Darpa/Navy Contract No. N00014-92-J-1809

ControlShell: A Real-Time Software Framework

Quarterly Progress Report — October – December, 1992

P.I.'s: Prof. J.C. Latombe, Prof. R.H. Cannon, Jr, Dr. S.A. Schneider

Executive Summary

We are creating a new paradigm for building and maintaining complex real-time software systems for the control of moving mechanical systems. This objective is being met through the *simultaneous* development of both a powerful software environment and cogent motion planning and control capabilities. Our research concentrates on three key areas:

Building an innovative, powerful real-time software framework,

Implementing new distributed control architectures for intelligent mechanical systems, and

Developing distribution architectures and new algorithms for the computationally "hard" motion planning and direction problem.

Perhaps more importantly, we are working on the *vertical integration* of these technologies into a powerful, working system. It is only through this coordinated, cooperative approach that a truly revolutionary, usable architecture can result.

Summary of Progress

This section highlights some of our achievements for this first quarter. During this period, we have:

Provided clear semantics for resolution of multiple producer conflicts, and consumer update rates and guarantees for the NDDS.

Started a seminar series (in conjunction with NASA Ames Research Center) on real-time software architectures. These seminars will enable us to closely compare and contrast our designs with other research activities.

Defined and implemented a preliminary version of the Robot Interface, including command definitions and responses.

STANFORD UNIVERSITY STANFORD, CALIFORNIA 94305

ROBERT H. CANNON, JR.
CHARLES LEE POWELL PROFESSOR
DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS

DURAND BUILDING, ROOM 275
TELEPHONE: (415) 723-3601
FAX: (415) 725-3377
E-MAIL: CANNON@SIERRA.STANFORD.EDU

September 3, 1993

Scientific Officer Code: 1133
Dr. Ralph Wachter
Office of Naval Research
800 North Quincy Street
Arlington, VA 22203-1714

Re: Department of the Navy Ref. N00014-92-J-1809

Dear Dr. Wachter:

Enclosed herewith are quarterly progress reports covering research on ControlShell: A Real-Time Software Framework, for the period October-December 1992 and January-March 1993.

We thank you for your interest in and valuable support of this research.

Sincerely yours,

[illegible]

Robert H. Cannon

Jean-Claude Latombe (JL)

Jean-Claude Latombe

Encl

cc: LTC Erik Mettala, PhD
Administrative Grants Officer
Director, Naval Research Laboratory ✓
Defense Technical Information Center
Victor Sosa, SPO, Stanford

Implemented the World Modeller interface defined last quarter.

Begun receiving parts for our robot hardware upgrade.

Expanded a parallel version of our Randomized Path Planner (RPP), and measured its performance on a Silicon Graphics 4D/240 multi-processor machine, and on a local-area network of UNIX-based workstations.

Made a documented Randomized Path Planner package available to other research institution on the computer network.

Build a Sensor-Based Localization Toolkit for navigation systems.

Our research is progressing according to schedule.

Chapter 1

Introduction

The goal of this research project is to build a new paradigm for building and maintaining complex real-time software systems for the control of moving mechanical systems. This objective is being met through the *simultaneous* development of both a powerful software environment and cogent motion planning and control capabilities. Our research concentrates on three areas:

> Building an innovative, powerful real-time software development environment,
>
> Implementing a new distributed control architecture, and using it to deftly control and coordinate real mechanical systems, and
>
> Developing a computation distribution architecture, and using it to build on-line motion planning and direction capabilities.

We believe that no technology can be successful unless proven experimentally. We are thus validating our research by direct application in several disparate, real-world settings.

This concurrent development of system framework, sophisticated motion planning and control software, and real applications insures a high-quality architectural design. It will also embed, in *reusable* components, fundamental new contributions to the science of intelligent motion planning and control systems. Researchers from our three organizations, the Stanford Aerospace Robotics Laboratory (ARL), the Stanford Computer Science Robotics Laboratory (CSRL), and Real-Time Innovations, Inc. (RTI) have teamed to cooperate intimately and directly to achieve this goal. The potential for advanced technology transfer represented by this cooperative, vertically-integrated approach is unprecedented.

Framework Development This research builds on an object-oriented tool set for real-time software system programming known as *ControlShell.* It provides a series of execution and data interchange mechanisms that form a framework for building real-time applications. These mechanisms are specifically designed to allow a component-based approach to real-time software generation and management. By defining a set of interface specifications for inter-module interaction, ControlShell provides a common platform that is the basis for real-time code exchange and reuse.

Our research is adding fundamental new capabilities, including network-extensible data flow control and a graphical CASE environment.

Distributed Control Architecture This research combines the high-level motion planning component developed by the previous effort with a deft control system for a complex multi-armed robot. The emphasis of this effort is on building interfaces between modules that permit a complex real-time system to run as an interconnected set of distributed modules. To drive this work, we are building a dual-arm cooperative robot system that will be able to respond to high-level user input, create sophisticated motion and task-level plans, and execute them in real time. The system will be able to effect simple assemblies while reacting to changing environmental conditions. It combines a world modelling system, real-time vision, task and path planners, an intuitive graphical user interface, an on-line simulator, and sophisticated control algorithms.

Computation Distribution Architecture This research thrust addresses the issues arising when computationally complex algorithms are embedded in a real-time framework. To illustrate these issues we are considering two particular problem domains: *object manipulation by autonomous multi-arm robots* and *navigation of multiple autonomous mobile robots in an incompletely known environment*. These two problems raise a number of generic issues directly related to the general theme of our research: motion planning is provably a computationally hard problem and its outcomes, motion plans, are executed in a dynamic world where various sorts of contingencies may exist.

The ultimate goals of our investigation are to both provide real-time controllers with on-line motion reactive planning capabilities and to build experimental robotic systems demonstrating such capabilities. Moreover, in accomplishing this goal, we expect to identify general guidelines for embedding a capability requiring provably complex computations into a real-time framework.

Chapter 2

ControlShell Framework Development

This section describes our progress in developing the ControlShell framework and underlying architecture. Two fundamental extensions to ControlShell are being pursued:

Distributed information sharing paradigms, by Gerardo Pardo-Castellote and Stan Schneider.

Graphical Computer Aided Software Engineering (CASE) environments, by Stan Schneider and Vince Chen.

2.1 Distributed Information Sharing Paradigms: NDDS

Last quarter we completed the initial implementation of the NDDS system and resolved the multiple producers and consumers issues. We also implemented a system that handles update rate guarantees and deadlines in a stateless system.

This quarter, we concentrated on deploying the NDDS system in several different environments and putting it to its initial use tests.

The Network Data-Delivery Service model has proved to be a powerful mechanism to speed the integration. The different modules have been developed in very diverse platforms (DEC workstations for the Computer Science Robotics Laboratory and Sun workstations and Real-Time VME-based computers for the Aerospace Robotics Laboratory . The Network Data-Delivery Service has allowed us to connect these platforms without any changes to the application program. Its functionality has allowed us to run simultaneous copies of *the same* software in a distributed and transparent fashion *without any software changes*. For example we have been able to demonstrate simultaneous cooperation of two people operating the same robot using identical copies of the same graphical user-interface.

2.2 ControlShell Upgrade

In this third quarter, Real-Time Innovations (RTI) released version 4.1 of ControlShell. This version encapsulates several new features, many bug fixes, and a considerable amount of deferred maintenance. With this release, we are well-positioned to directly address the major new work to be done under this contract.

One of the most important new advances is a new file and build structure to allow easy support of multiple architectures. The environment allows multi-platform support for both UNIX (Sun and DEC) workstations, and real-time VxWorks targets of several different architectures. It can be easily configured for local file structures, compiler usage, etc. at each site. We have also taken an important step toward facilitating shared components between applications by providing a structure for installing and building locally-defined components. That encourages the user community to share components on a small scale; RTI can then take these more tested pieces of software and incorporate them into the supported library.

ControlShell version 4.1 also supports dynamically-loaded component objects. By this, we mean that ControlShell is able to read component instantiation requests from the system configuration files. If a component is requested that is not already loaded onto the system, ControlShell will locate the object and load it at run-time. This allows individual users to load and run any of the defined components without pre-linking, or worrying about where the objects are loaded from. It also allows users to easily over-ride the standard component objects with custom modifications, improvements, etc. Several new components are also included, including a full interface to the real-time vision system, standard linear estimator and controller components, etc.

A direct module selection facility was also added. This facility provides a simpler interface than the doubly-recursive dependency-graph traversal module selection method. It should expedite the incorporation of the graphical Data-Flow Editor, to be started on in the next quarter. The ability to create a "configuration" of executable modules directly from the current system state is another new facility. This allows saving and restoring an execution state quickly and easily.

Finally, this version considerably upgraded the system DFE file parser. The parser is now much more general (so it can be used with our other file formats, such as the FSM editor) and reentrant (so multiple copies can be running simultaneously). It is also now a two-pass parser—all the strict data types are instanced in the first pass (CSMats, component data sets), the functional modules are instanced in the second pass. This allows functional modules to assume that their data vectors are already initialized at instance time.

To support this functionality, the parser interface now accepts multiple file names; the entire list is parsed as a self-consistent unit.

Chapter 3

Distributed Control Architectures and Interfaces

This section covers our research in software architectures, communication protocols and interfaces that will advance the state-of-the-art in the prototyping-development-testing cycle of high-performance distributed control systems. These interfaces will be implemented within the framework described in Chapter 2. The results of this research will be applied to the vertical integration of planning and control and demonstrated by executing a set of challenging tasks on our two-armed robot system.

There are three main thrusts to this research:

Development of inter-module interfaces for distributed control systems, by Gerardo Pardo-Castellote.

Development of a control methodology capable of executing high-level commands, by Gerardo Pardo-Castellote, Tsai-Yen Li, and Yotto Koga.

Hardware development and experimental verification, by Gerardo Pardo-Castellote and Gad Shelef.

This quarter, we focused our effort in achieving the first experimental milestone which will not only hand provide a test and proof-of-concept of the control and interface technologies developed so far as well as guide our future developments. We believe that early experimental verification is key to focusing research into the relevant real-life problems faced by distributed control systems.

3.1 Inter-Module Interfaces

The world model and robot interface have been successful at connecting the Planner. Simulator and Robot. We have added some functionality to the interface so that the planner can access details in the internal robot state (such as the joint angles) that were previously hidden from it. This is a small departure from our previous philosophy that the planner should operate only at the object level and has been motivated by the need to resolve kinematic ambiguities (the same tool position and orientation can be achieved with different robot configurations). This is significant for collision avoidance purposes and for performance reasons (given more information, the planner can use it to speed its computations). We have maintained the approach that the planner should still command the robot at the object level, although its commands can be augmented to resolve kinematic ambiguities. This will leave necessary freedom so that the robot can use the control policy most appropriate to the task at hand. For example, if an object needs to be manipulated and/or we are going to stablish a contact with the environment, we should use a control policy that takes into account the interaction forces (for example object impedance control).

3.2 Control Methodology Development

Figure 3.1 shows the setup of our first experimental milestone. The task its simple both because of the limitations of the current hardware (we will be installing the upgraded arms with extended reach in January 1993) and the fact that the purpose of this experiment is to test the fundamentals of our architecture and interfaces and we don't want to obscure these with unnecessary complication.

This quarter we have completed the final integration of all the components required for the robot to execute the commands received from the robot-interface and. for the world model module to generate all the information required by both the planner and the user-interface.

The control methodology described in previous quarterly reports has been tested in this first experimental milestone. This milestone which will be fully discussed in the next section. consisted in the autonomous two-arm acquisition and manipulation of an object in between workspace obstacles. This experiment has provided a proof-of-concept for the control methodology and further refinements resulted from this experimental test.

3.3 Hardware Development and Experiments

The task is to acquire an object specified by the user and deliver it to the desired destination. First this task is executed using the simulator. The user can set up a configuration with several obstacles in different locations of the workspace via the simulator's graphical interface shown in figure 3.2. The simulator makes this information available to the planner using the *world model interface*. The user then selects an object and requests it to be delivered to a desired destination. The

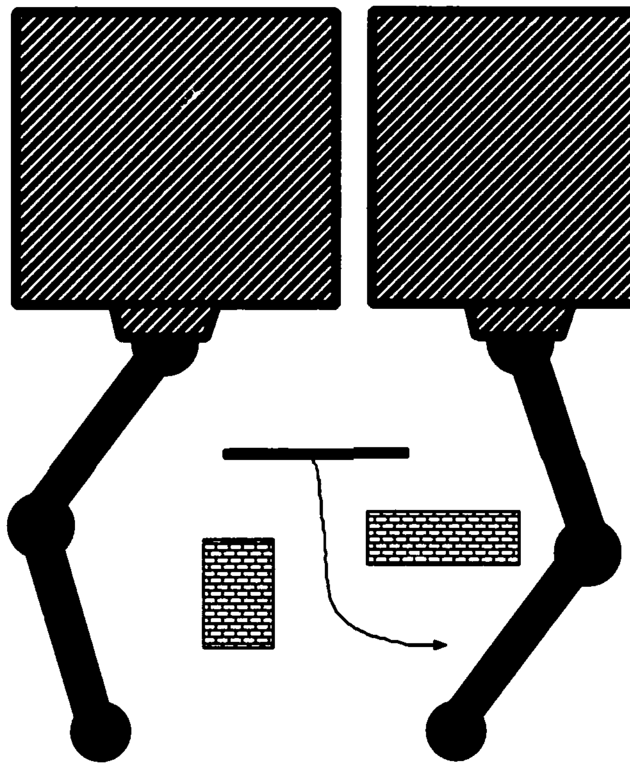

Figure 3.1: **Setup for the first experimental milestone.**

This experiment consists on the acquisition and placement of a user-selected object in a user-specified location. This requires object identification and localization using a vision system. Generation and execution of a multi-step plan involving arm motions, object acquisition and re-grasping.

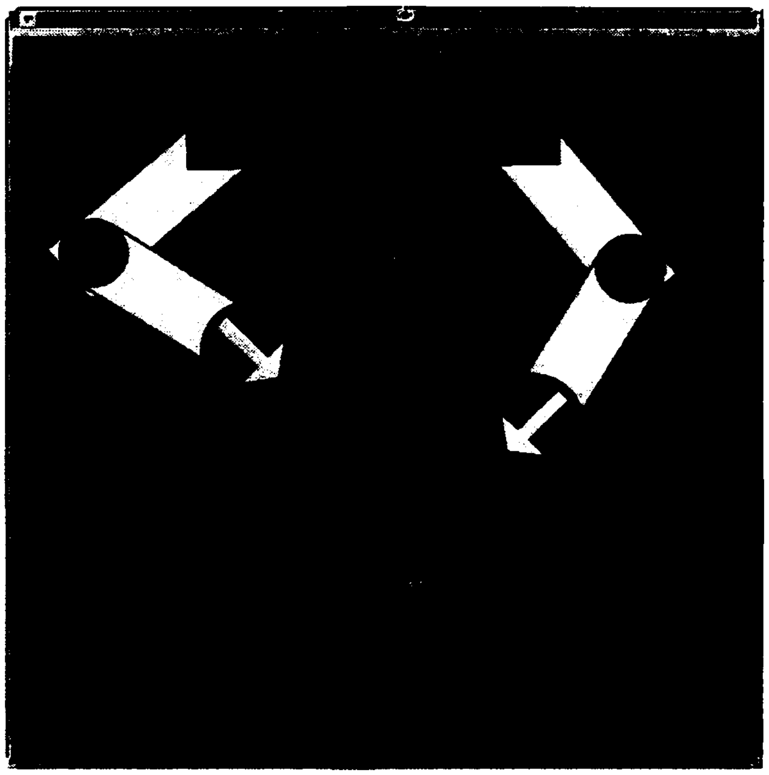

Figure 3.2: **Graphical Simulator.**

The simulator masquerades the robot and sensor systems. It produces "simulated" sensor information on the robot and location of the different workspace objects and receives robot commands. The simulator executes robot commands providing the user a graphical simulation of what the real system would do. The user can also to drag the objects and the robot around the workspace creating different configurations in which the planner and other modules of the system can be tested.

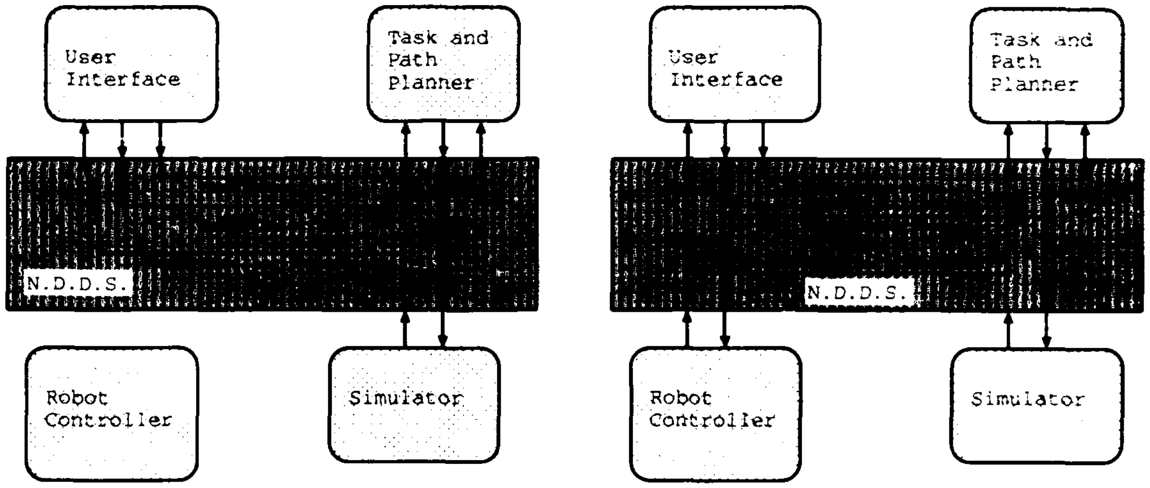

Figure 3.3: **Information flow for the first experimental milestone.**

Two modes of operation are demonstrated in this experiment. First (figure on the left), the robot isn't operational and the simulator masquerades the robot executing commands from the planner and user-interface and producing world-model information. In the second mode, the robot is now operational. Both robot and simulator execute robot-commands and produce information on the state of the world. However, the User Interface and Planner only receive the state information from the robot because it is of higher precedence. All information flows using one of the three interfaces: The Task Interface (for task-commands), the Robot Interface (for robot commands) and the World Model Interface (for state information of the robot and different workspace objects). These interfaces are built on top of NDDS. NDDS takes care of all this information flow and precedence-resolution. All the modules are unaware of who else is producing or consuming the information.

planner, using the information provided by the simulator generates a plan. This plan is composed of multiple steps including both dual-arm and object paths, interleaved by object acquisitions and releases (necessary for regrasping). This commands are sent to the robot using the *robot interface*. The planner continuously monitors the successful completion of each task. The simulator simulates execution of each command and provides feedback to the planner using the *world model interface*. Figure 3.3 illustrates the flow of information during task execution. The robot simulator has been developed by the Computer Science Robotics Laboratory and is described in section 4.5 of our last quarterly report. The planner has also been developed by the Computer Science Robotics Laboratory .

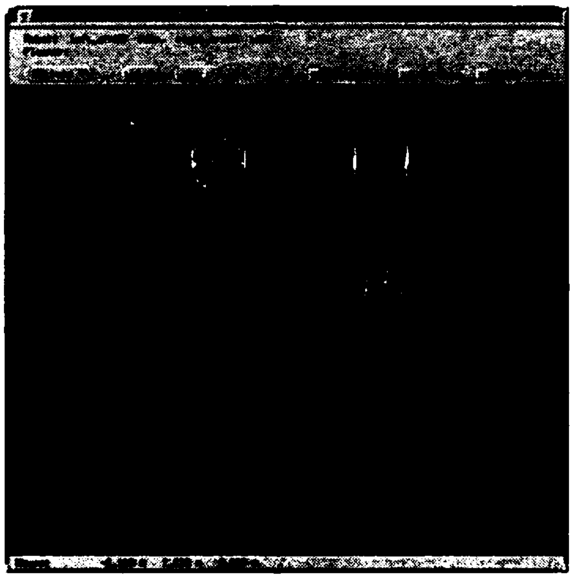

Figure 3.4: **Graphical User Interface**

The user specifies robot tasks with simple mouse commands. These tasks include desired goal positions for the different objects. In addition, it provides graphical feedback to the user.

Following this, we execute the same task with the difference that the robot is now operational. The simulator is still active but now, the world-model information coming from the robot takes precedence[1]. The existence, shape and location of the robot and objects is know sensed by the robot and distributed using the same world-model interface. The experiment proceeds as before with the exception that now the robot executes the planner's commands. Figure 3.3 illustrates the flow of information during task execution.

[1]This uses the multiple-producer conflict resolution mechanism of the Network Data-Delivery Service , already described in the last quarterly report.

Task specification by the user is performed using a graphical user interface. The user specifies a task with simple point-and-click operations on the graphical display. This interface shown in figure 3.4 also provides graphical feedback on the "state of the world", that is, location of the arms, objects, status of the planner, robot etc. The interface has been developed at the Aerospace Robotics Laboratory .

A videotape of this experiment showing the simulator, task specification with the user-interface and task execution by the physical robot is attached to this report.

Chapter 4

On-Line Computation Distribution Architectures

This research addresses technical issues arising when computationally complex algorithms are embedded in a real-time framework. To illustrate these issues we consider two particular problem domains: *object manipulation by autonomous multi-arm robots* and *navigation of multiple autonomous mobile robots in an incompletely known environment.* These two problems raise a number of generic issues directly related to the general theme of our research: motion planning is provably a computationally hard problem and its outcomes, motion plans, are executed in a dynamic world where various sorts of contingencies may happen.

The ultimate goal of our investigation, concerning the two problem domains mentioned above, is to both provide real-time controllers with on-line motion reactive planning capabilities and build experimental robotic systems demonstrating such capabilities. Moreover, in accomplishing this goal, we expect to elaborate general guidelines for embedding a capability requiring provably complex computations into a real-time framework.

This quarterly report covers work done towards this goal during the period of October, November, and December 1992. During this period, our work addressed on the following areas:

1. Distribution of Path Planning, by Tsai-Yen Li.
2. Parallelization of Path Planning, by Lydia Kavraki and Tsai-Yen Li.
3. New Methods for Fast Path Planning, by Tsai-Yen Li.
4. Time-Optimal Multi-Arm Manipulation Planning, by Yotto Koga.
5. Experiments in Manipulation Planning, by Tsai-Yen Li and Yotto Koga.
6. Landmark-Based Mobile Robot Navigation, by Anthony Lazanas.

7. Mobile Robot Navigation Toolkits, by Claudio Facchinetti, Mark Yim and David Zhu.
8. Multi-Mobile Robot Simulator, by David Zhu.

Areas 1 through 5 are mainly related to the first problem domain, i.e., *object manipulation by autonomous multi-arm robots.*

Areas 6 through 8 are mainly related to the second problem domain, i.e., *navigation of multiple autonomous mobile robots in an incompletely known environment.*

Participating Ph.D. Students: Lydia Kavraki, Yotto Koga, Anthony Lazanas, Tsai-Yen Li, Mark Yim.

Participating Staff: Claudio Facchinetti, Randall Wilson, David Zhu.

4.1 Distribution of Path Planning

In order to allow path planning techniques to be exploited in a real-time framework, we are exploring a new distributed software architecture to provide the "best" on-line planning service to robot systems. In previous reports, we identified and analyzed axes along which planning can be distributed: processors, methods, approximations, commitment over time, and time. In addition, we have obtained encouraging preliminary results in distributing our RPP planner (Randomized Path Planner) along some of these axes. However, it is not clear as yet how one should distribute RPP along multiple axes to obtain minimal planning time.

Although planning distribution may be problem specific, we hope to identify some general distribution invariants to develop a distribution architecture that applies to a wide range of problems. The architecture we are currently developing is hierarchical and consists of three levels of distribution. The computer agent at the first level, called the "task coordinator," takes the high-level task description from the user interface, breaks the task into subtasks (each formulated as a path planning problem), and sets task priorities for these problems. Problems with high priorities will be solved before problems with low priorities (and the later will be solved while the plan generated for the former are being executed). Hence, at this level, distribution is essentially done along the time dimension.

The output of the first layer consists of prioritized path planning problems and computing resource assigned to these problems according to their priorities. It is sent to the second layer, which consists of agents called "planning coordinators." Given a path planning problem, a planning coordinator selects appropriate planning methods and parameters to solve the problem according to the assigned computing resource. At this level, we achieve distribution along three axes: planning methods, problem approximations, and commitment over time.

The third layer consists of agents called "planning agents." The output of the planning coordinators is transferred to those agents, which use the selected path planners to solve the submitted planning problems.

Motion plans and the status of planning agents are reported back to the planning coordinators. The planning coordinators, in turn, report back to the task coordinator the best results that have been obtained so far. Due to possible changes in the state of the world and possible time constraint for the task to be accomplished, the higher-level agents must have the capability of interrupting the work of the lower-level agents, invalidating a path, accommodating a change, and then redistributing the new tasks accordingly.

4.2 Parallelization of Path Planning

4.2.1 Previous Work

We are investigating the parallelization of our Randomized Path Planner, RPP (see previous Quarterly Reports). In the previous reports, we focused our work on the most time-consuming subset of the RPP algorithm. This subset constructs a collision-free path connecting the initial and goal configurations of the robot by iteratively escaping local minima by concatenating random and down (negated gradient) motions. (The other two subsets of the RPP algorithm are those which precompute potential functions and smooth the constructed path.) We conduct our study, with two computer architectures: a small-scale shared memory parallel multiprocessor machine and a network of regular UNIX-based workstations. Results obtained with these two architectures were presented in previous reports. No new significant results can be presented this quarter.

4.2.2 Parallelization of the Preprocessing Part of RPP

During this quarter, we worked on parallelizing the preprocessing part of RPP. Previous work has shown that a significant amount of time is spent by RPP doing collision checking. However, it is possible to reduce this time at the expense of a rather time-consuming preprocessing step. It is this preprocessing step that we wish to parallelize in order to reduce its running time.

RPP represents the robot's configuration space (C-space) as a discrete space (grid) where each degree of freedom of the robot can assume only a finite number of values. A precomputed and stored C-space bitmap that explicitly represents the free part of the C-space (the 0's) versus the part that gives rise to collision with obstacles (the 1's), reduces collision checking for a rigid robot to an $O(1)$ operation. In our simulations, we used robots that are planar sets of K rigid bodies connected by prismatic and revolute joints. Collision checking for these robots can be done by checking separately for collision each of their links. Hence, during the preprocessing step of RPP, we can compute K bitmaps, each representing the C-space of one link if it were free to translate and rotate. Once these bitmaps are available, collision checking is an $O(K)$ operation.

The construction of a C-space bitmap for a rigid robot can be done in a variety of ways. Latombe[1] surveys many algorithms that analytically compute the C-space for robots and obstacles of fixed geometric shapes. In the context of RPP, these methods can be used to obtain an analytic expression of the obstacles in the C-space and then map these obstacles onto a bitmap. Methods that directly compute bitmaps have also been proposed,[2] but they require a lot of storage. In our search for a method that will be suitable for parallelization on a small-scale shared memory multiprocessor, such as the SGI, we ended up formulating a new method for computing C-space bitmaps. The proposed algorithm is inherently parallel and it is based on the Fast Fourier Transform (FFT).

Our approach derives from the observation that when the robot is a rigid object that can only translate, the C-space is a convolution of the workspace and the robot. We compute this convolution using the FFT algorithm. To do this we employ the convolution theorem. According to this theorem, a convolution of the workspace bitmap and the robot bitmap can be computed by (i) taking the Fourier Transform (FT) of the workspace bitmap and a properly shifted bitmap of the robot, (ii) multiplying these transforms pointwise, and (iii) computing the inverse FT of their product. We have established[3] under which conditions the FFT algorithm can be used in the above process and how our input data can be changed to match these conditions.

Although the basic FFT-based algorithm for computing C-space bitmaps applies to translating only robots, it can be extended to handle rotation. In the case of planar robots this is done by discretizing the range of orientations of the robot and building a three-dimensional C-space bitmap slice by slice. In theory, this extension also applies to the case of a robot that translates and rotates in a three-dimensional workspace, but the dimension (6) of the C-space in that case makes the construction of an explicit bitmap unrealistic. If the rotation is restricted to occur about a single axis, a relatively coarse four-dimensional bitmap can be constructed as a set of three-dimensional slices, each computed at a discretized orientation of the robot.

In the next quarter, we plan to implement the above method and experiment with it. We are also interested in comparing its performance with existing methods for computing C-space bitmaps.

4.3 New Methods for Fast Path Planning

Efficient motion planning algorithms are the core component of any robot planning system. The more efficient the motion planning algorithms, the more likely we can bring them on-line for real-time robot execution. Several efficient path planning algorithms have been developed in the our laboratory over the past few years. Among them, RPP applies to robots with many degrees of freedom. Although experiments with RPP have often be very successful (RPP is now also used in a variety of other organizations), we do know that there are some pathological cases where it works

[1] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[2] W. Newman, M. Branicky, "Real-Time Configuration Space Transforms for Obstacle Avoidance," *The Int. J. of Rob. Research*, 10(6), 1991, 650–667.

[3] L. Kavraki, *Computation of Configuration-Space Obstacles Using the Fast Fourier Transform*, Technical Report STAN-CS-92-1425, 1992.

poorly (some of these cases have been pointed to us by external users). Because of that, we have started research exploring other efficient planning algorithms in high-dimensional configuration spaces. Although the motivation for developing new algorithms was to solve the pathological cases for RPP, the preliminary results we obtained show that these algorithms work better than RPP in many examples. Thus, these algorithms can be used as complementary counterparts of RPP - the axis of distribution over planning methods identified in previous quarters - to create a more complete and efficient planning system.

We call one of the new developed planners the *vector-based planner*. Its algorithm uses the vector connecting the initial and the goal configuration to guide the search. This search is bi-directional, i.e., both the initial and goal points in the C-space may move toward each other. As in RPP, the search consists of pairs of down motion and random motion. The main difference is that down motions are guided by the length of the vector rather than by precomputed potential fields. If the part of the C-obstacle along which the endpoints of the vector slides is convex, the vector converges and shrinks to a point rapidly. This property gives an efficient way for doing local motion planning.

As in the RPP algorithm, the search in this new algorithm may get trapped into local minima. This occurs when both endpoints of the vector are at a concave portion of a C-obstacle boundary. Then a random walk is performed as follows: One of the vector endpoints, called the leader, makes a random walk, while the other endpoint, called the follower, follows the direction of motion of the leader whenever it is possible. Because the random walk is random in every direction, after some length of the walk, the follower may leave the C-obstacle boundary. To avoid the follower from getting trapped into previous local minima and to detect whether the path is connected during the random walk, we shrink the length of the vector periodically from the direction of the follower. At the end of the random walk, as long as one endpoint moves to the convex portion of the C-obstacle boundary, the local minimum is escaped by executing the next down motion. This algorithm was implemented and experiments show that it is very efficient for many cluttered environments, especially when the regions around the initial or goal configurations are cluttered by obstacles. The bi-directional search helps the planner move out of the cluttered regions which are difficult to get in, but easier to get out. Another advantage of this algorithm over RPP is that there is no need to precompute potential fields. These are not cheap to produce for three-dimensional workspaces.

However, because the algorithm only uses local properties of the C-obstacles' boundary, there still exist pathological cases where both endpoints of the vector are trapped by highly concave portions of C-obstacle boundary. In these case, there is no global information available that can help the planner avoid or escape the local minima. Currently, we are working on a modified version of this planner that uses potential fields to help escape local minima. Hopefully, this new version can be as efficient as the pure vector-based planner, while avoiding some pathological cases.

4.4 Time-Optimal Multi-Arm Manipulation Planning

We conduct research on time-optimal control algorithms in connection with RPP. We have implemented a manipulation planner that can generate the motion of two arms such that they move an object to a desired location in a collision-free manner. However, due to random walks utilized in the planning, the trajectory may contain dance-like motions; consequently, it requires further smoothing. We propose using a time-optimal control algorithm to smooth the path to one that the robot can follow in minimum time. Furthermore, by doing so, we simultaneously obtain a path that is optimally time parameterized, that is, we have a trajectory that can be directly sent to a robot controller for actual execution.

Bobrow et al.[4] have developed a time-optimal control algorithm for an open chain robot following a specified path. Using this algorithm to set the cost of a given path (minimum time of travel) the path can then be perturbed in the direction of the negative gradient of this cost to some locally minimum solution. The resulting path is one that the robot can follow to the goal in locally minimum time. This would be the desired smoothing of robot motions mentioned previously. However, manipulation paths consist of closed-chain robot motions (multi-arms grasping the same object). In the last few quarters, we have extended Bobrow et al.'s algorithm to find path-constrained time-optimal motions of closed-chain robot systems. In addition, we have added a "no slip" constraint to ensure that the robots do not lose their grasp of the object while executing these trajectories.

Unfortunately, in actual application these optimal-time motions may not be feasible trajectories for the robot controller to track. The reason is as follows: In the optimization strategy reported last quarter we take the actuator torque limits of each arm as input and then determine the optimal-time parameterization for the prescribed path. However, the controller strategy for multi-arm cooperative motion may be incompatible with this optimization strategy. For example, the robots in the ARL use "object impedance control", where the controller takes the object trajectory as the input and then determines through a partitioning scheme the effort that each arm must exert to track the object trajectory. The danger here is that during the execution of the optimal motion the controller may request one or more of the actuators to exceed their capacity.

During this quarter we have solved this problem by adding consideration of the controller strategy to the optimization algorithm. In particular, we consider the object impedance control strategy since we ultimately want to execute these optimal motions on the ARL robots. With object impedance control the arm dynamics are decoupled from the motion of the object through a partitioning scheme. By adopting this scheme the optimization problem in fact reduces to the original Bobrow et al.'s algorithm applied to each arm acting as open chain robots. We can now find optimal time motions of closed-chain robot systems that are consistent with the object impedance control strategy.

We are currently working to execute these optimal motions on the ARL robots and to apply the original and modified Bobrow etal.'s algorithm as cost functions in the proposed smoothing

[4]See J.E. Bobrow, S. Dubowsky, and J.S. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *The International Journal of Robotics Research*, MIT Press, 4(3), 1985.

optimization of manipulation paths.

4.5 Experiments in Manipulation Planning

Our research on distributed motion planning will yield a set of software modules running under the ControlShell system to control actual manipulator robots available in ARL (Aerospace Robotics Laboratory at Stanford). Our goal is to demonstrate manipulation of multiple movable objects by two cooperating arms in an uncertain, dynamic environment.

We are working on a first integrated demonstration using a manipulation planner that we developed over the past year.[5] This planner generates synchronized paths of two arms for manipulating movable objects. It also inserts grasp/ungrasp/regrasp operations between paths. Currently, it runs off-line. Our first task will be to make it run on-line under ControShell, assuming a static environment.

This investigation requires distributing the code in the ControlShell environment and developing/using various interface standards. To facilitate present and future transfer of code between our laboratory and ARL, we are developing a simulator which will allow us to check that some major constraints are satisfied, before actually transferring the code.

The need of developing a good software simulator emerges from the following facts:

Software simulation is safer than hardware in developing stages.

Developing software simulator is cheaper than purchasing hardware.

Software simulation is more flexible for changes in robot configuration.

Once the simulator is well developed, the setup time for experiments is less for software simulation.

Software simulation can help predict the potential problems of hardware design.

Software simulation can verify the correctness of a planning software, and reduce the cycle time of developing new planning algorithms.

We are building a simulator which emulates the ARL robot system in order to test the planning module in the development stage. The simulator takes the same interface between planning and the robot system in the ARL. It is capable of building the world model, simulating the kinematic motion, and providing the network data delivery service (NDDS). A graphical user interface using the MOTIF(TM) widget set under the X-Window environment has been developed to help users conduct experiments. The simulator is also designed in a modular fashion such that the same

[5]This work was done under DARPA contract DAAA21-89-C0002.

graphics functions can be reused in other displaying programs and more detailed simulation modules like dynamics, control, and contact modules can be added in the future. Currently this simulator is capable of simulating the kinematic motion of the ARL robots and has also been tested with the manipulation planner developed in the last quarter.

Once the simulator has been integrated with the network delivery service package, it will be interchangeable with the robot system and more experiments will be conducted and reported later.

The ultimate goal of this work is to develop and demonstrate an integrated autonomous robotic system which is capable of manipulating objects in a dynamic environment. Such an autonomous system consists of several modules including a robot controller, a task planner, a manipulation planner (itself combining several path planners), a simulator, and a user interface. Among these modules, the task planner is the one that acts as the interface between the high-level graphical user interface and the manipulation planner. The output of the manipulation planner (and the path planners it invokes) is a motion path that is sent to the robot controller. In this quarter, we have implemented the task-planning module and have successfully integrated it with the rest of the system.

The task planner is triggered by user-specified high-level task instructions, such as moving an object from an initial configuration to some goal configuration. The task planner also subscribes to the sensory data server through the Network Data Delivery Service(NDDS). This data is converted into the manipulation planner format and transmitted to it. When the manipulation planning is undertaken, the task planner reports progress of the planning activities back to the user interface. If the manipulation planner cannot find a path, the task planner also reports the possible causes for the failure to help the user specify a new feasible task. If the manipulation planner succeeds in finding a feasible path, the planned path is returned by the task planner to the robot controller for actual execution. This path typically consists of several subpaths separated by grasping and ungrasping operations. These subpaths are sent to the robot controller one at a time while the task planner monitors their execution to ensure safe motion. If an unexpected event is detected, the task planner reports failure or replans.

In the future, the task planner will be further integrated with the distributed planning architecture mentioned in previous sections to provide more efficient on-line planning service.

4.6 Landmark-Based Mobile Robot Navigation

In robotics uncertainty exists at both planning and execution times. Effective planning must make sure that enough information becomes available to the sensors during execution, to allow the robot to correctly identify the states it traverses. It involves selecting a set of states, associating a motion command with every state, and synthesizing functions to recognize state achievement. These three tasks are often interdependent, causing existing planners to be either unsound, incomplete, and/or computationally impractical. In our work during the previous quarters, we partially broke this interdependence by assuming the existence of landmark regions in the workspace. We defined such regions as "islands of perfection" where position sensing and motion control are accurate. Using this notion, we developed and implemented a sound and complete planner of polynomial complexity.

The plans created by our planner are distributed over the workspace. Each landmark region has an associated motion command, which is executed whenever the robot enters the region. A motion command consists of an exit region (a subset of the landmark region), a commanded direction of motion, and a termination set of landmarks. The robot should first enter the exit region and then attempt to follow the commanded direction of motion d (subject to some control uncertainty modelled by an angle θ) until it enters one of the landmarks in the termination set.

During this quarter, we have studied the dependency of a plan on the control uncertainty θ. We have shown that the structure of a reliable plan only changes at critical values of this uncertainty. We have introduced the notion of a generalized backprojection, defined as the four-dimensional set of backprojections for all possible values of d and θ. We have shown that this four-dimensional set is sufficiently represented by a polynomial number of two-dimensional backprojections. Indeed, the planner builds the backprojections in order to find out which initial-region disks are covered by it, and which landmark disks (currently not being backprojected) are intersected by it. The answers to these questions change only when a *critical event* occurs, i.e. when a ray of the backprojection becomes tangent to a target disk, or when a spike (a left and a right ray with a common endpoint) intersects a target disk. The number of critical events is a polynomial function of the complexity of the workspace.

Each critical event corresponds to a critical curve in the $d\theta$-plane. These curves divide the $d\theta$-plane into a polynomial number of cells. All the points in a cell are equivalent in the following sense: A backprojection computed for any combination of d-θ values within the cell includes the same initial-region disks and intersects the same non-backprojected landmark disks. Thus, the planner need compute only one backprojection per cell.

All critical curves are straight lines with slope ± 1, except those that correspond to the intersection of a disk by a spike. The equation of this curve is complicated. We spent a significant amount of time investigating its properties, and we proved that it has a single maximum and only one intersection with straight lines of slope ± 1. This is significant in proving that the number of cells is polynomial.

The applications of this research can be multiple. It can be used to create a complete polynomial planner in the case where directional uncertainty is controllable by the robot (e.g., by adjusting its speed, or the resources that monitor the straight-line motion). It can also be used to create non-guaranteed plans (if no guaranteed ones exist) that maximize probability of success. In our near-term research, we plan to implement these algorithmic results and experiment with them.

4.7 Mobile Robot Navigation Toolkits

During this quarter, we continued our work on establishing a Software Engineering Standard to facilitate the development of the Toolkits. In establishing this standard we are taking the following issues into consideration:

1. Sharing of common functions among the toolkits. Several toolkits may use the same, or similar functions such as line intersections. These functions should be written in such a way that they can easily be shared by many toolkits. In particular, this concerns the interaction between the Geometry Engine Layer and the Building Blocks Layer.

2. Interaction among the different building blocks. Since several building blocks may be integrated together to build a navigation system, the internal relationship among the different building blocks is important.

3. Interface between the Building Block Layer and the System Developer Layer. We must provide a convenient way to assemble the building blocks together to build a navigation system.

We have established a preliminary standard for the Toolkits. This standard will help us coordinate the efforts of different programmers in the development team. However, we expect to refine this standard as the development of the Toolkits progresses.

We have implemented a first-cut version of several toolkit modules that we believe to be essential to the development of a reliable robot navigation system:

1. Robot Modeling Toolkit: This toolkit allows to set parameters describing the geometry and capabilities (mainly sensors) of a robot. These parameters are accessible through functions by other toolkits, which can be written independently.

2. World Modeling Toolkit: This toolkit provides functions to describe and access a geometric model of the robot environment.

3. Path Planning Toolkit: This toolkit provides functions to generate channels and paths. These are generated using an approximate cell decomposition method.

4. Control Toolkit: This toolkit is an interface between the other toolkits and the actuator control functions offered by the robot. Hence, if the later are changed, only this toolkit need be modified.

5. Virtual Sensor Toolkit: This toolkit is an interface between the other toolkits and the robot's sensors. It fills generic paramaterized data structures. For example, a range sensor is parameterized by its minimal range, maximal range, resolution, precision, angle of views, number of rays, etc. Hence, the data provided by sonar proximity sensors, infra-red proximity sensors, and laser/camera range sensors are used to fill the same data structures (but with different parameters). Other toolkits can access these data structures as virtual sensors.

6. Vector Field Toolkit: This toolkit is a sophisticated one that allows the robot to track the flow of a vector field, while monitoring a variety of conditions. The routine computing the vector field is

given as a parameter. Monitored conditions are stopping conditions and velocity tuning conditions.

7. Sensor-Based Localization Toolkit: This toolkit makes use of all available sensing (including odometric sensing and environmental sensing) to locate the robot relative to its environment. It uses fast matching techniques to correct odometric errors. This toolkit, the most involved of all toolkits, was outlined in the previous quarterly report.

8. Map Building Toolkit: This toolkit uses environmental sensing to build and update a map of the environment. This map is represented as a fine-resolution bitmap. Every obstacle detected by sensors is represented as a set of points in this map.

9. Navigation Toolkit: This toolkit makes use of all the other toolkits to provide different navigation strategies. Three strategies have been implemented so far: navigation in a channel (the most reliable), navigation along a path (a bit less flexible), and greedy navigation using local potential fields (the less reliable).

4.8 Simulator for Multiple Robots

We have started the implementation of the multiple robot simulator based on the set of requirements that we have established. Currently we have a prototype robot simulator that supports the simulation of two robots. This simulator has the following characteristics:

1. It supports the simulation of two robots with different sensing capabilities. A client process can request the server to create a robot with specified sensing capabilities.

2. It supports multiple client processes to control the same robot, i.e., a client process can request the server to connect to a robot that has already been created.

3. When creating a connection to a robot in the server, a client process can specify the priority of the connection. A request from a client that has a higher priority connection will be processed first by the server.

4. It supports synchronous communication between client and server.

5. It supports synchronous communication between two client processes controlling the same robot or controlling two different robots.

We have performed several experiments concerning multiple robot navigation in an effort to test the simulator. For example, a "robot rendez-vous" program was implemented and tested successfully using this robot simulator.

4.9 Summary of Main Results Obtained So Far

1. Identification of several axes for distributing path planning software in an on-line architecture.

2. A documented Randomized Path Planner package has been made available to other research institution on the computer network. Several organizations are using it.
3. Implementation of parallel versions of RPP on a Silicon Graphics 4D/240 multiprocessor machine and on a local-area network of UNIX-based workstations.
4. Definition of a new, FFT-based method to compute obstacles in configuration space.
5. Definition and implementation of a new path planning method (the vector-based planner) to generate paths for robots with many degrees of freedom.
6. Design and implementation of an optimal-time motion planner for closed-loop kinematic chains.
7. Design and implementation of a new landmark-based mobile robot planning method.
8. Definition of the layout of a software toolkit to efficiently develop new navigation systems. Implementation of several toolkits.
9. Partial development of a powerful mobile robot simulator to facilitate the development and debugging of mobile robot programs.

Publications So Far:

Y. Koga and J.C. Latombe, "Experiments in Dual-Arm Manipulation Planning," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Nice, May 1992, pp. 2238–2245.

Y. Koga, T. Lastennet, J.C. Latombe, and T.Y. Li, "Multi-Arm Manipulation Planning," *Proc. of the 9th Int. Symp. on Automation and Robotics in Construction*, Tokyo, June 1992.

A. Lazanas and J.C. Latombe, *Landmark-Based Robot Navigation*, Rep. No. STAN-CS-92-1428, Dept. of Computer Science, Stanford U., May 1992.

A. Lazanas and J.C. Latombe, "Landmark-Based Robot Navigation," *Proc. of the 10th Nat. Conf. on Artificial Intelligence, AAAI-92*, San Jose, July 1992, pp. 816–822.

A. Lazanas and J.C. Latombe, "Landmark-Based Robot Motion Planning," *Proc. of the AAAI Fall Symp.*, Boston, MA. October 1992, pp. 98-103.

L. Kavraki, *Computation of Configuration-Space Obstacles Using the Fast Fourier Transform*, Technical Report, STAN-CS-92-1425, 1992.

4.10 Status

Our research progresses according to schedule.

Chapter 5

Applications and Technology Transfer

It is not possible to develop generic technology without multiple, specific applications to test and refine the ideas and implementations. As such, we are actively seeking sites, both internally and externally to provide the compelling test beds that will make this project succeed. These driving applications span a variety of the most important target users: high-performance control, intelligent machine systems, underwater vehicle command and control, and remote teleoperation. Several of these projects will reach for new limits in advanced technology and system integration; others will address real-world problems in operational systems.

With the reduced funding levels, we will not have the resources to support all of the originally proposed technology evaluation sites. However, we believe these sites are crucial to the development of ControlShell into a viable technology for "real-world" use. Thus, we have actively pursued alternative means of supporting external sites. We have been successful in securing several new test applications. These sites will either function with minimal support, or fund their own support.

This chapter highlights some of the activities of these projects. The highlights this quarter include:

Two NASA centers have become ControlShell sites.

A summary of the MBARI underwater vehicle activity is presented.

The currently-active ControlShell applications are:

Robotic Composite Layup, by Rick Hosey (joint project between the ARL and the Stanford Structures and Composites Laboratory).

Precision Machining, by The Stanford Quiet Hydraulics Laboratory.

Underwater Vehicle Control, a joint project between the ARL and the Monterey Bay Aquarium Research Institute. (progress report below)

Intelligent Machine Architectures, by Lockheed Missiles and Space Corporation.

Remote Teleoperation, by Space Systems Loral Corporation.

Space-based Mobile Robot Systems, by several ARL students (NASA-sponsored).

High-Performance Control of Flexible Structures, by several ARL students (AFOSR-sponsored).

Two NASA sites have selected ControlShell for their applications.

5.1 Underwater Vehicle Application

Unmanned underwater vehicles (UUV's) have proven themselves to be useful tools in the exploration of biological, geological and chemical phenomena in the deep ocean. However, the operational utility of current vehicles is bounded by the limited capability of current control systems. To advance the technology of robotic vehicles used in ocean exploration, the Monterey Bay Aquarium Research Institute (MBARI) and the Stanford Aerospace Robotics Laboratory (ARL) have begun a joint program where students, scientists and engineers will pool their talents to tackle issues in unmanned underwater vehicle control.

The ability to test software before full implementation on a expensive testbed vehicle cannot be underestimated in value, not only for the opportunity to find errors which may cause expensive failures but also for the capability of software and algorithm development independent of hardware which is often being modified and thus inoperable. Indeed the time where all of the hardware systems are up and running is very limited in a experimental setting. To this end we have made built a simulator of our vehicle that is able to test fully the code that will actually control the undersea robot.

Our simulation capability was relatively easily developed in the ControlShell environment. ControlShell allows us to build control and filter components whose output can be linked to the real hardware or to a simulation module. Real sensor signals and simulated sensors can be mixed as input to the control system in a similar manner. Thus we have the ability to test real hardware components individually while running the actual algorithms which will be controlling the vehicle. For those components which are non-operational, simulated modules are inserted in their place.

We can observe and record data using two different tools. One is StethoScope which gives us the capability of real-time data acquisition and display of data flowing through the control system. Another tool uses the Network Data Delivery Service (NDDS) as a means of retrieving data across the network from the real-time systems to a graphical display running on a Unix workstation. NDDS has the tremendous advantage of providing a simple and uniform interface to the complex problem of distributed control and data across multiple processor and architecture systems.

For example, we are running both Sun and HP Unix workstations as our development and user interface environment. The control of the vehicle resides onboard under the VxWorks operating system on 680x0 single board processors in a VME cage. The communications task between processes and data distributed across this heterogeneous environment is tremendously simplified by using a communications package/protocol that allows data transfer at high data rates. Because it is opaque to each module where the data is generated or being sent, attaching simulated modules generating test data or planning modules generating vehicle commands does not need to affect the other modules consuming or producing the associated data and commands.

We are now exploring the use of ControlShell's Finite State Machines to implement the coordination level of our vehicle. It is at the coordination level where simple tasks that the vehicle can perform autonomously are encoded. With these tasks, such as transit to a specified location, search for and track a marine creature, or pick up a geological specimen, scientists will be able to direct an underwater vehicle to perform their research. Tasks are semi-autonomous actions which are clearly defined and can be enacted solely through sensor feedback. In the creation of a task, the programmer is encoding knowledge and prediction of how the environment should change under the actions being commanded. When certain conditions arise, different pre-programmed actions will take place. The coordination of these actions and the reaction of the control system to the changes is what need to be encoded in a straight forward manner.

ControlShell's Finite State Machine (FSM) provides the mechanism in which actions can be coordinated and reaction to external events achieved. Control programmers can quickly build states and transitions associated with stimuli that will guide the vehicle through the completion of a task. The task-level controller of the OTTER vehicle will be mainly instantiated using FSM's. NDDS serves as the communications layer between the various distributed modules of the vehicle system.

5.2 NASA sites

A project at NASA's Langley Research Center, led by William Doggett, has elected to utilize ControlShell. The project focuses on end-effector development, system integration, and tools to ease robotic space construction and planning. The system will control a Puma-configuration robot building a representative space structure.

We are quite excited about working with this group at NASA Langley. They will provide a significant external application test for our environment. We plan to ship a version of ControlShell to them in early January.

Another NASA lab, the intelligent mechanisms group at NASA Ames, is now planning to utilize ControlShell for a mobile robotics project. This project's goals include studying intelligent exploration algorithms and robotic control architectures. We re targeting a shipment to them in Q2 93.