Concurrent Planning and Execution for Autonomous Robots

Reid Simmons

The performance of autonomous robots may be increased significantly by concurrently planning and executing actions. The *Task Control Architecture (TCA)* provides communication and coordination facilities to construct distributed, concurrent robotic systems. This paper describes the use of TCA in a system that walks a legged robot through rugged terrain. The walking system, as originally implemented, had a sequential sense-plan-act control cycle. Utilizing TCA features for task sequencing and monitoring, the system was modified to concurrently plan and execute steps. Walking speed improved by over 30%, with only a relatively modest conversion effort.

Coordinating Planning and Execution

The coordination of planning and execution is a significant problem for autonomous robotic systems. To achieve continuous motion, the robot must decide what to do next by the time its previous actions are completed. This real-time constraint can often be met through the use of concurrency: while one step is executing, the system can plan the next action.

The problem remains that the planning and execution tasks must be tightly coordinated. Concurrent processes must be prevented from temporally interacting, particularly when they contend for common resources. The *Task Control Architecture (TCA)* has been developed to facilitate this and other robotic control problems [1]. TCA provides a general framework of hierarchical task decomposition augmented with reactive behaviors, plus utilities for communication, coordination, and monitoring of distributed robotic systems. In particular, explicit temporal constraints between subtasks are used to synchronize processes.

The Task Control Architecture has been used in conjunction with the Ambler, a six-legged robot designed for planetary exploration (Fig. 1). TCA integrates perception, planning, and real-time control to autonomously navigate the Ambler over rugged terrain [2], [3]. The control architecture for the Ambler is quite deliberative: a host of geometric and terrain constraints on the robot's motion are analyzed in order to maximize the safety and energy efficiency of planned moves. In contrast, many walking systems utilize a more reactive approach, in which relatively simple rules are used to choose actions based on the current environment [4]-[6]. A deliberative architecture was chosen for the Ambler because reliability and power consumption are overriding concerns for planetary missions; it is a feasible architecture because the Ambler is statically stable (c.f., [7]).

One result of the Ambler's deliberative nature is that planning a step takes an appreciable amount of time. In particular, the walking system was initially developed with a sequential sense-plan-act cycle, which left the robot idle for much of the time. In analyzing the system performance, it was clear that concurrent planning and execution could result in continuous, or nearly continuous, motion of the mechanism.

Concurrency was achieved by modifying the original system, using facilities provided by TCA for specifying, executing, and monitoring hierarchical plans. The average walking speed improved by over 30%, with competence comparable to the sequential system. Notably, by using TCA the conversion effort involved only minor modifications to the existing software. Specifically, design, modification, and testing took only a few man-weeks of effort, compared to the several man-years needed to implement the original system.

Experience with the Ambler walking system supports the general strategy of first developing competent sequential systems, then increasing performance by adding concurrency. Sequential systems are easier to develop and debug, since temporal interactions are not a factor. Once developed, TCA can be used to facilitate conversion to concurrent operation. This same methodology has also been employed for an indoor mobile manipulator [8].

The next section briefly describes the Task Control Architecture and its salient features for supporting concurrency. The following section introduces the Ambler and the sequential walking system. The modifications to produce concurrent operation are then presented, followed by some empirical performance results.

The Task Control Architecture

The Task Control Architecture provides a general framework for controlling distributed robot systems [1], [8]. TCA can be thought of as a high-level robot operating system that provides common facilities needed by many robotic applications. In particular, TCA supports 1) distributed processing, 2) hierarchical task decomposition, 3) temporal synchronization of subtasks, 4) execution monitoring, 5) resource management, and 6) exception handling.

A robot system built using TCA consists of a number of task-specific *modules* (processes), and a general-purpose *central control* module (Fig. 2). The modules communicate with one another (and with the central control) by passing messages. The modules register message formats and message handling procedures with TCA, and the central control has responsibility for routing messages to the appropriate modules.

TCA provides several classes of messages. *Query* messages enable modules to obtain information from other modules. They are blocking, pending receipt of a reply. *Goal*, *command*, and *monitor* messages are used to create hierarchical plans and to react to plan failures. These messages are non-blocking, and the TCA central control takes responsibility for scheduling and executing the messages, and preempting them in case of plan failures.

TCA has several mechanisms for coordinating messages: resources, hierarchical task trees, and temporal constraints. A TCA

Revised version of a paper presented at the 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, April 7-12, 1991. This work was supported by NASA under Contract NAGW-1175. *R. Simmons is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.*

0272-1708/92/$03.00©1992IEEE

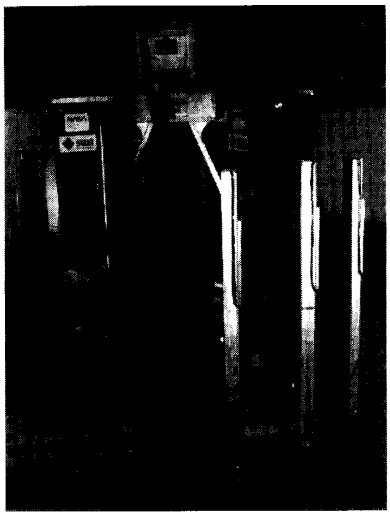

Fig. 1. The Ambler.

resource is defined by a set of message handling procedures and a capacity. By default, all messages handled by a module are grouped as a resource of unit capacity. This permits TCA to send only one message to a module at a time, with additional messages queued pending resource availability. A module can also *reserve* a resource, giving it exclusive access and preventing other modules from utilizing the resource until the reservation is released. This facility can be used to synchronize resources: a perception module might reserve the real-time controller module, for instance, to prevent the robot from moving while images are being acquired.

TCA directly supports the creation and execution of hierarchical plans. For each message sent, TCA records the relationship between sender (goal) and receiver (subgoal/command) in a hierarchical *task tree*. In Fig. 3., query messages are indicated by double-headed shaded arrows, while single-headed shaded arrows indicate task decomposition. These task trees are created dynamically, and can grow to arbitrary width and depth. Task trees also explicitly represent temporal constraints on the planning and execution of tasks (indicated by the thick horizontal arrows). These constraints indicate which tasks to schedule, with TCA queuing subtasks until temporally prior ones have completed.

A *sequential-achievement* constraint between two nodes is interpreted by TCA to mean that all the command and monitor messages under the first node (the leaves of its subtree) must be handled before any of those under the second node. For example, in the Ambler task tree (Fig. 3), the sequential-achievement constraint between the *placeLeg* goal and the *moveBody* command implies that the commands to move a leg must complete before the body can begin to propel forward.

A *delay-planning* temporal constraint indicates that TCA should delay handling a goal message (i.e., expanding it) until all commands and monitors under the previous node have been executed. Without this constraint, TCA would allow the goal message to proceed and create a plan, although, if a sequential-achievement constraint has been added, that plan will not actually be executed until the previous subtree has completed. For example, the delay-planning constraint between the *traverseArc* goals in Fig. 3 indicates that the steps for traversing the second arc should not be planned out until the robot has successfully navigated the first arc. (The Appendix presents a formalization of TCA's temporal constraints; TCA represents them using an arithmetic reasoning system called the *Quantity Lattice* [9].)

When advance planning is performed, execution monitoring and error recovery become necessary. TCA provides several different constructs for monitoring progress of a plan and for raising exceptions. In particular, a *point monitor* (Fig. 4) consists of a condition message (a query) and an action message (a goal or command). When the appropriate temporal constraints are met, the condition query is sent. If the condition holds, the action message is sent, which can replan by modifying parts of the task tree.

To facilitate error recovery, TCA provides utilities for modules to trace through and modify task trees. For example, a module can find the parent or children of a node in the tree. A module can also terminate execution of a subtree, and can insert new nodes and temporal constraints into the tree. Once added, the new nodes are expanded (i.e., planned out) and executed using the normal TCA mechanisms.

The Ambler

We have built and are currently testing the Ambler, a six-legged robot designed for planetary exploration [2]. The 4-m tall Ambler has been designed to stably traverse a 30° slope while crossing meter-sized surface features. The robot is configured with six legs, arranged in two stacks on central shafts (Fig. 1). Each leg is an orthogonal mechanism that decouples horizontal and vertical motions, thereby increasing walking efficiency and simplifying control. The stacked legs and large body cavity enable the Ambler to exhibit a unique *circulating gait*, in which trailing legs recover through the body cavity and past the other legs to become new leading legs. The use of a circulating gait enables the Ambler to use approximately one-half to one-third the footsteps of a follow-the-leader type walker, such as the ASV [10].

The Ambler utilizes a number of sensors to monitor its progress and safety. A scanning laser rangefinder on top of the robot provides accurate perception of the terrain. All joints have absolute and incremental (optical) encoders for servo control and dead-reckoning. Six-axis force/torque sensors mounted on each foot are used to detect terrain contact. Two inclinometers on the body indicate tilt and roll from the horizontal plane.

We have developed a software system integrating perception, planning, and real-time control that enables the Ambler to autonomously navigate over hills, boulders, and trenches [3], [11]. The walking system consists of a number of modules connected via the TCA central control module (Fig. 2).

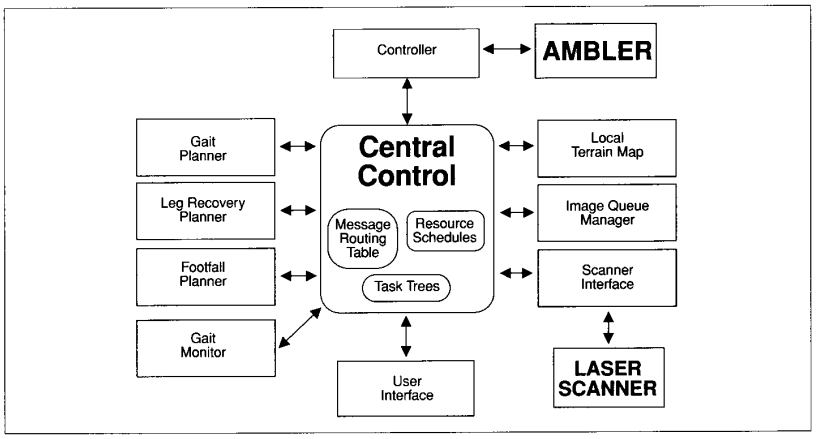

Fig. 2. Modules for the Ambler walking system.

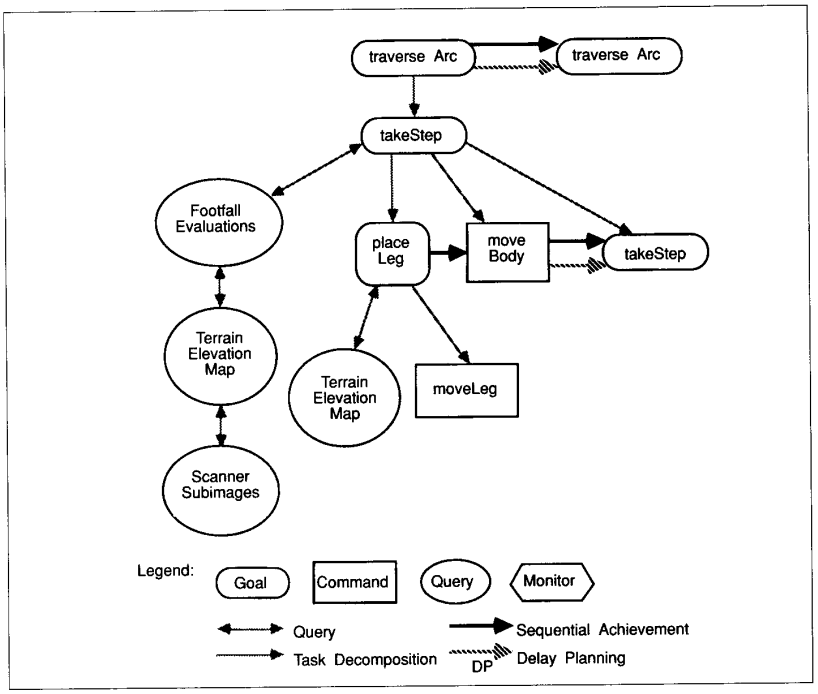

Fig. 3. Task tree for the sequential walking system.

To begin walking, a series of arcs is input by the user. TCA forwards the first arc to the Gait Planner, which issues a *takeStep* message to begin the planning process (Fig. 3). The Gait Planner analyzes geometric constraints on the Ambler's motion to find a body move that maximizes progress while keeping the body stable. Based on this move, the planner constructs a region of geometrically feasible footfalls. The Gait Planner then queries the Footfall Planner to provide evaluations of footfall quality within that region. The Footfall Planner requests an elevation map from the Local Terrain Map module, which in turn obtains scanner subimages from the Image Queue Manager (the Scanner Interface module acquires images and asynchronously sends them to the Image Queue Manager following each body move).

Upon receiving the footfall evaluations, the Gait Planner chooses the best move and sends the chosen footfall and body move to the TCA central control. If the Ambler has not reached the end of the current arc, the planner also sends a message to take the next step. The Leg Recovery Planner is invoked to determine an efficient path for the leg that avoids terrain collisions, and the trajectory is executed by the Controller. After a successful leg move, TCA forwards the body move to the Controller, which actuates the horizontal joints to simultaneously translate and rotate the body. Because of the rigidity and accuracy of the mechanism, joint conflict is negligible, even with powering all twelve horizontal joints. Finally, TCA forwards the next *takeStep* message to the Gait Planner to plan another step.

The planning and execution of steps are coordinated via the TCA temporal constraints. For one, all goals and commands are constrained to be achieved sequentially (Fig. 3). In addition, a delay-planning constraint between the *moveBody* and *takeStep* goals forces the planning of one step to wait until the previous step is completed. This yields a sequential plan-act cycle.

Concurrency

In analyzing the performance of the sequential Ambler walking system, it became clear that performance could be increased significantly by executing one step while planning the next. In fact, timing studies indicated that continuous walking could be attained since the execution time of the Controller is greater than the sum of the planning and perception times.

There are two main concerns in specifying concurrent operation for autonomous robotic systems. One is indicating which tasks may occur in parallel. The other is constraining the concurrency sufficiently so that unexpected temporal interactions (e.g., resource contention) do not occur. This section describes how TCA was used to specify concurrent operation for the Ambler.

The first step in converting to concurrent operation was to specify that planning and execution may occur in parallel. This merely involved removing a single line of code in the original system that added the delay-planning temporal constraint (Fig. 3). With that constraint removed, TCA is free to forward a *takeStep* message as soon as it is issued. Thus, the *takeStep* subtree is expanded (although not executed) while the previous leg and body moves are being executed (Fig. 4).

In the original sequential system, the Gait Planner queried the Controller for the Ambler's current position before planning a move. To plan in advance, the *takeStep* message format was augmented to include the expected position of the Ambler. This change

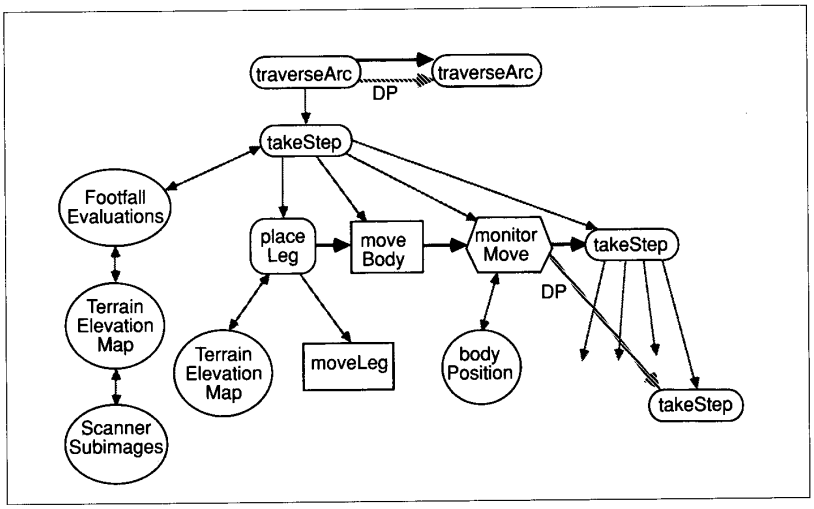

Fig. 4. Task tree for the concurrent walking system.

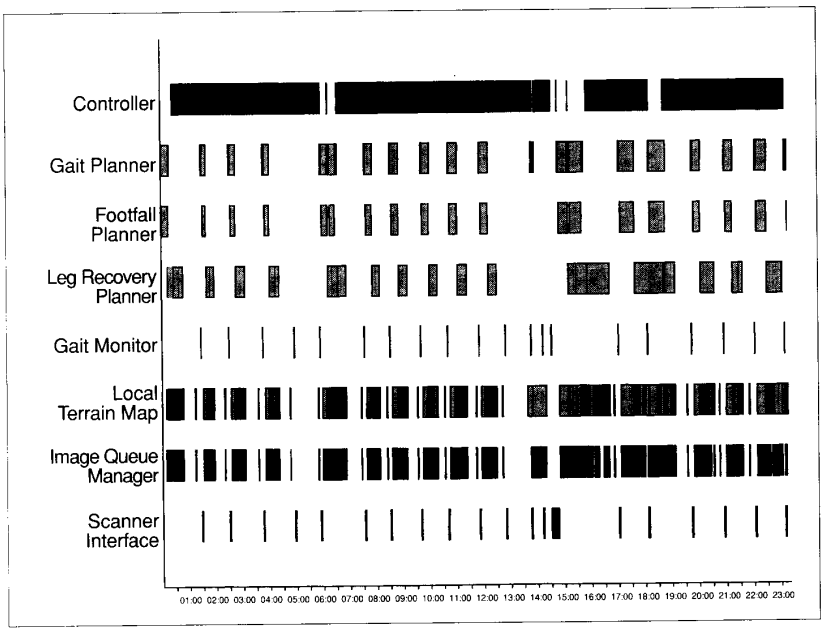

Fig. 5. Timing chart for the concurrent walking system.

was straightforward using the TCA message passing utilities.

In enabling concurrent planning, perception, and execution, undesirable temporal interactions between modules have to be avoided. For the Ambler system, there are several potential interactions in operating concurrently: 1) a planned body move that fails would invalidate assumptions made in planning the next step; 2) planning could exceed the perception horizon; 3) the robot might be moving when images are acquired.

With concurrent planning and execution, the plan for one step is based on the assumption that execution of the previous step succeeds as planned (as described above, the planned position is passed in the augmented *takeStep* message). In particular, if a body move fails the next planned moves might not be achievable. To handle this, a TCA point monitor is inserted in the task tree after the *moveBody* command and preceding the next gait planning cycle (Fig. 4). The condition query of the monitor checks whether the planned and actual body positions differ by some threshold. The action message uses TCA facilities to find and terminate the *takeStep* node (and its subtree) that follows the monitor. A new *takeStep* message is then issued with the correct current body position, which is subsequently planned out and executed, as per usual.

Another temporal interaction stems from the fact that the walking system plans steps significantly faster than it executes them. Thus, if not controlled, the planning would soon get several steps ahead of the execution and would be operating at the limits of the perceived terrain (where the scanner's resolution is low). To prevent this, we limit the system to plan only one step ahead. This is accomplished by adding delay-planning constraints between *takeStep* messages and the monitor messages one level up the tree from them (Fig. 4). While this method could be easily used to provide multi-step lookahead, for the Ambler one step lookahead provides an appropriate balance between performance and accuracy.

The third potential interaction is that the Scanner Interface might try to acquire a range image while the mechanism is moving. This could lead to both image blurring and inaccurate determination of the scanner position at the time of image acquisition. Thus, it is desirable for the Ambler to be stationary while the laser scanner is operating. The necessary synchronization was easily achieved using TCA's resource mechanisms. The Scanner Interface issues a message to reserve the Controller module, queries for the current body position, acquires the image, and then releases the reservation. TCA ensures that 1) the reservation is not granted until the Controller is available (i.e., the Ambler is idle), and 2) subsequent commands are queued while the reservation is in effect.

The modifications to produce concurrent walking took only a few man-weeks of effort, including the time to analyze, design, code, and test the concurrent system. Most of the modifications were initially developed and tested using a graphical simulator. Testing on the Ambler itself revealed the need to synchronize the Scanner Interface and Controller modules.

Results

The competence of the concurrent system is comparable to the sequential system: although neither are perfect, both can reliably navigate through rough terrain.

The performance of the system increased substantially with concurrency, in many cases achieving nearly continuous walking. Fig. 5 presents a timing chart for a typical walk, produced from TCA log files. The darkly shaded areas of the chart are times when modules are computing; lightly shaded areas are when they are awaiting replies from other modules. The chart indicates that the Ambler took 20 steps in 23.5 min, averaging 35 cm/min. This is 34% faster than the sequential system in navigating the same terrain (Fig. 1). The only nontrivial idle times of the Controller are while the first step of each arc is being planned (since the *traverseArc* goals are constrained to operate in delay-planning mode). Overall, the Controller is active about 84% of the time and, discounting the first step of each arc, is active 95% of the time, which represents nearly continuous walking.

The utilization of the TCA central control module is about 3%. While some researchers have warned about the potential bottleneck inherent in centralized control (e.g., [12]), empirical evidence with the Ambler walking system indicates otherwise. This observation is also supported by our experience using TCA to control an indoor mobile manipulator [8].

The timings presented in Fig. 5 are typical of runs where everything happens according to plan. Sometimes, however, a leg will unexpectedly collide with an obstacle, or a body move will fail to achieve the requested position. When this occurs, the TCA point monitor acts to initiate replanning from the actual Ambler position, ensuring that the competence of the system to navigate rugged terrain is not impaired. While the motion is no longer continuous, the time performance is still no worse than the sequential system (and since failures occur only rarely, the overall performance is usually still much better).

Conclusions

The primary conclusion from this work is how relatively simple it was to convert from sequential to concurrent operation by using TCA. This compares favorably with the often

significant effort needed to achieve concurrent operation [13], [14]. We demonstrated a significant increase in performance with no discernible impact on competence, and with relatively modest effort to modify the existing code.

Much of the power in using TCA comes from its explicit representation of hierarchical task trees and temporal constraints. Only recently was explicit temporal information included in AI planners (e.g., [15], [16]), and it is still fairly uncommon in mobile robot systems. We believe that the ability to concisely specify, reason about, and manipulate temporal constraints on tasks is crucial in building complex robotic systems. To this end, we are beginning to investigate the use of formal methods (e.g., [17]) to verify the temporal consistency of robot designs.

The results also indicate TCA's usefulness in evolving systems to be faster (through concurrency) and more robust (by adding monitors and exception handlers). Our experience with the mobile manipulator also bears out the utility of TCA in evolving complex robot systems [8], but this is our first quantitative assessment of the gain in performance and incremental development time needed.

We have recently extended the Ambler system to include concurrent perception as well as planning and execution [11]. We intend to further parallelize and robustify the system as our timing analyses point out other performance bottlenecks and temporal interactions.

Appendix
TCA Temporal Constraints

Each node in a TCA task tree is associated with a message and a set of children nodes (denoted Ch_n) issued in handling that message. A node has several temporal intervals, defined by their start and end points (*I.start* and *I.end*, with *I.start* < *I.end*). Every node has a *handling* interval (Hnd_n) which denotes the time when its associated message is being handled.

Nodes associated with goal, command, and monitor messages also have an *achievement* interval (Ach_n). For command and monitor nodes, the achievement interval contains the handling interval of the node:

$Ach_n.start = Hnd_n.start \wedge$
$Ach_n.end \geq Hnd_n.end.$

For goals, achievement and handling intervals overlap:

$Ach_g.start = Hnd_g.start \wedge$
$Ach_g.end \geq Hnd_g.end.$

In addition, the achievement interval of a node contains the achievement intervals of all its children:

$\forall(c \in Ch_n)\ [Ach_n.start \leq Ach_c.start \wedge Ach_n.end \geq Ach_c.end].$

Finally, nodes have a *planning* interval (Pln_n). For commands and monitors, the planning and handling intervals are identical. For goals, the planning interval contains the handling interval of the node, and contains the planning intervals of all its children (analogously to the above formulae).

In handling messages, TCA keeps track of the special time point **now**, which moves in relationship to the handling intervals of the nodes. When the message for a node is being handled, TCA asserts that:

$Hnd_n.start \leq *now* < Hnd_n.end.$

After a message has been handled, TCA retracts the above assertions and instead asserts that $*now* \geq Hnd_n.end.$

Finally, TCA schedules a node to be handled whenever, given the existing temporal constraints, it cannot be proven that $*now* < Hnd_n.start.$

Using these intervals, TCA defines the *sequential-achievement* constraint between two nodes as $Ach_{n1}.end \leq Ach_{n2}.start.$

Similarly, the *delay-planning* constraint is defined as $Ach_{n1}.end \leq Pln_{n2}.start.$

Acknowledgment

C. Fedor, G. Roston and D. Wettergreen aided in modifying the walking system for concurrent operation. In addition, numerous members of the Planetary Rover group were responsible for developing and testing the original Ambler walking system.

References

[1] R. Simmons, "An architecture for coordinating planning, sensing, and action," in *Proc. DARPA Planning Worksp.*, San Diego, CA, Nov. 1990.

[2] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. Whittaker, "Ambler: An autonomous rover for planetary exploration," *IEEE Comput.*, vol. 22, pp. 18-29, 1989.

[3] R. Simmons and E. Krotkov, "An integrated walking system for the Ambler planetary rover," in *Proc. IEEE Int. Conf. Robotics and Automation*, Sacramento, CA, Apr. 1991, pp. 2086-2091.

[4] R. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Auto.*, vol. RA-2, no. 1, 1986.

[5] R. Brooks, "A robot that walks: Emergent behavior from a carefully evolved network," in *Neural Computation,* vol. 1, no. 2, pp. 253-262, Summer 1989.

[6] F. Pfeiffer, H.J. Weidermann, and P. Danowski, "Dynamics of the walking stick insect," *IEEE Control Syst. Mag.*, vol. 11, pp. 9-13, Feb. 1991.

[7] M. Raibert, *Legged Robots That Balance.* Cambridge, MA: M.I.T. Press, 1986.

[8] R. Simmons, L.J. Lin, and C. Fedor, "Autonomous task control for mobile robots," in *Proc. IEEE Symp. Intelligent Control*, Philadelphia, PA, Sept. 1990.

[9] R. Simmons, "Common sense arithmetic reasoning," in *Proc. AAAI-86*, Philadelphia, PA, Aug. 1986, pp. 118-124.

[10] S. Song and K. Waldron, *Machines That Walk: The Adaptive Suspension Vehicle.* Cambridge, MA: M.I.T. Press, 1989.

[11] R. Simmons, E. Krotkov, and J. Bares, "A six-legged rover for planetary exploration," in *Proc. AIAA Computing in Aerospace 8*, Baltimore, MD, Oct. 1991.

[12] J. Connell, "A behavior-based arm controller," *IEEE J. Robot. Auto.*, vol. 5, no. 6, pp. 784-791, 1989.

[13] F. Noreils and R. Chatila, "Control of mobile robot actions," in *Proc. IEEE Int. Conf. Robot. Auto.*, 1989, pp. 701-712.

[14] C. Thorpe, Ed., *Vision and Navigation: The Carnegie Mellon Navlab. Norwell, MA: Kluwer, 1990.*

[15] J. Allen, "Towards a general theory of action and time," *Artif. Intell.*, vol. 23, no. 2, pp. 123-154, 1984.

[16] T. Dean and D. McDermott, "Temporal data base management," *Artif. Intell.*, vol. 32, no. 1, pp. 1-55, 1987.

[17] E. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent system using temporal logic specifications," *ACM Trans. Programming Languages Syst.*, vol. 8, pp. 244-263, Apr. 1986.

Reid G. Simmons earned the B.A. degree in computer science from SUNY at Buffalo in 1979. He received the M.S. degree in 1983 and the Ph.D. degree in 1988 from M.I.T. in the field of artificial intelligence. His doctoral thesis focused on the combination of associational and causal reasoning for solving complex planning and interpretation tasks. Since coming to CMU as a Research Faculty member in 1988, his research has concentrated on developing self-reliant robots that can autonomously operated over extended periods of time in unknown, unstructured environments. The work involves issus of task-level robot control architectures that combine deliberative and reactive control, selective perception, and robust error detection and recovery. The ideas are currently being applied to the six-legged Ambler Planetary Rover and an indoor mobile manipulator.