1989 Year End Report
Autonomous Planetary Rover at Carnegie Mellon

William Whittaker   Takeo Kanade   Tom Mitchell
Principal Investigators

CMU-RI-TR-90-04 2

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

February 1990

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 United States Government.

Contents

List of Figures

Abstract

This report describes progress in research on an autonomous robot for planetary exploration performed during 1989 at the Robotics Institute, Carnegie Mellon University. The report begins with an introduction, summary of achievements, and lists of personnel and publications. It also includes several papers resulting from the research.

The research program includes a broad agenda in the development of an autonomous mobile robot. In the year covered by this report, we addressed two key topics:

Six-Legged Walking Robot — To overcome shortcomings exhibited by existing wheeled and walking robot mechanisms, we configured the *Ambler* as a walking robot. The fundamental advantage of the Ambler configuration—which has implications for efficiency, mechanism modeling, and control simplicity—is that actuators for body support are independent of those for propulsion; a subset of the planar joints propel the body, and the vertical actuators support and level the body over terrain. During 1989 we configured, designed, and constructed the Ambler. In addition, we developed models of its dynamics, and studied leveling control.

Integrated Single Leg Walking — We implemented and tested an integrated system capable of walking with a single leg over rugged terrain. A prototype of an Ambler leg is suspended below a carriage that slides along rails. To walk, the system uses a laser scanner to find a clear, flat 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. In addition, we conducted preliminary experiments with concurrent planning and execution, and developed a leg recovery planner that generates time and power efficient 3D trajectories using 2D search.

Mobile Manipulation with Hero Robot — Indoor mobile manipulator tasks include collecting cups from the lab floor, retrieving printer output, and recharging when its battery gets low. The robot monitors its environment, and handles exceptional conditions in a robust fashion. For example, it uses vision to track the appearance and disappearance of cups, uses on-board sonars to detect imminent collisions, and monitors battery level periodically.

This research is primarily sponsored by the National Aeronautics and Space Administration. Portions of this research are also supported by the National Science Foundation and the Defense Advanced Research Projects Agency.

Introduction

This report reviews progress during 1989 at the Robotics Institute, Carnegie Mellon University, on research sponsored by NASA titled "Autonomous Planetary Rover." This report begins with an overview and a summary of achievements. It then lists the members of the research group supported by, or directly related to the contract, and their publications. Finally, it includes three detailed papers representative of specific areas of research.

Overview

The CMU program to develop an Earth-based prototype of an autonomous planetary rover is organized around three teams that are developing the locomotion, perception, and planning subsystems. A joint task is to integrate the three subsystems into an experimental robot system. We will use this system for evaluating, demonstrating, and validating the concepts and technologies developed in the program.

The technical objectives of the research include the following:

- To develop and demonstrate an autonomous Earth-based mobile robot that can survive, explore, and sample in rugged, natural terrains analogous to those of Mars.
- To provide detailed, local representations and broad, 3-D descriptions of rugged, unknown terrain by exploiting diverse sensors and data sources.
- To demonstrate robot autonomy through a planning and task control architecture that incorporates robot goals, intentions, actions, exceptions, and safeguards.

Accomplishments

This section describes key accomplishments of the project research from January 1989 to December 1989. We present these accomplishments in three parts: the first includes all activities related to construction of the Ambler[1]; the second includes those activities related to integrated walking; the third covers other activities.

[1]An acronym for Autonomous MoBiLe Exploration Robot.

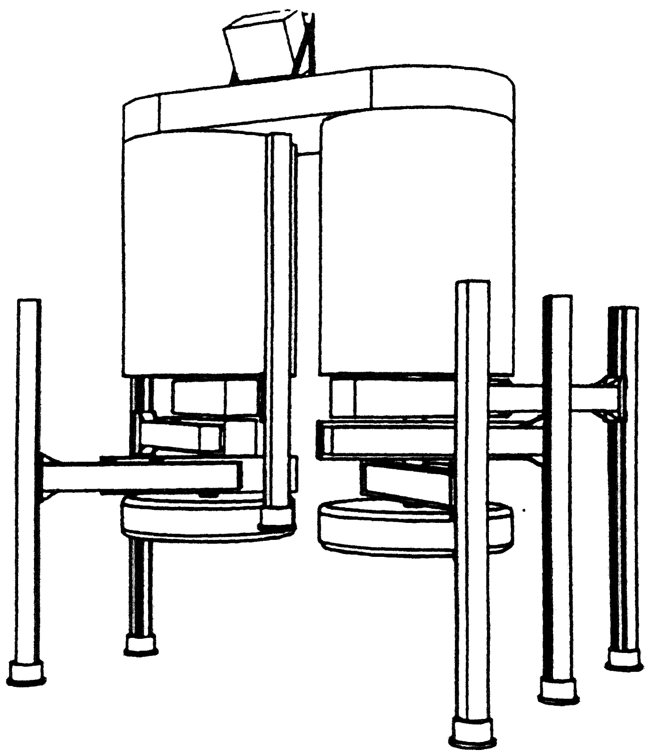


Figure 1: Ambler configuration

Ambler

A major accomplishment of 1989 was to reconfigure, design, and build the six-legged walking machine. Using all six legs, we demonstrated body motion (lift, advance) and leg recovery (circulation between stacks). These first steps of the Ambler are a significant project milestone.

Configuration — We reconfigured the earlier Ambler designs to have two stacks, with six circulating legs (Figure 1). Each leg is a rotary-prismatic-prismatic orthogonal leg. The configuration enables level body motion, a circulating gait, conservatively stable gaits, high mobility, and many sampling deployment options.

Design — We detailed the relative leg link scale, duplicated components when possible, and augmented our efforts with results from a prototype leg testing program. Also in the design process we identified worst cases for structural loads, drivetrain loads, power, and link speeds. We made a number of key design decisions: to use aluminum as our primary material; to equip all axes with spur gear drives; to outfit the prismatic links with linear bearings; to incorporate separate slipring units in each leg; to have shoulders ride not on each other but on a central shaft; and to construct the superstructure from aluminum instead of composites.

Fabrication — As we completed designs, we began fabrication but continued to alter them slightly to simplify assembly. An intensive effort to put all the pieces together culminated in a complete vehicle in December.

Electronics and Sensing — We designed and implemented a variety of electronic devices to link computing, actuation, and the physical mechanism. We established signal paths to provide machine status—including drive train, positions, and forces—to computing. To reduce the number of cables required, we designed and built a high-speed multiplexor that provides real-time data transmission of analog and digital signals. We built a tether to carry all signals to and from the machine. The tether is 46m of protective fabric sheathing that contains 130 shielded twisted pairs, 30 coaxial cables, and power cables. To ensure safe operation of the machine, we implemented a three state finite state machine safety circuit that allows manual control, computer control, and provides graceful termination upon certain conditions.

Real-Time Controller — We have developed a real-time controller based on VME hardware and the VxWorks operating system. Multiple processors synchronize input/output and motion control. Creonics motion control cards receive encoder feedback and amplifier status signals, and transmit motor command and amplifier control signals. Digital boards route signals for brake control, the safety circuit interface, and force sensor control. Up to 64 A/D converter channels read signals from the force sensors, absolute encoders, and inclinometers.

Mechanism Modeling — We formulated two models for the Ambler mechanism: a comprehensive model and a planar model. The comprehensive model incorporates non-conservative foot-soil interactions in a full non-linear dynamic formulation. We employed it for performance evaluations such as assessment of power consumption, potential for tipover, and foot slippage, and continue to use it to develop body leveling control algorithms. We used the second, planar model to evaluate mechanism designs and to investigate joint driving configurations for propulsion.

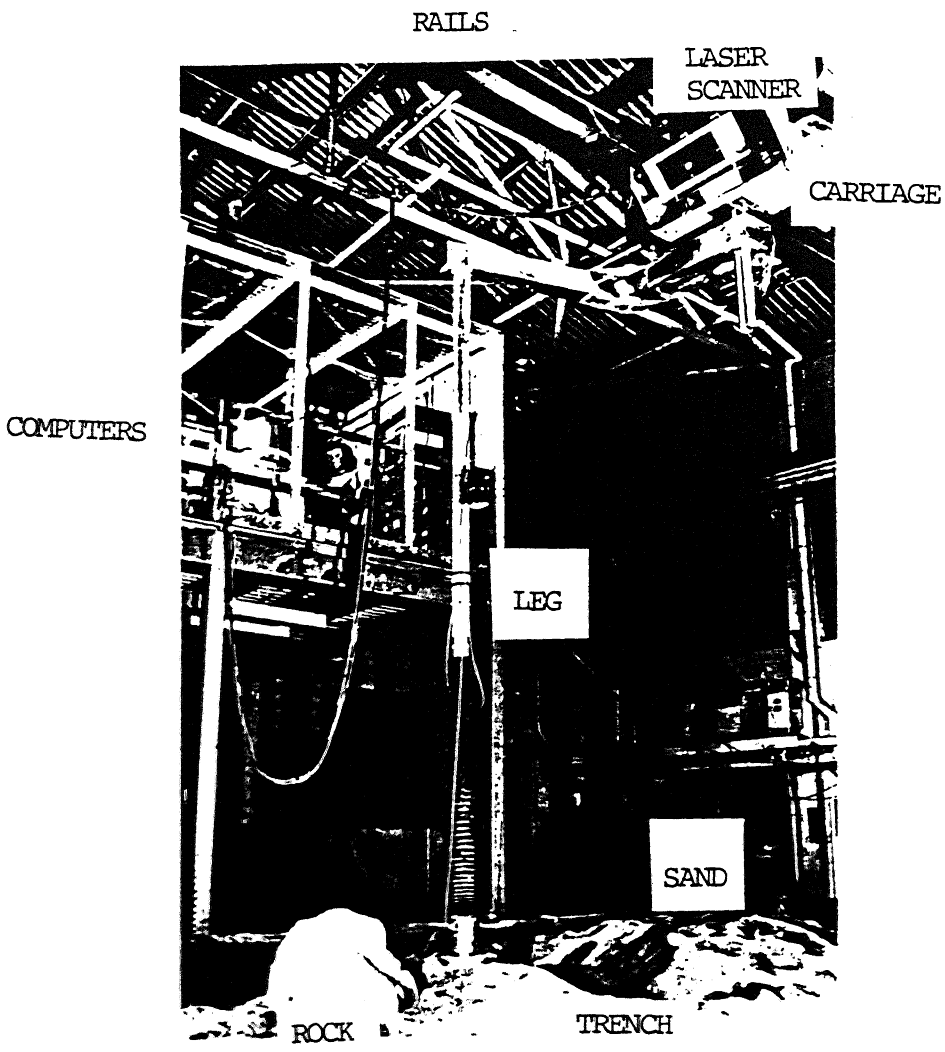


Figure 2: Single leg testbed

Integrated Walking

We implemented and tested an integrated system capable of walking with a single leg over rugged terrain. A prototype of the Ambler leg is suspended below a carriage that slides along rails (Figure 2). To walk, the system uses a laser scanner to find a foothold, computes an efficient trajectory to 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, briefly described below) and a central control process that directs the flow of communication between processes. With this system we experimented with extensions to support concurrency and error recovery.

Task Control Architecture — We implemented the Task Control Architecture (TCA) and used it to integrate the various components of the walking system. TCA provides a number of important 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 the environment, and exception handling. Using TCA the system consists of a number of task-specific processes and a central control process that directs the flow of communication between modules.

Real-Time Controller — We implemented a real-time control system for the single leg. This system, which runs under the VxWorks operating system, communicates via the TCA, moves the leg and carriage and reports their positions, and handles asynchronous interrupts generated by the Creonics motion control boards.

Perception using Elevation Maps — We implemented a perception system to build elevation maps from sequences of range images. In addition to the elevation, the system computes the elevation uncertainty, local slope, visibility, and foothold goodness (measure of terrain flatness in a foot-size neighborhood). The system executes approximately 20×10^6 instructions to build a 400 point map. In parallel, we developed techniques for matching long sequences of range images and for merging them stochastically into a composite map (Figure 3), and conducted experiments in updating satellite maps from local data.

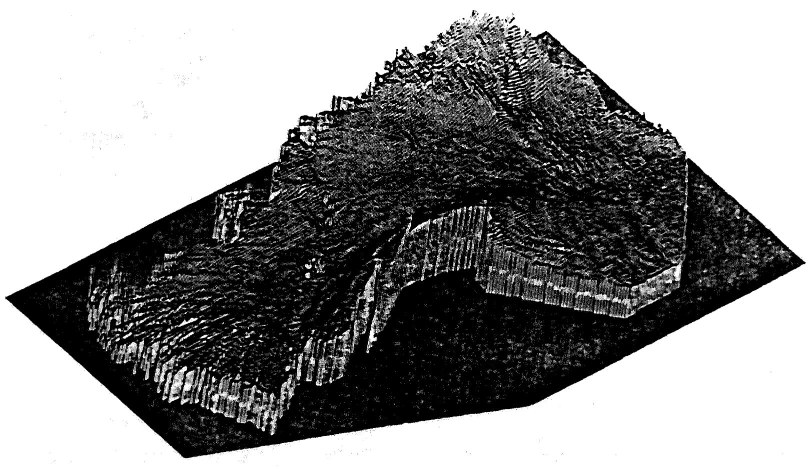
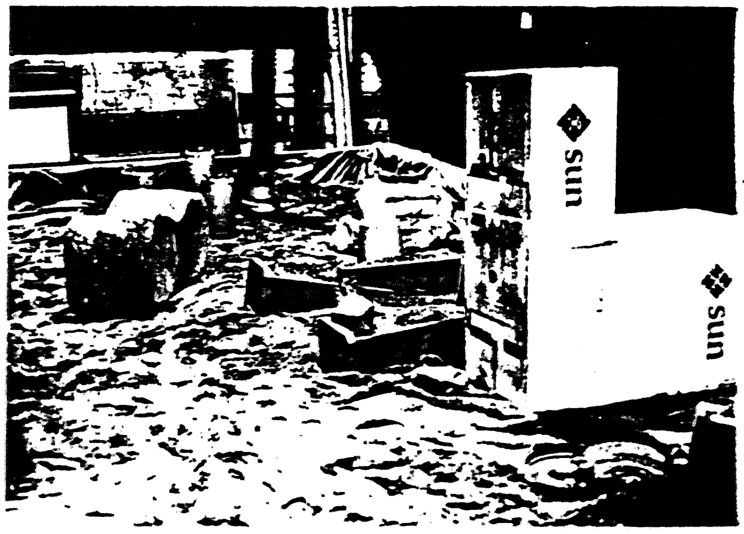


Figure 3: Composite elevation map

This map was built by matching 125 Erim range images acquired by the Autonomous Land Vehicle as it traversed a 40m path (right to left), including a 30 degree left turn, at an outdoor site in Colorado. The matching between consecutive range images was performed by first matching features to obtain an initial estimate of the displacement, and then using that estimate to seed an iterative minimization procedure.

Planning — We developed and implemented two planning modules: the Gait Planner and the Leg Recovery Planner. The Gait Planner determines leg sequencing, body trajectory, and foothold location. The Leg Recovery Planner generates trajectories that avoid obstacles and minimize an objective function of time and energy. It plans three-dimensional trajectories while searching a two-dimensional space, which reduces computation time substantially.

Single Leg Walking Experiments — We conducted a series of experiments and demonstrations using the Single Leg Testbed. For the first stage of testing, we levelled the terrain and did not alter it between runs. We began with a minimal set of processes, and incrementally added processes. For the second stage of testing, we executed the same processes, and walked over different terrains. We began with level ground, and graduated to successively more difficult terrain. Figure 4 shows an obstacle course that the integrated system traversed more than 30 times, and the elevation map built by the perception system.



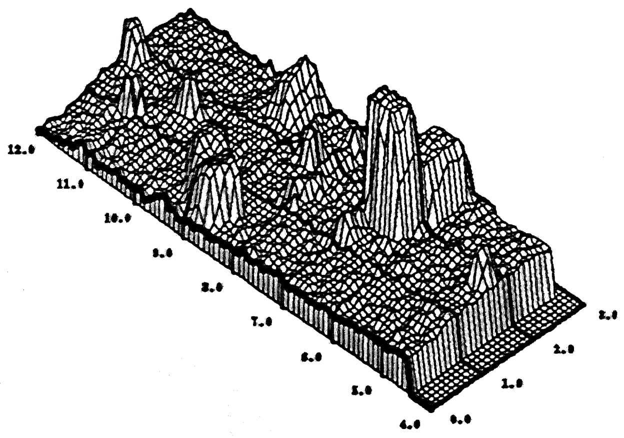


Figure 4: Obstacle course

The obstacle course consists of a small obstacle (upside down basket, lower right), 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.

The perception system built this elevation map from approximately five range images acquired at different positions. The labels indicate metric units in the global reference frame, where $0 \leq X \leq 3$ and $4 \leq Y \leq 12$. The map resolution is 10cm.

Other Activities

Mobile Manipulator Testbed — At the Mobile Manipulator Testbed we developed and tested advanced TCA features such as monitors, task tree management, temporal constraints, exception handling, and resource allocation. Using these features, a Hero robot successfully demonstrated several tasks: cup collection, retrieval of printer output, delivering objects to workstations, recharging its battery, using on-board reflexive procedures to detect and react to imminent collisions. We also achieved substantial progress toward a number of other capabilities, including navigation based on sonar, learning to approach and recognize objects, and learning stimulus–response action rules.

Simulator — We developed a simulation system on a Titan supercomputer (Figure 5). Capabilities include three-dimensional solid and kinematic models of the six-legged Ambler, generation and display of synthetic terrain (rocks, hills, craters, etc), and acquisition of synthetic range images of terrain.

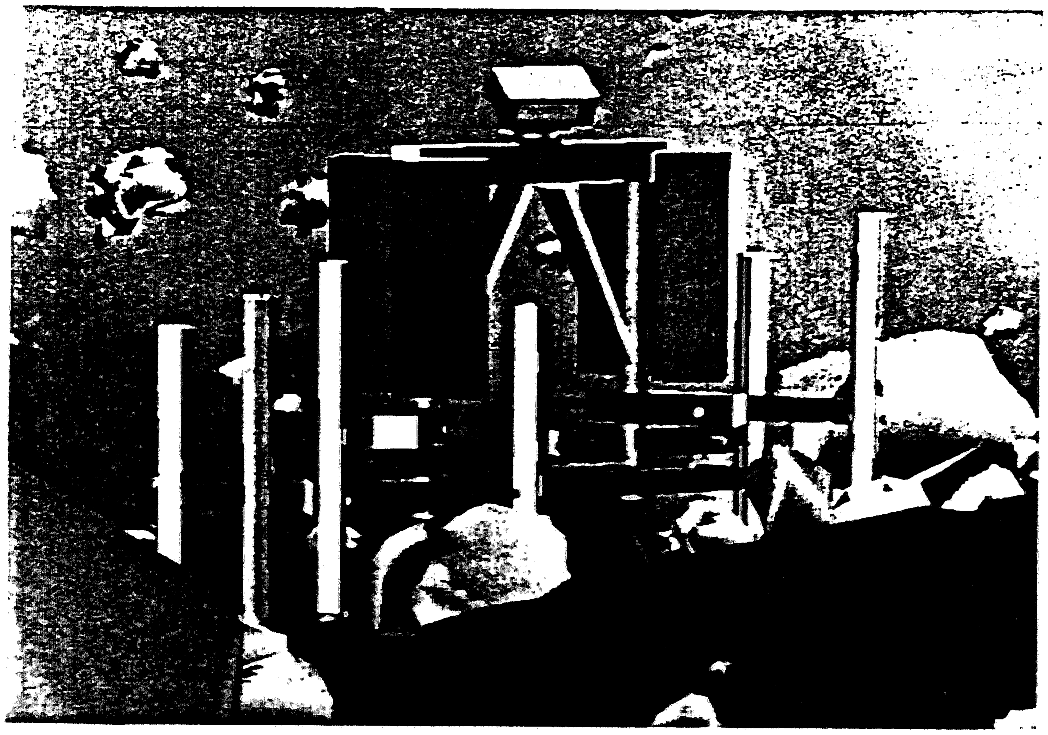


Figure 5: Simulated Ambler on synthetic terrain

Personnel

The following personnel were directly supported by the project, or performed related and contributing research in 1989:

Faculty: Martial Hebert, Katsushi Ikeuchi, Takeo Kanade, Chelva Kumar, Eric Krotkov, Tom Mitchell, Reid Simmons, Chuck Thorpe, William Whittaker.

Staff: Brian Albrecht, Purushothaman Balakumar, Gary Baun, Mike Blackwell, Kevin Dowling, Christopher Fedor, Kerien Fitzpatrick, Joe Hirsch, Regis Hoffman, Ralph Hyre, Jim Martin, Clark McDonald, Jim Moody, Dave Pahnos, Henning Pangels, Gerry Roston, Kevin Ryan, Jay West, David Wettergreen.

Visiting Scientists: Jim Blythe, Claude Caillas, Herve Delinguette, Bao Xin Wu.

Graduate Students: John Bares, Lonnie Chrisman, Richard Goodwin, Goang Tay Hsu, In So Kweon, Long-Ji Lin, David Manko, Peter Nagy, Ming Tan.

Undergraduate Students: Steve Baier, Jonathan Burroughs, John Greer, Nathan Harding, Chris Ivory, Susan Kane, Nina Koros, Terry Lim, Eric Miles, Sundip Patel, Naeem Shareef, Hans Thomas, Rob Wolpov, Kurt Zimmerman.

References

[1] J. Bares. Orthogonal Legged Walkers for Autonomous Navigation of Rugged Terrain. December 1988. Ph.D. Thesis Proposal, Department of Civil Engineering, Carnegie Mellon University.

[2] 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.

[3] J. Bares and W. Whittaker. Configuration of an Autonomous Robot for Mars Exploration. In *Proc. World Robotics Conf. on Robotics Research: The Next Five Years and Beyond*, pages 37–52, Gaithersburg, Maryland, May 1989.

[4] J. Bares and W. Whittaker. Orthogonal Legged Walking Robot. December 1988. United States Patent Application.

[5] J. Bares and W. Whittaker. Orthogonal Legged Walking Robot CIP. May 1989. United States Patent Application.

[6] C. Caillas. Autonomous robot using infrared thermal camera to discriminate objects in outdoor scene. In *Proc. SPIE Conf. Applications of Artificial Intelligence VIII*, Orlando, Florida, To appear, April 1990.

[7] C. Caillas. Imaging Sensing to Identify Footfall Positions for a Legged Robot. In *Second Workshop on Military Robotic Applications*, Royal Military College of Canada and Civil Institute of Environmental Medicine, Kingston, Ontario, August 1989.

[8] 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*, pages 244–250, Tsukuba, Japan, September 1989.

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

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

[11] Planetary Rover Group. Experiments in Perception, Planning, and Control for the Carnegie Mellon Mars Rover. December 1988. Robotics Institute, Carnegie Mellon University, unpublished working document.

[12] Planetary Rover Group. Integration of Perception, Planning, and Control in the Carnegie Mellon Mars Rover. December 1988. Robotics Institute, Carnegie Mellon University, unpublished working document.

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

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

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

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

[17] 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.

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

[19] 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 Conf. on Space Telerobotics*, Jet Propulsion Laboratory, Pasadena, California, January 1989.

[20] E. Krotkov, G. Roston, and R. Simmons. *Integrated System for Single Leg Walking.* Technical Report PRWP-89-3, Robotics Institute, Carnegie Mellon University, 1989.

[21] 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.

[22] I. S. Kweon, M. Hebert, and T. Kanade. Perception for Rugged Terrain. In *Proc. SPIE Mobile Robots III Conf.*, Society of Photo-Optical Instrumentation Engineers, Cambridge, Massachusetts, November 1988.

[23] I. S. Kweon, M. Hebert, and T. Kanade. Sensor Fusion of Range and Reflectance Data for Outdoor Scene Analysis. In *Proc. NASA Workshop on Space Operations, Automation, and Robotics, NASA Conf. Publication 3019*, pages 373–382, Dayton, Ohio, July 1988.

[24] L.-J. Lin, T. 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.

[25] 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.

[26] S. Mahalingam. *Terrain Adaptive Gaits for the Ambler.* Master's thesis, Department of Mechanical Engineering, University of North Carolina, 1988.

[27] S. Mahalingam and W. Whittaker. Terrain Adaptive Gaits for Walkers with Completely Overlapping Leg Workspaces. In *Proc. Robots 13*, pages 1–14, Gaithersburg, Maryland, May 1989.

[28] D. Manko. Models of Legged Locomotion on Natural Terrain. June 1988. Ph.D. Thesis Proposal, Department of Civil Engineering, Carnegie Mellon University.

[29] D. Manko and W. Whittaker. Inverse Dynamic Models used for Force Control of Compliant Closed-Chain Mechanisms. In *Proc. ASME Design and Automation Conf.*, pages 61–66, Montreal, August 1989.

[30] D. Manko and W. Whittaker. Planar Abstraction of a Prototype Walking Machine. In *Proc. 20th Pittsburgh Conf. on Modeling and Simulation*, pages 1817–1823, Pittsburgh, Pennsylvania, May 1989.

[31] P. Nagy. Attitude and Altitude Control for a Novel 6-Legged Robot. January 1989. Research Prospectus, Mechanical Engineering Department, Carnegie Mellon University.

[32] P. Nagy. Coordinated Compliant Motion Control for Multi-Legged Walking Vehicles on Rugged Terrain. January 1990. Ph.D. Thesis Proposal, Department of Mechanical Engineering, Carnegie Mellon University.

[33] P. Nagy and W. Whittaker. Experimental Program for the CMU Mars Rover Single Leg Testbed. In *Proc. 20th Pittsburgh Conf. on Modeling and Simulation*, pages 1825–1829, Pittsburgh, Pennsylvania, May 1989.

[34] P. Nagy and W. Whittaker. Motion Control for a Novel Legged Robot. In *Proc. IEEE Symp. Intelligent Control*, pages 2–7, Albany, New York, September 1989.

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

[36] R. Simmons and T. Mitchell. A Task Control Architecture for Mobile Robots. In *Proc. AAAI Spring Symposium*, Stanford, California, March 1989.

[37] C. Thorpe. Outdoor Visual Navigation for Autonomous Robots. In *Proc. Intl. Conf. on Intelligent Autonomous Systems*, pages 530–544, Amsterdam, Netherlands, December 1989.

INVERSE DYNAMIC MODELS USED FOR FORCE CONTROL OF COMPLIANT, CLOSED-CHAIN MECHANISMS

D. J. Manko and W. L. Whittaker
Field Robotics Center
Carnegie Mellon University
Schenley Park, Pennsylvania

BSTRACT

A general inverse dynamic model is presented that is applicable to iechanisms incorporating member, joint and base compliance. revious approaches for defining inverse dynamic models of compliant iechanisms have been approximations or limited to simple mechanism eometries and open-chain mechanisms. Hence, the motivation for a iore general approach. Inverse dynamic equations for compliant iechanisms modeled with and without constraint equations are shown be solvable sets of differential/algebraic equations (DAE's); relevant iaracteristics and solutions of DAE systems are discussed. An nportant application for inverse dynamic models of compliant echanisms is model-based force control of closed-chain mechanisms. he formulation and solution procedures discussed in this paper have en successfully applied to model legged locomotion on natural rrain.

INTRODUCTION

An inverse dynamic model of a mechanism is an application of the uations of motion for a system where joint trajectories are defined, d the actuator forces and interaction forces required to produce these otions are calculated. (In contrast, a forward dynamic model is used calculate mechanism motions in response to a set of applied forces.) ilculations of these forces are useful for sizing members and tuators during the mechanism design phase. Additionally, mputationally fast versions of the model can be incorporated into odel-based control schemes. An important application is model-sed force control of closed-chain mechanisms where the constraint or eraction forces (e.g., foot forces for a walking machine) are sampled antities. The inverse dynamic model provides estimates of these eraction forces, which may be used as control set points.

The following section of this paper discusses existing methods for fining inverse dynamic models of compliant mechanisms. A general mulation of inverse dynamic models for compliant mechanisms odeled with and without constraint equations is described in Section The resulting inverse dynamic equations are shown to be sets of ferential/algebraic equations (DAE's) after substitution of specified nt motions. Relevant characteristics and solution procedures for E systems are considered in Section 4. The system index (which relates with solution difficulty) of compliant mechanism, inverse namic models are defined in Section 5 of this paper; identifying the tem index assures that stable and accurate numerical solutions can calculated by the methods described in Section 4.

2. BACKGROUND

An inverse dynamic model is obtained by substitution of specified coordinate trajectories into the equations of motion developed for the system. For non-compliant mechanisms, all coordinate trajectories can be specified because rigid body motions completely define the system kinematics. Alternately, inverse dynamic models for compliant mechanisms are complicated by the fact that trajectories cannot be specified *a priori* for the deflection variables which model the system compliance. In general, deflection variables are not directly controlled (i.e., applied forces equal to zero) so it is inappropriate to presume that motions of a deflection variable can be specified. Responses of the deflection variables are unknown quantities that are calculated along with the required actuator forces to produce the specified joint motions.

An approximate approach for formulating inverse dynamic models of compliant mechanisms is to impose inertial loads on a flexible, static mechanism model for deflection calculations [Dado 86]. The inertial loads used in the analysis are obtained from the kinematics of the mechanism considering it to be ideally rigid. Although this method is relatively simple to implement, it does not account for the coupling between joint and deflection variables which limits its application to relatively slow moving mechanisms. Also, time dependent deflection response is not considered.

The inverse dynamic model for a compliant, 2 dof cylindrical arm [Forrest-Barlach 87] was obtained by substitution of dynamic equations for the deflection variables (where the deflection variables are defined as functions of the joint variables) into the remaining dynamic equations corresponding to the joint variables. The resulting joint variable equations are fourth order differential equations requiring joint trajectory planning of both jerk and jerk rate. The approach of eliminating deflection variables from the equations of motion can only be accomplished for relatively simple systems.

The finite element method was used to discretize the equations of motion for open-chain mechanisms having structural flexibility [Bayo 88]. An inverse dynamic model was obtained by specifying trajectories for a subset of the deflection variables which decouples the equations for an individual link. Joint torques required to produce a desired end effector motion were calculated using an iterative solution scheme. Specification of trajectories for a subset of the deflection variables is not appropriate for all compliant systems (e.g., a mechanism on a compliant base).

3. COMPLIANT MECHANISM INVERSE DYNAMIC EQUATIONS

3.1 Mechanisms Modeled Without Constraint Equations

The equations of motion (before substitution of specified joint trajectories) for open-chain and closed-chain mechanisms (rigid or compliant), where motions are defined in terms of an independent set of generalized coordinates (q), have the following form,

$$M(q)\ddot{q} = f(q,\dot{q},t)$$

Substitution of joint trajectories into the equations of motion for a non-compliant mechanism results in straightforward evaluation of joint forces. Alternately, substitution of trajectories into the equations of motion for a compliant mechanism and conversion to standard form (discussed in Section 4) results in a coupled set of first order differential and algebraic equations (DAE's).

The planar manipulator on a vertically compliant base shown in Figure 1 is used to illustrate formulation of an inverse dynamic model for compliant mechanisms modeled without constraint equations. The planar mechanism has the dynamic equations shown in Figure 2 after substitution of joint trajectories (all joints must be powered to produce a controlled motion) and conversion to state space form. Gaussian elimination of derivative terms from the joint variable equations results in the standard form equations shown in Figure 3 where the differential equations correspond to the vertical deflection variable and the algebraic equations correspond to the joint variables. The joint variable dynamic equations always reduce to algebraic equations while the deflection variable equations remain as differential equations for any compliant mechanism modeled without constraint equations.

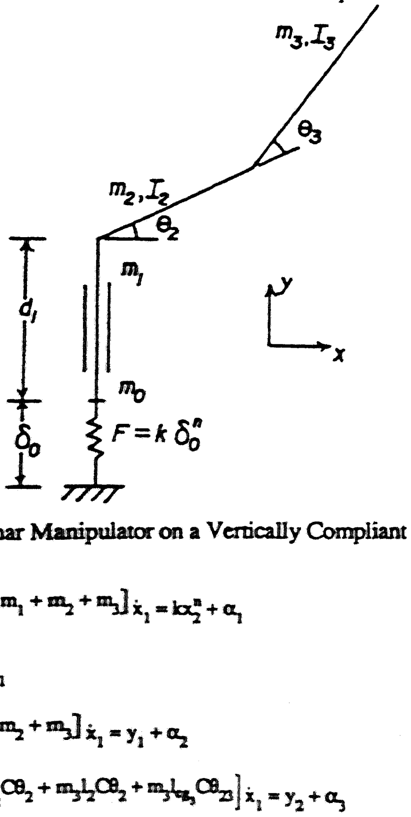


Fig. 1 Planar Manipulator on a Vertically Compliant Base

$$[m_0 + m_1 + m_2 + m_3]\,\dot{x}_1 = kx_2^n + \alpha_1$$

$$\dot{x}_2 = x_1$$

$$[m_1 + m_2 + m_3]\,\dot{x}_1 = y_1 + \alpha_2$$

$$\left[m_2 l_{cg_2} C\theta_2 + m_3 l_2 C\theta_2 + m_3 l_{cg_3} C\theta_{23}\right]\dot{x}_1 = y_2 + \alpha_3$$

$$m_3 l_{cg_3} C\theta_{23}\dot{x}_1 = y_3 + \alpha_4$$

where: $x_1 = \dot{\delta}_0, x_2 = \delta_0, y_1 = F_1, y_2 = \tau_2, y_3 = \tau_3$

$\alpha_1, \alpha_2, \alpha_3, \alpha_4$ - constants

Fig. 2 Dynamic Equations in State Space Form for a Planar Manipulator on a Vertically Compliant Base

$$\left.\begin{aligned} &[m_0 + m_1 + m_2 + m_3]\,\dot{x}_1 = kx_2^n + \alpha_1^* \\ &\dot{x}_2 = x_1 \end{aligned}\right\} F_1(x,\dot{x},y,t) = 0$$

$$\left.\begin{aligned} &0 = \frac{-k\,[m_1 + m_2 + m_3]}{m_0 + m_1 + m_2 + m_3}x_2^n + y_1 + \alpha_2^* \\ &0 = \frac{-k\left[m_2 l_{cg_2} C\theta_2 + m_3 l_2 C\theta_2 + m_3 l_{cg_3} C\theta_{23}\right]}{m_0 + m_1 + m_2 + m_3}x_2^n + y_3 + \alpha_3^* \\ &0 = \frac{-km_3 l_{cg_3} C\theta_{23}}{m_0 + m_1 + m_2 + m_3}x_2^n + y_3 + \alpha_4^* \end{aligned}\right\} F_2(x,y,t) = 0$$

where: $\alpha_1^*, \alpha_2^*, \alpha_3^*, \alpha_4^*$ - modified constants

Fig. 3 Dynamic Equations in Standard DAE Form for a Planar Manipulator on a Vertically Compliant Base

Inverse dynamic solutions for compliant mechanisms modeled without constraint equations are calculated by first integrating the differential equations for the mechanism deflections. The solved deflections are then substituted into the algebraic equations to obtain the joint forces required to produce the desired motions. The two part solution is possible because the differential equations are decoupled from the algebraic equations (i.e., independent of the joint forces) as shown in Figure 3.

3.2 Mechanisms Modeled With Constraint Equations

The equations of motion (before substitution of specified joint trajectories) for closed-chain mechanisms have the following form

$$M(q)\ddot{q} = f(q,\dot{q},t) + G(q)\lambda$$

$$\Phi(q) = 0$$

where $\dfrac{\partial\Phi}{\partial q} = G^T$

λ - Lagrange multipliers

$\Phi(q)$ - constraint equations

The closed kinematic chains are enforced with algebraic constraint equations and Lagrange multipliers in the above equations.

For non-compliant mechanisms, the constraint equations are identically satisfied by the specified joint trajectories so these equations provide no useful information for the inverse dynamic model. The number of unknowns (joint and constraint forces) may exceed the number of available dynamic equations for inverse dynamic models of closed-chain, non-compliant systems. This occurs when the number of actuated joints is greater than the number of system dofs. In such cases, an infinite number of possible force solutions exist for a given trajectory. Many of the possible solutions correspond to actuator conflict where powered joints act in an isometric (i.e., non-productive) manner. An example of actuator conflict is shown in Figure 4 for a non-compliant four bar linkage.

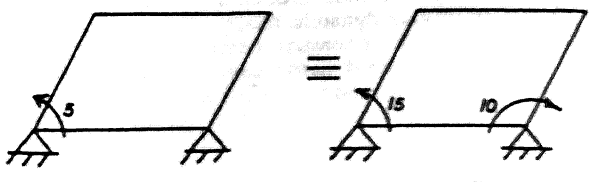


Fig. 4 An Example of Actuator Conflict for a Non-Compliant, Four Bar Linkage

When a closed-chain mechanism has compliance in each closed kinematic chain, the inverse dynamic solution is determinate for any number of actuated joints because opposing actuator forces result in deformation that distributes internal forces. The compliance must be arranged so that no mechanism motion is rigidly constrained for the problem to be determinate. Otherwise, the degree of indeterminacy would be reduced and not eliminated dependent on the number of constraint equations that are no longer applicable. When sufficient system compliance exists for the inverse dynamic solution to be determinate considering all joints to be actuated, the mechanism will be referred to as fully compliant.

All joints in a closed-chain mechanism do not have to be powered to produce a controlled mechanism motion. The existence of a determinate, inverse dynamic solution for a less than fully compliant, closed-chain mechanism is dependent on having an unactuated joint with a specified trajectory in each closed kinematic chain corresponding to an eliminated constraint equation. (Constraint equations cannot be eliminated for fully compliant, closed-chain mechanisms because deflection variables are present in these equations.) If a joint is unpowered (i.e., its trajectory is unknown) in a fully compliant, closed-chain mechanism, the corresponding joint motion is governed by the deflection variables in addition to the constraint equations. Either a trajectory or force, but not both, is specified for each joint of a fully compliant, closed-chain mechanism because the unknown deflection variables and constraint forces affect the unspecified quantity.

The inverse dynamic model of a compliant mechanism modeled with constraint equations is obtained by substitution of the defined joint trajectories into the equations of motion for the system; conversion of the equations to standard form results in a coupled set of DAE's. An example of two rigidly connected, planar manipulators on independent, vertically compliant bases shown in Figure 5 is used to illustrate formulation of an inverse dynamic model for compliant mechanisms modeled with constraint equations. The constraint equations given in Figure 6 ensure compatability at the end-effectors. Joints d_{12}, Θ_{22} and Θ_{32} were considered to be powered which results in a determinate solution. (If, in addition, joints Θ_{21} and Θ_{31} were considered to be powered, the first and third constraint equations would be eliminated and the inverse dynamic solution would become indeterminate.)

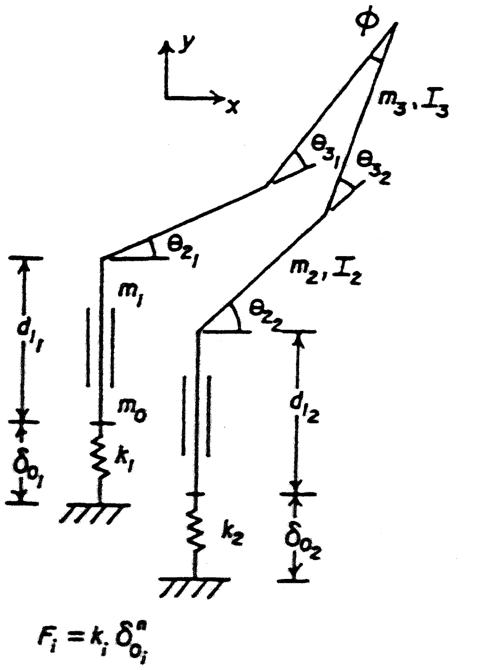


Fig. 5 Two Rigidly Connected, Planar Manipulators on Independent, Vertically Compliant Bases

$$p_{x_1} + L_2 C\theta_{2_1} + L_3 C\theta_{23_1} = p_{x_2} + L_2 C\theta_{2_2} + L_3 C\theta_{23_2}$$

$$p_{y_1} + \delta_{0_1} + d_{1_1} + L_2 S\theta_{2_1} + L_3 S\theta_{23_1} = p_{y_2} + \delta_{0_2} + d_{1_2} + L_2 S\theta_{2_2} + L_3 S\theta_{23_2}$$

$$\theta_{2_2} + \theta_{3_2} - \theta_{2_1} - \theta_{3_1} = \Phi$$

where: (p_{x_1}, p_{y_1}), (p_{x_2}, p_{y_2}) - manipulator base positions

Fig. 6 Constraint Equations for Two Rigidly Connected, Planar Manipulators on Independent, Vertically Compliant Bases

The inverse dynamic equations have the functional form shown in Figure 7 after substitution of the joint trajectories and conversion to state space form. Gaussian elimination of derivative terms from the dynamic equations for joint variables having defined trajectories (i.e., d_{12}, Θ_{22} and Θ_{32}) results in the standard form equations shown in Figure 8. The differential equations correspond to deflection and joint variables having unspecified motions while the algebraic equations correspond to constraint equations and joint variables with specified trajectories.

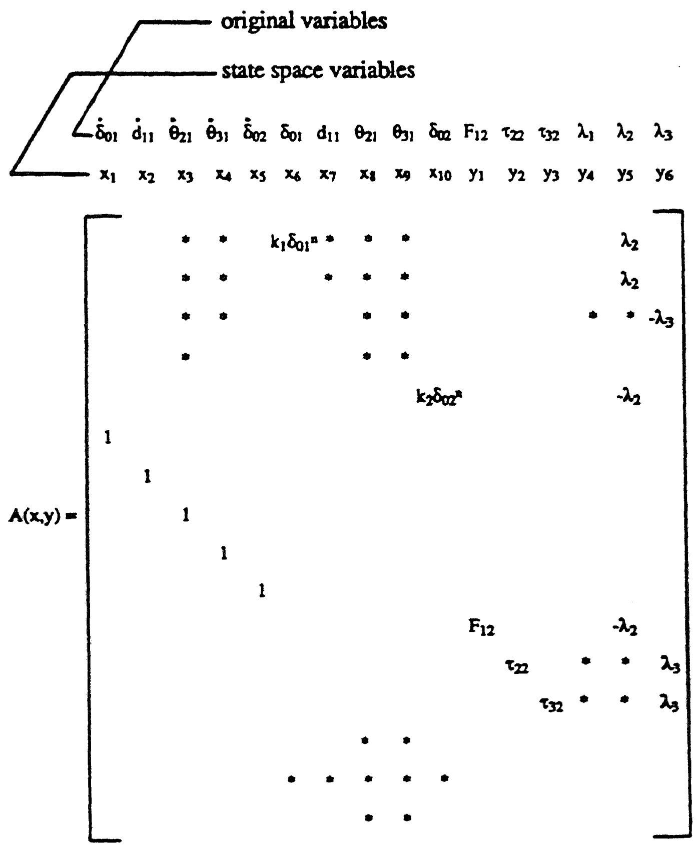
$$M \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = A(x,y) + B$$

where: B - constant vector

$$\begin{array}{c|cccccccccccccccc}
\text{original variables} & \ddot{\delta}_{01} & \ddot{d}_{11} & \ddot{\theta}_{21} & \ddot{\theta}_{31} & \ddot{\delta}_{02} & \dot{\delta}_{01} & \dot{d}_{11} & \dot{\theta}_{21} & \dot{\theta}_{31} & \dot{\delta}_{02} & \dot{F}_{12} & \dot{\tau}_{22} & \dot{\tau}_{32} & \dot{\lambda}_1 & \dot{\lambda}_2 & \dot{\lambda}_3 \\
\text{state space variables} & \dot{x}_1 & \dot{x}_2 & \dot{x}_3 & \dot{x}_4 & \dot{x}_5 & \dot{x}_6 & \dot{x}_7 & \dot{x}_8 & \dot{x}_9 & \dot{x}_{10} & \dot{y}_1 & \dot{y}_2 & \dot{y}_3 & \dot{y}_4 & \dot{y}_5 & \dot{y}_6
\end{array}$$

$$M = \begin{bmatrix}
m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} & & & & & & & & & & & & \\
m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} & & & & & & & & & & & & \\
m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} & & & & & & & & & & & & \\
m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} & & & & & & & & & & & & \\
 & & & & m_{5,5} & & & & & & & & & & & \\
 & & & & & 1 & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & \\
 & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & 1 & & & & & & \\
 & & & & m_{11,5} & & & & & & 0 & & & & & \\
 & & & & m_{12,5} & & & & & & & 0 & & & & \\
 & & & & m_{13,5} & & & & & & & & 0 & & & \\
 & & & & & & & & & & & & & 0 & & \\
 & & & & & & & & & & & & & & 0 & \\
 & & & & & & & & & & & & & & & 0
\end{bmatrix}$$

Fig. 7 Dynamic Equations in State Space Form for Two Rigidly Connected, Planar Manipulators on Independent, Vertically Compliant Bases



where: * - indicates a non-zero entry

Fig. 7 (cont'd) Dynamic Equations in State Space Form for Two Rigidly Connected, Planar Manipulators on Independent, Vertically Compliant Bases

$$F_1(x,\dot{x},y,t) = 0$$

$$F_2(x,y,t) = 0$$

where: $F_1(x,\dot{x},y,t)$ - same as first 10 equations of Figure 7

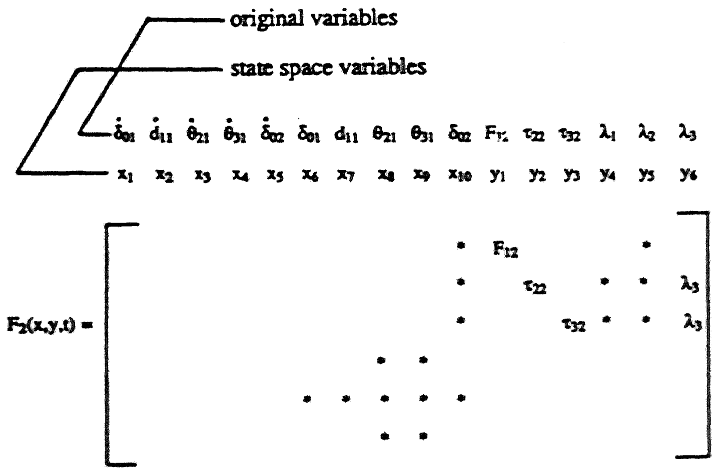


Fig. 8 Dynamic Equations in Standard DAE Form for Two Rigidly Connected, Planar Manipulators on Independent, Vertically Compliant Bases

The dynamic equations for joint variables having defined trajectories reduce to algebraic equations while the constraint equations remain as algebraic equations for any compliant mechanism modeled with constraint equations. Also, the dynamic equations for joint and deflection variables having unspecified motions remain as differential equations. The difference between inverse dynamic models for systems modeled with and without constraint equations is the additional algebraic (constraint) equations of the former. These additional equations serve to couple the differential and algebraic equations of the inverse dynamic model through the Lagrange multipliers; this eliminates the possibility of separate solutions (as for systems modeled without constraint equations). Relevant characteristics of DAE systems are considered below so that appropriate solution techniques can be defined for inverse dynamic models of compliant mechanisms modeled with constraint equations.

4. DIFFERENTIAL/ALGEBRAIC EQUATIONS

A coupled set of DAE's, which are also described as singular systems of differential equations [McClamroch 86], can be expressed in the following standard form

$$F_1(x,\dot{x},y,t) = 0$$

$$F_2(x,y,t) = 0$$

Systems of DAE's cannot be solved directly using numerical methods intended for ordinary differential equations (ODE's); an equation transformation or special numerical techniques must be considered. A singularity measure of a set of DAE's is given by its index (or nilpotency) and the solution difficulty increases as the index increases [Petzold 82]. The index of a system is determined by transforming the set of equations into canonical form and observing the size of the coefficient matrix for the non-state variables (i.e., variables not having any derivative terms). An alternate approach for determining the index [Gear 88] is to count the required number of differentiations of the algebraic equations to produce a set of ODE's. The forward dynamic model of a closed-chain mechanism is shown [Petzold 86] to be index 3 or index 2 when position or velocity constraints are defined, respectively.

Solutions to determinate systems of DAE's can be obtained by substituting backward difference formulas (BDF) for the derivative expressions and solving the resulting set of simultaneous equations iteratively using Newton's method [Gear 71]. (There are no proven optimization techniques for indeterminate DAE systems except for the preliminary work in [Lotstedt 84].) For a general system of DAE's, the Jacobian matrix used in Newton's method is

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial x} + \frac{\alpha_0}{h}\frac{\partial F_1}{\partial \dot{x}} & \frac{\partial F_1}{\partial y} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial y} \end{bmatrix}$$

where h - stepsize

α_0 - BDF coefficient

A unique solution exists if the Jacobian is non-singular. An obvious requirement for an invertible Jacobian is that $\partial F_2/\partial y$ be non-singular, which is an alternate definition of an index 1 system [Lotstedt 86]. For this reason, index 1 DAE systems can be readily solved using BDF substitution and Newton's method with little more difficulty than solving ODE's. A mathematically precise requirement for a unique solution to exist is that the Schur complement [Cottle 74] of the above matrix must be non-singular. This explains why solutions can be obtained for certain higher order systems (i.e., index greater than 1) where $\partial F_2/\partial y$ is singular. A solvable index 2 system has non-zero rows of $\partial F_2/\partial y$ that are linearly independent [Lotstedt 86].

The error in solving index 1 systems, solvable index 2 systems and index 3 mechanical systems of DAE's using a constant stepsize BDF is $O(h^k)$ [Lotstedt 86], where k is the order of the difference expression and h is the stepsize. The use of variable stepsizes is difficult because the normal error definitions used for ODE's can be completely incorrect for DAE's as a result of the non-state variable contributions. A suitable error definition used for variable stepsize control of DAE systems is discussed in [Petzold 82].

A method for reducing the index of a set of DAE's [Gear 88] is to differentiate the constraint equations to produce a DAE system with an index that is one lower for each differentiation. Taken to the extreme, this approach can result in a system of ODE's. Since the ODE system is equivalent to the original DAE system, calculated solutions will depend on differentials of state variables, non-state variables and input functions; any discontinuities in the latter can result in non-defined solutions. Similarly, consistent initial conditions that satisfy not only the constraint equations but derivatives of the constraint equations are essential; otherwise, errors in the initial conditions will contaminate the solution.

5. EQUATION INDEX OF COMPLIANT MECHANISM INVERSE DYNAMIC MODELS

The inverse dynamic equations of motion for a compliant mechanism modeled without constraint equations have been shown by this study to be a set of DAE's where the differential equations correspond to unknown deflection variables and the algebraic equations correspond to unknown joint forces. Converting the equations to standard form results in an index 1 system because $\partial F_2/\partial y$ is non-singular. A single unknown joint force is unique to each algebraic equation which makes the rows of $\partial F_2/\partial y$ always linearly independent as shown by the example equations for the planar manipulator on a vertically compliant base given in Figure 9.

original variables

state space variables

$$\begin{array}{cccc} & F_1 & \tau_2 & \tau_3 \\ & y_1 & y_2 & y_3 \\ \partial F_2/\partial y = & \left[\begin{matrix} 1 & & \\ & 1 & \\ & & 1 \end{matrix}\right. & & \left.\right] \end{array}$$

$$\partial F_2/\partial y = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$$

Fig. 9 $\partial F_2/\partial y$ for a Planar Manipulator on a Vertically Compliant Base

If the mechanism has closed kinematic chains modeled with constraint equations, additional algebraic equations and non-state variables corresponding to the constraint equations and Lagrange multipliers, respectively, are included in the equations of motion. These additional equations make $\partial F_2/\partial y$ singular because the constraint equations are independent of the non-state variables (i.e., variables not having derivative terms in the equations). The non-zero rows of $\partial F_2/\partial y$ are always linearly independent because a single unknown joint force is unique to each algebraic equation that corresponds to joints with specified trajectories (i.e., actuated joints). This linear independence is shown by the equations given in Figure 10 for two rigidly connected, planar manipulators on independent, vertically compliant bases. Therefore, the inverse dynamic equations of motion for a compliant, closed-chain mechanism modeled with constraint equations are a solvable set of index 2 DAE's.

original variables

state space variables

$$\begin{array}{c|cccccc} & F_{12} & \tau_{22} & \tau_{32} & \lambda_1 & \lambda_2 & \lambda_3 \\ & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \\ \hline & 1 & & & & * & \\ & & 1 & & * & * & 1 \\ \partial F_2/\partial y = & & & 1 & * & * & 1 \\ & & & & 0 & & \\ & & & & & 0 & \\ & & & & & & 0 \end{array}$$

where: * - indicates a non-zero entry

Fig. 10 $\partial F_2/\partial y$ for Two Rigidly Connected, Planar Manipulators on Independent, Vertically Compliant Bases

5. SUMMARY

A general formulation of inverse dynamic models for all compliant mechanisms is presented where the model is obtained by substitution of prescribed joint motions into the equations of motion for the system. The resulting equations for compliant mechanisms modeled with and without constraint equations are shown to be solvable index 2 and index 1 DAE systems, respectively. Relevant characteristics of DAE systems and solutions of these equations are discussed. Stable and accurate numerical solutions can be obtained by BDF substitution and application of Newton's method to the resulting set of equations.

The formulation and solution procedures have been used to define the inverse dynamic model of legged locomotion on natural terrain. Foot-soil interactions are modeled with non-linear force-deflection relationships resulting in a fully compliant model and determinate solutions. The mechanism structure and joints are considered to be rigid while joint damping and backdrive effects are included in the model. Stable solutions are being calculated with two to five Newton iterations per timestep. The locomotion model is currently being used for gait and control system simulation studies of a multi-legged robot under development at the Field Robotics Center, Carnegie Mellon University.

REFERENCES

Bayo, E., 1988, "Computed Torque for the Position Control of Open-Chain Flexible Robots," Proceedings 1988 IEEE International Conference on Robotics and Automation, Vol. 1, pp. 316-321

Cottle, R. W., 1974, "Manifestations of the Schur Complement," Linear Algebra Appl., Vol. 8, pp. 189-211

Dado, M., and Soni, A. H., 1986, "A Generalized Approach for Forward and Inverse Dynamics of Elastic Manipulators," Proceedings 1986 IEEE International Conference on Robotics and Automation, Vol. 1, pp. 359-364

Forrest-Barlach, M. G., and Babcock, S. M., 1987, "Inverse Dynamics Position Control of a Compliant Manipulator," IEEE Journal of Robotics and Automation, Vol RA-3, No. 1, pp. 75-83

Gear, C. W., 1971, "Simultaneous Numerical Solution of Differential/Algebraic Equations," IEEE Trans Circuit Theory, CT-18, pp. 89-95

Gear, C. W., 1988, "Differential-Algebraic Equation Index Transformations," SIAM J. Sci. Stat. Comput., Vol. 9, No. 1, pp. 39-47

Lotstedt, P., 1984, "Numerical Simulation of Time-Dependent Contact and Friction Problems in Rigid Body Mechanics," SIAM J. Sci. Stat. Comput., Vol. 5, No. 2, pp. 370-393

Lotstedt, P., and Petzold, L., 1986, "Numerical Solution of Nonlinear Differential Equations with Algebraic Constraints I: Convergence Results for Backward Differentiation Formulas," Mathematics of Computation, Vol. 46, No. 174, pp. 491-516

McClamroch, N. H., 1986, "Singular Systems of Differential Equations as Dynamic Models for Constrained Robot Systems," Proceedings 1986 IEEE International Conference on Robotics and Automation, Vol. 1, pp. 21-28

Petzold, L., 1982, "Differential/Algebraic Equations are not ODE'S," SIAM J. Sci. Stat. Comput., Vol. 3, No. 3, pp. 367-384

Petzold, L., and Lotstedt, P., 1986, "Numerical Solution of Nonlinear Differential Equations with Algebraic Constraints II: Practical Implications," SIAM J. Sci. Stat. Comput., Vol. 7, No. 3, pp. 720-733

A Perception System for a Planetary Explorer

M. Hebert, E. Krotkov, T. Kanade [1]
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

To perform planetary exploration without human supervision, a complete autonomous robot must be able to model its environment and to locate itself while exploring its surroundings. For that purpose, we propose a modular perception system for an autonomous explorer. The perception system maintains a consistent internal representation of the observed terrain from multiple sensor views. The representation can be accessed from other modules through queries. The perception system is intended to be used by the Ambler, a six-legged vehicle being built at CMU. A partial implementation of the system using a range scanner is presented as well as experimental results on a testbed that includes the sensor, one computer controlled leg, and obstacles on a sandy surface.

1 Introduction

The unmanned exploration of planets, such as Mars, requires a high level of autonomy due to the communication delays between a robot and the Earth-based station. This impacts all the components of the system: planning, sensing, and mechanism [6]. In particular, such a level of autonomy can be achieved only if the robot has a perception system that can reliably build and maintain models of the environment. We propose a perception system that is designed for application to autonomous planetary exploration. The perception system is a major part of the development of a complete system that includes planning and mechanism design. The target vehicle is the Ambler, a six-legged walking machine being developed at CMU (Figure 1, [1]).

The perception system can be viewed as an intelligent memory that can be interrogated by external modules (e.g., path planning modules) while maintaining an internal representation of the world built from sensors as the vehicle navigates. The paper addresses the choice of the basic representation maintained by the system in Section 2 and the architecture of the perception system in Section 3. Although the architecture is designed to handle a variety of sensors, we have focused on the use of a laser range finder, since the first requirement for safe navigation of the robot is reliably modeling the geometry of the surrounding terrain. Section 4 describes the algorithms developed for the construction of terrain models from range images. Finally, Section 5 describes the experiments that were conducted to evaluate the perception system.

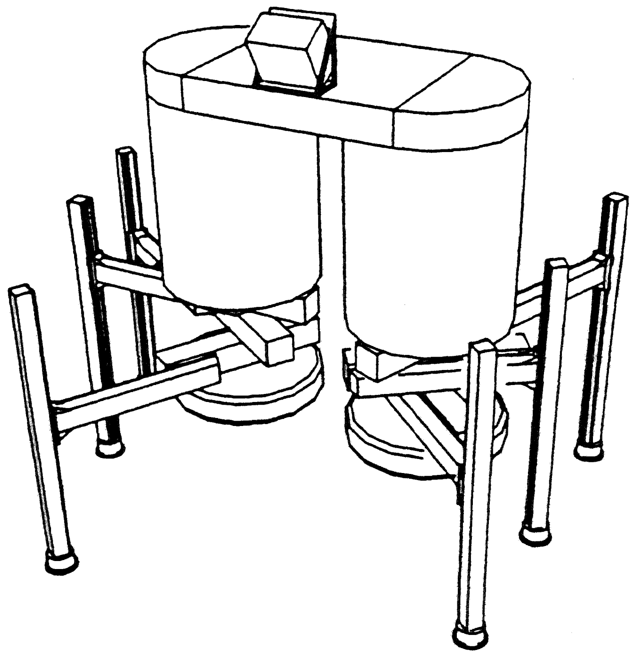


Figure 1: The Ambler

2 Terrain Representation

The basic internal representation used by the perception system is a grid, the local terrain map, each cell of which contains attributes of the terrain. A cell must contain at least the elevation of the terrain and the uncertainty on the elevation due to sensor noise. The uncertainty is modeled as a Gaussian distribution, whose standard deviation σ is stored in the terrain map. Other attributes may include the slope, the surface texture, etc. In addition, attributes that are stored in a cell may be of non-geometric nature, such as the color of the terrain. Several resolutions of the grid may be maintained simultaneously.

On top of the base grid, higher level information can be represented in the form of labeled features in the grid, such as topographic features (hills, ravines, etc.), regions of homogeneous terrain type, objects of interest that have been extracted (boulders, rocks, etc.).

Other terrain representations are possible. The surface could be represented directly by 3-D patches that either are approximations of the measured surface or are built directly upon the set of data points. In both cases, however, retrieving a region of interest from the map becomes a complex operation. Another possibility

[1]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 US Government.

is to represent only a higher-level description of the terrain, such as a segmentation of the surface. This is not appropriate in our case since some planning tasks need information at the lowest level (elevation map). One example for the Ambler is to estimate how stable a foot placement on the terrain would be, in that case a surface description at a resolution that is well below the size of the foot is needed. By contrast with the alternative representations, the terrain map representation as an elevation map is simple to manipulate, can include high-level information as well as high resolution elevation data, and can be accessed by external modules in a simple way by giving the boundary of the region of interest in the map.

3 Architecture

The perception system is divided into six logical modules (Fig. 2). The system communicates with external modules using messages that are routed through a central message handler [4]. The perception system is controlled by a front end—the Local Terrain Map Manager (LTMM)—that receives the messages. Once a message requesting data is received, the LTMM checks whether it is available in the current internal terrain map. If not, then the LTMM instructs the Imaging Sensor Manager (ISM) to take a new image from the relevant sensors, and the terrain map from the new image is merged in the current terrain map. The internal representation is a terrain map built with respect to a fixed reference frame, the *global* frame $\mathcal{G}$. All the operations in the overall Ambler system are expressed with respect to $\mathcal{G}$. A separate module provides the vehicle pose in $\mathcal{G}$. Since the terrain map is of interest only in a region around the vehicle, useless parts of the terrain map are discarded by another module, the Scroller.

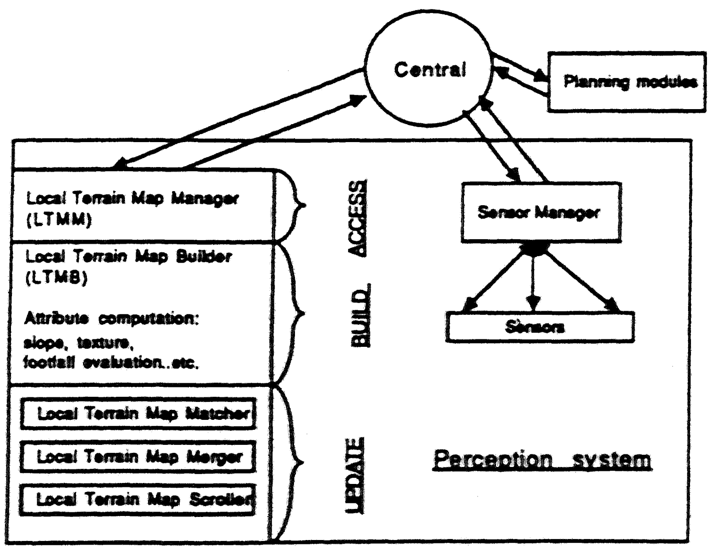

Central
Planning modules
Local Terrain Map Manager (LTMM)
Local Terrain Map Builder (LTMB)
Attribute computation: slope, texture, footfall evaluation..etc.
Local Terrain Map Matcher
Local Terrain Map Merger
Local Terrain Map Scroller
ACCESS
BUILD
UPDATE
Sensor Manager
Sensors
Perception system

Figure 2: Architecture of the perception system

3.1 Accessing the Perception System

External modules communicate with the perception system by exchanging messages (Fig. 2). Two types of messages are used: *queries* that are request for data, and *replies* that are used for sending data in response to a query. The queries and replies are routed and synchronized by a central module [4]. The simplest example of a query from a planning module would be a request for an elevation map in a given area for clearance checking. The main advantage of this mode of access is that external modules do not need to know the internal workings of the perception system such as the sensor used or the format of the internal representation.

The LTMM is the front end to the perception system, and is responsible for processing queries and for activating the proper submodules. When a query is received, the manager first checks if the area of interest has been already processed at the requested resolution, if that is the case the requested information is extracted from the existing terrain map, otherwise the manager requests a new image from the ISM that is processed and merged with the current terrain map.

To be processed all queries must contain three pieces of information: a polygon that is the boundary of the region of interest; a resolution that indicates at what level of detail the requested calculations must be carried out on the terrain map;[2] and the type of information requested (elevation, uncertainty, slopes, etc.). Because all queries are expressed in $\mathcal{G}$, external modules do not need to know the pose of the sensors. The transformation between a sensor and the vehicle's base frame is stored internally by the perception system, while the current vehicle pose with respect to $\mathcal{G}$ is requested each time a query is received.

3.2 Acquiring Sensor Data

Instead of hardcoding the sensor interface into the LTMM, sensor data is obtained through the same query mechanism. Whenever an image is requested, the requesting module sends a query to the ISM that includes the type of sensor and the type of data desired. The ISM is responsible for activating the requested sensor. The ISM can be viewed as a virtual sensor that hides the details of the sensors' interfaces from the perception system, thus allowing for a more flexible way of changing sensor specifications. Because all queries are expressed in $\mathcal{G}$, the ISM is also responsible for requesting the position of the vehicle with respect to $\mathcal{G}$ from a module that keeps track of the position of the robot either by dead reckoning or by using a navigation system. The other transformation that is needed in order to use the sensor data is the transformation between sensor frame and vehicle frame; this transformation is pre-computed by a calibration procedure and stored by the ISM at initialization time. The composition of those two transformations, that is the transformation between sensor frame and $\mathcal{G}$, is returned to the perception system along with the sensor data (Fig. 3).

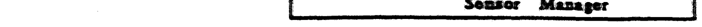

LTMM
Request for data + sensor type
Sensor data + transformation T
Sensor frame
T
Site frame
Vehicle frame
Sensor Manager

Figure 3: The Imaging Sensor Manager

[2]For example, a resolution of several tens of centimeters is sufficient for checking that the path of the body is clear. Analyzing the stability of one foot of the Ambler requires a resolution of a few centimeters.

3.3 Building the Terrain Map

Once a query is received, a terrain map must be built within the boundary of the region of interest. This is the role of the Builder (Fig. 2) which constructs a terrain map given sensor data taken at one position of the vehicle. The terrain map is computed at the requested resolution and includes the elevation and the uncertainty at each point. In addition to those two attributes, the Builder also implements the algorithms for computing other local attributes such as slope or surface texture as well as non-geometric attributes such as color or terrain type depending on the available sensors. In addition to computing the local attributes, the Builder also identifies the portions of the map that are outside of the fields of view of the sensors, and those that are occluded by parts of the terrain.

The Builder is optimized in several ways: It maintains maps at different resolutions so that it is not necessary to always compute the map at finest resolution. Also, if a map at the desired resolution does not exist, the Builder will create one, thus allowing for arbitrary resolutions. The Builder minimizes the amount of computation by remembering both the regions of the world that have already been computed and by storing the past images so that if a query falls within the field of view of an existing image, it is not necessary to acquire and process a new image.

An implementation of a Builder that uses range images is described in Section 4.

3.4 Updating the Terrain Map

Since the Builder constructs a terrain map from a single image, new sensor data has to be acquired each time a new query is received. This is sufficient as a first approximation, however the perception system should be able to handle terrain maps built from sensor data acquired at different positions. There are two motivations for handling multiple frames. It is obviously more efficient to remember terrain maps built from previous frames rather than recomputing everything at each step. A more compelling motivation is that merging multiple frames may be the only way to provide the requested data. Such a situation occurs when parts of the vehicle, usually a leg, lie within the field of view of the sensor and therefore occludes a part of the terrain map, in that case it may not be possible to extract the region of interest from the current position. A second case in which multiple frames are needed is when data that is outside of the current field of view of the sensor is needed. In the case of the Ambler, this is actually the standard situation since, in the normal walking mode, the leg further behind the body is moved to the front of the body which requires data *behind* the body so that the path of the leg can be checked for clearance. For these reasons, the perception system must include the capability to merge terrain maps from successive frames into a single terrain map.

The responsibility for the management of multiple terrain maps is shared by two modules, the Matcher and the Merger (Fig. 2). The Matcher estimates the displacement between a new terrain map and the current internal terrain map. The displacement is in general a 3-D transformation. It is estimated by matching features extracted from the maps, or by using a correlation technique that compares the two maps directly. Section 4 briefly describes an implementation of the latter in the case of terrain maps built from range images. An initial estimate of the displacement is always available either from dead reckoning or from a navigation system. Once the displacement is computed, the Merger is responsible for merging the new map into the current map. Actually, only the part of the map that is within the requested region of interest is actually merged for efficiency reasons. The maps are merged by combining the elevation values at each location of the map using the uncertainty values to obtain the maximum likelihood estimate of the elevation. The Merger must also update the occluded areas of the current map.

It is important to logically separate the matching and merging operations. First of all, the matching operation may or may not be necessary depending on the accuracy of the positioning system. If a navigation system provides displacement estimates that are well below the resolution of the grids, the estimates will not be improved by terrain matching. Second, if raw sensor data is stored along with past terrain maps, a new query requires only merging portions of the terrain maps since the displacements have already been computed at the time the images were acquired. Finally, separating the two modules allows for experimenting with different matching algorithms, presumably the most difficult part of the system, while retaining the same structure for the rest of the system.

Since the terrain map must grow as the vehicle moves and as new sensor data is acquired, a third module, the Scroller, is responsible for discarding the part of the map that is too far from the vehicle to be useful. This can be viewed as sliding a window centered on the current position of the vehicle; only data within this window is retained. The Scroller is motivated both by the need to prevent the size of the terrain map from expanding during the course of a long mission, with the risk of memory overflow, and by the fact that only the most recent terrain maps can be used with confidence due to the accumulation of errors in the displacement estimates between maps.

4 Elevation Maps from Range Data

The perception system is designed to use multiple sources of data. Because geometric information is most important for local navigation, we consider in this section the case of data from an active range scanner.

We use the Erim laser scanner, which delivers 64×256 range images by measuring the phase difference between a laser beam and its reflection from a point in the scene [7]. The scanner measures the range ρ in a spherical coordinate system in which ϕ and θ are the vertical and horizontal scanning angles, corresponding to row and column positions in the image.

Prior to operation, the position of the sensor with respect to the vehicle's coordinate frame must be computed. This is done by a calibration procedure that computes the position by observing markings on the leg using the range scanner at different known positions of the leg. A least-squares estimation algorithm estimates the transformation between the coordinate system of the scanner and the coordinate system of the vehicle. This transformation is compounded with the transformation between vehicle and global frames by the ISM each time a new image is acquired.

The easiest way to convert the range images to elevations maps is to convert each pixel (ρ, θ, ϕ) to a point in space (x, y, z), which is straightforward knowing the geometry of the sensor and the transformation between sensor and global frames. This approach has some severe drawbacks, however, such as the need for interpolation, the dependency on a particular coordinate system, and the fact that it is not possible to limit the computation to a region of the terrain map because we do not know apriori where this region is in the image. Instead, we use the locus method described in [3]. This approach has many advantages including the explicit detection of range shadows, the representation of uncertainty, the independence of the algorithm with respect to a reference frame, a straightforward extension to the case of multiple frames of data. Furthermore, a major feature of the locus method is its ability to limit the computation of the terrain map to any region in space, thus facilitating the computation of the maps within the boundaries of the queries of Section 3.1.

The terrain map building algorithms were evaluated on range images taken by the Erim scanner. The test images were taken in a construction site that exhibits the type of rugged terrain that we are interested in. Fig. 4 shows a map built from one range image using the locus algorithm. The resolution is 10 cm over a 10 × 10 m square.

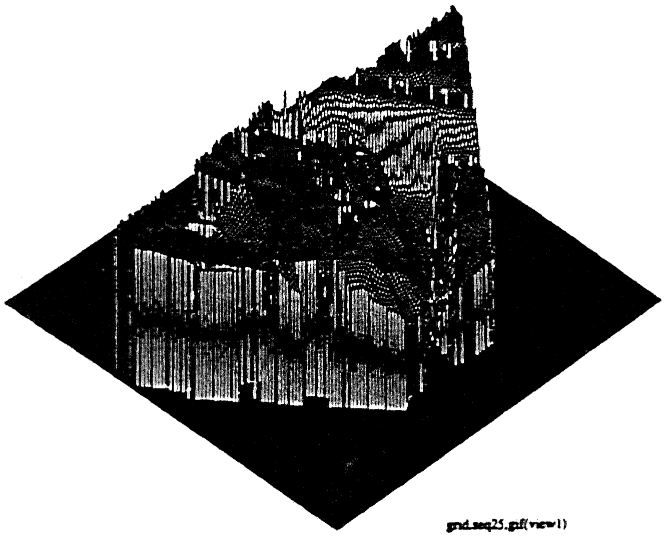


Figure 4: Elevation map built from one range image

An extension of the locus algorithm allows for matching and merging terrain maps built from images taken from different positions. The matching algorithm computes the best 3-D transformation between maps, while the merging algorithm computes optimal combination of the elevation from the two maps given this transformation. The map building from multiple frames was tested on sequences of Erim images as well as on synthesized images. Fig. 5 shows the terrain map obtained by merging data from four successive range images. The resulting terrain map is about thirty meters long. In this example the images were collected along a general path including a sharp turn (about 30°). The matching between consecutive terrain maps was performed by first matching features to obtain a first estimate of the transformation [3], and by using the estimate as a starting point for the minimization of the difference between the two terrain maps that is used as the final transformation for the merging. Experiments on synthesized images for which the transformation between images is known show that the error on the resulting transformation can be as small as the resolution of the grid. The error in elevation is of the order of a few centimeters, increasing with the uncertainty as the points are further away from the sensor.

These experiments show that the algorithms developed for range data provide the type of terrain maps required for rugged, unstructured environments including variable resolution, arbitrary reference frame, explicit uncertainty representation, and representation of occluded areas.

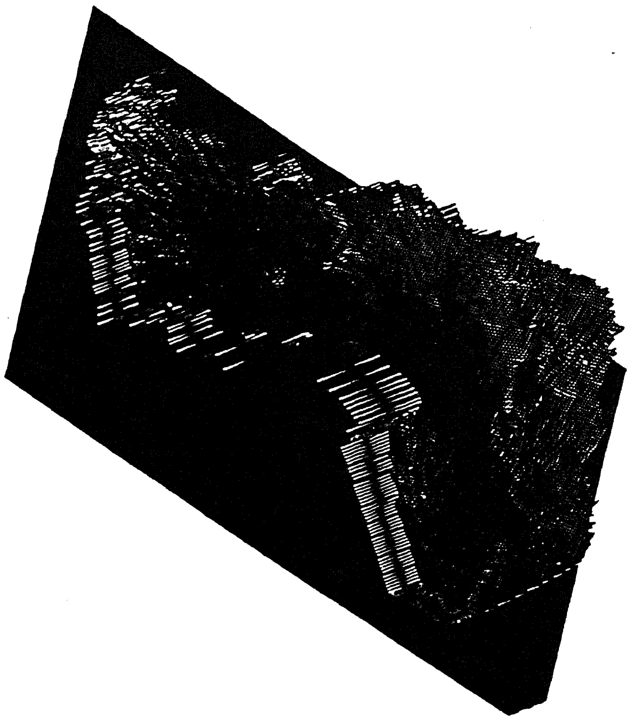


Figure 5: Elevation map from 125 range images

5 Experimentation

A first version of the perception system of Fig. 2 is implemented. This version includes the LTMM, LTM Builder, and ISM. This implementation includes the algorithms of Section 4 and uses the Erim scanner. This implementation of the system is used to validate the interface on single range images. The system builds terrain maps with currently two attributes: uncertainty and footfall evaluation. The latter is a measure of how good a footfall each location in the map would be, based on the local shape of the terrain. The algorithms used for the footfall evaluation are described in [2].

Three types of queries are currently recognized:

- Elevation map: This is a request for an elevation map within a given region (polygon) with a given resolution.
- Elevation and uncertainty map: This is basically the same query except that the uncertainty at each point of the terrain map is returned as well.
- Footfall evaluation: This is a request for the best position of the foot within a region. Currently this request is processed by computing the stability of a circular foot at each point of the terrain map by using only the geometry of the terrain [2].

Fig. 6 shows the result of processing a footfall evaluation query. The lower left view displays an overhead view of the site with the region of interested displayed as a shaded polygon. The three other views are the map computed from a range image. The lower right map is a map of the footfall evaluation in which the highest values correspond to the best footfall locations. The dimensions on the lower left diagram are in meters. The resolution of this query is 10 cm.

A testbed was built in order to test the fully integrated planning/perception/mechanism system. The testbed (Fig. 7) includes a single leg, the range finder mounted on top of the "body" of the vehicle, and a $25m^2$ sandbox that simulates the terrain in which

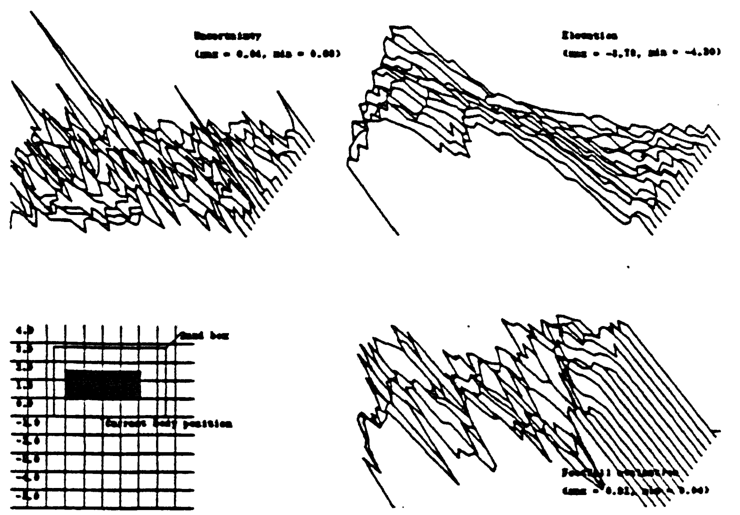


Figure 6: Processing a footfall query

the rover will navigate. The testbed leg was built from an earlier design, as a result it is slightly different from the legs in Figure 1. The main difference is that the testbed leg uses rotational joints while the design of Figure 1 uses prismatic joints. A real-time controller drives the leg to specified locations in Cartesian or joint space, allowing for constraints on the velocity of the leg and the forces applied to the foot. In addition, the body can be translated along two parallel rails by controlling the two horizontal joints of the leg while keeping the foot on the ground, thus simulating the motion of the body in the actual rover. The testbed is equipped with a linear position sensor and two clinometers that together give an estimate of the position and orientation of the rover with respect to the global frame.

Figure 7: The single leg testbed

The most complicated task that is used in order to test the perception system as part of the complete single leg testbed is the so-called "move-body" task in which, given a desired length of travel and a desired step length, the leg takes a series of steps, pulling the body forward after each step. The locations of the footfalls as well as the trajectory of the leg are computed using the terrain maps from the perception system.

For each step, the sequence of operations is as follows.

1. A region in which the foot may be placed to achieve the next step is computed by the gait planner module.
2. The gait planning module queries the perception system for the best footfall position within this region. This query activates the whole cycle of taking an image, computing the terrain map, computing the footfall evaluation attribute, and replying.
3. Given the footfall position, the leg recovery planning module computes a path for the leg and sends a region around this path to the perception system requesting a map.
4. Perception answers the query by sending back a map within the specified region, including the uncertainty attribute.
5. The planning module uses the map to compute the locations of intermediate points along the path of the leg. The leg is moved to the goal location and the foot is lowered onto the terrain using position control. The uncertainty is used as a safety margin both for the travel of the leg and for the actual footfall. In the latter case the foot is lowered to a position that is 2σ above the nominal value reported in the terrain map, and then lowered using force control until it contacts the soil.

Repeated experiments with the "move-body" scenario with different terrain shapes and different initial and goal configurations of the leg have shown conclusively that the first version of the perception system performs reliably and allow the system to safely walk around obstacles.

Several lessons were learned during these experiments. Good calibration between the sensor and the leg is essential for computing reliable elevation value in the vehicle's reference frame. It is important to use information already extracted when possible, if an image is taken whenever a query is received we then run the risk to have the leg in the field of view of the sensor occluding the region of interest. The solution to this problem is to include in the perception system the algorithms that extract the relevant information from the existing terrain map before acquiring new data. Finally, it is clear from those experiments that more development is needed as far as the computation of attributes is concerned. The only attributes are currently the footfall evaluation and the uncertainty.

6 Discussion

We have presented a perception system for an autonomous vehicle designed for planetary exploration. The perception system uses terrain maps as the basic internal representation that is accessed by external modules. Parts of the system have been demonstrated using algorithms for building terrain maps from range images. The current version of the perception system has been included in a complete single leg testbed.

6.1 Improvements

Several improvements are needed in the current system. First, calibration is of critical importance for the successful operation of the overall system. We therefore need to improve the calibration procedure to the point at which the errors due to miscalibration are minimal compared to the other sources of errors. This involves in particular a more detailed analysis of the geometry of the mechanism, a more accurate model of the sensor, and more reliable algorithms for the detection of calibration targets.

The quality and accuracy of the terrain maps may also be improved. In particular, the uncertainty model may reflect the actual environment by using a more detailed model of the sensor measurements. Another improvement is the extensive use of map merging to produce more accurate maps by combining many measurements at each point in the map.

The last improvement is in the area of exception handling. Currently, the perception system cannot recover gracefully from errors such as corrupted sensor data, bad transformation from corrupted position readings, or bad message handling. In order to have a robust system we need to design a mechanism to detect and recover from these conditions.

6.2 Extensions

Further work is required to demonstrate a perception system that can handle the tasks of a complete autonomous system. Other sensors must be used in conjunction with the laser range finder in order to compute non-geometric types of information such as the type of the terrain in a region. This is important both for sampling tasks, which require the identification of specific types of terrain, and for the evaluation of footfall selection since the soil compliance depends on the type of terrain. The best candidates are color cameras and thermal cameras. We are working on integrating those sensors into the perception system.

Other sensors that should be added to the perception system include sensors for short-range perception, such as proximity sensors. Those sensors would be used in the final phase of the footfall to provide better control of the foot contact with the terrain. Currently, the foot is lowered to a nominal value given by the elevation map, after which point it is slowly lowered until a given force reaction is observed. This is a potentially dangerous approach if the map is inaccurate at that point, or if the terrain has changed between the time the map was built and the time the foot is moved. A proximity sensor would guarantee that the foot does not attempt to penetrate the ground.

The perception system uses only local information from its sensors. A possible extension would be the addition of more global information such as a large-scale map from an orbiter. The main issue is then to establish the relationship between the low-resolution global map and the high-resolution local maps. This is essentially a matching capability that can greatly enhance the performances of the rover. For instance, the rover could register itself with respect to large-scale terrain features from the global map.

Finally, we must complete the inclusion of the matching of multiple frames in the system. Map matching will give the rover a "self-localization" capability, that is the ability to register itself with respect to its environment without relying entirely on special-purpose position sensors (clinometers, dead reckoning, INS). It has been our experience that those sources of position information cannot be relied upon at all time because they do not necessarily give an accurate description of the position and orientation of the sensor at the time an image is taken. Furthermore, they have to be carefully calibrated with respect to the perception sensors which add another level of complexity to the already difficult calibration problem. The solution will be to use the output of the position sensors as an initial estimate for the map matching process which will provide the accurate position estimate actually used.

6.3 Remaining Issues and Lessons Learned

In the course of developing this system we have encountered the usual fundamental issues in the design of autonomous systems [5], and had to make choices to overcome those problems. Two issues were of special interest: the mode of synchronization between the perception system and the other modules, and the limitations due to message-passing between modules.

The architecture is currently entirely query-driven in that the terrain maps are computed only in response to a specific query from another module. This may not be the best strategy in a system that includes many other computation-intensive modules. In that case, the perception system would be idle most of the time. A different strategy would be for the perception system to keep computing the terrain map around the vehicle even if no query has been received. That way, the perception would take advantage of the idle time to perform some additional computations. The main issue is for the perception system to be able to predict the regions of the environment that will be "useful" to compute for the future queries. This also requires a careful analysis of the synchronization between modules so that this self-driven approach does not accidentally slow down the other modules.

Our experience with this system has been that the communication bandwidth using conventional network technology is not a limitation. In this application, shipping images and maps between the different modules of the perception system and the other modules does not affect the performance of the overall system significantly. There are still some synchronization issues to be addressed, however. The most important one is to guarantee that the position of the vehicle is correctly read at the time that an image is taken (Section 3.2), which is not possible if there is too much of a delay between the ISM and the module that sends the vehicle position. One solution is to bypass the central message handler completely for some of the low-level operations such taking an image so that the communications are performed by direct memory transfer with minimal delay.

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] 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*, Tsukuba, Japan, September 1989.

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

[4] R. Simmons and T. Mitchell. A Task Control Architecture for Mobile Robots. In *Proc. AAAI Spring Symposium*, Stanford, California, March 1989.

[5] T. Stentz and C. Thorpe. Against Complex Architecture. In *Proc. AAAI Spring Symposium*, Stanford, California, March 1989.

[6] B. Wilcox. Session 3: Planetary Rovers. In *Proc. SPIE Vol. 1007: Mobile Robots III*, 1988.

[7] 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.

Experience with a Task Control Architecture for Mobile Robots

Long-Ji Lin
Reid Simmons
Christopher Fedor

Abstract

This paper presents a general-purpose architecture for controlling mobile robots, and describes a working mobile manipulator which uses the architecture to operate in a dynamic and uncertain environment. The target of this work is to develop a distributed robot architecture for planning, execution, monitoring, exception handling, and multiple task coordination. We report our progress to date on the architecture development and the performance of the working robot. In particular, we discuss temporal reasoning, execution monitoring, and context-dependent exception handling.

1. Introduction

The principal goal of this work is to develop a distributed robot architecture to support robot planning, execution, monitoring, exception handling, and multiple task coordination. We have been developing such a robot architecture, called the Task Control Architecture (TCA) [15]. TCA is designed for controlling mobile robots that have limited computational and sensory resources, operate in uncertain, changing (but relatively benign) environments, have multiple goals, and have a variety of strategies to achieve goals and handle exceptions.

We have been developing TCA concurrently on two testbeds -- the CMU six-legged Planetary Rover [3] and the Heath/Zenith Hero 2000 mobile manipulator robot [12]. The CMU Rover project is an attempt to develop an autonomous robot that can survive, navigate, and acquire samples on the Martian surface. The Hero testbed is an indoor platform that has been used to drive the architecture design. The current capabilities of the Hero include collecting cups in the laboratory and recharging itself.

Our initial implementation on the Hero robot [12], which was developed in an ad hoc manner, had several shortcomings. It was slow and slack in reacting to environmental changes. It could not protect itself and recover from failures properly. It also could not change its focus to higher-priority tasks or respond to requests from human advisors. After re-implementing the testbed using mechanisms and functions provided by TCA, most of these shortcomings have been minimized. The robot is now faster and more robust. It can react to environmental changes in a reasonable time frame, and it has a variety of strategies to recover from failures.

The following are the capabilities that TCA currently supports.

- **Concurrent planning and execution.** Robots often take a significant amount of time in constructing plans. Since planning and execution are activities that often need different resources, both can occur concurrently. However, this concurrency sometimes needs to be constrained. In many cases, the robot must act on an incomplete plan and defer some specific decisions until more information can be acquired. On the other hand, to minimize risk to the robot, one might want to completely plan out a goal before executing any of its sub-commands.
- **Reacting to environmental changes.** To accomplish tasks, and even to survive, the robot must be reactive. It must always be aware of environmental changes, and respond to them appropriately and in a timely manner. Some environmental changes invalidate current plans, while others may demand the robot to change its focus completely.
- **Error recovery.** In complicated, changing environments, failures are bound to occur. When they do occur, the robot must change its plan to meet the new situation. Error recovery is often context-dependent, that is, the same failure may have to be handled differently, depending on the robot's intentions. Since in a benign environment, the failed plan is often close to being correct, it is desirable for the robot to be able to fix and re-use the problematic plan, instead of always replanning from scratch.
- **Coordinating Multiple Tasks.** With many simultaneous goals but limited resources, the robot must be able to dynamically prioritize and schedule its various tasks based on their urgency, relative costs, likelihoods of success, etc. Currently, only simple-minded strategies can be specified using TCA, but we envision taking a more knowledge intensive approach in the near future.

Various TCA mechanisms have been developed to support these capabilities.

- **Distributed processing.** TCA is a distributed architecture with centralized control. A robot system using TCA includes a central control and a number of concurrent, application-specific processes. We believe that a centralized control architecture facilitates the coordination of multiple complex robot behaviors, while the distributed processing allows for concurrency in planning, execution, and perception.

- **Resources.** TCA provides a mechanism to schedule the use of the robot's limited computational and physical resources. A task is automatically queued by TCA until the needed resources are available. Resource reservation, together with temporal constraints (see below), provide synchronization mechanisms to control distributed robot systems.
- **Task trees and temporal constraints.** In TCA, planning and execution are separate activities and can be performed concurrently. The interleaving of these activities can be constrained by imposing temporal constraints among the planning and achievement times of subgoals. TCA explicitly maintains the goal/subgoal hierarchies, called *task trees*. Task trees, together with the temporal constraints, are TCA's representation of plans.
- **Concurrent monitors.** Concurrent monitors enable the robot to watch for environmental changes in parallel with normal task execution. Because task execution and monitoring occur concurrently, the performance of tasks will not be (significantly) slowed down, while still enabling environmental changes to be detected as early as possible.
- **Exception handling.** TCA provides a general mechanism for handling planning time failures, execution time errors, and contingencies. The robot implementor can specify different strategies for handling the same exception in different contexts. One benefit of having this mechanism is to allow the user to separate robot behaviors for normal situations from these that handle failures or contingencies. In this way, complex robot behaviors can be developed incrementally, and exception handling can be flexibly defined. At present, the mechanism is still under construction but some primary results have been obtained.

Table 1-1 summarizes the supporting relationships between the TCA mechanisms and desired robot capabilities. A mark "X" in an entry of the table indicate that the mechanism in that column is used to support the capability in that row. Note that although synchronization by itself is not a capability needed by robots, it plays an important role in the distributed environment of TCA.

Table 1-1: The supporting relationships between mechanisms and capabilities

Mechanisms / Capabilities	Distributed Processing	Resources	Task Trees & Temporal Constraints	Concurrent Monitors	Exception Handling
Synchronization		X	X		
Concurrent Planning & Execution	X		X		
Reacting to Changes	X			X	X
Error Recovery			X		X
Coordinating Multiple Tasks		X	X		

The rest of this paper presents the Hero robot system, the Task Control Architecture, and their performance. Section 2 describes the hardware setup of the system and gives a scenario to illustrate how the Hero robot performs tasks. Section 3 discusses the various mechanisms of TCA. Section 4 describes the robot system in detail. Performance of the robot and TCA is evaluated in Section 5. Comparisons with related work are given in Section 6. Finally the paper is concluded in Section 7.

2. Scenario

Our mobile manipulator robot, the Heath/Zenith Hero 2000, is a commercially available wheeled robot with a two-finger hand (see Figure 2-1). The robot operates in an unstructured laboratory, which is observable through a ceiling-mounted camera (see Figure 2-2). The Hero robot has three sonar sensors: a rotating sonar on top, a forward-pointing sonar fixed to its base, and one mounted on the robot's hand which can be repositioned relative to the body. In addition, the robot has a battery charge level sensor, a rotating light intensity sensor, and touch sensors on the fingers. Using existing vision software [10], we developed a 2D vision subsystem for the ceiling camera. We also developed algorithms for navigation and manipulation in the indoor environment.

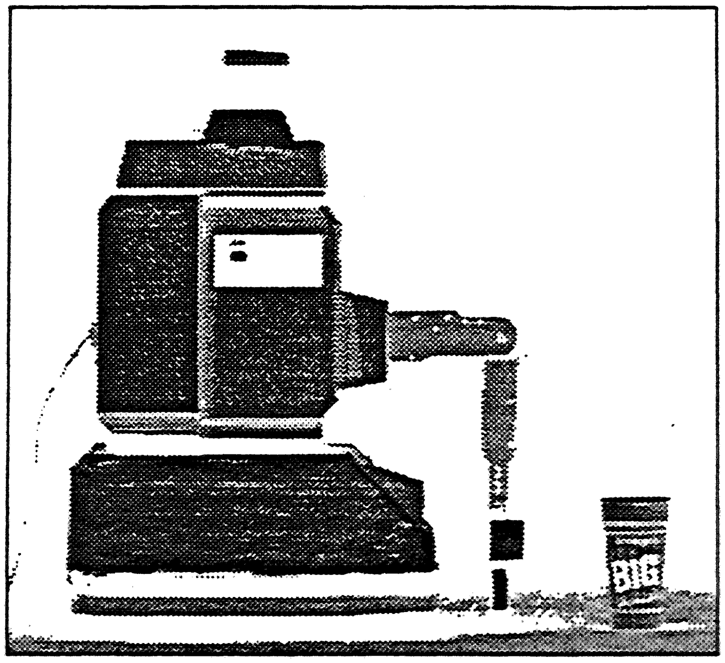


Figure 2-1: The Hero 2000 Robot

When the system is started up, the robot is given several high-level goals, including (1) collecting cups discovered on the lab floor and placing them in a receptacle, (2) avoiding obstacles, and (3) recharging its battery when necessary. The rest of this section presents a scenario to illustrate how the robot achieves and coordinates these goals.

For the cup collection task, the robot monitors its 2D vision map for the appearance of cups on the floor. An asynchronous perception process continually takes a picture and updates a world map. Once a new map is built, the robot scans the map to find cup-like objects. In this scenario, two cup-like objects are spotted, and the system sets up two *cup-collection* goals and temporally orders them so that the closer object will be explored first.

The robot then plans and executes a path to the first object. While moving, it monitors for obstacles in its path. A monitor, whose temporal extent continues until the object is picked up, is created to ensure that the target object does not disappear (e.g., someone else may pick it up). Upon arriving near the object, the robot uses its wrist sonar to measure the height and width of the object and matches them against its cup models. If a satisfactory match is found, the robot plans and executes actions to pick up the object. In parallel with measuring and picking up the object, the robot uses its overhead vision map to pre-plan a path to the receptacle so that a path plan is ready for execution when the cup is picked up. The

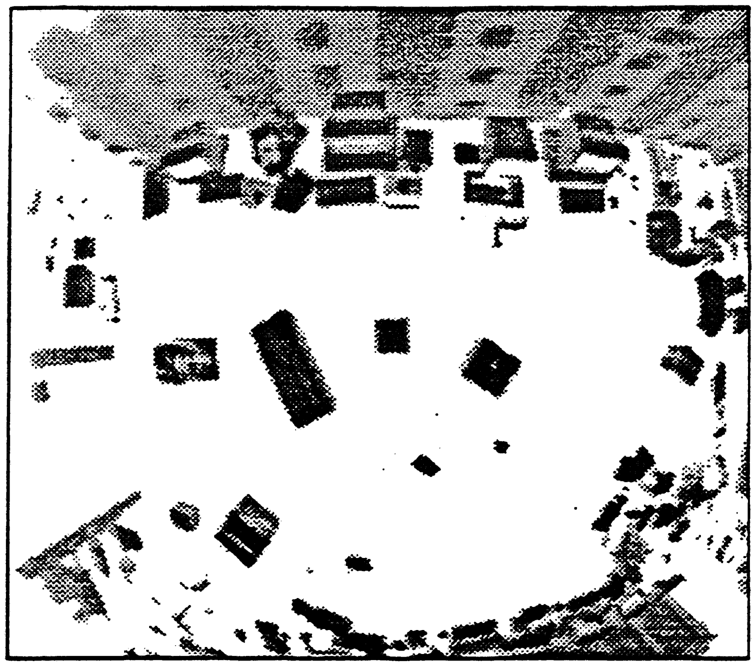


Figure 2-2: Overhead View of Laboratory as Seen by Robot

robot then uses the plan to navigate to the receptacle, where it deposits the cup.

Next, the robot attends to collecting the other object. While moving toward the object, the robot notices (from either its overhead vision or its sonar sensors) that an object appears in into its path. The robot stops immediately and waits to see if the object will move away soon. If the obstacle does not move, the robot plans a detour by modifying the blocked path plan. If no detour can be found, the robot replans a path from scratch. If still no path can be found, the robot abandons this cup-collection goal.

In this scenario a detour is found, so the robot continues to navigate to the object. The robot finally arrives near the object and starts measuring it. At this point, the battery charge monitor notifies the robot that its battery charge is getting low. Based on the simple-minded strategy: "if the robot has arrived near the object complete the task before going over to recharge", the robot creates a *recharge* goal with temporal constraints indicating that the new goal will be attended to after the cup-collection goal is achieved or aborted. The robot continues and subsequently discovers that the object is not a cup at all. It gracefully terminates all ongoing and pending activities and monitors that were set up for collecting the object, and then it chooses to pursue its next goal, which is the *recharge* goal.

3. The Task Control Architecture

TCA is designed to implement capabilities we believe to be necessary for autonomous robots. TCA is a distributed architecture with centralized control. An application of TCA includes a central process and a number of concurrent, application-specific processes, called *modules*. Communication occurs via coarse-grained message passing between modules, with all messages being routed through the central process.

To facilitate experimentation with different control schemes, TCA is built as a layered system so that an implementor can choose which layers to use -- higher layers provide more functionality specific to robot control, but lower layers provide flexibility to implement alternative control schemes.

At present, the implemented layers include:

- Communication layer that supports distributed processes under centralized control;
- Behavior layer for querying the environment, specifying goals, executing commands, and altering the robot's internal state
- Resource layer for allocating and managing physical and computational resources;
- Task management layer for building hierarchical plan structures and specifying temporal constraints between planning and execution of various goals in the plan;
- Monitor layer for concurrently monitoring user-selected aspects of the robot's external and internal environments;
- Exception handling layer for specifying context-dependent strategies for handling plan failures, execution errors, and environmental changes.

In addition, other layers to support multi-task coordination and user interaction are planned.

3.1. Communication Layer

The base layer of functionality provided by TCA is the sending and receiving of messages between modules. Modules can be written in different languages (currently both Lisp and C are supported) and run on different machines (using the UNIX TCP protocol). In essence, TCA provides a simple *remote procedure call*(RPC) interface from a caller in one module to a procedure in a possibly remote module. The main difference between typical RPC implementation and TCA is that the central control determines which module handles messages and in what order they are handled.

A potential problem with centralized control is that the central process may become a bottleneck. Experimentally, a round-trip time for messages of under 10K bytes is about 50 milliseconds. Since this time is small compared with the time taken by image processing, planning, and the robot's actuators, the centralized control has not been a problem on our current testbeds. Besides, the potential bottleneck problem can be overcome by using high-speed hardware (e.g., the Nectar [2]) and adhering to some conventions, such as using coarse-grained behaviors to limit the amount of module-to-module communication.

3.2. Behavior Layer

TCA provides several types of primitive building blocks needed to construct robot behaviors. The primitive behaviors are implemented as different classes of messages, built on top of the communication layer. The classes differ mainly in their control flow. For example, query messages block the user's code until a reply is received, while goal and command messages are non-blocking and report success or failure directly to the central control.

- **Query messages** are requests to provide information about the external or internal environment, such as obtaining a world map or determining the robot's dead-reckoned position.
- **Goal messages** are intended to support top-down, hierarchical planning. A typical response to a goal message would be to issue other (sub)goal and/or command messages based on the results of planning. Unlike queries, goal messages are asynchronous and non-blocking. That is, the central control may queue the goal until resources become available; in the meanwhile, the module sending the goal message can continue. The rationale is that non-blocking goal messages give the implementor greater flexibility in controlling the achievement of goals

(e.g., interleaving planning and execution).

- **Command messages** are used to execute actions. Like goal messages, command messages are asynchronous and non-blocking. Distinguishing goal from command messages is done mainly for interleaving planning and execution.
- **Constraint messages** provide a way to alter the robot's internal state. For example, constraint messages can be used to add expectations about its future behaviors.

3.3. Resource Layer

It is crucial for an autonomous agent to effectively allocate its limited resources in order to satisfy its goals. The robot must detect when tasks need competing resources, and must prioritize and schedule tasks when conflicts occur. In TCA, a resource is an abstract entity that is used to manage the handling of messages. A resource may be associated with a computational entity, such as a module, or with a physical entity, such as a motor or camera.

Resources are created with a capacity - the number of messages the resource can handle simultaneously. A message received by the central control is queued until the resource that handles the message has available capacity. Currently, messages to the same resource are handled in FIFO order, subject to the temporal constraints imposed by the task management layer.[1]

Sometimes, a module might need control over a resource for some period of time, particularly one associated with a physical item. For example, if a vision module is acquiring an image, it might want to ensure that the robot does not move during that period. To facilitate this, TCA includes mechanisms for reserving resources, in effect, preventing other modules from utilizing the resource until the reservation is explicitly canceled. Resource reservation is one of the synchronization constructs in TCA.

3.4. Task Management Layer

The task management layer provides mechanisms for organizing sets of messages into hierarchical *task trees* (see Figure 3-1). For each goal, command, or monitor message sent by a module, TCA adds a node to the task tree as a child of the node that issued the message. The resulting tree is an execution of graph of messages used to complete a given task. In addition, facilities have been developed for tracing and manipulating the task tree, such as killing off subtrees, suspending them, and adding new nodes. These facilities will provide functionalities needed by some of the higher layers, such as the exception handling layer (see Section 3.6) and the planned multi-task coordination layer.

Another important purpose of this layer is for scheduling tasks. The layer contains a general facility for reasoning about time. In TCA, by default planning and execution can occur concurrently. Interleaving of planning and execution can be constrained by imposing temporal constraints on the planning times of goals and achievement times of goals, commands, and monitors. For example, a module might specify that the achievement time of G1 precedes that of G2, but the planning time of G2 precedes that of G1 (e.g., first achieve pick up the cup, then bring it to the receptacle, but plan the route to the receptacle before planning how to pick up the cup). Similarly, a module might constrain a goal to be completely planned before any of its sub-commands can start being achieved.

[1]We plan to add more sophisticated scheduling mechanisms in the future.

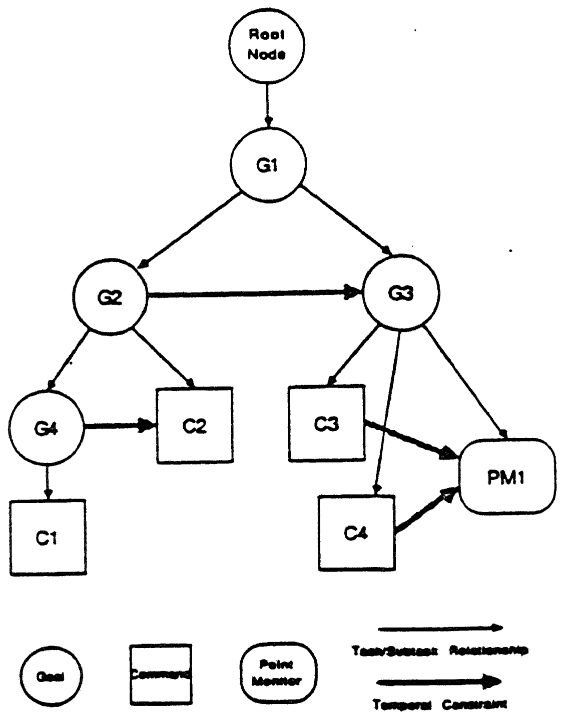


Figure 3-1: Sample task tree

The mechanisms for reasoning about temporal constraints are based on the *Quantity Lattice* [13], an arithmetic reasoning system, that integrates *relationships, arithmetic expressions, qualitative* and *quantitative*[2] information to perform a wide range of common arithmetic inferences. In TCA, it is used to maintain a consistent partial order of time points and to answer queries about relationships between time points and about the durations of intervals.

With the temporal mechanisms provided, robot implementors can formulate a fairly wide range of different constraints to take advantage of concurrencies in the distributed environment of TCA. Together with resource reservation, the temporal constraints provide synchronization mechanisms to control distributed robot systems.

3.5. Monitor Layer

To react to environmental changes, robots must first be able to monitor the environment and detect changes in time. Although in the real world many things may go wrong at any time, robots with limited sensory resources, such as ours, cannot afford to monitor everything that goes on in the environment. The monitor layer provides mechanisms to monitor user-selected aspects of the environment and report detected changes to the central control for handling. Monitors in TCA run concurrently with normal task execution. For example, the Hero robot attends to the cup collection goal while monitoring for obstacles and its battery charge.

A monitor specifies the condition to be monitored, and the time, relative to other messages, when monitoring is to take place. When the condition holds, a typical action would be to send an exception message to the central control, which will decide what to do based on the environment and context in

[2]The quantitative reasoning capability of the Quantity Lattice is not yet utilized by TCA.

which the exception occurred (see Section 3.6).

Two classes of monitors are implemented: *point monitors* and *interval monitors*. Point monitors, which test the monitor's condition just once, are useful for checking static, execution time conditions, such as checking the pre-condition or post-condition of a command or goal. Interval monitors, which have a temporal extent, are useful for checking for environmental changes over time.

TCA has two variations of interval monitors: *polling* and *demon monitors*. Polling monitors implement synchronous polling of conditions at a fixed frequency, while demon monitors implement asynchronous demon-invocation. For instance, the battery monitor of the Hero robot, which is a polling monitor, periodically checks the battery charger and raises an exception if a low charge is detected. The cup appearance monitor, implemented as a demon monitor, is invoked whenever a world map is updated by the asynchronous perception process, and checks the world map for cup-like objects, raising exceptions if such objects are found.

Monitors can also be used to construct conditional plans. For instance, suppose there are two strategies to achieve goal G, but we do not know in advance which one will be applicable. We can set up a monitor to check the environment and choose the appropriate strategy at execution time.

3.6. Exception Handling Layer[3]

Exceptions can be divided into three classes, according to the ways they are detected.

- failures detected in planning (e.g., no path to the cup);
- errors detected in executing commands (e.g., wheel slippage);
- contingencies detected by monitors (e.g., low battery charge).

TCA employs the same mechanisms to handle the three different types of exceptions.

Exception handling is often context-dependent: the same exception might need be handled differently, depending on the environment and where in the plan the exception occurs. For example, a wheel blockage is a failure if it is detected when the robot is navigating in an open space. But it could be a signal of a successful docking if the robot's goal is to dock on the charger. To facilitate context-dependent exception handling, TCA supports mechanisms for associating exception handlers with contexts at planning time and automatically invoking the handlers when exceptions are raised. Various utilities are also provided to enable handlers to fix problematic plans.

The context of an exception handler is established by attaching the handler to a task tree node. This association is done dynamically as the task tree is created. When an exception is raised, TCA searches up the task tree, starting from the node where the exception arose, to find a handler specific to that exception. The first matched handler is then invoked to handle the exception.

Exception handling is achieved by editing the task tree, for example, by deleting part of it and inserting some new nodes. The exception handlers can use the task tree operations provided by the task management layer to access, scrutinize, and then modify the task tree. Modifications to task trees may include terminating or suspending the execution of subtrees, and adding new nodes to the task tree, which

[3]Currently, only the framework of the exception handling layer has been implemented, and various supporting mechanisms are still under construction.

is then expanded using the normal TCA mechanisms. To illustrate, Figure 3-2(a) shows a situation where a *battery charge* monitor is set up and the robot is actively attending to the *cup-collection* goal. When the monitor detects a low battery charge, the *low battery charge handler* attached to the root node is chosen to handle it. After checking the battery charge and the progress of the cup collection, the handler decides to recharge first and finally ends up with the situation in Figure 3-2(b), where the monitor has been canceled, the *cup-collection* goal has been suspended, and the *recharge* goal has been added and become the current goal.

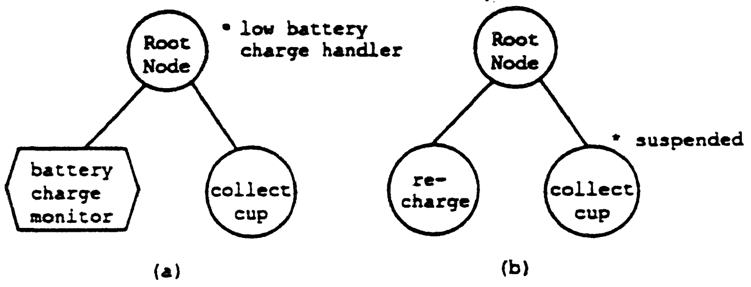


Figure 3-2: Exception handling

If an exception handler finds it cannot actually handle the situation, it can raise an exception itself. When the central control receives an exception from an exception handler, the search for a capable handler is resumed, starting from the node where the previous handler was found and searching up the task tree. This process is repeated until the exception is successfully handled. As a catchall, TCA attaches a general exception handler to the root node of the task tree. When invoked, this general handler simply deletes the failed task along with all its subtasks.

This TCA approach to exception handling is efficient. First, the invocation of exception handlers is fast, because only a simple search on the task tree is involved. Second, TCA allows a problematic plan to be fixed and re-used as much as possible. For example, when moving obstacles appear unexpectedly, the Hero robot first waits for obstacles to move away. If they do not move away, it tries to plan a detour by modifying the blocked path plan. If no detour is found, a new path is planned from scratch. Only if no path is found is the task terminated.

4. The Hero Robot System

The Hero robot system, which uses TCA, presently consists of five modules plus the central control (see Figure 4-1). In this section, we describe the functionalities of the modules and how they interact with each other.

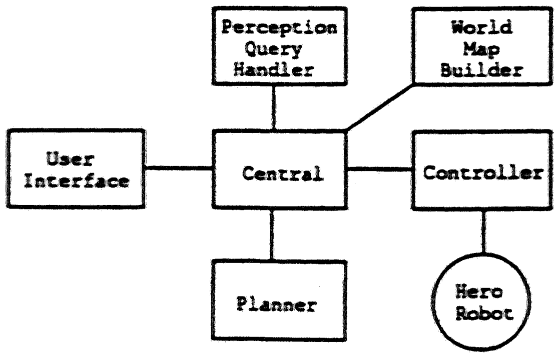


Figure 4-1: Organization of the robot testbed

Controller. This module, which controls the robot via either a radio link or an RS232 cable, executes navigation commands (e.g., turn, move) and manipulation commands (e.g., raise arm, open grippers). It also handles queries that involve using sensors on the robot, for example, reading the battery charge level, and measuring the height of an object using the wrist sonar.

The Controller also keeps track of the robot's trajectory and handles trajectory queries. Because of the control error, the uncertainty about the robot's position will grow over time. The Controller utilizes a covariance matrix representation [16] to model the control error, and compounds the uncertainty whenever the robot moves or turns. This uncertainty information is primarily used by the Perception Query Handler to determine the likelihood of hitting obstacles in the course of navigation.

We also implemented reflexive guarded move commands directly on-board the Hero. These give the robot a higher degree of reactivity than could be gotten from centralized control. While the robot is moving or turning, the on-board CPU detects wheel slippage and blockage by monitoring the motor encoders. At the same time, the sonar sensors are used to detect obstacles in the robot's trajectory. In both cases, the reflex action is to stop the robot immediately, stabilizing it. Then the Controller signals a failure so that the system can rectify the situation using the exception handling mechanisms.

World Map Builder. This module continually takes and processes images of the lab (every 20 seconds or so), and updates a world map, which is then forwarded to the Perception Query Handler. We have found that this asynchronous process has substantially increased the performance of the robot compared with our previous system. For example, since a relatively up-to-date world map is always available, the robot does not need to wait for processing an image in order to find a cup-like object or to plan a path.

To identify the robot in the image, the World Map Builder first gets the robot's dead-reckoned trajectory from the Controller. Based on the trajectory and other information such as the size of the robot, the robot region can often be distinguished from other object regions. Two failures, however, can be encountered. First, the robot may not be successfully spotted, because the robot region, for example, overlaps another visual region. This failure is handled by taking an image, moving the robot a few inches, taking another image, and comparing the differences in the images to spot the robot. The second failure occurs when the light in the lab is turned off. This exception is handled by asking humans to turn on the light or going to sleep (i.e., turning off the power to all circuitry except the memory) if no help is secured.

Perception Query Handler. The Perception Query Handler provides three kinds of functionality. First, it updates the world map upon receiving a new map from the World Map Builder. Second, it handles perception demons. When a new world map is received, perception demons are invoked to check conditions that they monitor. Presently there are two kinds of demons that can be set up - cup appearance monitors and object monitors (for checking if an object remains at a specified position on the floor).

The third task of this module is to handle perception queries, including

- calculating the vicinity of an object in order to approach it,
- checking if a path is clear, based on uncertainty reasoning,
- reducing the uncertainty about the robot's location and orientation by using vision.

As mentioned previously, the Controller explicitly models the uncertainty of the robot's status. When the robot is executing a path plan, the Perception Query Handler, given the uncertainty information, would be asked to determine (1) if the path is clear, (2) if yes, how far the robot may safely proceed along the path before the uncertainty cone overlaps object regions (see Figure 4-2). If the uncertainty has grown to the extent that collisions with obstacles are possible, the Perception Query Handler uses vision to reduce the uncertainty. To do this, it first takes a picture of the robot and calculates the robot status (including visual uncertainty) based on properties of the robot's shape and internal model of sensor uncertainty. A new robot status is then obtained by merging the observed and expected status [16].

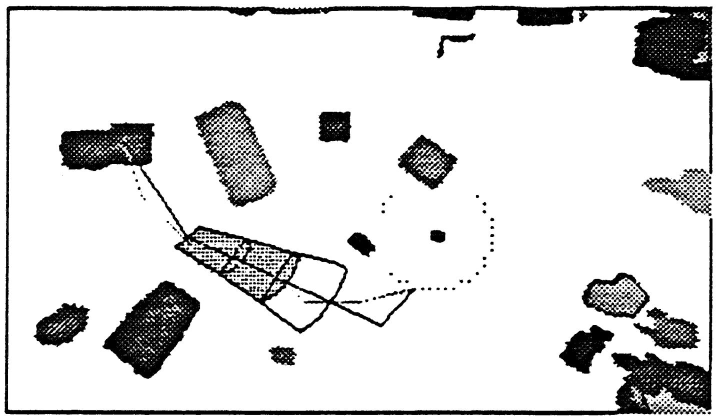


Figure 4-2:
Interpreted Version of the Image from Figure 2-2 with Planned Path and Uncertainty Cone. The brightened line shows the final computed path to a cup-like object, while the dimmer line is the original path before optimization. The shaded area in the uncertainty cone indicates how far the robot may safely proceed.

Planner. At present most of the navigation and manipulation planning is done in this module. The Planner has a collection of procedures, each of which is intended to achieve a goal. When executed to achieve goals, the procedures typically send queries, create subgoals, issue commands, set up monitors, specify temporal constraints, and/or associate exception handlers with contexts.

As an example, the procedure for handling the cup collection goal does the following:

1. Adds *approach object* goal. The first step is to navigate to the vicinity of the target object. In the course of navigation, the robot models uncertainty and watches out for obstacles.
2. Sets up *object monitor*. This monitor watches for the disappearance of the target object. Temporal constraints are added to indicate that the monitor starts from the beginning of the cup collection goal and ends at the beginning of the *grasp cup* goal (see below).
3. Adds *servo to object* goal. Once arriving near the object, the robot utilizes its wrist sonar to estimate its distance and orientation relative to the object. This information is used to compute the locomotion commands to reduce the differences between the estimated and desired distance and orientation. To overcome sensing and control errors, this goal is re-generated recursively until the differences are within acceptable limits. This recursive implementation makes it possible to break the time-consuming servoing loop for handling contingencies.
4. Adds *identify object* goal to measure and classify the object.
5. Adds *grasp cup* goal. If the object is a cup, it is grasped by a procedure specific to that cup. A point monitor, which utilizes the base sonar, is set up for checking if the grasping

succeeds.

6. Adds *approach receptacle* goal. Once picked up, the cup is brought to the receptacle. However, temporal constraints are imposed so that the path planning can begin once the robot arrives near the cup.
7. Sets up *holding monitor*. This interval monitor periodically reads the sensors on the fingers to make sure that the cup does not drop on the way to the receptacle.
8. Adds *deposit* command to drop off the cup in the receptacle.
9. Associates appropriate exception handlers to various task tree nodes.

User Interface. Presently the User Interface merely allows the user to enter commands, add goals, and set up monitors. Facilities for supporting a friendly user interface are being planned.

5. Performance

Our experience with the testbed shows that TCA is a helpful tool for building robot behaviors.

- TCA is easy to use and programs developed under TCA are usually easy to extend and modify. This is partly because TCA encourages modularity of programs. For example, normal robot behaviors, monitors, and exception handling can be developed separately.
- TCA provides a fair amount of expressive power to facilitate implementing complex robot behaviors. For example, TCA makes it easy to specify and control the interleaving of planning and execution, concurrent monitors, and exception handling.

Due to its deliberative nature, TCA cannot be used to implement low-level reflex behaviors that demand sub-second responses to environmental changes. To minimize the interval between the time an exception is detected and the time the exception handler gets executed, the implementors themselves must adhere to a principle: each of the robot's primitive actions must be designed to finish in a small time frame. In other words, a time-consuming action must be repeatedly divided into smaller ones, so that each does not take much time. The reason is that when an exception is raised, the chosen exception handler might be blocked by other ongoing primitive actions, because of resource conflicts. If so, the handler must wait for these actions to finish. Guaranteed reactivity is an interesting research area and we plan to investigate it in the near future.

Roughly speaking, the robot system described above is quite successful in surviving, collecting cups, and maintaining battery charge. It typically takes about 3-5 minutes to collect a cup, depending on the difficulty of individual tasks (e.g., smaller cups usually demands more time). If a cup is placed away from the perimeter of the visual view and not occluded, the robot can locate and collect it most of the time. Although the vision subsystem can be easily fooled by small non-cup objects (e.g., small box, sneaker), those objects are usually identified as non-cups by the sonar sensors when the robot approaches the objects (but they can result in considerable wasted time).

The robot system is about twice as fast as the previous sequential version. This is mainly because the world map is updated by an asynchronous process; this is a big win, because image processing takes much time. Another speed-up results from concurrent monitors and concurrent planning and execution.

The robot system is also relatively robust compared with the previous version. This is mainly because the concurrent monitors enable exceptions to be found early and the robot has a variety of strategies for handling exceptions. It is also helped by the reflexive guarded commands and their integration into the

TCA mechanisms.

The robot, however, is still susceptible to dangers. These dangers mainly arise from the robot's inability in sensing. For example, the robot has no sensor to detect imminent arm collisions and prevents them in advance. The vision processing is slow, so the robot might use out-of-date information and make wrong decisions. Although these problems can be minimized (but not overcome) by adding more sensors and using faster hardware, that is not the purpose of this work.

6. Related Work

An alternative approach to building reactive and robust robots is that taken by the subsumption architecture [4]. The main features of this approach are (1) hard-wired, layered robot behaviors, (2) no explicit internal model of the world, (3) no explicit representation of goals and plans, (4) no central control, and (5) continual monitoring. Many of these characteristics are shared by some other approaches, such as [1] and [11]. In contrast to these architectures, TCA has a centralized control and makes the notion of goals explicit, allowing the robot to reason about them. These differences make TCA more flexible in coordinating complex robot behaviors. The use of explicit plan representations enables TCA to pre-plan for the future, not just figure out "what to do next". TCA advocates selective monitoring, because sensors are often scarce resources and the use of them should be carefully scheduled. These differences result in two architectures with very different capabilities [6]. While the subsumption architecture is good at handling low-level sensor and effector actions (e.g., car chasing), it is not yet clear how complex behaviors (e.g., planning, exception handling) can be coordinated in the architecture. On the other hand, while with TCA fairly complex behaviors have been realized on the Hero robot, it is not well-suited to handling low-level reflex activities. Rather than competing architectures, however, it is reasonable to combine the strengths of both approaches, for example, by using the subsumption architecture for reflexive control, which talks to TCA for higher-level control. In fact, our experience with the guarded move commands (see Section 4) suggests that this might be a promising way to implement robust, intelligent robots.

The Procedural Reasoning System (PRS) [7] consists of four main components: a database of beliefs about the world, a goal stack, a library of procedural plans, and an interpreter. PRS is similar to TCA in several aspects. For example, both are concerned with combining planful, reasoned behaviors with reactivity. The goal stack and procedural plan representation used in PRS is similar to our task tree structure plus temporal constraints. The main difference between the two systems is that PRS is more concerned with reasoning and planning, while TCA mainly focuses on the execution, monitoring, and exception handling.

The Reactive Action Package (RAP) system [5], which is very similar to PRS, is another work which addresses reactivity and adaptive execution of plans. Like TCA, the RAP system provides various mechanisms for supporting resource reservation, temporal constraints, monitoring, and exception handling. The RAP system, which is a sequential system, is based on the idea of situation-driven execution, much like the subsumption architecture. This viewpoint is different from that of TCA. While supporting reactivity, TCA still allows the robot to plan for the future. For example, the Hero robot can measure the potential cup, monitor its battery charge, and pre-plan the path to the receptacle concurrently. Both systems also differ in the ways exceptions are handled. When exceptions are raised, the RAP system examines the context at run-time to find the appropriate method for re-achieving the failed task, while in TCA only a simple search on the task tree is needed.

The exception handling mechanisms of TCA are similar to those in some programming languages such as Ada [9] -- when an exception occurs, program execution is transferred to the exception handler with a matched name that is closest to the exception point in the context (i.e., the runtime call-stack in Ada or the task trees in TCA). However, they differ in three aspects. First, TCA allows the exception handlers to manipulate the task trees explicitly, while explicit manipulation of the call-stack in Ada is prohibited. Second, popping and pushing the call-stack is always simpler than killing and adding new subtrees, because of the temporal constraints placed on the task trees. Maintaining the desired temporal constraints between tree nodes while modifying the task trees is a difficult problem, which we have not solved completely. Third, task tree nodes are not killed while TCA is searching for capable handlers, so the exception handlers can examine the failed node and its ancestors to help in debugging [14].

7. Conclusion

We have designed and implemented TCA, a general-purpose task control architecture, for the control of mobile robots. TCA is designed to be used for robots with multiple tasks, and limited computational and physical resources, that operate in an uncertain and changing, but relatively benign, environment. The design of TCA is based partly on experience gained from our first version of the Hero testbed. That version, developed in an ad hoc manner, had several shortcomings, such as brittleness, unawareness of environmental changes, etc. By using TCA, we have re-implemented the system in a more disciplined way. The current robot can navigate in a changing (indoor) environment, avoid obstacles, collect cups on the floor, and at the same time watch for failures and contingencies, recover from failures, and go recharge when necessary.

The features of TCA that result in the Hero robot's success and that, we believe, will facilitate the building of intelligent, robust robots are (1) distributed processing, (2) resources, (3) task trees and temporal constraints, (4) concurrent monitors, and (5) context-dependent exception handling. The distributed environment enables robot activities such as planning, sensory data processing, monitoring, and plan execution, to be performed concurrently. The resource mechanisms enable robots to schedule the use of their limited resources. By using the temporal mechanisms, the user can implement intelligent robots that are able to act on an incomplete plan when not enough information is available to make a decision, and to take advantage of parallelism by planning ahead when needed information is obtainable. Concurrent monitors, which allow robots to acquire information from the environment while executing tasks, gives robots the opportunity of reacting to environmental changes and changing their focus for contingencies or opportunities. The exception handling mechanisms enable robots to dynamically choose context-dependent strategies for handling contingencies, planning time failures, and execution time errors. The mechanisms also allow robots to re-use a failed plan by making changes in it, or even to change their focus completely.

Another important feature of TCA is that it facilitates modular and incremental design of complex robot systems. In TCA, planning, execution, monitoring, and exception handling are all logically and functionally separate activities. This enables one to build systems incrementally -- first building behaviors that plan and execute, then adding features (usually by adding new code with few changes to the existing programs) to take advantage of concurrency in planning and execution, to monitor for exceptional situations, and to handle those situations intelligently.

Despite these encouraging results, much more work remains to be done. In particular, we plan to extend TCA to support various knowledge-intensive decision-making capabilities [8], such as, scheduling various tasks based on their urgency and relative cost, choosing optimal plans based on the analysis of various plans' strength, limitation, resource usages, time constraints, etc.

Although building complex, robust robot systems is still very much an art, we believe that with the use of high-level architectures, such as TCA, we can make the process easier. Through experience with different robot systems (the CMU planetary Rover also uses TCA), and analysis of the requirements for different environments and robot configurations, we are converging on a set of mechanisms to support the building of such robot systems.

8. Acknowledgements

We express our gratitude to Robert Eric Wolpov, who has been adding new sensors to the Hero robots and maintaining them in good health. We thank Tom Mitchell and Andrew Phillips, who helped develop the first version of the testbed. Lonnie Chrisman has provided valuable comments on TCA design and Kevin Ryan helped develop the guarded commands. We are grateful to them. We also thank John Allen, Jim Moody, and Steve Shafer for their assistance in setting up the testbed. This research has been supported by NASA under Contract NAGW-1175.

References

[1] Agre, P.E., Chapman, D.
Pengi: An Implementation of a Theory of Activity.
In *Proceedings of AAAI-87*, pages 268-272. 1987.

[2] Arnould, E.A., Bitz, F.J., Cooper, E.C., Kung, H.T., Sansom, R.D., and Steenkiste, P.A.
The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers.
Technical Report, CMU-CS-89-101, Carnegie Mellon University, 1989.

[3] Bares, J., et al.
Ambler: An Autonomous Rover for Planetary Exploration.
In *IEEE Computer, Vol. 22, No. 6.* 1989.

[4] Brooks, R.A.
A Robust Layered Control System for a Mobile Robot.
In *IEEE Journal of Robots and Automation, vol. RA-2, no. 1.* 1986.

[5] Firby, R.J.
Adaptive Execution in Complex Dynamic Worlds.
Technical Report, YALEU/CSD/RR #672, Yale University, 1989.

[6] Flynn, A.M., and Brooks, R.A.
MIT Mobile Robots - What's Next?
In *Proceedings IEEE Robotics and Automation*, pages 611-617. April, 1986.

[7] Georgeff, M.P.
A System for Reasoning in Dynamic Domains: Fault Diagnosis on the Space Shuttle.
Tech Note 475, AI Center, SRI International, 1986.

[8] Georgeff, M.P., Ingrand, F.F.
Decision-Making in an Embedded Reasoning System.
In *Proceedings of IJCAI-89*, pages 972-978. 1989.

[9] Habermann, A.N., and Perry, D.E.
Ada for Experienced Programmers.
Addison-Wesley Publishing Company, Inc., 1983.

[10] Hamey, L., Printz, H., Reece, D., and Shafer, S.A.
A Programmer's Guide to the Generalized Image Library
Carnegie Mellon University, 1987.

[11] Kaelbling, L.P.
An Architecture for Intelligent Reactive Systems.
Tech Note 400, AI Center, SRI International, 1986.

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

[13] Simmons, R.
Commonsense Arithmetic Reasoning.
In *Proceedings of AAAI-86*, pages 118-124. 1986.

[14] Simmons, R.
A Theory of Debugging Plans and Interpretations.
In *Proceedings of AAAI-88.* 1988.

[15] Simmons, R., Mitchell, T.M.
A Task Control Architecture for Mobile Robots.
In *Stanford Spring Symposium.* 1989.

[16] Smith, R.C., and Cheeseman, P.
On the Representation and Estimation of Spatial Uncertainty.
In *The International Journal of Robotics Research*, pages 56-68. 1986.