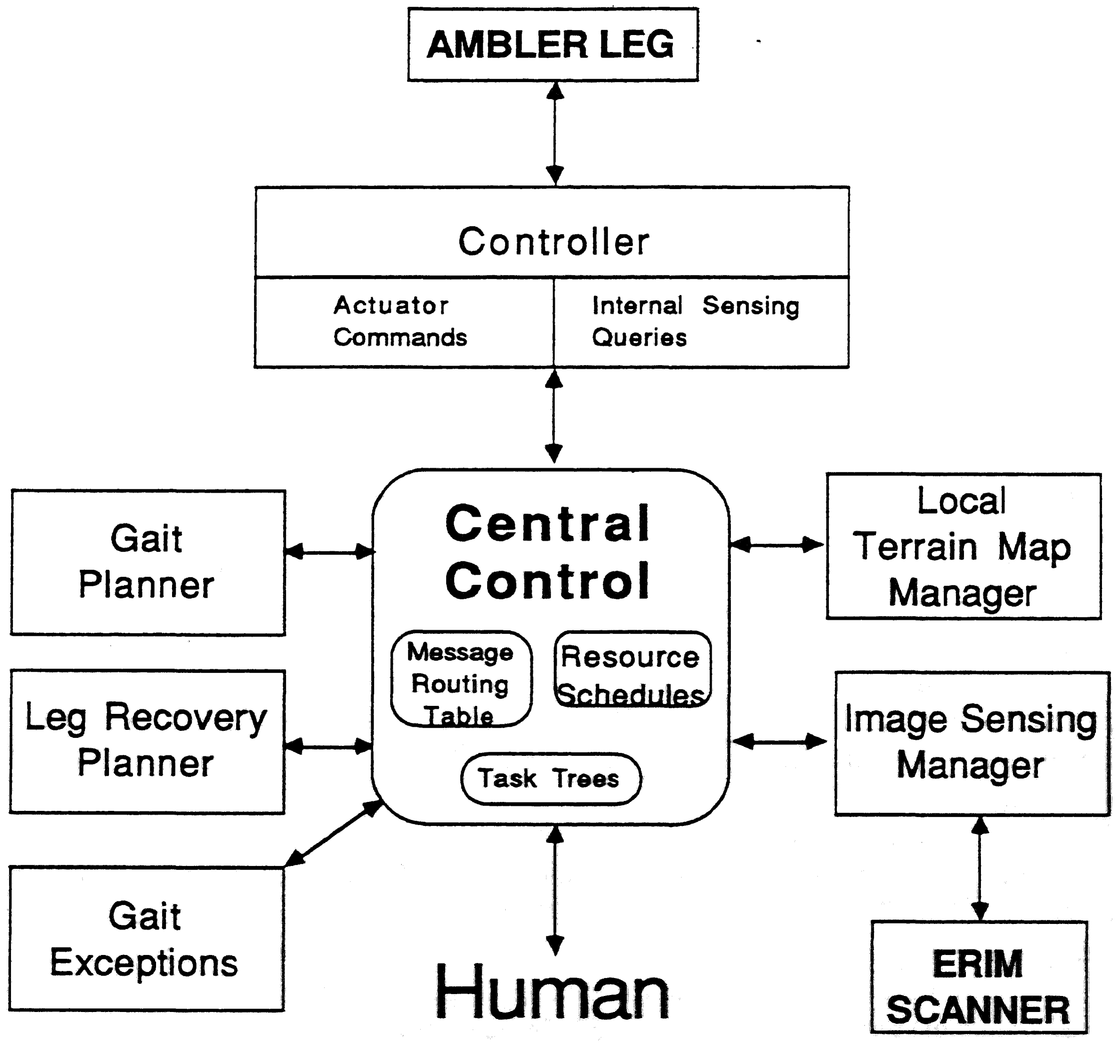

Figure 4: Modules Used in the Single-Leg Integrated System

by synchronizing them, allocating resources, and determining which tasks have priority. Second, centralized control makes the system more understandable and hence easier to modify. Since there is a single point through which all communication flows, one can easily monitor and analyze the communication. Finally, we have not found centralized control to be a system bottleneck. TCA can process a message in approximately 80msec, and for a typical walking cycle the central control module is idle about 97% of the time.

3.2 Description

The Task Control Architecture consists of a general-purpose central control module and utilities for enabling user-defined modules to send messages to other modules and to the central control. Messages sent to other modules are typically used for data flow, such as for querying the state of the environment, or for describing what tasks need to be achieved. Messages sent to the central module are used for specifying control flow, such as detailing the hierarchical structure of plans, temporal constraints between actions, and exceptions.

Modules can connect with the central control module in any order, and at any time. In fact, if a module crashes it can be restarted and reconnected to TCA without bringing down the rest of the system. When modules connect, they indicate which messages they can handle by registering with TCA the message name, a handler procedure, and the data format of the message. The data format is a string that describes a C or Lisp data structure. TCA contains routines for parsing fairly arbitrary data structures, passing them as byte streams over the Ethernet, and re-assembling them to be passed to a message handler. Thus, all aspects of low-level communication are transparent to a user module. TCA also contains facilities for displaying arbitrary data structures, used for logging message traffic.

All messages, even those meant for user-defined modules, are routed through the central control. A module sends a message name and message data using TCA-defined utility functions, passing as arguments standard C or Lisp data structures [9]. The central control determines which module can handle the message, and when it is appropriate to forward the message. This gives developers the flexibility of substituting one module for another (e.g., replacing the real-time controller with a graphical simulator) without changing the code in other modules.

In TCA, planning and executing a task occurs by having modules send a series of messages to one another (via the central module). For the single-leg walking system, after all modules have connected to the central control (Figure 4) and registered their messages and handlers, a message is sent to the Gait Planner instructing it to begin planning. To plan and execute a complete step involves sending about 25 messages.

TCA supports several message classes, each with a different semantics and different effects. *Query* messages are used to obtain information about the external or internal environment. Query messages are blocking pending the receipt of a reply. The single-leg system uses query messages to obtain elevation and footfall evaluation maps from the Local Terrain Map Manager, to obtain images from the Image Sensing Manager, and to obtain leg and body positions from the Controller.

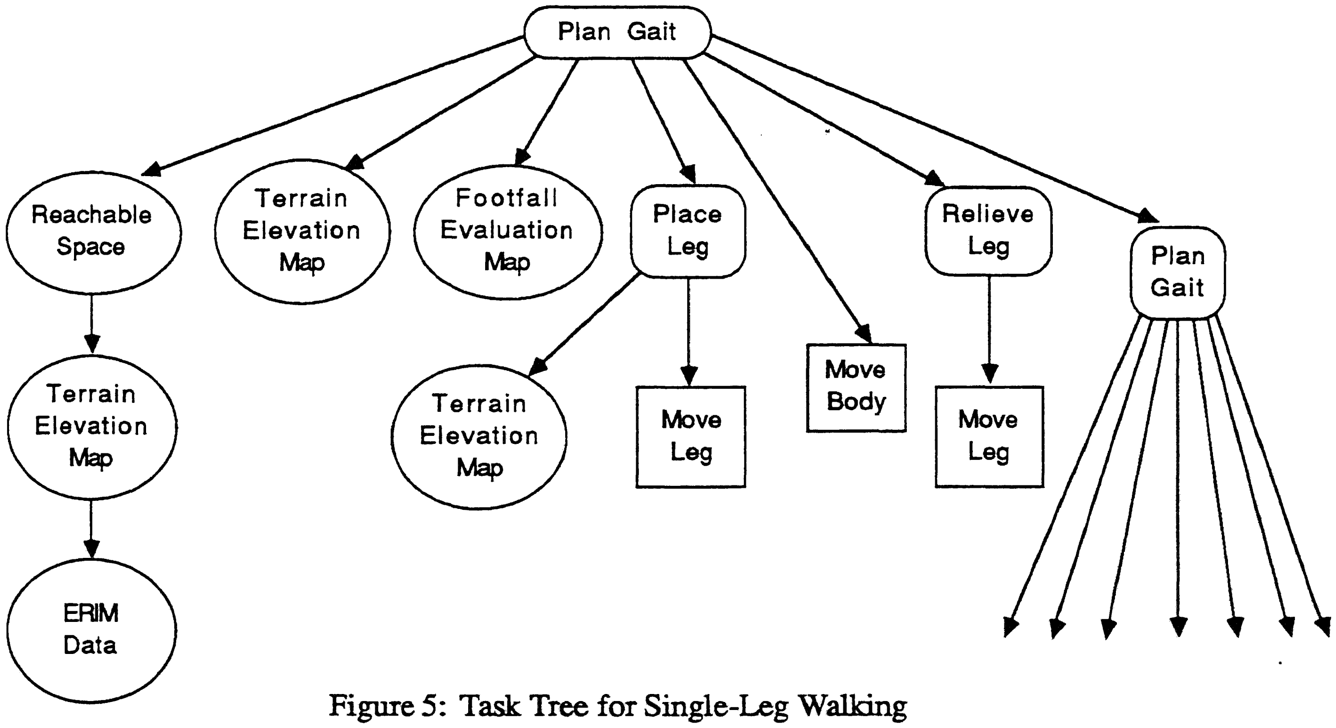

Figure 5: Task Tree for Single-Leg Walking

The *goal* message class provides a mechanism for hierarchical planning. Each time a goal message is issued, TCA creates a reference to the message and adds it as a child of the handler that issued the message. These message references form a hierarchical *task tree* that TCA uses to schedule planning and execution of tasks (Figure 5). The task trees can also be examined and manipulated by other modules, a utility that is useful in handling exceptions.

Command messages are requests for some action to be performed. Like goal messages, they are added to the task tree, typically as the leaf nodes of the tree.

In addition to maintaining parent/children relationships between messages, TCA provides mechanisms for temporally constraining the relationships between nodes in a task tree. Essentially, the task trees plus the temporal constraints form TCA's representation of plans. For example, one can specify that a *legMove* command must be executed before the next *bodyMove* command is started, or that a *planGait* goal cannot be planned until the previous *bodyMove* command has finished (see Figure 5). TCA maintains separate constraints for the planning and achievement of tasks. Thus, one could specify that the robot should go to a sample site and then acquire a sample, but that it should plan how (and if) it can acquire the sample before planning how to navigate to the site.

Unlike query messages, goal and command messages are non-blocking. That is, a goal or command message has not necessarily been handled by the time control returns to the module issuing the message. Although this asynchronous control is sometimes difficult for implementors to get used to, it makes the overall system much more reactive since TCA has control over when to schedule tasks and when to preempt them. The non-blocking nature of goal and command messages also makes it easy to do planning in advance of execution: the planning modules merely send messages that create task trees, and TCA ensures that the tasks will be executed at the appropriate times [27].

TCA also provides mechanisms for exception handling. Goal and command messages, when they detect plan-time and execution-time failures, respectively, issue exception messages. The central control suspends the current task and routes the exception to the appropriate user-defined handler. There, the exception handler can decide how to recover from the error, for instance, by killing part of the current task tree, or by adding new nodes to the tree to patch the plan. Currently, the single-leg system makes only rudimentary use of the TCA exception-handling facilities.

3.3 Discussion

TCA has proven very useful in building and debugging the single-leg system. First, the message passing conventions forced us to design and enforce clean interfaces between modules. Second, the ability to start and stop modules independently meant that the system did not need to be completely re-initialized in order to debug a single module. Third, the ability to log all message traffic has proven invaluable in detecting timing problems and interface errors.

In addition, TCA has proven quite useful in controlling and coordinating a complex system. The exception handling mechanism enables us to build reliability into the system incrementally, by first specifying the normal behavior of the system, and then adding new modules to handle the exceptional behaviors. The task tree and temporal constraint mechanisms provide for sophisticated planning and execution strategies. For example, we have recently extended the walking system to concurrently execute one step while planning the next step. TCA enables us to switch between sequential and concurrent mode by changing only a few of the temporal constraints. This has enabled us to achieve a significant increase in performance with minimal modifications to the existing system [27].

4 Real-Time Control

This section describes the software that actually controls the leg and the non-imaging sensors. The Controller module performs several tasks: (1) it communicates with people via a terminal and the TCA central control module, (2) it moves the leg and carriage as commanded and reports their positions as requested, and (3) it handles asynchronous interrupts that are generated by the motion control cards. In addition, when a motion request is made, the Controller responds (to the terminal or TCA) whether the command succeeds or fails.

The control software runs on the real-time system under the vxWorks operating system. It communicates with the rest of the walking system over the Ethernet via TCA. Although to the rest of the system the Controller appears to be a single module, it is best thought of as four separate, but tightly coupled tasks running concurrently. In addition to the four main tasks, there are several others that handle asynchronous interrupts generated by the motion control cards, shut down the leg in case of an abnormal abort, etc.

4.1 Rationale

The Controller module was designed with three major objectives: speed, reliability, and extensibility. The Controller obviously has to perform at levels capable of closed loop control of the leg. This involved using both specialized hardware (e.g., the Creonics motion control cards) and a real-time operating system (vxWorks).

To maintain reliability, the Controller was designed to monitor sensors and the motion control cards for possible failures (mainly using interrupts and special device drivers). The Controller was designed to gracefully shut down the leg and inform TCA when errors are detected. Such errors include passing limit switches, amplifier faults, servo errors, excessive force readings, and "kill" messages from users. In addition, the control software is written to permit recovery from hardware errors without restarting the Controller module.

Finally, the Controller was designed to be extensible. The user and TCA interfaces can easily be modified. New sensors, and interrupt handlers for them, are readily incorporated into the Controller. All this has made the single-leg testbed a useful experimental tool for conducting our research.

4.2 Description

The four main tasks that make up the Controller module are the Host Interface Task, the User Interface Task, the Position Sensing Manager Task, and the Walk Motion Manager Task.

The Host Interface Task (HIF) handles the TCA interface to the rest of the system. This task registers all of the necessary TCA messages, and awaits incoming requests for the Controller. When a request is received, the HIF forwards the message to the appropriate task, after adding some information for local usage. The messages handled by the Controller through the HIF are the following:

bodyMove This command causes the carriage to be propelled along the rails until the requested position is reached. The requested position is given in world (global Cartesian) coordinates, as determined by the linear position transducer (Section 2.2).

legMove This command moves the leg through a variable number of way-points. In addition, there is an option to monitor the force/torque sensor during the last way-point, and to stop if a specified force is reached. This *transition mode* is used to contact the terrain.

setTransForce This command sets the value, in pounds, of the force to servo to during transition moves.

bodyPosition This query returns the position and orientation of the carriage in world coordinates. The carriage location is determined using the linear position transducer and the two inclinometers.

jointLegPosition This query returns the current position of the leg in joint coordinates. It is determined by reading the leg encoders through the Creonics motion control cards.

cartLegPosition This query returns the current position of the leg in Cartesian coordinates, local to the frame of the carriage. It is determined by computing the inverse kinematics of the joint positions.

globalLegPosition This query returns the current position of the leg in world coordinates. It is determined by adding the body position to the local Cartesian leg position.

readForce This query returns the current force and moments as indicated by the force/torque sensor.

The User Interface Task (UIF) is analogous to the HIF in that it waits for requests from a user, then issues the command to the other tasks. The UIF interacts with the user through a text-based terminal interface. The UIF accepts the same requests as the HIF, as well as numerous other commands and queries for monitoring the health of the real-time system, changing dynamic tuning parameters, etc. The user can also issue emergency stop commands through the UIF if a commanded action is not behaving as expected.

The Position Sensing Manager Task (PSM) is responsible for reporting sensor data (joint angles, body position, and leg forces). The joint angle can be reported in three ways: as joint angles in radians, as local Cartesian coordinates in meters, or as global Cartesian coordinates in meters. Body position is reported as a Cartesian vector giving the displacement from the world origin in meters and the rotations about the world axes in radians. Since the carriage is confined to move along the rails, the only meaningful values are the displacement along the Y axis and the rotations about the X and Y axes. Forces are returned as a three-vector of forces along the foot Cartesian axes and a three-vector of the moments about these axes.

The Walk Motion Manager Task (WMM) is responsible for handling all actuator motions. This task handles leg move and body move commands. The leg is moved through a series of user-supplied way-points, which are given in joint space. The WMM calculates the amount of time required for the slowest joint to move between successive way-points and then scales the speeds of the other joints such that all joints will arrive at each way-point simultaneously. To smooth the motion, the way-points are linked with constant velocity segments connected together by constant acceleration segments [7]. The user has the option of specifying whether the last path segment is made in transition mode. In transition mode, the force/torque sensor is monitored and the motion is stopped if a specified force is achieved before the way-point is reached. If the way-point is reached first, an exception is raised to TCA.

To move the body, the force on the leg is first increased to 800 pounds, to compress the underlying terrain. The force is then relieved to 500 pounds, which is usually sufficient to provide good traction. The shoulder and elbow joints are then actuated, thereby propelling the carriage. Although superficially similar, the body move procedure is actually significantly different from the leg move procedure. First, since the carriage is constrained to move only in a single direction, a Cartesian control scheme is needed to properly move it. Second, to minimize the possibility of the leg's breaking free, the motions of the two joints need to be tightly coordinated.

The desired Cartesian velocity of the carriage is calculated using a clipped, linear function of the error between the present carriage position (as read from the transducer) and the commanded goal position. This velocity is converted into joint velocities using an inverse Jacobian function. The joints are commanded to move at a given velocity, not to a given position. Compliance in the mechanisms causes overshoot of the joints from the positions they would be expected to have given a rigid kinematic description of the leg. This overshoot takes the form of stored strain energy and must be relieved before the leg is lifted from the ground, otherwise the foot will drag across the terrain, possibly hitting an obstacle.

The body move control loop is operated at a frequency of about 60 Hz. This differs sufficiently from the natural frequency of the mechanism so that resonance does not occur. During the control loop, the Controller also monitors the forces exerted on the foot, and an emergency stop is issued if the force rapidly drops off, indicating that the foot may have broken free.

4.3 Discussion

The major difficulty in controlling the single leg is the compliance in the mechanism. This proved to be a problem with the body move procedures, and to a lesser extent with specifying leg moves.

For the leg moves, the problem stemmed from our initial assumption that the leg was fairly rigid, and from the incorporation of that assumption into our kinematic and inverse kinematic procedures. This caused problems with the planning modules, which planned moves in Cartesian coordinates (the coordinate space of the terrain elevation maps provided by the perception subsystem) and then had to convert to joint space in order to issue leg move commands. The actual compliance of the leg meant that it often collided with terrain features.

We partially solved this problem by measuring the deflections in the leg and updating the kinematic routines using a simple deflection model fit to the data. This improved the accuracy of the leg moves, as measured in Cartesian space, from about 20cm down to about 5cm.

More troublesome was the body move procedure. Our initial implementation commanded the position of the horizontal joints to follow a linear trajectory. Due to the compliance of the mechanism and the friction between the carriage and rails, this procedure proved to be very inaccurate. We often witnessed errors of more than 40cm over a (commanded) one meter body move.

The remedy was to use the velocity control procedure described above. With this method, the error was reduced to less than 5cm. The new body move algorithm was subjected to extensive testing to gain confidence in its performance. Over 1000 moves were performed with the leg starting at various X, Y locations relative to the carriage. The resultant data revealed the accuracy of body moves for each point on a 25cm grid. This data not only confirmed the general accuracy of the body move procedure, but also provided a "map" for the Gait Planner to indicate the goodness of footfalls with respect to body advance (see Section 6.2.1).

Fortunately, these compliance-related issues are only an artifact of the single-leg design, and do not carry over to the full six-legged vehicle. Those legs, based on a different design, are extremely rigid [3]. Our experience to date with that mechanism indicates that we can do leg and body moves to within a centimeter of commanded positions. Nevertheless, even with these problems, we have found the single-leg testbed to have been a valuable tool for testing the component subsystems and their integration.

5 Perception

This section describes the perception subsystem that we developed for the Planetary Rover project. We present the design rationale, describe the major software modules, and discuss future improvements and extensions. Readers interested in a higher level account of the perception subsystem should consult references [1] and [12].

5.1 Rationale

The purpose of the perception subsystem is to build and maintain representations of the environment. Without these representations, the Ambler would be "blind," and thus could safely negotiate only the simplest terrain. At a minimum, the perception subsystem must be capable of performing two tasks: it must acquire information about the environment by sensing, and it must interpret the sensor signals in terms that are meaningful for different behaviors (e.g., locomotion, navigation, manipulation).

In designing and implementing the perception subsystem, we have separated these tasks into two major modules: one that senses the environment with a scanning laser rangefinder, and another that constructs elevation maps from the rangefinder data. A third module is concerned with interactively displaying terrain data.

We use a laser scanner, which measures both reflectance and range, because it directly recovers the environment's three-dimensional structure. Therefore, it supplies 3D data more rapidly and reliably than passive vision techniques such as binocular stereo and motion.

We use an elevation map as the primary terrain representation because 1) it provides a representation that is appropriate for a wide variety of tasks, 2) it is simple to manipulate, 3) it can be accessed in a simple way (by the boundary of the region of interest), and 4) it can be hierarchical and so accommodate high-level information as well as high resolution elevation data. An alternative terrain representation might consist of 3D patches that either are approximations of the measured surface or are built directly upon the set of data points. Another possibility is to represent only a higher level description of the terrain, such as a segmentation of the surface. Future work may explore these alternative representations.

5.2 Description

This section describes the major software modules of the perception subsystem: the Imaging Sensor Manager (ISM), the Local Terrain Map (LTM) Manager, and the Perception Display Manager (PDM).

5.2.1 Imaging Sensor Manager

The ISM operates the system's imaging sensors. The ISM's functions include initialization, status determination, data acquisition, calibration, and aiming. The ISM has been implemented and tested for the Erim and Perceptron scanning laser rangefinders (see Section 2.2). These sensors may be real (i.e., they acquire data in real-time from the physical sensor) or virtual (i.e., they access previously acquired images from storage). We have found virtual sensors and images to be very useful for developing and testing code.

The ISM initializes a real sensor by downloading interface code from a host to special-purpose boards, and initializes a virtual sensor by opening sequences of image files stored on disk. The ISM tests the status of a sensor by checking that it has been properly initialized.

The calibration procedure identifies the relationship between two reference frames: the body frame $\mathcal{B}$, and the scanner frame $\mathcal{S}$ (Figure 6). Commands to the Controller module are issued with reference to $\mathcal{B}$, while the perception subsystem makes measurements of the terrain with reference to $\mathcal{S}$. Thus, knowledge of the relationship between these two frames is essential for connecting perception to action (walking).

The calibration procedure collects data by moving a reflective calibration target attached to the leg to a number of different *poses* (positions and orientations). At each pose, it queries the Controller for the position of the target referred to $\mathcal{B}$, and acquires a range/reflectance image pair. After collecting this data, the calibration procedure uses image processing techniques to identify the target position in the images, and refers it to $\mathcal{S}$. Then it computes the homogeneous transformation ${}^{\mathcal{S}}T_{\mathcal{B}}$ from the scanner frame to the body frame (Figure 7). For details, see reference [16].

The ISM acquires data by reading from the memory-mapped sensor, and tags the acquired data with the pose of the sensor at the time of acquisition. Specifically, it computes the homogeneous transformation referring the locus frame $\mathcal{L}$ (which is closely related to $\mathcal{S}$) to the global frame $\mathcal{G}$:

$$ {}^{\mathcal{L}}T_{\mathcal{G}} = {}^{\mathcal{L}}T_{\mathcal{S}}\,{}^{\mathcal{S}}T_{\mathcal{B}}\,{}^{\mathcal{B}}T_{\mathcal{G}} \quad , $$

where

- ${}^{\mathcal{L}}T_{\mathcal{S}}$ is fixed,
- ${}^{\mathcal{S}}T_{\mathcal{B}}$ is determined by the calibration procedure, and
- ${}^{\mathcal{B}}T_{\mathcal{G}}$, which represents the pose of the body frame at the time of image acquisition, is maintained by the Controller.

5.2.2 Local Terrain Map Manager

The LTM Manager constructs and maintains a local terrain map (LTM). Modules external to the perception subsystem can use the terrain map for locomotion guidance, short-range navigation, and sampling operations. An LTM describes the environment in the immediate vicinity of the

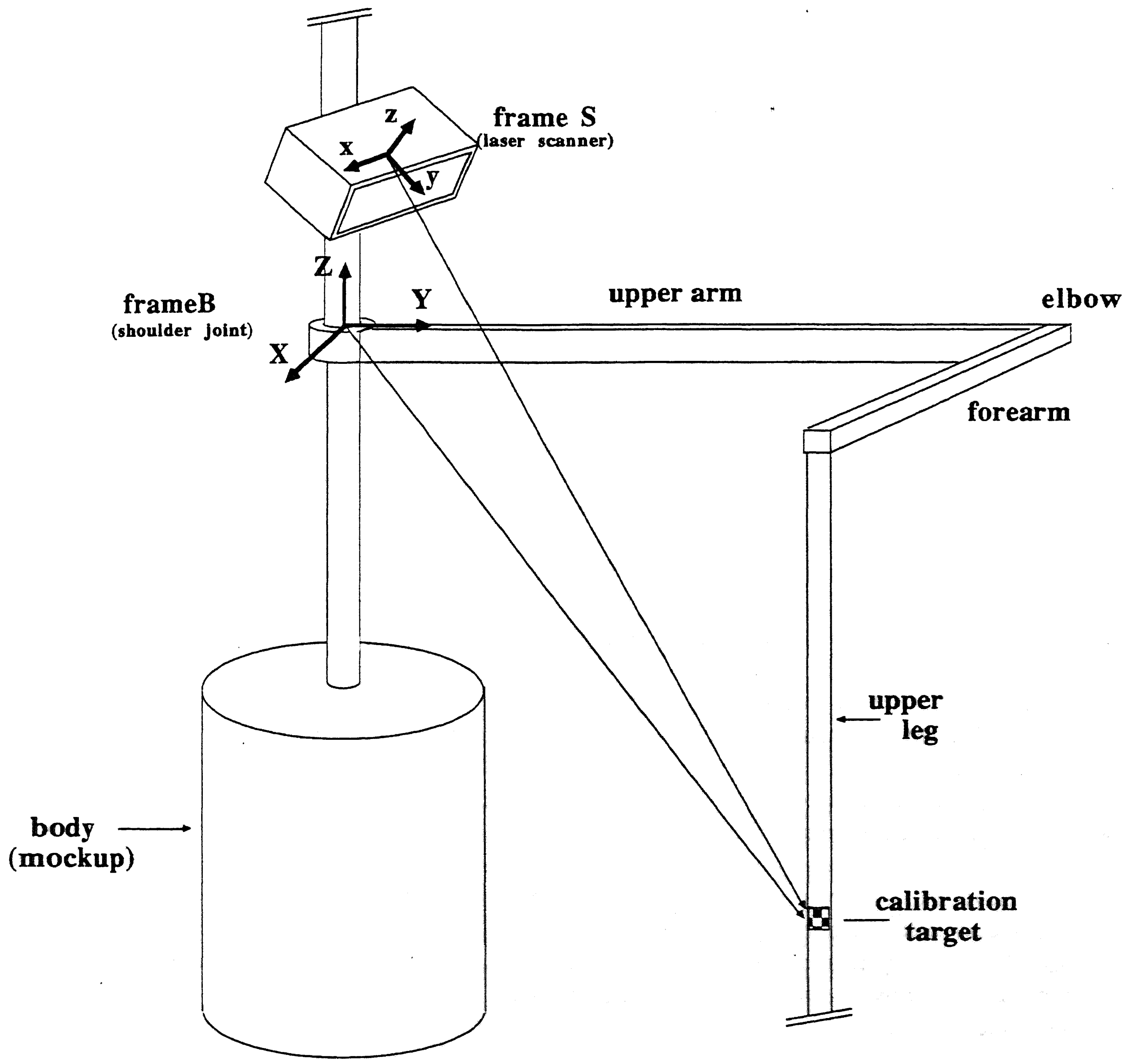

Figure 6: Reference Frames Used by the Calibration Procedure

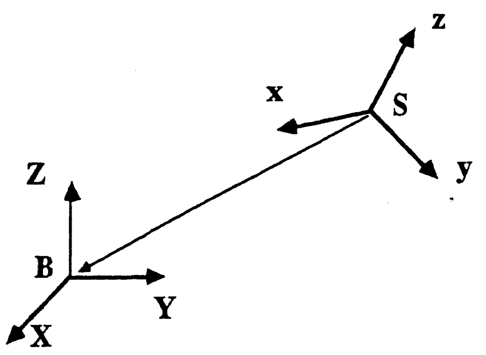

Figure 7: Transformation to be Estimated by Calibration Procedure

vehicle, and may extend up to tens of meters on a side. Strictly speaking, however, an LTM is not a single map, but a collection of maps whose descriptions of the environment may include geometric characteristics as well as material properties of the terrain.

The LTM Manager has been implemented and tested for the Erim and Perceptron scanners. We have organized the software into three major submodules: one that builds an LTM, one that merges LTMs, and one that focuses attention on parts of the LTM that are close to the vehicle.

The LTM Builder constructs an LTM from a single frame of sensor data by transforming the raw sensor observations into a structured description of the terrain in the local vicinity of the vehicle. The implementation uses the *Locus Method* to transform the raw range images into an elevation map [11, 18]. In addition, the LTM Builder computes the uncertainty of the estimated elevations [18], analyzes elevation map patches for their potential goodness as footfall locations [6], and estimates the mean slope over elevation map patches.

The LTM Merger maintains the LTM to reflect the information contained in a sequence of maps constructed by the LTM Builder. The merging operation is necessary because maps created from a single frame of data do not, in general, contain enough information about the local terrain to accomplish even simple tasks. For example, consider the task of planning the trajectory of a recovering leg. Because the scanner looks forward, the map constructed from a single forward-looking range image cannot possibly see obstacles either below or behind the vehicle. This may pose threats to the recovering leg, which must follow a trajectory that avoids collision with obstacles. Thus, a merging operation is necessary to create an LTM that provides a wider coverage of the terrain than is possible with a single frame of data.

Conceptually, the implementation of the LTM Merger accepts the following input,

1. the LTM L_k^0 constructed from range images $I_0, I_1, \ldots, I_k$,

2. the LTM L_{k+1}^{k+1} constructed from range image I_{k+1},

and generates as output the LTM L_{k+1}^0 by replacing overlapping elevation measurements with the maximum likelihood estimate of the elevation [18].

The LTM Scroller matches the size of the LTM to the size of available local memory. This is necessary because as the vehicle advances, the merged LTM becomes larger and larger, but the perception subsystem runs on machines that have a fixed memory size. To accommodate this fixed memory size, the LTM Scroller maintains a window around the LTM, outside of which LTM values are paged out, or clipped. Currently, the size of the window is 20×20 meters.

The rest of the single-leg system interfaces with the LTM Manager through TCA queries that request various maps of specified regions (elevation, elevation and uncertainty, footfall evaluation, and slope maps). A region of interest is specified by giving a map resolution and a set of points that defines a 2D polygon. The LTM Manager returns a rectangular grid that is big enough to contain the polygon. Grid cells outside the polygon (or outside the scanner field of view) are labeled *unknown*. Cells occluded by other objects are labeled as such, along with the maximum known elevation of the cell, given the available information. All other grid cells are given real values corresponding to the value (elevation, uncertainty, etc.) of a representative point within the cell (our current implementation uses the mid-point of the cell).

The requested maps are computed on demand, but are cached so that future queries that request the same (or overlapping) regions do not have to recalculate the values. While this is relatively efficient, we are looking at pre-computing some maps concurrently with the planning and execution of walking commands.

5.2.3 Perception Display Manager

The PDM displays the maps constructed by the LTM Manager. The user specifies a particular rectangular region to display and a type of map (or maps). The PDM queries the LTM Manager for the data and displays them as elevation maps (Figure 8).

Displaying maps is an important function, because graphics display provides information vital to validating and debugging both perception and planning software. There are a number of reasons for creating an independent module to manage map display. First, the user interface can be more flexible, because map display is not coupled temporally with map construction. Thus, the user can display maps when it is convenient or desirable, not only when they are being constructed. This can also increase the performance of the overall system, since the LTM Manager can be concurrently processing maps while the PDM is displaying information needed for debugging. Since graphics display takes a significant amount of time, the savings are considerable. Another advantage is that it is easier to extend the graphics interface if it is well modularized. For example, in addition to merely displaying map data, the PDM has been extended to indicate requested 3D positions on the map. This has proven useful in debugging the correlations between planned and sensed footfall locations.

5.3 Discussion

This section discusses improvements and extensions to the existing perception subsystem that are needed or desired for use on the six-legged Ambler. By "improvements" we mean increasing the

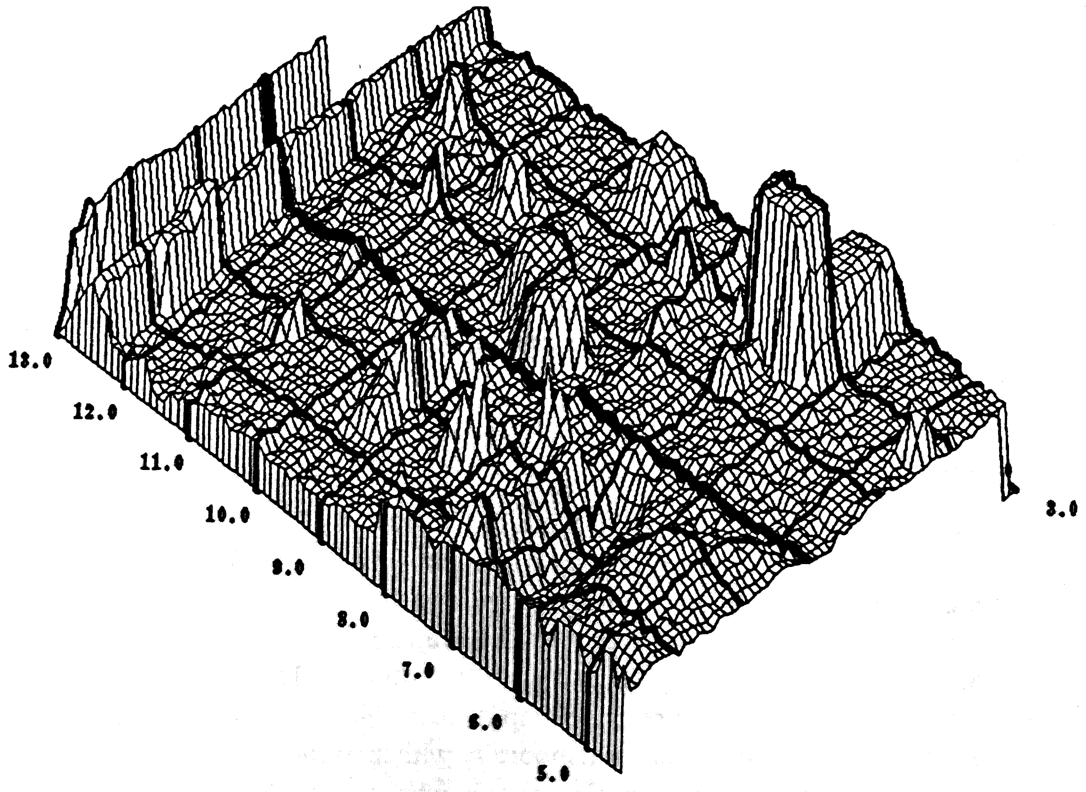

Figure 8: Display Produced by the Perception Display Manager

performance of current tasks, and by "extensions" we mean increasing functionality to perform new tasks.

Improvements

The desired improvements for the ISM are relatively minor. They consist of improving the readability of the code, and producing more comprehensive documentation.

Most of the improvements desired for the LTM Manager involve processing speed. In particular, the algorithms for, and implementations of, the Locus Method and uncertainty map computations must be enhanced. In addition, the LTM Scroller should store clipped LTM values in secondary memory (e.g., on disk) instead of deleting them.

Processing perception data concurrently with planning and executing of commands has the potential for significantly increasing the overall performance of the walking system. In one design, the LTM Manager would have a subtask that would compute elevation maps in regions and at resolutions that are likely to be subsequently useful to the planners. With the current system, the decision about what regions to preprocess does not have to be very sophisticated. The LTM Manager is usually idle enough to compute the complete map in front of the current foot location. Although nearly continuous walking can be achieved without concurrent perception [27], experience with the Ambler will indicate the utility of concurrent perception to enhance performance.

Extensions

We plan to extend significantly the functionality of the ISM and LTM Manager, and to include a new module, the Global Terrain Map (GTM) Manager.

We plan to extend the ISM so that it makes decisions about aiming the sensors (by controlling panning or tilting mechanisms), and about when to take images. We may implement the ISM so that it runs on a processor coupled to the real-time system, so that if images are acquired while the Ambler is moving the scanner pose at the time of image acquisition can be determined more accurately.

We plan to extend the LTM Manager to identify the displacement of the sensor between images. In the current implementation, the Controller computes the sensor displacement between images. For the single-leg testbed, this is satisfactory due to the stability of the sensor platform (the carriage) and the accuracy of the position/velocity transducer. However, the Ambler may be less stable and the position and orientation measurements may be less accurate. Therefore, it is desirable for the perception subsystem to augment the dead-reckoning of the vehicle by identifying the sensor displacement. To accomplish this, we will implement the LTM Matcher, which will accept the following input,

1. the LTM L_k^0 constructed from a sequence of range images $I_0, I_1, \ldots, I_k$,

2. the most recent range image I_{k+1},

3. an estimate of the displacement ${}^{k}\hat{T}_{k+1}$ between viewing stations k and $k+1$,

and compute as output the best displacement ${}^{k}T_{k+1}$ in the least-squares sense. References [10, 19] describe in more detail our proposed approach to matching.

We plan several other extensions to the LTM Manager:

- To represent discrete objects, such as boulders.
- To represent material properties of terrain as well as geometric properties. In particular, we plan to produce maps that represent ground compliance, coefficients of static surface friction, and object density.
- To analyze the traversability of elevation map patches on the basis of the estimated roughness of the surface [13].

The GTM Manager will manage the construction and maintenance of a global terrain map (GTM). A GTM differs from an LTM in scope of information and in scale, extending hundreds or thousands of meters on a side. The GTM is to be used for long-range navigation and route planning. It can be constructed by piecing together a number of LTMs [10, 19], or by utilizing information not acquired by the Ambler (for example, telemetry from Earth or an orbiting satellite). Important issues include managing large amounts of data, multiple sources of data (including off-board sources), and multiple resolution data (e.g., low resolution satellite data and high resolution scanner data).

6 Planning

As described previously, the single leg walks by moving the leg, planting it firmly on the ground, and then pulling the carriage along using the horizontal actuators. This sequence is continued until the carriage reaches a given goal position along the rails.

The planning problems (for both single-leg and six-leg walking) include deciding where to plant the leg, how to move it through three-dimensional space to that spot, and how far to move the body at each step. Our approach utilizes features of the robot's design to plan movements that are time and power efficient, while at the same time maintaining the stability and integrity of the mechanism. This combination of using design constraints to choose efficient and reliable moves, while trying to maximize progress, has led to a planning system that performs well in rough terrain.

6.1 Rationale

One guiding factor in developing the single-leg planning was that the results should be applicable to the Ambler. To this end, some of the algorithms are more complex than are justified by the problem of moving the single leg, but we are gaining insights (and code) that we believe will transfer to the complete walker.

Other factors influencing the planning algorithms were related to constraints imposed on an autonomous planetary explorer. In particular, a planetary explorer would need to be both power efficient and reliable, while trying to maintain good progress towards its goals. For example, [24] estimates that a complete rover would operate on 500 watts. To this end, the planning modules attempt to find leg movements that minimize both the power usage of the mechanism and the time spent in movement.

Above all, a planetary explorer needs to ensure its own survival. It needs to avoid movements that could result in damage to the machine, such as excessive force being applied to a leg, crashing into terrain features, and (in the case of six legs) crashing into other legs and tipover. The planning algorithms developed are sensitive to the applicable geometric and terrain constraints of the rover, attempting to maximize forward progress while choosing footfalls that are conservative with respect to the constraints.

6.2 Description

The planning subsystem consists of two modules: the Gait Planner chooses footfall locations and body advances, and the Leg Recovery Planner chooses trajectories from the current leg position to the planned footfall location.

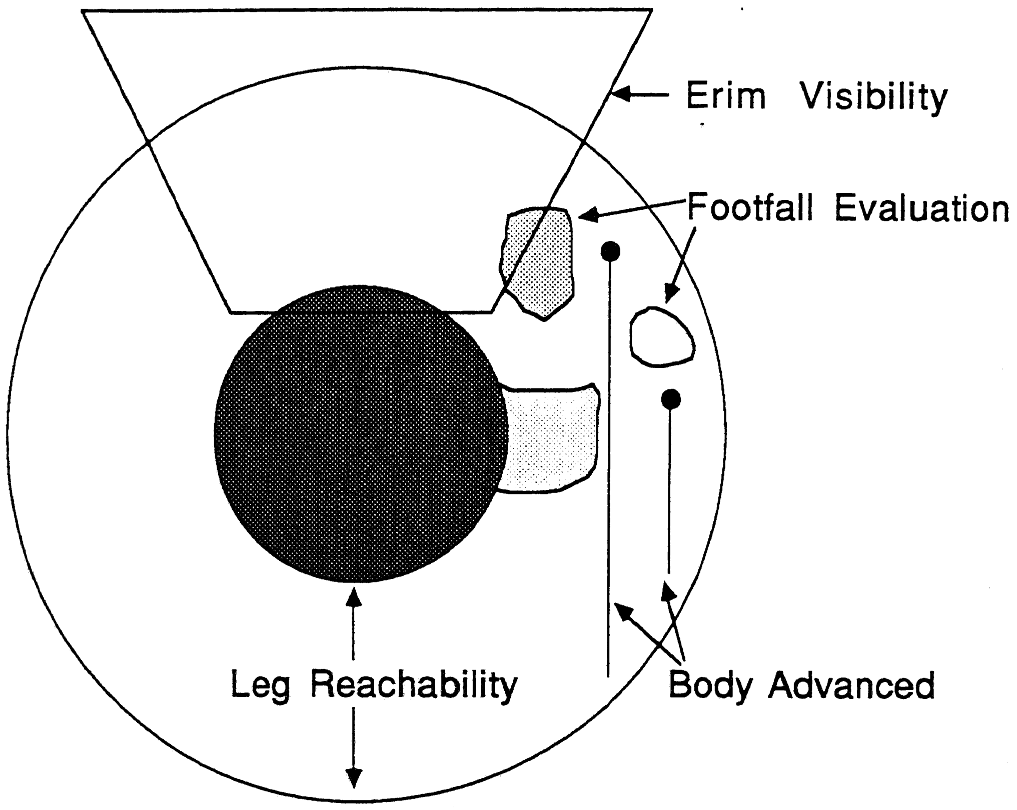

Figure 9: Constraints on Choosing a Good Footfall

6.2.1 Gait Planner

The Gait Planner plans footfalls by combining various geometric and terrain constraints for each possible footfall location. The constraints are computed on a 10cm grid for all points reachable by the leg (actually, the reachable space forward of the carriage). For each constraint, a *cost map* is created that indicates the goodness of the constraint within each grid cell. The cost maps are combined into a composite map using a weighted sum, and the grid cell with the lowest cost is chosen as the footfall location. The Gait Planner then chooses a body move that is the minimum of 1) the best possible advance from the chosen location, and 2) a user-defined threshold (we typically constrain the body advance to 1.5m to get a reasonable number of footfalls over the length of our 11m obstacle course).

Advantages of this constraint-based approach are that 1) the planner does not have to commit *a priori* to which constraint is most important, 2) and it is easy to add new constraints as relevant ones are identified [28]. Although this approach evaluates a large number of grid cells, in practice the gait planning is fast relative to other computations.

Relevant constraints used by the Gait Planner were derived from both analysis and experimental evidence. They include (Figure 9):

Footfall Relatively flat terrain is preferable both for stability and for providing traction in moving the body. The LTM Manager computes a statistic that indicates the flatness of the terrain around the foot (see Section 5). To account for the limited resolution of the scanner and inaccuracies in the mechanism control, the Gait Planner also produces a filtered footfall cost map. This map indicates the maximum footfall value in some neighborhood of the grid cell, weighted inversely by the distance from the cell. The filtered footfall cost map has the effect of steering the vehicle away from locations that are adjacent to obstacles.

Body Advance The body advance cost map indicates how far the carriage can travel from a given footfall location until the leg is extended to its maximum. Originally, this map was computed using the kinematics of the leg, but this proved to be too optimistic, given the compliance in the leg and the friction in the overhead rails. The body advance cost map is now based on empirical values derived from our testing program of the Controller's body move procedure (Section 4.2).

Scanner Visibility The leg should not be in the field of view of the laser scanner, since it can occlude large areas of the terrain. The visibility cost map indicates whether the horizontal links intersect the scanner field of view after the body move is done, assuming that the body moves by its maximum from each leg position. Currently, the cost map is just binary (in view or not), but we intend to extend it to compute the percentage of the field of view obstructed.

Leg Reachability An area might contain good footfalls, but not be acceptable if it is unreachable: either beyond the mechanical limits of the leg, or surrounded by very high obstacles (including other legs, for the six-leg case). The Leg Recovery Planner calculates a binary reachability map that takes feasible leg movements into account.

Terrain Elevation Simply, the leg cannot reach areas that are too high or too low (because the body height is fixed).

Other constraints that we have identified as relevant, but not yet incorporated into the Gait Planner are the average slope of the terrain (useful for determining the traction available for body moves), and the lateral forces exerted on the foot during body moves. These can combine with previous studies made on the pullout forces of the leg at various slopes [23] to gain a more accurate picture of the reliability of potential footfalls. The forces also give an indication of the efficiency of the body moves, enabling us to plan power-efficient moves.

In combining the above cost maps, the footfall and filtered footfall maps are given high weights, and the scanner visibility and body advance maps are given lower and approximately equal weights. The leg reachability and terrain maps are used as binary constraints: if the location is not reachable, it is eliminated from consideration, regardless of the other values.

6.2.2 Leg Recovery Planner

While the Gait Planner decides where to put the foot, it is the responsibility of the Leg Recovery Planner (LRP) to determine the trajectory that will get the leg to that position without hitting any obstacles.

The LRP uses a novel algorithm that finds time and power efficient moves through three-dimensional space, while searching only a 2D grid, thus increasing the planning efficiency considerably. The planner starts by creating a configuration search space for the elbow and shoulder joints [22]. It divides the space into a discrete grid approximately 0.1 radian wide, and fills the grid with obstacles by growing terrain features and other legs (for the six leg case) by the radius of the foot plus an uncertainty factor.

The LRP searches this space using the *Envelope Trajectory Finding Algorithm* (ETFA) to find the least-cost path to the goal location. The ETFA may choose to move the leg either over or around an obstacle (or some of both), depending on the overall cost (a weighted sum of power and time). Estimating power usage is fairly simple: the power needed to reach a grid cell from an adjacent cell is the sum of the power needed to move the elbow and shoulder joints to the cell, plus the power needed to raise the leg above the elevation associated with that cell. This, of course, assumes that the power consumption in each joint is independent of the others, which is a reasonable approximation given the slow speeds of our machine.

It is more difficult to estimate the time needed to get to a cell. To see this, consider the case in Figure 10a. If we add up the times to get to each individual cell, path *A* is the quickest way to get to point *X*. To get to point *Y*, however, path *B* is actually better, since in path *A* the horizontal joints must stop and wait for the vertical lift, while in path *B* the leg is lifting while it is moving horizontally.

In essence, we need to keep track of all possible paths that the leg can take in reaching a particular grid cell. This is the "envelope" part of the ETFA. The algorithm keeps track of the maximum and minimum heights that the leg can reach in any particular cell, assuming that the leg lifts/lowers at full speed while moving horizontally (Figure 10b). Thus, the leg can reach anywhere within the envelope in the same amount of time. Only if the terrain is above the top of the envelope (e.g., point *Z*) does the leg have to stop moving horizontally and lift.

The LRP uses the ETFA and A* search to find the minimum cost path (weighting power and time by a user-specified ratio). At the end of the search, the planner determines an actual trajectory through the envelope space by choosing vertical moves that minimize the risk to the machine while maintaining the optimality of the path found. In particular, this means performing all purely vertical lifts at the start and delaying all purely vertical lowers until the end of the move (Figure 10c).

6.3 Discussion

While the Gait Planner and LRP have proven themselves in actually moving the single leg through rough terrain, there are several areas for improvement.

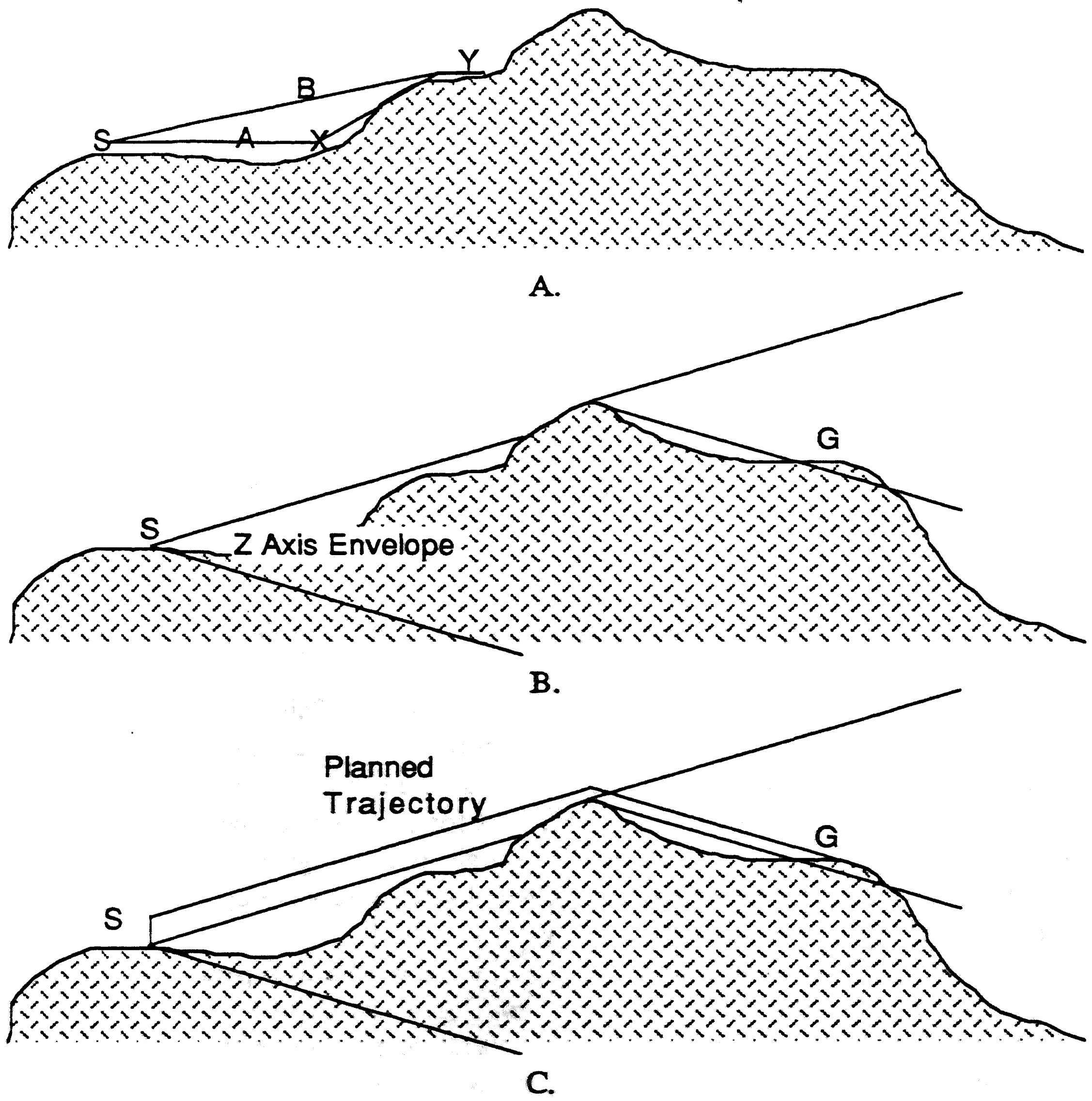

Figure 10: The Envelope Trajectory Finding Algorithm

First, a difficulty with the constraint-based Gait Planner is in choosing the weights for combining the cost maps. Our current methodology is empirical: trying the system on a variety of terrains and tweaking the weights to reflect the results of the experiments. While we have tuned the parameters to walk over a wide variety of terrain (see Section 7), it is still a matter of guesswork. To make the process of choosing weights less *ad hoc*, we are considering the use of adaptive algorithms that autonomously adjust the constraint weights based on the difference between the planned moves and actual outcomes.

As with the perception subsystem, substantial increases in performance can be obtained by taking advantage of the concurrency inherent in the TCA. In particular, moving the leg and body takes substantially longer than the planning and perception times combined. Thus, nearly continuous motion can be achieved by planning the next step while executing the current one. We have recently modified the planners to operate concurrently, and the results are very encouraging [27]. We achieved an average performance increase of about 30%, with only minor modifications to the existing code.

Exception handling is another important planning issue that we are beginning to address. TCA provides mechanisms for detecting and handling plan-time and execution-time failures. Some of these techniques have already been incorporated into the system. In particular, if the Controller cannot perform a leg or body move, the current step is terminated and the Gait Planner is re-invoked to plan from the current position. If a hardware failure is detected, an exception handler kills the task entirely, bringing the system to a graceful halt.

Several aspects of the LRP are worth noting. First, there is the problem of grid bias in the search: moves in the shoulder/elbow space must be along grid lines, so the planner either moves the shoulder alone, the elbow alone, or both the same distance. This often results in plans with many turns (Figure 11a). Executing the resultant path produces undesirable vibrations in the leg due to the frequent changes in direction. We have addressed this problem by including a factor in the search evaluation function that adds a time cost for each turn in the path. Essentially, this *turn penalty* "encourages" the planner to pursue straight paths (Figure 11b), with the degree of encouragement dependent on the size of the turn penalty (currently we operate with a penalty of 0.5 seconds per turn).

Another problem with grid bias (which is exacerbated by the turn penalty, in fact) is that it sometimes leads to non-optimal paths. While the grid forces the shoulder and elbow to move the same distance each time, often the optimal path involves moving the joints non-uniformly. Although some researchers have used relaxation algorithms to eliminate grid bias [30], the techniques developed so far are not applicable to 3D paths. In any case, the maximum deviation from optimal caused by the grid bias is only about 8% [32], and in practice the degradation in performance is not noticeable.

Finally, it should be pointed out that there are pathological cases (which we have not yet seen in practice) that could cause the algorithm to choose a sub-optimal path. Currently, we are attempting to categorize this phenomenon. We believe, however, that the plan-time efficiency of the ETFA more than outweighs the occasional sub-optimal plans that it may produce.

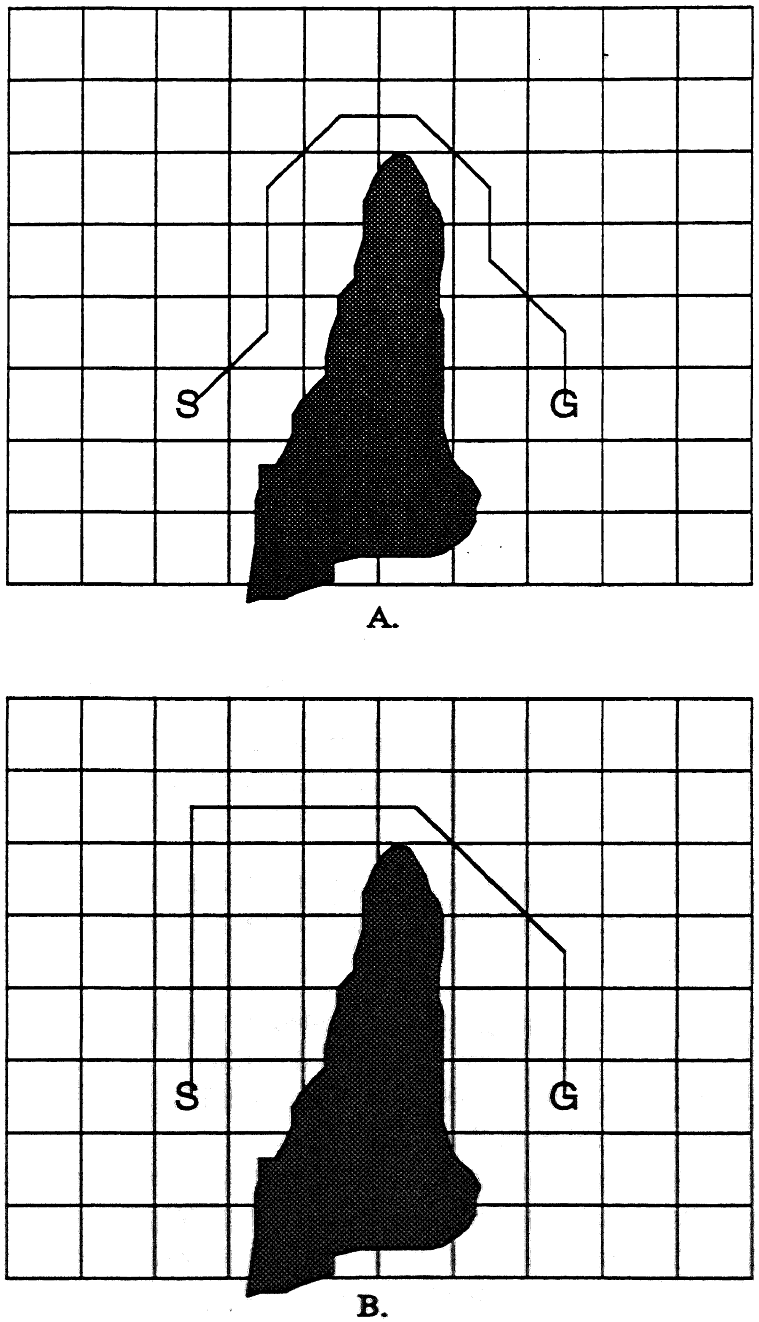

Figure 11: The Problem of Grid Bias

7 Experiments

After the single-leg testbed became operational in May 1989, we performed two series of autonomous walking tests. One series took place during the Summer of 1989, the other during the Autumn. The overall purpose of the experiments was to validate the integrated system under realistic conditions. This section describes the testing program and some of its results.

7.1 Summer Tests

We performed the Summer walking tests in two stages. For the first stage, we fixed the terrain and gradually increased the number of software modules. For the second stage, we fixed the number of software modules and varied the terrain.

Fixed, Level Terrain

For the first stage of testing, we leveled the terrain and did not alter it between runs. We began with a minimal set of modules (the TCA central control and real-time Controller), and incrementally added modules, as follows (see also Figure 4).

1. A User Interface module is added. This module enables us to interactively issue *legMove* and *bodyMove* commands to advance the carriage from one end of the testbed to the other.

2. The LTM Manager, ISM, and LRP modules are added. In the User Interface module, in addition to issuing *legMove* and *bodyMove* commands, we also issue queries that exercise the LTM Manager, ISM, and LRP.

3. We replace the User Interface module with the Gait Planner module, which issues the appropriate sequence of queries and commands to autonomously move the leg and carriage the length of the testbed.

By the end of the first stage of testing, the integrated system was able to walk reliably on level terrain. This is not a dramatic accomplishment, but neither is it trivial. More precisely, the main challenge in level terrain walking involved the compatibility of the software modules. We encountered two types of problems during this stage of testing.

The first type of problem involved misunderstandings in interfaces, as illustrated by the following examples. Some of the modules used units of radians to describe angles, others used units of degrees; this caused conflict between the different modules resulting in untimely termination. Some of the modules used underscores in message names, others used hyphens; this caused TCA to route messages incorrectly. Some of the modules interpreted the roll angle as a rotation about the *Z*-axis, others as a rotation about the *X*-axis; this caused the elevation maps to rotate. Besides trial and error, this type of problem was alleviated by carefully documenting the interfaces and making all interface data structures globally accessible to all groups.

The second type of compatibility problem concerned inconsistent software environments. For example, some modules did not assign values to required environment variables (UNIX shell variables that users can alter dynamically) or set them incorrectly; this caused programs to terminate abnormally. Similarly, different programmers and users run incompatible window managers (namely, X and Suntools); this caused difficulties in operating and debugging the system. We found that documentation and experience in a heterogeneous environment ameliorated this problem.

Varied, Non-Level Terrain

In the second stage of Summer testing, we used the same six modules as above, but walked over increasingly difficult terrain. Beginning with level ground, we graduated to gently rolling hills, trenches, a few small obstacles, a dense field of small obstacles, and large obstacles. By the end of this stage of testing, the integrated system was able to traverse reasonably challenging terrain. Obstacle Course 1 (Figure 12) is an example of terrain successfully negotiated by the system.

Although the system demonstrated competence, it performed unreliably. In some cases, it could walk once but not twice over the same terrain. In the unsuccessful cases, the leg collided with and stepped on obstacles, and the foot pulled out when walking up slopes.

We found that these problems were not easy to solve. For example, it was often difficult to decide what went wrong when the leg struck an obstacle. The fault could be in any of the subsystems. The LTM Manager might have computed an inaccurate map, incorrectly determining the obstacle location. The Gait Planner might have chosen a poor footfall location, or the LRP might have chosen an erroneous leg trajectory. The Controller may have executed poorly a perfect plan. Or the fault could be due to a combination of these factors.

After analyzing the results of the experiments, we identified several key areas requiring improvement.

1. Body moves were inaccurate: the difference between commanded and achieved propulsion could be as large as 40 cm. We adopted a different body move procedure, and reduced the error to under 5 cm (see Section 4.3).

2. Leg moves were inaccurate: the error in the position of the foot could be as large as 20 cm. We calibrated the leg and implemented an inverse kinematics function that takes into account rotations caused by torsional deflections of the rails. This reduced the error to around 5 cm.

3. The Gait Planner chose questionable footfalls at times: it appeared to select locations that either were uncomfortably close to obstacles or did not permit significant body advances. We rewrote the Gait Planner module to use a constraint-based approach (Section 6.2.1) and developed debugging aids, including visual display of cost maps.

4. The foot sometimes slid into obstacles when lifted. This occurs due to strain in the leg after body moves. Our solution is to relieve the tension by moving the leg so that the final joint angles correspond with the expected (i.e., planned) Cartesian position of the leg.

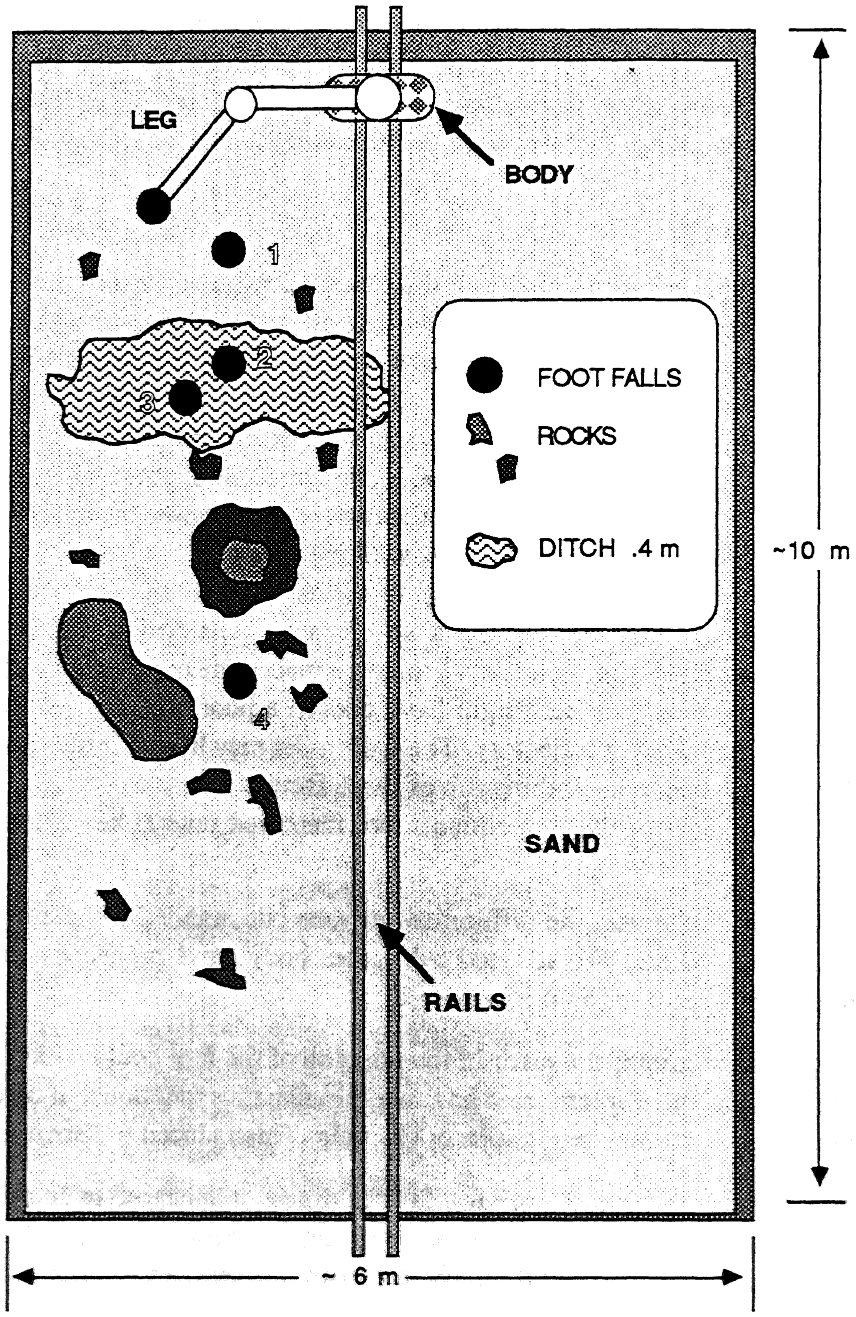

Figure 12: Diagram of Obstacle Course 1

7.2 Autumn Tests

In the Autumn of 1989, once the above changes were developed and tested, we repeated the second stage of Summer experiments on a variety of obstacle courses. Obstacle Course 2 (Figures 13 and 14) is representative of the terrain we successfully negotiated during this round of experiments. As the figures indicate, these obstacle courses were significantly more complex in terms of the number and arrangement of obstacles than the ones used in the Summer tests.

Again, we experimented incrementally, starting with the steeplechase arrangement of pylons. For several weeks, the integrated system succeeded only sporadically in stepping through the pylons. It would negotiate the terrain once or twice, and fail on the next trial by striking or stepping on one of the obstacles. After some modifications to the Gait Planner and LRP, the integrated system walked repeatedly through the pylons. We then made the terrain rougher by placing obstacles at footfall locations selected during previous trials. The system selected new footholds, and successfully negotiated each of the new configurations.

Another challenge was Obstacle Course 3 (Figure 15) which differs from Obstacle Course 2 by the large obstacle (the box labeled "Sun") at the beginning of the course. Figure 16 illustrates an elevation map of Obstacle Course 3 constructed by the LTM Manager. This new obstacle is tall enough so that the leg can not reach over it. Thus, the LRP must plan a trajectory that skirts the obstacle.

The integrated system traversed Obstacle Course 3 more than 30 times. During one afternoon, the system successfully negotiated the obstacle course (with minor variations) seven consecutive times. While it rarely performed exactly the same footfalls or leg trajectories, it did negotiate the terrain in a predictable and repeatable manner.

Obstacle Course 4 (Figure 17) is the same as Obstacle Course 3 on the right-hand side of the testbed (as seen by the scanner), but has a new constellation of obstacles on the left-hand side. The new obstacles include pylons so closely spaced that there is no room for the foot among them, a hill of sand, and a trench deep enough so that the leg can not reach the bottom.

The system traversed Obstacle Course 4 by walking forward up one side of the course and then walking backward down the other side (typically we walked forward on the right-hand side, but in some experiments we switched directions). The system cannot use the laser scanner to guide itself in walking backward because the scanner is fixed to point forward. Instead, the system uses the local terrain map built by the LTM Manager while walking forward. We found walking backwards to be surprisingly easy: we could usually follow a successful forward traverse by a successful backward traverse.

A typical time breakdown for traversing Obstacle Course 3 is presented in Figure 18. The darkly shaded areas of the chart represent times when a module is computing; lightly shaded areas are times when a module is awaiting a reply from another module. The system takes six steps in 13.5 minutes, while traveling about 8 meters (60cm/min). Figure 18 indicates that about 60% of the time is spent by the Controller in moving the leg and carriage. Conversely the ISM, which spends only one half second for each of the seven images it acquires, is nearly always idle.

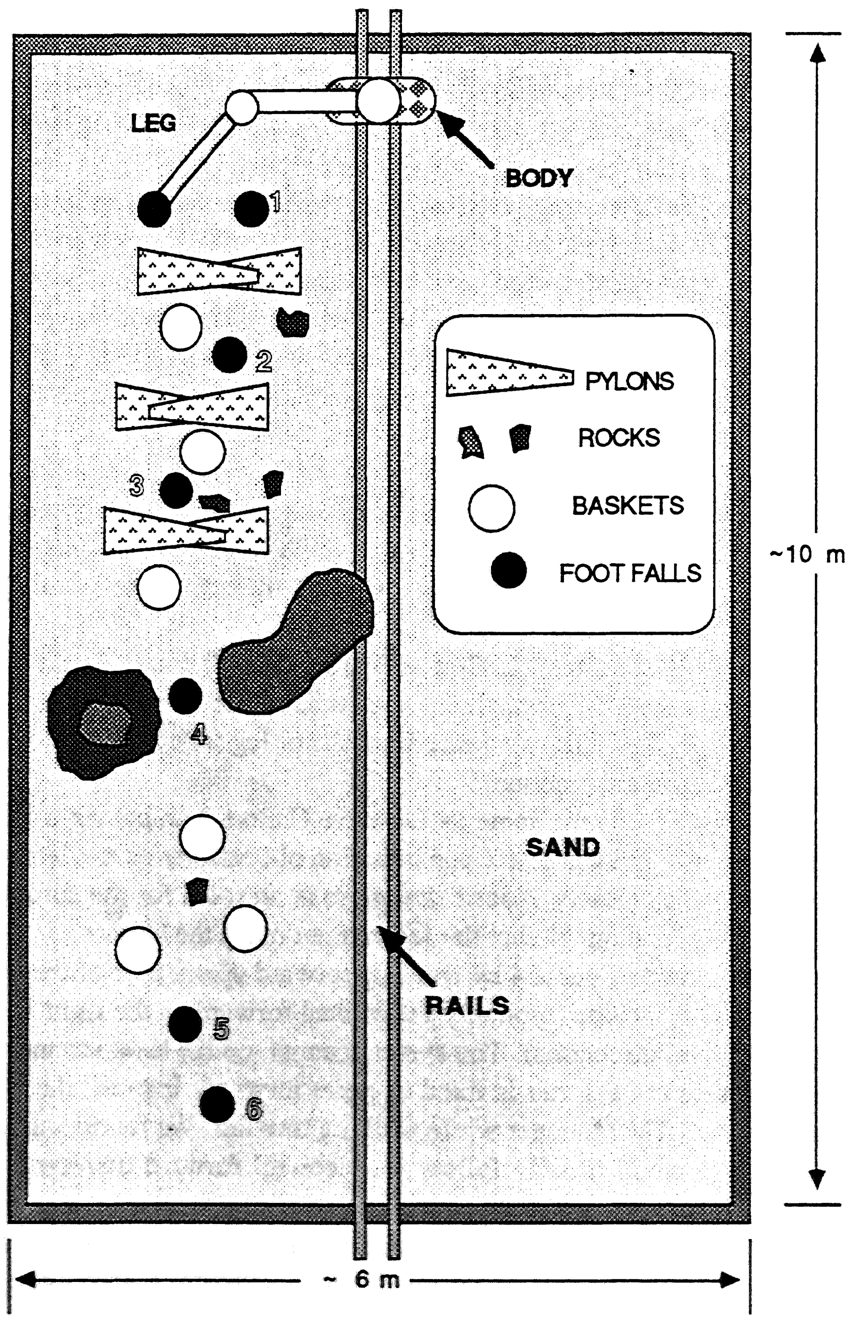

Figure 13: Diagram of Obstacle Course 2

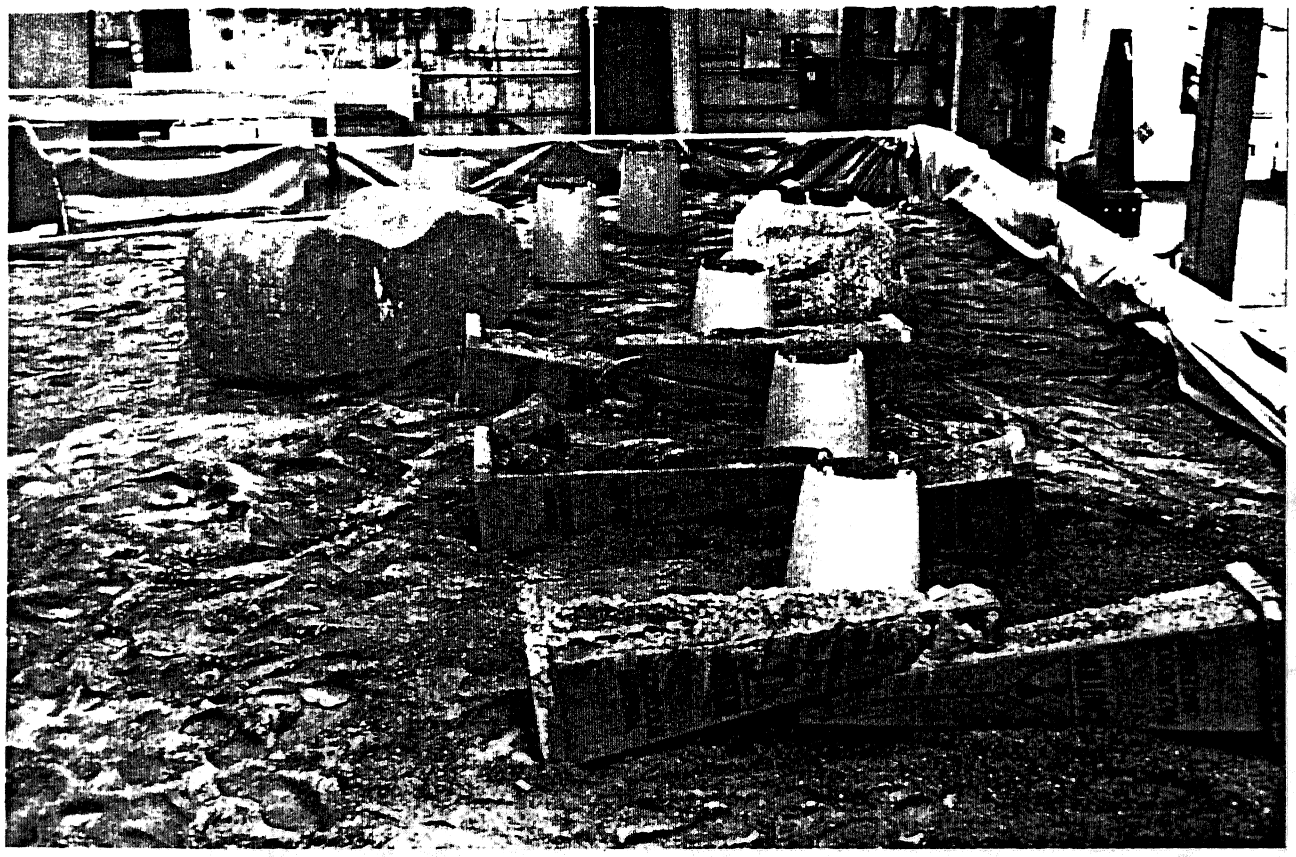

Figure 14: Photograph of Obstacle Course 2

G
PITT

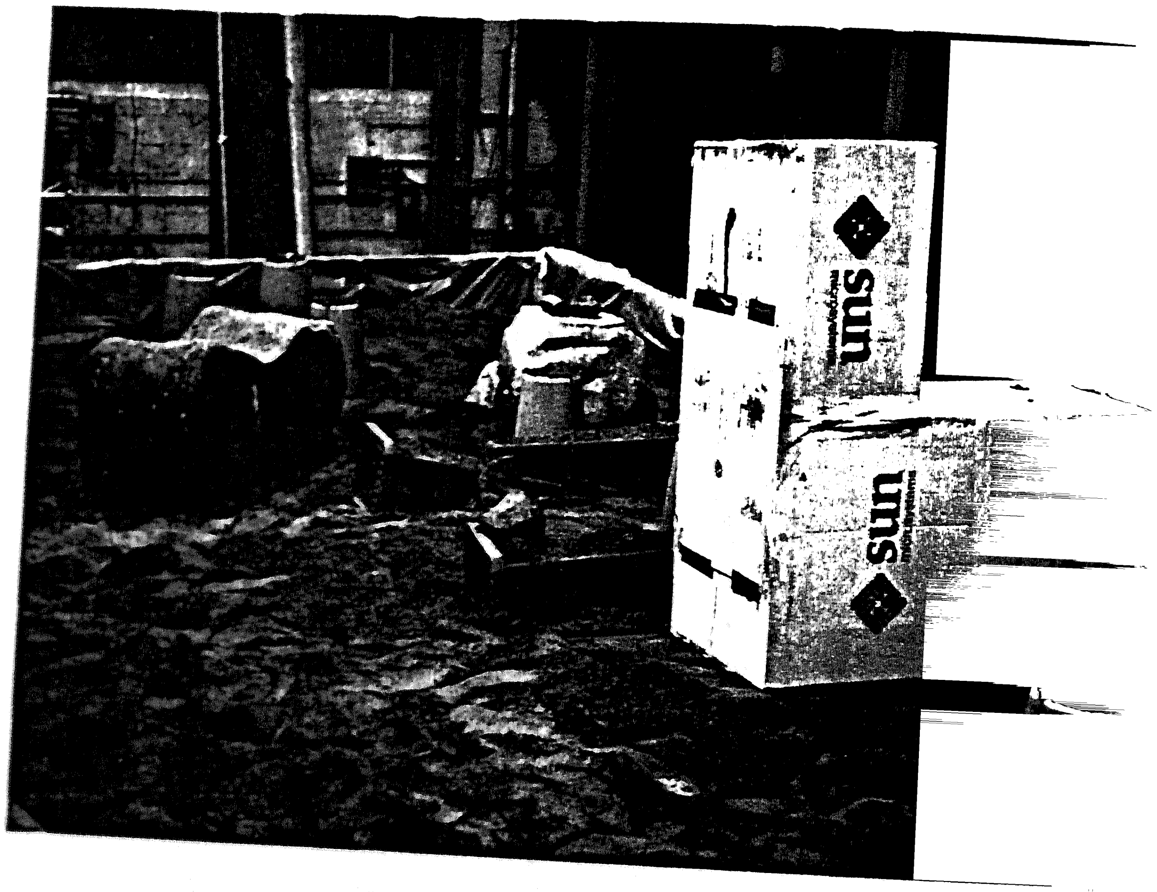

Figure 15: Photograph of Obstacle Course 3

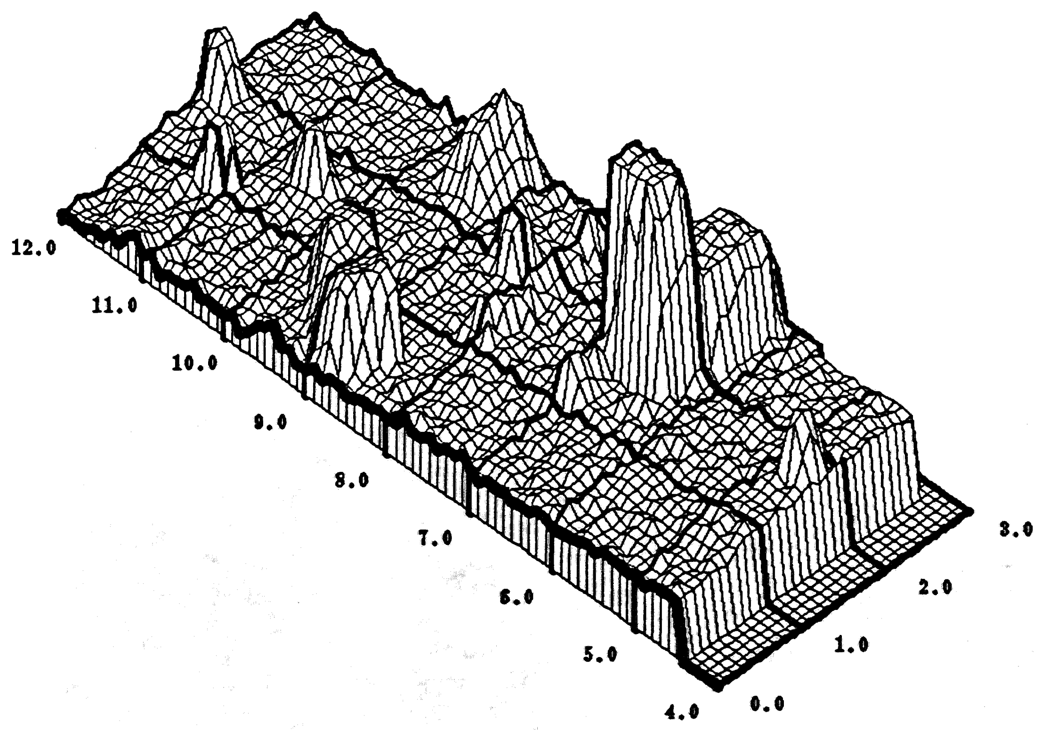

Figure 16: Elevation Map of Obstacle Course 3

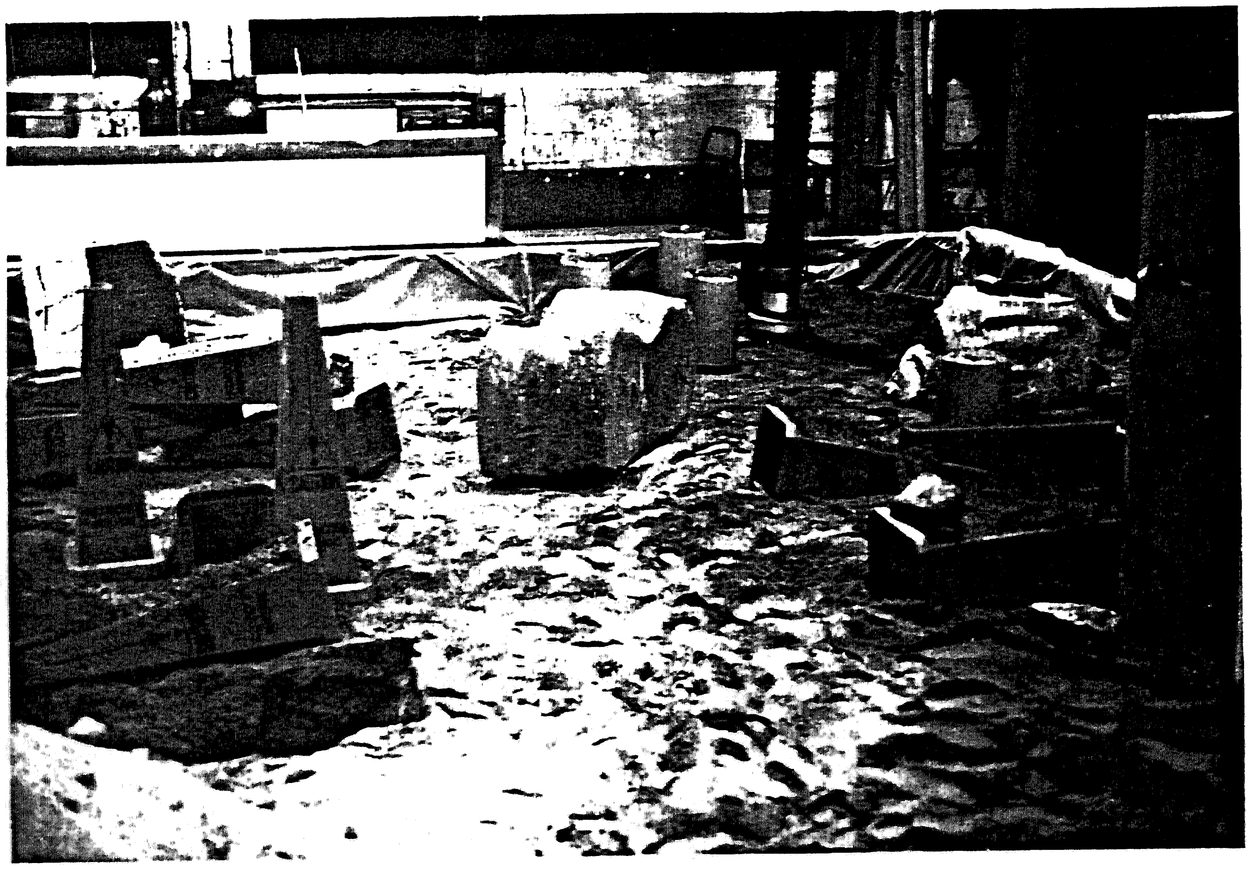

Figure 17: Photograph of Obstacle Course 4

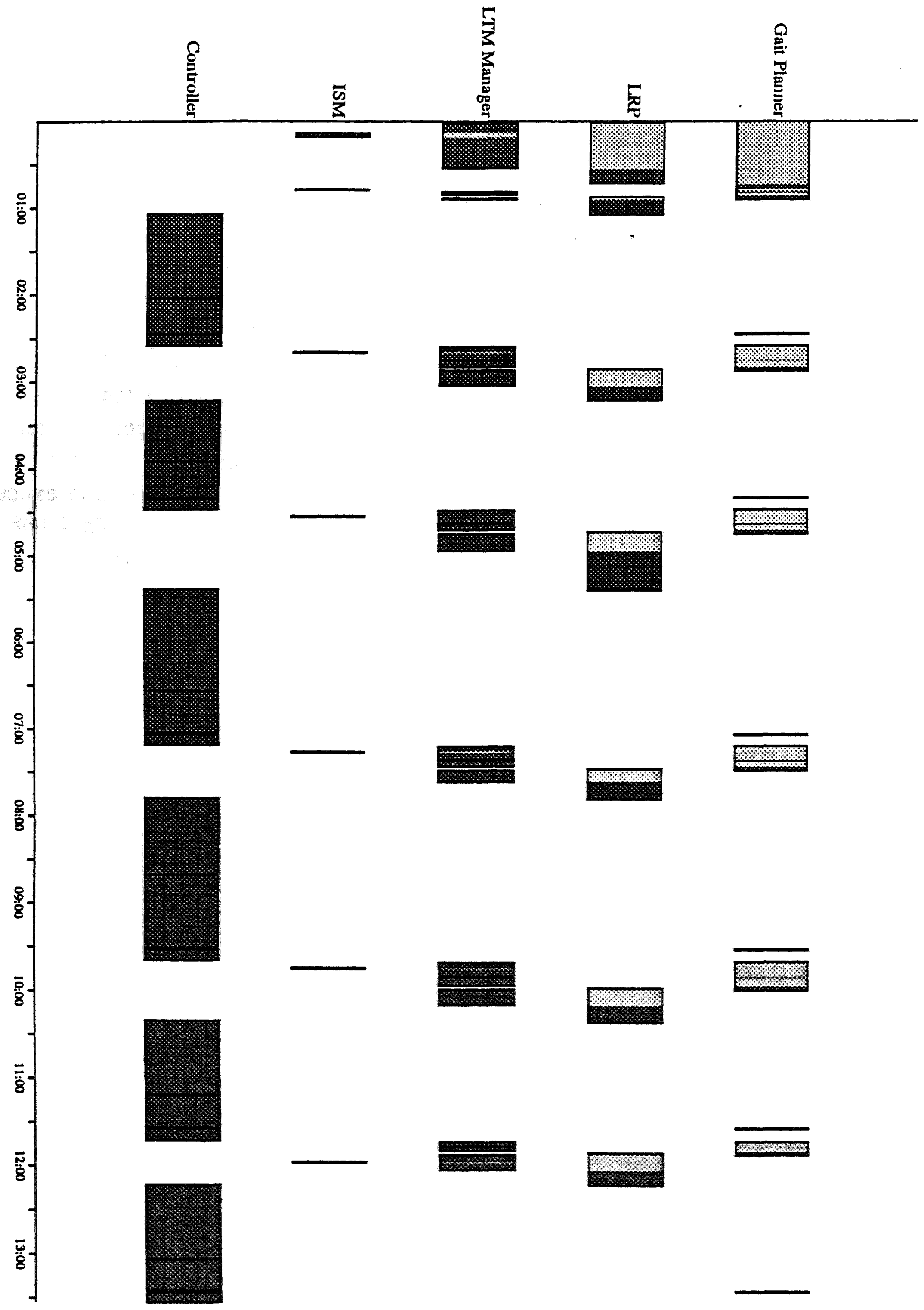

Figure 18: Timings for Single-Leg Walking Modules

To reduce the complexity of the chart, the 71 leg and body position queries handled by the Controller are not illustrated. In any event, they have negligible effects on the timings, since they are handled in less than 50msec each. Also not shown is the time spent by the TCA central control module. This time, which is included in the times presented in Figure 18, account for about 3% of the total operating time of the integrated system.

Note that the times for the Gait Planner and LRP are somewhat deceiving since they include the time spent waiting for the LTM Manager to compute requested terrain maps. Exclusive of those wait times, the Gait Planner averages about 2 seconds per step and the LRP about 12 seconds. Note also that the first footfall takes significantly longer to plan than subsequent ones. This is mainly due to the fact that the LTM Manager must calculate the first terrain map from scratch, but for subsequent steps it can use some of the cached maps to save computation.

As mentioned, we have done some experiments with concurrent planning and execution of steps, with the aim of achieving nearly continuous walking. The use of concurrency has reduced the typical traversal time to about 8.75 minutes for six steps, a 35% performance improvement [27]. At this speed, the mechanism itself is the system bottleneck, except for a one minute period at the beginning when the first step is being planned.

8 Discussion

8.1 Integration Issues

Knowing that integration could prove difficult, given the number of people and amount of software involved (over 50,000 lines of C code), we decided early on to try to standardize the integration effort. In part, this was facilitated by the message-passing conventions of TCA, which forced us to detail our interfaces. However, there were many interface problems that could not be solved just by agreeing on data structures. We needed to decide on the semantics of those structures: whether we would use radians or degrees, body-centered or world-centered coordinates, etc. Although these issues may seem trivial, they in fact are responsible for much of the difficulty in integrating large systems.

We adopted several approaches to facilitate the integration effort (some of them, admittedly, *post facto*). First, we instituted regular weekly meetings of the principals in each of the component areas. It was the responsibility of this group to decide on the semantics for all interfaces between modules. Although the interfaces evolved over the course of the research, the integration group functioned fairly well as a clearing-house for any changes. In retrospect, relatively few of our failures were attributable to misunderstood interfaces, and many of the misunderstandings that did occur were by people who had not been attending the integration meetings.

A second approach to integration was rigorous documentation. In the early part of the project, efforts were made to record the design decisions for each of the modules. This proved less than successful, mainly due to the tension between getting the code to work and keeping the documentation up-to-date. A more successful documentation effort has been to create UNIX manual pages for each module and for each message in the system. Typically, these describe the I/O behavior of a module or message. Since the document is relatively short (typically one page), keeping it current is less difficult. In addition to being available on-line, the manual pages are bound and kept at the testbed control room for reference.

Finally, we have attempted to standardize our software structure. This includes consistent naming conventions for variables, message names, data types, etc. [8], and consistent directory/sub-directory structures. Both are efforts to make common code more accessible to developers. Unfortunately, this change was made rather late in the project, causing a fair amount of added work during the conversion process, but since then the standardization has worked to our advantage.

One of the practical difficulties was getting all of the experimenters at the testbed at the same time. There are many reasons for this, including other obligations, and frustration with slow progress. Pagers supplied to key members of the project proved to be useful in communicating quickly with absent personnel.

8.2 Surprises

When the project began, we all had assumptions about what parts would be easy and hard. To a certain extent, these assumptions were valid, but in addition a number of unforeseen difficulties arose. In addition to describing the difficulties, this section tries to anticipate how we can overcome them in the full six-legged walker.

Calibration Probably the most problematic issue was (and continues to be) the calibration of the leg and scanner frames. In order to get reliable footfalls, if perception finds a clear spot in the terrain, we want the Controller to put the foot there. Inaccuracies in the calibration may cause the leg to step on obstacles.

Our current approach is to reference both to a common world frame, and do all our calculations in that frame. The problem is that due to inaccuracies in the laser scanner and leg mechanism and due to inaccuracies in our models of how they work, the calibration to the world frame is never exact. We have spent much effort in calibrating the Erim scanner and in devising better kinematic models for the leg, but residual errors still exist. For the Ambler, it might be useful to augment these model-based calibrations with an empirical method that calibrates the scanner and the legs directly. One promising method would be to find the leg in many images and use a connectionist approach that essentially builds an inverse kinematics table [4].

Coordinate Frame The choice here was threefold: we could do our calculations in a body-centered Cartesian frame, in a body-centered joint space reference frame, or in world coordinates. Unfortunately, after much thought and discussion, no one clear reference frame seemed preferable. A Cartesian frame was better for the elevation maps, since they could be easily scrolled and merged with other images; a joint-centered frame was better for geometrical planning problems (especially the Leg Recovery Planner).

Finally, a global frame was chosen for most interfaces because of the possibility that the body could slip along the rails between the time that planning was done and the time that execution began. The major exception is the *legMove* command, which is given in joint space coordinates. This problem, which will be even more acute in the six-legged case, is not yet settled. In particular, by choosing a world reference frame we can more easily control the robot to avoid terrain obstacles, but a body-centered frame has advantages in controlling the robot as it moves between and around its own legs. Our current thinking is to provide both modes, and let the planners use the most relevant mode for each individual movement.

Compliance The single leg has a large amount of compliance, primarily due to its long vertical axis. This problem manifests itself in the problem of leg calibration, described above, but it is most problematic in the control of body moves. Our original body move software simply computed the joint angles necessary to move the body to the desired spot, and then controlled the joints to produce a linear motion. Unfortunately, the compliance of the leg often produced position errors as much as half a meter for a commanded one meter move. While the current software, which servos to the body position, is much more reliable, it will still occasionally fail to move the body adequately.

The compliance of the system was noted fairly early, and has led to a major redesign of the six-legged mechanism. While preliminary testing of the Ambler indicates that it is sufficiently rigid, we believe we have the experience from the single-leg testbed to handle any residual compliance.

Planning Footfall Locations When the six-legged walker was first proposed, it was suggested that the range of the machine made terrain considerations secondary to geometry in planning footfall locations. That is, the planner could choose a small region that was optimal for leg trajectories and body moves, and then search that area in the elevation map to find the flattest point. In practice, that strategy has not worked. In particular, the inaccuracies in the leg and scanner (see above) has led us to the conservative strategy of trying to find at least a 70cm diameter region in which to put the 30cm foot. This has led to a reversal of our planning strategy, in which now the terrain considerations are paramount. Although we are uncertain as to which constraint will dominate in the six-legged case, the constraint-based structure of the Gait Planner should enable us to experiment with various constraints and weightings without much alteration to the basic planner.

8.3 Conclusion

This report describes a successful integration effort that has led to fairly reliable single-leg walking on rough terrain. The research involved major efforts in the component areas of perception, real-time control, and planning. In addition, much time and effort was spent in integrating the components into a complete system, and testing the walking system over a wide range of terrains.

The major impetus for the single-leg walking program was to gain experience for six-legged walking. To that extent, the project was quite successful. We have gained much insight into controlling the legged mechanism, calibrating the leg and scanner, planning in the face of uncertainties and conflicting constraints, and coordinating a distributed software system. The task ahead is to apply our experiences and successes to an integrated system for the six-legged Ambler.

Acknowlegements

Many members of the Planetary Rover project contributed to the single-leg walking system. Significant contributions in the integration and testing of the system were provided by P. Balakumar, L. Chrisman, C. Fedor, R. Hoffman, M. Hebert, and D. Wettergreen. This research is supported by NASA under contract NAGW-1175.

References

[1] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. Whittaker.. Ambler: An Autonomous Rover for Planetary Exploration. *IEEE Computer*, 18–26, June 1989.

[2] J. Bares and W. Whittaker. Configuration of an Autonomous Robot for Mars Exploration. In *Proc. World Robotics Conference*, Society of Mechanical Engineers, Gaithersburg, Maryland, May 1989.

[3] J. Bares and W. Whittaker. Walking Robot with a Circulating Gait. In *Proc. IEEE International Workshop on Intelligent Robots and Systems*, Tsuchiura, Japan, July 1990.

[4] J. Barhen, S. Gulati, and M. Zak. Neural Learning of Constrained Nonlinear Transformations. *IEEE Computer*, 67–77, June 1989.

[5] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[6] C. Caillas, M. Hebert, E. Krotkov, I. S. Kweon, and T. Kanade. Methods for Identifying Footfall Positions for a Legged Robot. In *Proc. IEEE International Workshop on Intelligent Robots and Systems*, 244–250, Tsukuba, Japan, September 1989.

[7] J. Craig. *Introduction to Robotics, Mechanics and Control*. Addison-Wesley Publishing Company, 1986.

[8] C. Fedor, R. Hoffman, G. Roston, and D. Wettergreen. *Software Standards and Guidelines*. Technical Report PRWP-89-2, Robotics Institute, Carnegie Mellon University, 1989.

[9] C. Fedor and R. Simmons. *Task Control Architecture User's Manual*. Technical Report PRWP-89-1, Robotics Institute, Carnegie Mellon University, 1989.

[10] M. Hebert, T. Kanade, E. Krotkov, and I. S. Kweon. Terrain Mapping for a Roving Planetary Explorer. In *Proc. IEEE Robotics and Automation Conference*, 997–1002, Scottsdale, Arizona, May 1989.

[11] M. Hebert, T. Kanade, and I. S. Kweon. *3-D Vision Techniques for Autonomous Vehicles*. Technical Report CMU-RI-TR-88-12, The Robotics Institute, Carnegie Mellon University, 1988.

[12] M. Hebert, E. Krotkov, and T. Kanade. A Perception System for a Planetary Explorer. In *Proc. IEEE Conference on Decision and Control*, Tampa, Florida, December 1989.

[13] R. Hoffman and E. Krotkov. Terrain Roughness Measurement from Elevation Maps. In *Proc. SPIE Conference on Advances in Intelligent Robotics Systems, Mobile Robots IV*, Philadelphia, Pennsylvania, November 1989.

[14] T. Kanade, T. Mitchell, and W. Whittaker. *1988 Year End Report: Autonomous Planetary Rover at Carnegie Mellon.* Technical Report CMU-RI-TR-89-3, Carnegie Mellon University, January 1989.

[15] T. Kanade, T. Mitchell, and W. Whittaker. *1989 Year End Report: Autonomous Planetary Rover at Carnegie Mellon.* Technical Report CMU-RI-TR-90-4, Carnegie Mellon University, February 1990.

[16] E. Krotkov. *Laser Scanner Calibration Using the Legs of a Walking Vehicle.* In preparation.

[17] E. Krotkov, J. Bares, M. Hebert, T. Kanade, T. Mitchell, R. Simmons, and W. Whittaker. Design of a Planetary Rover. In *1988 Annual Research Review*, The Robotics Institute, Carnegie Mellon University, 9–24, 1989.

[18] E. Krotkov, C. Caillas, M. Hebert, I. S. Kweon, and T. Kanade. First Results in Terrain Mapping for a Roving Planetary Explorer. In *Proc. NASA Conference on Space Telerobotics*, Jet Propulsion Laboratory, Pasadena, California, January 1989.

[19] I. S. Kweon. *Modeling Rugged 3-D Terrain from Multiple Range Images for Outdoor Mobile Robots.* July 1989. Ph. D. Thesis Proposal, School of Computer Science, Carnegie Mellon University.

[20] L. J. Lin, T. M. Mitchell, A. Phillips, and R. Simmons. *A Case Study in Robot Exploration.* Technical Report CMU-RI-TR-89-1, Robotics Institute, Carnegie Mellon University, January 1989.

[21] L. J. Lin, R. Simmons, and C. Fedor. *Experience with a Task Control Architecture for Mobile Robots.* Technical Report CMU-RI-TR-89-29, Robotics Institute, Carnegie Mellon University, 1989.

[22] T. Lozano-Perez. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*, C-32:108–120, 1983.

[23] P. Nagy and W. Whittaker. Experimental Program for the CMU Mars Rover Single Leg Testbed. In *Proc. 20th Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, Pennsylvania, May 1989.

[24] D. S. Pivirotto, and W. C. Dias. *United States Planetary Rover Status – 1989.* Publication 90-6, Jet Propulsion Laboratory, Pasadena, California, May 1990.

[25] J. Randolph, ed. *Mars Rover 1996 Mission Concept*. Technical Report D-3922, Jet Propulsion Laboratory, Pasadena, California, 1986.

[26] R. Simmons and T. Mitchell. A Task Control Architecture for Autonomous Robots. In *Proc. Third Annual Workshop on Space Operations Automation and Robotics*, Houston, Texas, July 1989.

[27] R. Simmons. *Concurrent Planning and Execution for a Walking Robot*. Technical Report CMU-RI-90-16, Robotics Institute, Carnegie Mellon University, July 1990.

[28] A. Stentz. *The NAVLAB System for Mobile Robot Navigation*. PhD thesis, Technical Report CMU-CS-90-123, School of Computer Science, Carnegie Mellon University, March 1990.

[29] H. Thomas, D. Wettergreen, and C. Thorpe. Simulation of the Ambler Environment. In *Proc. Modeling and Simulation Conference*, Pittsburgh, Pennsylvania, May 1990.

[30] C. Thorpe. *The CMU Rover and the FIDO Vision and Navigation System*. PhD thesis, Carnegie Mellon University, 1984.

[31] D. Wettergreen, H. Thomas, and C. Thorpe. Planning Strategies for the Ambler Walking Robot. In *Proc. IEEE International Conference on Systems Engineering*, August 1990.

[32] P. Widmayer, Y. Wu, and C. Wong. Distance Problems in Computational Geometry with Fixed Orientations. In *Proc. First Annual ACM Conference on Computational Geometry*, 1985.

[33] D. Zuk, F. Pont, R. Franklin, and V. Larrowe. *A System for Autonomous Land Navigation*. Technical Report IR-85-540, Environmental Research Institute of Michigan, Ann Arbor, Michigan, 1985.