A TASK CONTROL ARCHITECTURE FOR AUTONOMOUS ROBOTS

Reid Simmons and Tom Mitchell

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present an architecture for controlling robots that have multiple tasks, operate in dynamic domains, and require a fair degree of autonomy. The architecture is built on several layers of functionality, including a distributed communication layer, a behavior layer for querying sensors, expanding goals, and executing commands, and a task level for managing the temporal aspects of planning and achieving goals, coordinating tasks, allocating resources, monitoring, and recovering from errors. Application to a legged planetary rover and an indoor mobile manipulator is described.

INTRODUCTION

We are currently developing a general-purpose task control architecture (called TCA) for controlling mobile robots. TCA is designed specifically for robots that need to be fairly autonomous and that operate in dynamic and uncertain environments, have multiple goals to achieve, have a variety of strategies to achieve those goals, and use a variety of sensors with different ranges and resolutions.

We are developing TCA concurrently on two testbeds — the CMU six-legged Planetary Rover, and the Hero, a commercially available mobile manipulator platform. The CMU Rover project is an attempt to develop an autonomous robot that can survive, navigate, and acquire samples on the Martian surface [1]. The Hero testbed is an indoor platform that is being used to study coordination of planning, execution, monitoring, error recovery, decision making, and user interaction [3].

TCA is a distributed architecture with centralized control. Communication occurs via coarse-grained message passing between modules, with all messages being routed through the central control. When the central control receives a message, it decides when the message should be attended to, and which module will handle it. We believe that a centralized control scheme will facilitate the coordination of the complex tasks needed for autonomous behavior on Mars. Due to its deliberative nature, however, TCA is not well-suited for robots that need very fast reflexes (e.g., race-car drivers). Our group is beginning research, however, on combining reflexive actions with a deliberative architecture.

TCA can be thought of as a high-level robot operating system — providing a shell for building specific robot control systems. Like any good operating system, the architecture provides communication with other tasks and the outside world, facilities for constructing new behaviors from more primitive ones, and means to control and schedule tasks and to manage computational and physical resources. At the same time, it tries to impose relatively few constraints on the overall control and data flow in any particular system. This should enable researchers to experiment easily with different instantiations of robot control schemes.

TESTBEDS

Our long-term goal is to produce a rover capable of reliable and robust behavior on another planet [1]. Such a system would have to be relatively autonomous, since it will receive infrequent commands from Earth (on the order of every 8 hours) and will have significant communication delay (on the order of 30 minutes).

The tasks envisioned for such a rover include 1) navigating to given sites, 2) acquiring rock and soil samples, 3) surveying for sites of scientific interest, such as regions of sedimentary rock or

underground water, 4) mapping the area traversed, 5) diagnosing system malfunctions, and 6) maintaining communication with Earth.

The rover is designed as a six-legged walking robot. The walker, which will stand around 15 feet high, features orthogonal legs and a split body to enable rear legs to recover past forward legs by passing between the body segments (Figure 1). A prototype leg of the walker has been built and is currently being tested. The leg, along with a laser range-scanner, is mounted on an overhead carriage that is free to roll along a rail (Figure 2). The single-leg testbed "walks" by choosing a footfall location based on elevation maps computed from laser scans, moving the leg to that location while avoiding obstacles, and pulling the carriage with the shoulder and elbow joints.

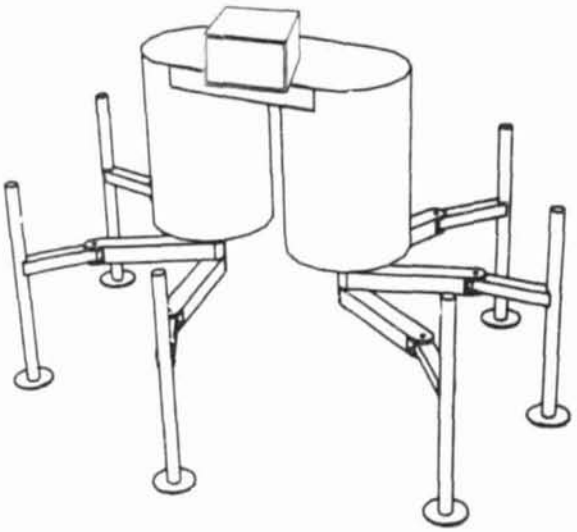

Figure 1. Design of Six-Legged Planetary Rover

Since the current rover testbed is limited in the tasks it can perform, we are exploring issues of coordinating multiple tasks, monitoring, error recovery, and decision making using the Hero testbed [3]. The testbed is based on the Heath/ Zenith Hero 2000, a wheeled robot with manipulator arm and on-board sonars (Figure 3). In addition, an overhead camera provides a two-dimensional plan view of the lab.

Our goal is to let the Hero operate unattended for hours or days at a time in our lab and nearby vicinity. The high level goals of the robot will include 1) collecting cups on the lab floor and placing them in a receptacle, 2) retrieving printer output when requested and

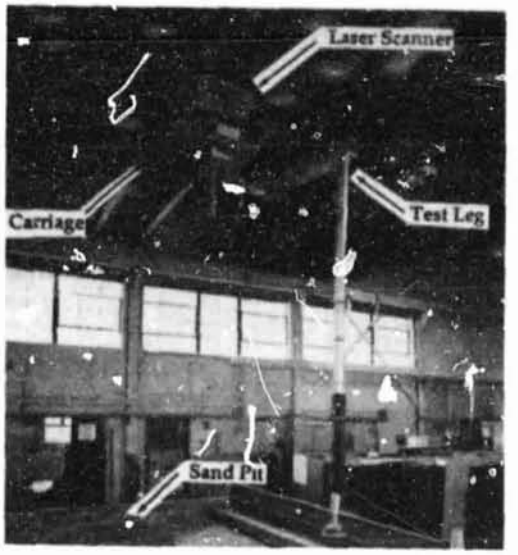

Figure 2. Single-Leg Testbed

delivering it to a workstation, 3) avoiding obstacles, 4) recharging batteries when necessary, and 5) exploring and mapping its environment, when not otherwise occupied. We have already implemented the first three tasks, and are working on the other tasks, as well as the problems of coordinating multiple tasks and error detection and recovery.

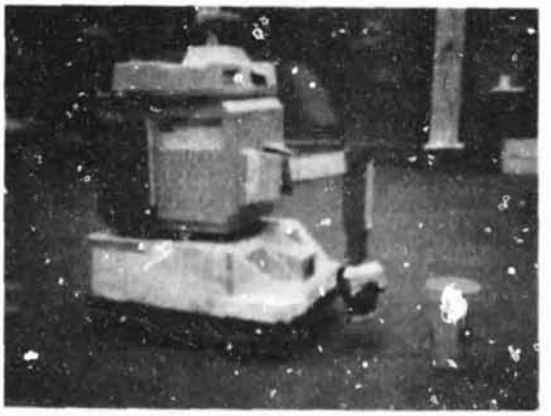

Figure 3. Hero 2000 Robot

SCENARIO

This section presents a scenario for the Hero robot that illustrates capabilities we want TCA to support. Similar scenarios are envisioned for the Planetary Rover.

The scenario begins with the Hero using its overhead camera to spot two cup-like objects on the floor. It forms two "cup-collection" goals

and prioritizes them by choosing to attend to the closer object first. The robot plans and executes a path to the object. While moving, it monitors for objects in its path; at the same time, it uses its overhead image of the lab to pre-plan a path from the object to the receptacle. Upon arriving near the object, the robot uses sonar sensors to determine the height and width of the object. If this matches its model of a cup, it plans how to pick up the cup, then executes the actions. The robot then uses the plan previously made to navigate to the receptacle, where it drops off the cup.

Next, the robot attends to collecting the other cup it had spotted previously While moving towards the second cup, it receives a printer retrieval request. Since this request is of higher priority, the robot suspends planning nd execution of the cup-collection goal. It plans and executes a path to the printer, monitors the printer to determine when printing is finished, and then picks up the paper. While waiting for the printer, it plans a path to the appropriate workstation. In this case, the workstation is located outside the lab, so it plans a path to the door, with the intention of using its sonar-based navigation once outside the room.

Before executing the plan, however, the robot notices (from its overhead vision) that an obstacle has appeared in a segment of the path. The robot attempts to replan that segment to detour around the obstacle. If no detour can be found, the robot replans higher level segments until a clear path is found.

At this point, the robot notices that its battery charge is getting low. It estimates that it has enough power left to deliver the output before it needs to recharge, so it continues towards the door. Upon finding the door closed, it re-prioritizes its goals and, after setting the printer output down near the printer, docks with its charger and waits until the battery is charged.

CAPABILITIES

The above scenario illustrates many of the capabilities we believe will be important for autonomous robots, capabilities that we desire the task control architecture to support.

Achieving Goals — the most basic need of a robot is to construct and execute plans to achieve given goals, such as collecting cups. The robot constructs plans based on the current (or projected) environment, available resources, and its other goals and beliefs. It should also be able to execute parts of its plans before specifying them completely, for instance, moving towards a potential cup before determining exact grasp points.

Coordinating Multiple Tasks — for autonomous robots with many goals but limited resources, prioritizing and scheduling its various tasks are crucial. The robot should make decisions about which goals to attend to by comparing their relative costs, benefits, likelihood of success, etc.

Reacting to Environmental Change — if a new cup is placed on the floor, or the battery charge is low, the robot should notice the change in a timely manner and attend to it, if necessary. Handling such changes may involve adding new subgoals (e.g., collect a new cup), re-prioritizing tasks (e.g., stop collecting the cup and go recharge), or replanning.

Error Recovery — part of reacting to changes includes noticing when a plan or action is failing. The robot should have general mechanisms for recovering from both execution and plan time errors. For example, if an obstacle appears in the robot's path, it might want to detour around the object, or plan a new path to its goal. Similarly, if the robot is unable to find a path, it may try replanning with less restrictive constraints (e.g., allowing more tolerance), or may just attend to a different goal altogether.

External Communication — although not illustrated in the scenario, interaction with humans is a necessity. The robot should be able to explain its decisions and actions. It also should request assistance when needed and allow people to add goals and to alter plans and decisions made by the system.

MECHANISMS

To facilitate experimentation with different control schemes, TCA is built as a layered system. The current layers of functionality include mechanisms for 1) communicating between distributed processes, 2) building behaviors out of more primitive behaviors, and 3) managing the planning and execution of tasks.

TCA is designed 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. For example, if the implementor finds the types of messages in the behavior layer insufficient, s/he can construct new types using the lower communication layer.

Communication Layer

The base layer of TCA supports the sending and receiving of messages between distributed processes (modules). Modules send messages to a central control module, which routes the messages to the appropriate modules to be handled. TCA supports the use of a variable number of modules — in fact, modules can be added or removed while the system is running. The communication layer contains mechanisms to route messages, notify modules when messages are pending, and send and receive data, even between modules written in different languages (currently both Lisp and C are supported).

Using a user-supplied description of the format of a message, TCA translates the message data into a linear stream of bytes, routes the data, and then reassembles it on the receiving end. All data transfer is transparent to the user. The data format language we developed allows for primitive data types (e.g., integer, float, string), and composite types (e.g., structures, arrays, pointers).

Communication via message passing encourages the use of good software engineering techniques — first defining the functionality and interfaces of the system, implementing them in a modular fashion, and treating the functions as "black boxes." We have found that this eases the effort to integrate different parts of the system. This contrasts with some architectures in which some functions are allowed to interfere with the internal workings of others (e.g., [2]).

Note that while perception, planning, and execution modules are distributed, control of how messages are handled is centralized. A major advantage of centralized control is that it facilitates coordination of the robot's behavior. All control decisions, such as which goals to pursue, or which modules should handle particular messages, are made centrally, where global information can be used to determine the best alternatives.

A potential problem with centralized control is that it may become a bottleneck. This may be overcome through conventions, such as using coarse-grained behaviors to limit the amount of process-to-process communication, and limiting the amount of information passed in each message. Although further experimentation might show that this is indeed a problem, the current message passing cycle time of around 50 milliseconds has proven to be sufficient for our applications.

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 flow of control. For example, query messages block the user's code until a reply is received, while goal 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 computing an elevation map or determining the robot's current position. Query messages are routed either to modules that have access to external and internal sensor data, or to the constraint data base (see below). The module issuing the query suspends execution pending the reply.

Goal messages are intended to support top-down, hierarchical planning. A typical response to a goal message would be to issue other (sub)goal or command messages based on the results of some queries.

Unlike queries, goal messages are non-blocking. That is, the central control may queue the goal until resources become available; in the meantime, the module sending the goal message can continue. This implies that a planning behavior cannot assume that the goal will actually be achieved after the goal message is issued. The rationale is that non-blocking goal messages give the implementor greater flexibility in controlling the achievement of goals, such as planning in advance of execution. If goal messages were blocking, planning would always be depth-first — the first subgoal would have to be completely planned before the next subgoal message could be issued.

Command messages, which are used to execute actions, are similar to goal messages. The difference is manifest only at the task level which distinguishes between order in which goals are planned and the order in which commands are achieved. For example, although the robot might be able to plan how to go from the printer to a workstation before planning how to pick up the printer output, it obviously should not actually go to the workstation before it has the output in hand.

Constraint messages provide a way to alter the robot's internal state, just as command messages alter the external environment. For example, constraint messages can be used to set the robot's desired average speed, or add expectations about its future behavior. We plan to implement a global data base (blackboard) to facilitate adding constraints and maintaining consistency among them. Currently, constraint messages are used to set global variables, whose values can then be accessed via queries.

Task Layer

The task layer provides mechanisms for maintaining hierarchical goal structures, allocating resources, monitoring the environment, recovering from execution and plan-time errors, and coordinating multiple tasks. The main representations in the task layer are goal structures, resources, and monitors.

While the behavior layer defines goal and command messages, the task layer contains mechanisms for constructing and analyzing goal hierarchies. For each goal, TCA maintains a subtree of the goal, command, and monitor messages (and their descendants) issued by the goal. Facilities exist for tracing goal/subgoal relationships, displaying the goal structure, and suspending or killing subtrees (needed for switching tasks and doing error recovery).

For the scheduling of tasks, TCA contains a general facility for reasoning about time [4] that enables modules to temporally constrain the planning and achievement of goals. A module specifies constraints on the planning intervals of goals (the time needed to completely expand a goal subtree) and the achievement intervals (the time needed to execute all the commands of the subtree). For example, a module might specify that the achievement of G1 precedes the achievement of G2, but that the planning of G2 precedes that of G1. Similarly, it might constrain G3 to be completely planned before any of its sub-commands can start being achieved (by default, planning and execution can occur concurrently). This temporal framework should enable implementors to take advantage of concurrencies in the distributed environment of TCA — for instance, planning routes from a given area while still travelling to the area.

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 range-scanner. Resources are created with a capacity — the number of messages the resource can handle simultaneously. By default, TCA associates a single resource of unit capacity with each module. In addition, a module can create additional resources and associate message handlers with them.

A message received by TCA is queued until the resource that handles the message has available capacity. Currently, messages are handled in FIFO order, subject to the temporal constraints imposed by the goal structure. In the future, we plan to add mechanisms for prioritizing messages. Since the prioritization is context dependent, it will be determined by user-supplied functions, accessed using "decision messages." A module can also explicitly reserve a resource, temporarily preventing other modules from using the resource. While taking an image, for example, a vision module might reserve the "robot motors" resource to ensure that the robot does not move during that period.

Monitors are mechanisms that query for specified changes in the environment, such as obstacles in the robot's path or low battery charge, and take some action based on the results. A monitor is specified by the condition to be monitored (a query message), an action to take if the condition holds (a goal, command, or monitor message), and the time, relative to other messages, when the monitoring is to take place. Point monitors, which test the condition just once, are useful for checking the preconditions or postconditions of an action, such as checking that a planned move succeeded in reaching the desired location. Interval

monitors, which have a temporal extent, are useful for checking for changes in the environment, such as a low battery charge or the appearance of a new cup. Two complementary implementations exist for interval monitors — synchronous polling at a fixed frequency, and asynchronous demon-invocation.

When a monitor detects an error condition, it sends a "failure message" to the central control. The architecture will then decide what to do based on the current environment and the goal structure for the goal that failed. The decision, made by user-defined handlers for the failure messages, may include replanning the goal with additional constraints, replanning a higher level goal, or adding a new subgoal to patch the initial plan. We believe that the goal structure dependencies maintained by TCA will prove useful in diagnosing and recovering from errors [5].

Since monitors must be coordinated by the central control, special reflex behaviors are needed to provide bounded-time reactions to imminent dangers, such as collisions. Such reflexes, which would be implemented outside the centralized TCA, would have a default response (e.g., "halt immediately") and then signal TCA that a reasoned response to the error is required. This strategy is being tested on the Hero testbed. We implemented a "guarded move" routine that checks the robot's encoders and sonars and stops the robot if it detects a collision or impending collision. Once stabilized, the robot notifies TCA so that an appropriate recovery can be planned.

CONCLUSIONS

Currently, we have implemented the communication layer, the behavior layer (except for the constraint data base), and most of the task layer (except for mechanisms to deal with decision and failure messages). We have used the communication and behavior layers of TCA to run the single-leg testbed for the CMU Rover. Our Hero testbed has recently been re-implemented to use the architecture, and we are currently experimenting with it to test out and expand the task layer.

We have described a task control architecture (TCA) that we believe will be useful in implementing control systems for autonomous mobile robots. TCA is designed around a distributed message-passing system that uses centralized control to queue and route messages. TCA is built using layers of functionality to provide flexibility in experimenting with different control regimes. In addition to the message-passing communication layer, higher layers provide mechanisms for building and coordinating complex robot tasks and behaviors, including mechanisms for goal structure manipulation, temporal reasoning, resource management, and monitoring.

ACKNOWLEDGEMENTS

The TCA was designed with the help of the CMU Planetary Rover group, under NASA contract NAGW-1175. In particular, Christopher Fedor and Long-Ji Lin have helped in both the design and implementation of TCA and the Hero testbed. Kevin Ryan implemented the Hero guarded move routine.

REFERENCES

1. Bares, John, Hebert, Martial, Kanade, Takeo, Krotkov, Eric, Mitchell, Tom, Simmons, Reid, Whittaker, William "An Autonomous Rover for Exploring Mars," IEEE Computer, Vol. 22, No. 6, June, 1989, pp. 18-25.

2. Brooks, Rodney, "A Robust Layered Control System for a Mobile Robot," IEEE Journal of Robots and Automation, Vol. RA-2, No. 1, 1986.

3. Lin, Long-Ji, Mitchell, Tom, Simmons, Reid, "A Case Study in Autonomous Robot Behavior," CMU-RI-TR-89-1, Robotics Institute, Carnegie Mellon University, January, 1989.

4. Simmons, Reid, "'Commonsense' Arithmetical Reasoning," Proceedings of AAAI-86, Philadelphia, PA, August, 1986.

5. Simmons, Reid, "A Theory of Debugging Plans and Interpretations," Proceedings of AAAI-88, St. Paul, MN, August, 1988.