IEEE International Workshop on Intelligent Robots and Systems
IROS '90

Single Leg Walking with Integrated Perception, Planning, and Control

Eric Krotkov, Reid Simmons, and Charles Thorpe
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We describe an integrated system capable of walking over rugged terrain using a single leg suspended below a carriage that rolls along rails. To walk, the system uses a laser scanner to find a foothold, positions the leg 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 rocks, 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. Implementing this integrated system is a significant step toward the goal of the CMU Planetary Rover project: to prototype an autonomous six-legged robot for planetary exploration.

1 Introduction

The goal of the CMU Planetary Rover project is to prototype an autonomous mobile robot for planetary exploration. The design is a six-legged walking robot with orthogonal legs and an overlapping gait [Bares, this proceedings]. To successfully walk over rugged terrain, the rover must combine perception, planning, and real-time control in an integrated system.

Recent research toward such an integrated system has concentrated on the task of single leg walking as a special case of six-legged walking. What distinguishes our work is the simplicity of the walking mechanism and the completeness and comprehensiveness of the controlling system. Other researchers use a single leg to isolate and study fundamental issues in balance and dynamics [9, 11]. Our reasons for using a simple mechanical system—it is stable both statically and dynamically—include testing algorithms with relative safety and ease, and coordinating design and development so that results from walking experiments influence more quickly the evolution of design of the six-legged walker. Previous efforts to create a coherent robotic walking system from component research results have generated significant advances, for example as reported in [5, 6, 7, 10] and many others. One of the reasons that these efforts have not proven entirely effective is the difficulty involved in developing each of the subsystems (e.g., locomotion, perception, planning, control); thus, each research effort has concentrated on a proper subset of the issues.

This paper describes the comprehensive system that we have implemented, and presents results from single leg walking experiments. We refer readers interested in the objectives and accomplishments of the overall project to [12].

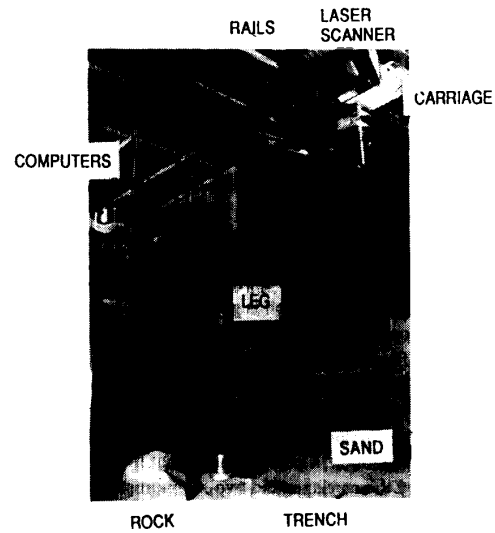

Figure 1: Single Leg Testbed

2 Single Leg Testbed

The single leg testbed contains over 40 tons of sand and a variety of obstacles in an 11×6m "sandbox" made from I-beams. It also includes a robot leg, sensors, and electronics (Fig. 1).

The robot leg is a prototype design not currently used on the six-legged Ambler (because integrated walking experiments revealed problems early). The leg has three joints: a revolute shoulder, a revolute elbow, and a prismatic vertical axis. Its horizontal length is 2.5m and vertical travel is 1.5m. In the shoulder and elbow axes, brushless DC servo motors couple to the joint axes by an 80:1 harmonic drive speed reducer and a 3:1 bevel gear. In the vertical axis, a brushless DC servo motor drives a 12:1 speed reducer and a lead screw.

The leg hangs from a carriage (or body) that rolls along rails. The leg "walks" by planting the foot on the ground and actuating the shoulder and elbow motors to push or pull the carriage.

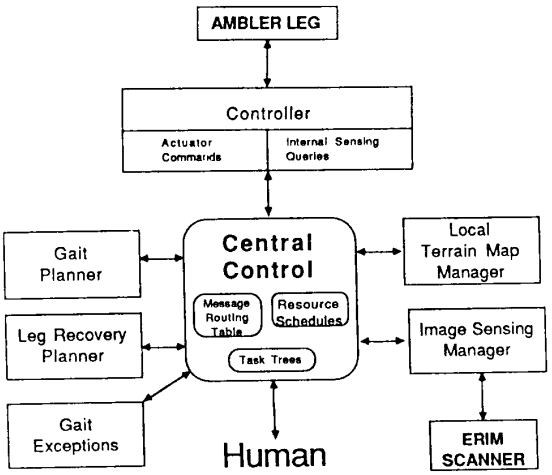

Figure 2: Modules of the Single Leg Walking System

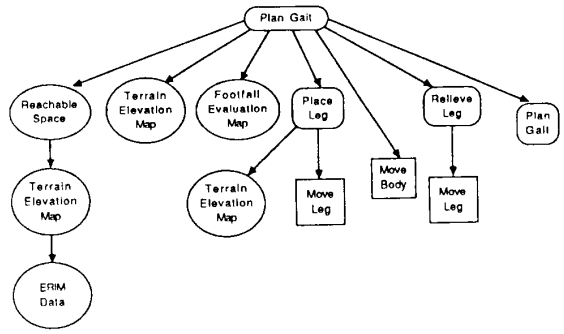

Figure 3: Task Tree
Bubbles represent query messages, rectangles represent command messages, and rounded rectangles represent goal messages.

Although the rails are I-beams, they exhibit significant deflection due to their length and the loads applied. This deflection provides compliance for the system (which is good), but changes as a function of the leg position (which is bad, because it makes accurate leg placement difficult).

Sensors on the leg include motor shaft encoders, limit switches, and a six-axis force/torque sensor mounted near the foot. Other sensors include a scanning laser rangefinder to sense the terrain in the sandbox, a potentiometer that measures the distance travelled by the carriage, and two inclinometers to measure rotation of the carriage with respect to gravity.

A control room houses three workstations connected by Ethernet, hardware to control the laser scanner, and a VME cage containing a real-time control system with its associated hardware (68020 single board computer, Ethernet controller, A/D converter, two 80186 motion control cards, and an interface card to connect them to the motor amplifiers).

3 Task Control Architecture

Simmons et al. designed the Task Control Architecture (TCA) to integrate sub-systems developed by different researchers into a complete robotic system [8]. TCA provides mechanisms to support message passing between distributed processes, hierarchical planning, plan execution, monitoring the environment, and exception handling. A system built using TCA consists of a number of task-specific processes, called *modules*, and a general *central control* process that directs the flow of communication between modules. The single leg walking system consists of six modules plus the central control (Fig. 2).

A prominent aspect of TCA is centralized control. Although researchers have recently advocated decentralized control for mobile robots, e.g [3], the TCA designers believe that centralized control has many advantages for supporting the above capabilities. First, it can more easily control multiple tasks by synchronizing them, allocating resources, and determining which tasks have priority. Second, centralized control makes the system more understandable and 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 in our applications; TCA can process a message in approximately 80msec, faster than either the perception or control systems operate.

Modules can connect with the central process in any order and at any time. If a module crashes, it can be restarted and reconnected without bringing down the rest of the system. Modules interact with TCA by calling utility functions, passing as arguments standard C or LISP data structures. When modules connect to the central control, they indicate which messages they can handle by registering the message name, a handler procedure, and the data format of the message. TCA also contains facilities for displaying arbitrary data structures; we use this to log all message traffic.

In TCA, planning and executing a task occurs by modules sending a series of messages to one another. For the single leg walker, after all modules have connected to central 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.

These messages are of various types—each message class has a different semantics and different effects. *Query* messages obtain information about the external or internal environment. *Goal* messages provide a mechanism for hierarchical planning. When a module issues a goal message, TCA creates a node and adds it as a child of the node associated with the handler that issued the message. These nodes form a hierarchical *task tree* that TCA uses to schedule planning and execution of tasks (Fig. 3). *Command* messages are requests for some action to be performed. Like goal messages, TCA adds them to the task tree; typically, they form the leaf nodes of the tree.

In addition to specifying parent/children relationships in the task tree, TCA provides mechanisms for temporally constraining the relationships between nodes in the tree. Essentially, the task trees plus the temporal constraints form TCA's representation of

plans. For example, one can specify that command A must be executed before command B is started, or that goal C cannot be planned until command B has finished. 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, i.e., a goal or command message has not necessarily been handled by the time control returns to the module issuing the message. This asynchronous control makes the overall system more reactive since the central process controls 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.

In addition to separating planning and execution, TCA uses a separate mechanism to perform 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 analyze the failing task tree and decide how to manipulate it to recover from the error, for instance, by killing part of the task tree, or by adding new nodes to the tree to patch the plan.

4 Real-Time Control

The control software runs on the real-time system under the vxWorks (TM) operating system, and communicates to the rest of the system through TCA. The controller performs three tasks: it executes leg and body movements and reports their positions; it communicates with the user (either a person or a process); and it handles asynchronous interrupts generated by the motion control cards. This section discusses only the first of these tasks.

Given a series of points in joint space (way-points), the controller actuates the leg motors so that the leg passes through each. It computes the time required for the slowest joint to move between successive way-points, and then scales the speeds of the other joints so that all arrive simultaneously at each way-point. To smooth the motion, the controller links way-points with constant velocity segments which in turn it connects by constant acceleration segments. If the user specifies the last path segment to be in transition mode, then the controller places the foot on the ground and loads it up to a specified force.

Once it achieves the desired load on the leg, the controller moves the body by actuating the shoulder and elbow joints to move at given velocities, not to given positions. It performs this at about 60Hz, which differs sufficiently from the natural frequency of the system to preclude resonance. The controller computes the joint velocities by applying the inverse Jacobian to the Cartesian body velocity, which is a clipped, linear function of the error between the current and commanded body positions. Due to non-linearities of the system, this causes overshoot of the joints from their nominal position given a perfectly linear system. This overshoot takes the form of stored strain energy. The controller dissipates the strain before unloading the leg, otherwise the foot could drag across the ground, possibly hitting an obstacle. If at any time the forces exerted on the foot decrease rapidly, indicating that the foot has lost contact with the ground, the controller halts.

5 Perception

The perception system consists of two major modules: the Imaging Sensor Manager (ISM), which senses the environment with a scanning laser rangefinder; and the Local Terrain Map Manager (LTM Manager), which constructs elevation maps from the rangefinder data. Readers will find a higher level account of the perception system in [1] and a lower level account in [4].

The ISM operates the imaging sensors, including initialization, status determination, data acquisition, calibration, aiming, and other operations. The ISM has been implemented and tested for the Erim and Perceptron scanning laser rangefinders. These sensors may be real (i.e., they acquire data in real-time from the physical sensor) or virtual (i.e., they acquire data from storage, not directly from the sensor). We have found virtual devices and virtual images to be useful for developing and testing code without hardware.

The LTM Manager constructs and maintains a local terrain map (LTM)[1] for locomotion guidance, short-range navigation, and sampling operations. An LTM describes the environment in the immediate vicinity of the Ambler, and may extend up to tens of meters on a side. An LTM is not, strictly speaking, a single map; in practice, it is a registered collection of maps, whose descriptions of the environment include geometric characteristics and material properties of the terrain. We have organized the software into three major submodules: one that builds the LTM, one that merges LTMs, and one that focuses attention on parts of the LTM that are closer to the vehicle (not described here).

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 [4] to transform the input raw range images into an output elevation map. In addition, the LTM Builder computes the uncertainty of the estimated elevations, analyzes elevation map patches as footholds, 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 implementation of the LTM Merger accepts as input the LTM L_k^0 constructed from range images $I_0, I_1, \ldots, I_k$, and the LTM L_{k+1}^{k+1} constructed from range image I_{k+1}. It generates as ouput the LTM L_{k+1}^0 by replacing overlapping elevation measurements with the maximum likelihood estimate of the elevation. The merging operation is necessary because maps created from a single frame of data do not, in general, contain enough information 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 can not possibly see obstacles either below or behind the vehicle. These obstacles pose real threats to the recovering leg, which must follow a trajectory that avoids collisions with them. Thus, the merging operation is necessary to create an LTM that

[1] To eliminate any possible confusion about the terminology, we mean to distinguish the LTM, which is a data structure, from the LTM Manager, which is a process.

provides a wider coverage of the terrain than is possible with a single frame of data.

6 Planning

The planning problems for single leg walking include deciding where to place the leg, how to move it there, and how far to move the carriage at each step. The planning system consists of two modules: the gait planner chooses footholds and body advances, and the leg-recovery planner identifies trajectories from the current leg position to the planned foothold.

The gait planner computes *cost maps* that indicate the "goodness" of each potential foothold on a 10cm grid. It assigns costs based on the following constraints: 1) flat terrain is preferable both for stability and for providing traction in moving the body; 2) the carriage can advance farther from some footholds than from others; 3) leg configurations in which horizontal links obstruct the scanner field of view are undesireable; 4) the leg can not reach areas outside its kinematic limits or ones surrounded by high obstacles (including other legs, for the six-leg case); 5) the leg can not reach terrain that is too high or too low (recall that the body height is fixed). The gait planner combines the cost maps using a weighted sum. It selects as the foothold the grid point with the lowest cost in the composite cost map. It plans the body move that is the minimum of 1) the largest advance possible from the chosen foothold, and 2) a user-defined threshold.[2]

Advantages of this constraint-based approach are that the planner does not have to commit *a priori* to which constraint is most important, and it is easy to add new constraints as relevant ones are identified. Although this approach could result in high computational costs, in practice the planner is fast relative to other computations.

While the gait planner decides where to set the foot, the leg-recovery planner determines the trajectory to that position without hitting obstacles. The leg-recovery planner uses a novel algorithm that finds time and power efficient moves through three-dimensional space while searching only a two-dimensional space, thus considerably increasing the efficiency of the planning.

The planner creates a configuration search space for the elbow and shoulder joints. It divides the space into a discrete grid approximately 0.1 radian wide, and fills the grid with obstacles. It grows terrain obstacles and other legs (for the six leg case) by the radius of the foot plus an uncertainty factor. The planner then searches this space using the A^* algorithm for the minimum cost path (weighting power and time by a user-specified ratio) to the goal, either by going around or over obstacles. It computes the power consumed to reach a grid cell from an adjacent cell as the sum of the power needed to move the elbow and shoulder joints to get to the cell, plus the power needed to raise the leg above the elevation associated with that cell. It computes the time required to get to a cell by keeping track of 1) all possible paths that the leg can take in reaching a particular grid cell, and 2) the maximum and minimum heights that the leg can reach at any particular cell, assuming that the leg lifts/lowers at full speed while moving horizontally. At the end of the search, the planner determines the final trajectory by selecting vertical moves that minimize the risk to the machine while maintaining the optimality of the path found.

[2]The threshold cannot exceed the maximum body advance of 3.9m. In order to take more steps per experiment, typically we use 1.5m.

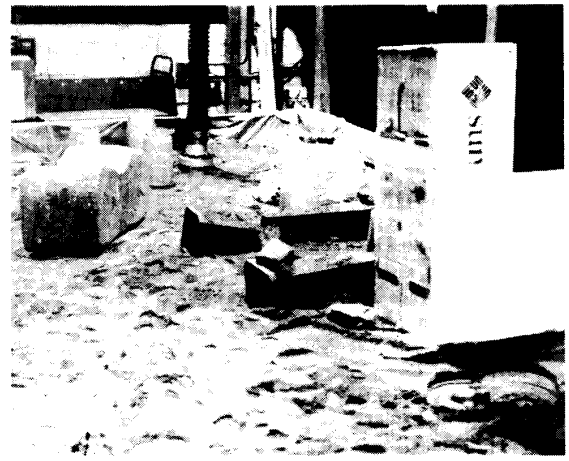

Figure 4: Obstacle Course
The obstacle course is 10m long and consists of a box (right) too tall for the leg to step over, a "steeplechase" arrangement of pylons (center) lying on the ground, two larger obstacles (left and upper center) separated by about 1m, and a dozen or so smaller obstacles.

7 Experiments

After the single leg testbed became operational in May 1989, we performed a series of tests on half a dozen obstacle courses. Fig. 4 shows one of them, and Fig. 5 illustrates a map of it constructed by the perception system. The courses combine obstacles that are too tall to step over, obstacles separated by distances smaller than the diameter of the foot, trenches too deep to step in, and sand hills with a variety of slopes. The criteria for a successful traversal are to reach the goal and to avoid contact with any obstacle.

For each trial, first we activate the integrated system shown in Fig. 2. Then we issue a command to walk forward to the end of the testbed (about 10m). After this, the integrated system is entirely autonomous as it plans and executes the walking cycle of moving the leg and propelling the body along the rails.

The integrated system successfully negotiated all of the obstacle courses. It traversed the course in Fig. 4 seven consecutive times during one afternoon. It traversed comparable courses more than thirty times. In some of the trials, the system also walked backward, using the map built by the perception system while walking forward. This was surprisingly easy; the system could always follow a successful forward traverse by a successful backward traverse.

Not all forward walking trials were successful. Failures include stepping on obstacles, and grazing them with the leg. To diagnose these failures is challenging. To illustrate the difficulty, suppose that the leg strikes an obstacle. What went wrong? The fault could be in any of the subsystems. The perception system may have computed an inaccurate map, incorrectly determining

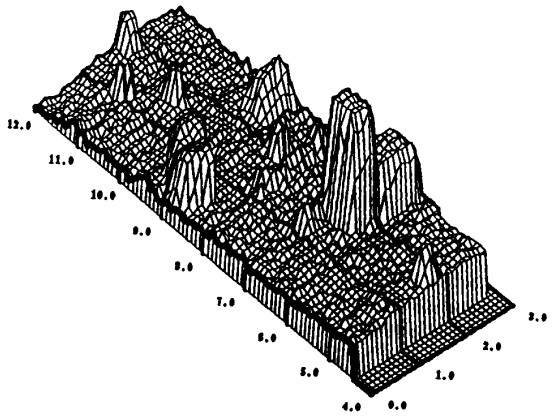

Figure 5: Elevation Map of Obstacle Course
The perception system built this elevation map from five range images acquired at different positions. The map resolution is 10cm, $0 \leq X \leq 3$m, and $4 \leq Y \leq 12$m.

the obstacle location. The planning system may have chosen a poor foothold or an erroneous leg trajectory. The controller may have executed poorly a perfect plan. Or any combination of the former factors could cause the fault.

During the walking trials, we identified the the following combination of factors to be responsible for the largest number of failures: 1) the perception system sometimes underestimates the possible error in the perceived location of an obstacle, 2) the controller does not compensate for the change in leg positioning accuracy as a function of body position,[3] 3) the gait planner, working without models of the above deficiencies, is sometimes more eager to advance to the goal (i.e., plan longer steps) than to steer clear of obstacles (i.e., plan steps that sacrifice body advance for obstacle avoidance).

We find the average walking velocity to be on the order of one meter per minute. Since we have not dedicated much effort to optimizing either hardware or software, this statistic may not be particularly meaningful. One version of the integrated system achieves nearly continuous walking (Fig. 6). It concurrently executes one step while planning the next step(s), exploiting the temporal constraint mechanisms of TCA.

8 Discussion

We have described an integrated system that combines advanced techniques in perception, control, and planning into a comprehensive whole. Experiments show that the system can perform capable and fairly reliable single leg walking on rough terrain.

[3]Due to variable compliance of the rails, the errors in leg positioning grow with the distance along the rails from the body to the closest anchoring wall. Thus, the errors are larger in the middle of the testbed than at the ends.

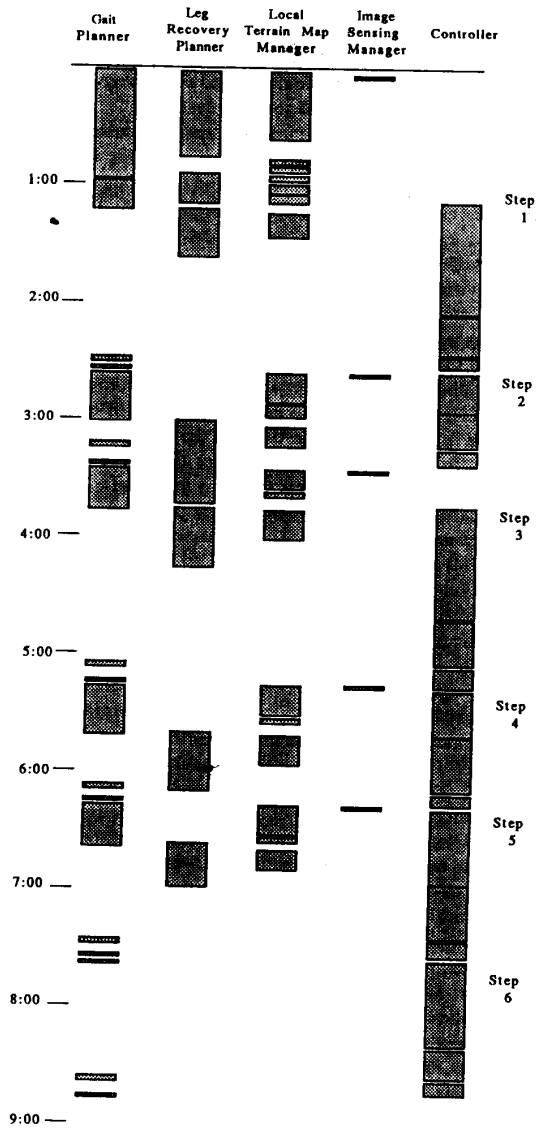

Figure 6: Nearly Continuous Walking
The figure shows when processes are active during one obstacle course traversal using concurrent planning and execution. Numbers in the left column indicate time. The controller is idle between Steps 2 and 3 because the leg recovery planner takes longer than usual to plan a complex trajectory around (not over) the large box shown in Fig. 4.

The major impetus for the single leg walking research program was to gain experience for six leg walking. In that regard, the project is quite successful. We have gained 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.

Future work will concentrate on applying our experiences to an integrated system for six leg walking. This will require significantly extending the controller and the planners, but only superficial changes to the perception system and the TCA.

One topic that we anticipate will be an issue in the future is calibration of the leg and scanner frames. For the six-legged walker, we may augment our current model-based calibration with a direct, empirical method that, for example, locates the leg in many images and uses a connectionist approach to build an inverse kinematics table [2].

Another topic for future work is better performance in situations that conflate errors in perception, errors in control, and optimism in planning (cf. the cause of the failures cited in Section 7). To better understand situations that cause failures we need more powerful debuggers (possibly graphical). To achieve more reliable performance we must develop robust error recovery mechanisms.

In this paper we have concentrated on the integrated system. We conclude by discussing briefly the process of system integration. Perhaps the most important lesson that we learned is that integration is a contact sport; it cannot succeed without significant "hands on" participation by researchers with a broad range of experience and expertise. While this may be self-evident, it is by no means easy to accomplish.

We have adopted several approaches to facilitate the integration effort. First, we conduct regular weekly meetings to identify the semantics for all interfaces between modules, answering questions about the type and units of information communicated. The message-passing conventions of TCA promote this because they force us to detail the interfaces. Second, we employ Unix manual facilities to document the I/O behavior of a module or message. We find this to be a useful lowest common denominator that all programmers and users can use to advantage and that is not so difficult to maintain. Third, we insist that software meet standards of internal documentation (comments for modules, files, and functions), follow conventions for naming (source files, include files, defines, type definitions, and variables), and obey a variety of other guidelines. We find this to be of great value both in debugging and development. Finally, we standardize our software structure in order to make common code more accessible. This includes enforcing consistent directory structures and naming conventions.

These and other approaches have significantly assisted us in integrating the single leg walking system. We intend to continue these practices as we combine our perception, planning, and control techniques into a comprehensive system for six-legged walking.

Acknowlegements

We would like to acknowledge contributions by all the members of the Planetary Rover project, and thank especially P. Balakumar, L. Chrisman, C. Fedor, M. Hebert, G. Roston, and D. Wettergreen for their assistance in integrating and testing the single leg walker.

This research was sponsored by NASA under Grant 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 US Government.

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*, pages 18–26, June 1989.

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

[3] 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.

[4] M. Hebert, E. Krotkov, and T. Kanade. A Perception System for a Planetary Explorer. In *Proc. IEEE Conf. on Decision and Control*, pages 1151–1156, Tampa, Florida, December 1989.

[5] S. Hirose. A Study of Design and Control of a Quadruped Walking Vehicle. *International Journal of Robotics Research*, 3(2):113–133, Summer 1984.

[6] Y. Ishino, T. Naruse, T. Sawano, and N. Honma. Walking Robot for Underwater Construction. In *Proc. Intl. Conf. Advanced Robotics*, pages 107–114, 1983.

[7] C. A. Klein, K. W. Olson, and D. R. Pugh. Use of Force and Attitude Sensors for Locomotion of a Legged Vehicle over Irregular Terrain. *International Journal of Robotics Research*, 2(2):3–13, 1983.

[8] 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.

[9] M. H. Raibert. *Legged Robots That Balance*. MIT Press, Cambridge, Massachusetts, 1986.

[10] S. Song and K. Waldron. *Machines that Walk: The Adaptive Suspension Vehicle*. MIT Press, Cambridge, Massachusetts, 1988.

[11] M. Sznaier and M. J. Damborg. An Adaptive Controller for a One-Legged Mobile Robot. *IEEE Transactions on Robotics and Automation*, 5(2):253–259, April 1989.

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