Preprints of the Third International
Symposium on Experimental Robotics
Kyoto, Japan, Oct. 28-30, 1993

Experimental Integration of Planning in a Distributed Control System*

Gerardo Pardo-Castellote†, Tsai-Yen Li‡,
Yoshihito Koga§, Robert H. Cannon, Jr.¶,
Jean-Claude Latombe‖, Stanley A. Schneider**
Stanford University
Stanford, California 94305

Abstract

This paper describes a complete system architecture integrating planning into a two-armed robotic workcell. The system is comprised of four major components: user interface, planner, the dual-arm robot control and sensor system, and an on-line simulator.

The graphical user interface provides high-level user direction. The motion planner generates complete on-line plans to carry out these directives, specifying both single and dual-armed motion and manipulation. Combined with the robot control and real-time vision, the system is capable of performing object acquisition from a moving conveyor belt as well as reacting to environmental changes on-line.

The modules communicate through a novel subscription-based network data sharing system called the *Network Data Delivery Service* (NDDS). NDDS allows the different modules to transparently share data, and thus be distributed across different computer systems. Its stateless protocol naturally supports multiple anonymous data consumers and producers, arbitrary data types, on-line reconfiguration and error recovery.

The control software is integrated within the *ControlShell* framework. *ControlShell* provides an object-oriented generic software framework for combining reusable software components into a working, complex system.

This paper presents an overview of the individual system components, as well as a summary of the architecture developed to integrate the system. Much of the paper focuses on the interfaces between components.

1. Introduction

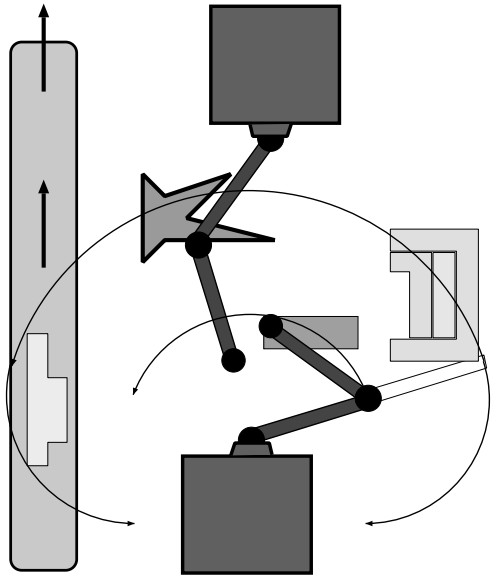

Figure 1. **Experimental Demonstration**

Experimental demonstration consisting of a robotic assembly in the presence of moving objects. The robot has two 4-DOF arms. The parts are delivered by a conveyor.

Automation and ease-of-operation are two goals of robotic systems. Ideally, one would specify a high-level task such as an assembly and have it executed automatically. To achieve these goals, sophisticated software modules such as planners, user interfaces, controllers etc. are being developed. However, the complexity of these modules and the fact that they are often developed at different times by different groups of people make system integration and testing very time consuming and often problem-specific.

*This work was supported in part by ARPA/Navy Contract No. N00014-92-J-1809.

†Department of Electrical Engineering.
‡Department of Mechanical Engineering.
§Department of Mechanical Engineering.
¶Department of Aeronautics and Astronautics.
‖Department of Computer Science.
**Real Time Innovations Inc.

In a joint effort, the Computer Science Robotics Laboratory and the Aerospace Robotics Laboratory at Stanford University have developed a flexible experimental test-bed to explore these issues. Our goal is to achieve task-level operation on a distributed robotic system in a dynamic environment.

The experimental demonstration is illustrated in Figure 1. Two 4-DOF arms manipulate parts in a dynamic environment containing both *static* and *moving* obstacles and parts. The parts are supplied by a conveyor. A vision system identifies and tracks the moving parts which are picked-up by the robot *while in motion*. Several efficient path planning modules are also implemented to find trajectories to deliver and assemble parts while avoiding the obstacles in the workspace. Due to their size and weight, some of the parts require cooperative manipulation and regrasping by the two arms whereas others are manipulated by a single arm. The user monitors and issues task-level commands using a graphical user interface.

2. System Architecture

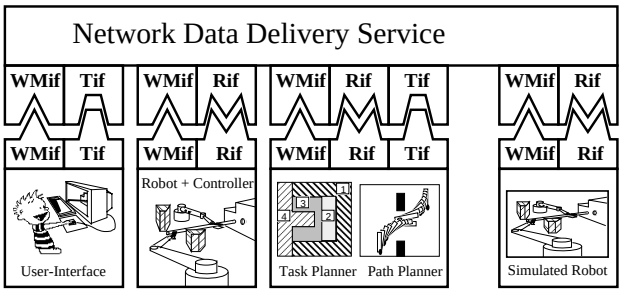

Figure 2. **System Architecture**

The overall system showing its four main modules. Each module communicates using one or more of the three interfaces: The World Model Interface (WMif), the Robot Interface (Rif) and the Task Interface (Tif). These modules are physically distributed. The Network Data Delivery Service plays the role of a bus providing the necessary interconnections.

Our experimental test-bed is composed of four modules as illustrated in Figure 2. The user interface receives state information from the robot and sensor systems and provides a graphical representation of the scene to the user. During operation, the high-level task is specified from the graphical user interface. The task planner/path planner receive continuous updates from the robot and sensors. The planners produce robot-commands primitives which are executed by the robot controller.

The simulator and the robot have the same interface to the other modules. This allows the simulator to masquerade as the robot for fast prototyping and testing of the rest of the system.

A novel *network-transparent*, *subscription-based* data-sharing scheme—the *Network Data Delivery Service* (NDDS)—facilitates communication among the different modules. It allows them to be distributed across different computer systems (with different processor architectures and operating systems[1]) and provides the necessary arbitration between data updates, enabling multiple users to operate and monitor the system concurrently.

The NDDS system [9] builds on the model of information producers (sources) and consumers (sinks). Producers register a set of data instances that they will produce, unaware of prospective consumers and "produce" the data at their own discretion. Consumers "subscribe" to updates of any data instances they require without concern for who is producing them. In this sense the NDDS is a "subscription-based" model. NDDS provides stateless (and hence robust) mechanisms to resolve multiple-producer conflicts and supports multiple-rate consumers.

The use of subscriptions to drastically reduces the update overhead over a classical client-server architecture. Occasional subscription requests, at low bandwidth, replace numerous high-bandwidth client requests. Latency is also reduced, as the outgoing request message time is eliminated.

NDDS differs from other distributed data-sharing schemes [17, 8, 3, 14] in its transparent support for multiple anonymous data-producers and consumers as well as in its fully-distributed, symmetrical implementation (which contains no privileged nodes).

All modules in the system communicate using one of the following three interfaces (built on top of NDDS): The *World Model* interface, the *Robot* interface and the *Task* interface. The functionality of two of these interfaces is summarized in tables 1 and 2.

This arrangement is analogous to a hardware bus as illustrated in Figure 2. NDDS plays the role of the physical interconnections. The three interfaces being similar to bus-access protocols. This approach is key to reducing system integration time and produces generic, reusable modules.

3. Controller

The robot system consists of two four degree-of-freedom SCARA manipulators equipped with joint

[1]The current demonstration involves DEC workstations, Sun Workstations and several VME-based real-time processors

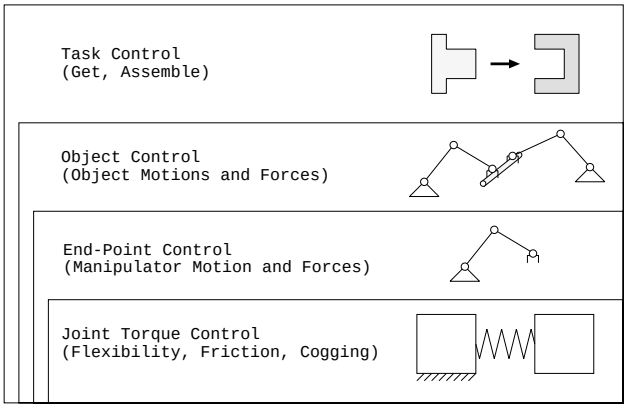

Figure 3. **Four-level control hierarchy for two-armed robot.**

We use a four layer hierarchy to control the two-armed robot. At the lower joint *level, we use joint-torque sensors to compensate for the non-idealities of the motor (cogging, non-linearity) and the joint dynamics induced by the joint flexibility. The next layer the* arm *level control can now assume ideal actuation (i.e. the motors deliver the desired torque to the link itself) and use a Computed Torque approach to compensate for the non-linear arm dynamics. The third* object *layer, is concerned with object behavior and assumes that the arms are virtual multi-dimensional actuators that apply torques to the object. The top layer implements elementary tasks such as object acquisition and release, insertions etc.*

torque sensors, joint encoders and an end-point 6-DOF force sensor. An overhead vision system provides global sensing of the position of both the robot and objects. The robot is controlled from a VME-based real-time computer system.

The control software is organized in a four-layer hierarchy originally presented in [10]. The highest level of the control hierarchy illustrated in Figure 3 uses Finite State Machines (FSMs) to coordinate the actions of the two arms and react to both external and internal events. The current implementation uses three FSMs: A global FSM and individual ones for each of the two arms. The global FSM receives commands from the task-planner and may initiate a cooperative two-arm action and/or send stimuli to the individual arm FSMs. Capture of moving objects from the conveyor is achieved using FSM subchains. Subchains are FSM subprograms analogous to subroutines in conventional programming. For example, the subchain in the FSM illustrated in figure 6 is used to perform single-arm object acquisitions from a conveyor.

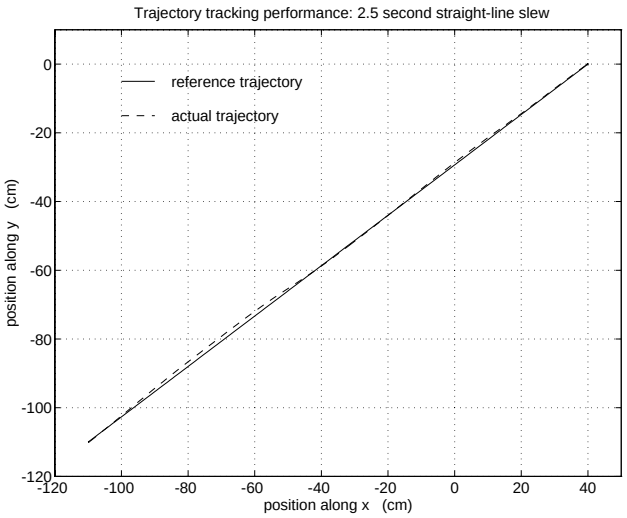

Figure 4. **Experimental tracking performance for right arm.**

Illustration of the tracking response of right arm. The reference is a fifth order polynomial trajectory for the arm endpoint commanding it to follow a 1.75 m straight line path in 2.5 sec. This trajectory requires accelerations of up to 4.3 m/s^2 (close to 1/2 g). The maximum tracking error is 1.4 cm.

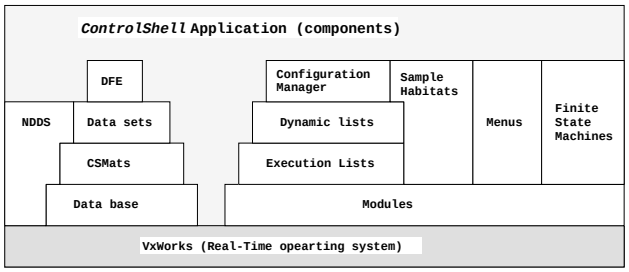

Figure 5. *ControlShell* **Structure.**

The right side of the diagram denotes the "execution" hierarchy; the left side is the "data" hierarchy. The application, consisting partially of a set of reusable components, has access to all facilities at every level. ControlShell *provides a layer on top of the real-time operating system VxWorks.*

The next level of the hierarchy (object control) commands the arms to achieve the desired object behavior. The controller used enforces the Virtual Object Impedance Control policy [12]. The third layer

uses a Computed Torque approach to achieve dynamic arm control and, at the lowest (joint) level, higher bandwidth joint-torque control-loops are used to control the joint flexibility and compensate for the non-idealities of the motors. Figure 4 illustrates the trajectory-tracking performance of a single arm following a straight line.

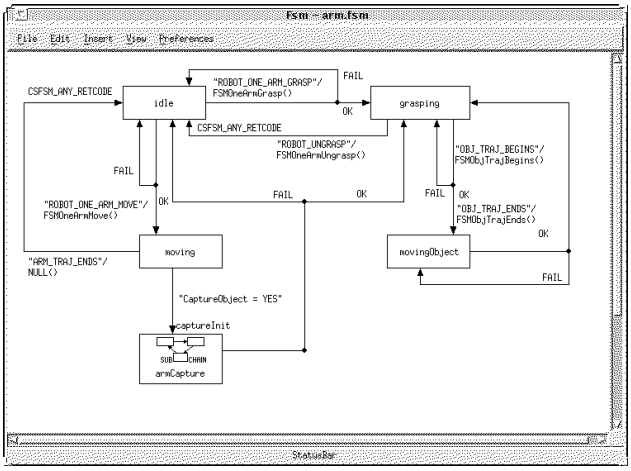

Figure 6. **Finite-State Machine (FSM) used to control an individual arm.**

Stimuli sent to the FSM (shown in quotes) cause the associated transition routines to execute. The return code of these transition routines determines the next state. Subchains are FSM "subprograms".

With the increased complexity demanded from distributed control-systems, the use of CASE tools that facilitate the programming task is becoming essential [2, 15, 13]. All the real-time software in this project has been implemented using the *ControlShell* framework. *ControlShell* [5] is a CASE tool that enables modular design and implementation of real-time software. It contains an object-oriented toolset which provide a series of execution and data exchange mechanisms that capture both data and *temporal* dependencies. This allows a unique *component-based* approach to real-time software generation and management. *ControlShell* defines temporal events, and provides mechanisms for attaching routines to those events. It provides data structure specifications, and mechanisms for binding data and routines while resolving data dependencies. *ControlShell's* event-driven finite state machine provides the means to weave asynchronous events into a sequential execution stream. *ControlShell* also includes numerous code-generation and maintenance tools such as a graphical component editor, a finite state machine editor, a data-flow editor, an interactive menu facility etc. The structure of *ControlShell* is summarized in figure 5.

Object information from World Model	
location	object location in the global reference frame.
grasps	positions within the object where a grasp is possible
properties	Mass and inertia of a object, whether it can be moved by the robots etc.
shape	object shape for collision avoidance purposes.
Robot information from World Model	
location	robot location in the global reference frame.
joint values	value of each of the joint coordinates
joint limits	limits on the joint coordinates
Denavit-Hartenberg Parameters	Description of robot kinematics
state	Whether the robot is moving, grasping an object etc.

Table 1. Information Available using the World-Model interface

4. Planner

The automatic generation of robot paths for accomplishing a user-specified task is one of the key elements in our automated robot system. Since our goal is to build such an interactive system, the on-line motion planning capability is crucial. Indeed, it is unacceptable for a user to specify a task and then have to wait a few minutes for the motion of the robots to be planned and executed. Due to the high computational cost of robot motion planning [1, 16], we focus our effort on developing effective strategies and efficient path planning algorithms to achieve on-line performance.

Alami, Siméon, and Laumond [11] have proposed and implemented an algorithm for the case of one robot and several movable objects. They assume both a finite number of possible grasps and a finite number of object placements within the workspace. The robot grasps one object at a time and the remaining (currently non-grasped) objects are treated as static obstacles. Their algorithm first considers all the possible arrangements of the robot grasping the object and then decomposes into cells the collision-free subset of these arrangements. These cells are subsequently connected by links which correspond to regrasping oper-

command	*meaning*
move object	Move an object that is being grasped. This command will provide a via-point collision-free path for the object. The robot is controlled using object impedance control.
move arms (operational space)	Move the arms. This command assumes the arms are not grasping an object. The command provides a via-point collision-free path for the arm-endpoints.
move arms (joint space)	Move the arms. This command assumes the arms are not grasping an object. The command provides a via-point collision-free path for the arms joints (This is provided to resolve kinematic ambiguities).
grasp	Grasp an object. This command specifies the object to be grasped.
un-grasp	Un-grasp an object.

Table 2. Robot-Command primitives available through the Robot interface.

ations of the robot. The resulting graph, called the *Manipulation Graph*, is then searched for a path.

In [4], the authors consider a similar scenario dealing with two robots and multiple moving objects on a conveyor belt. The authors present an A^* algorithm to search for a locally-optimal motion sequence for a given assembly represented as a AND-OR graph. In this work, they assume that only the last links of the arms may collide with each other and that the parts are fed slowly enough that the robots can always pick up objects from locations that do not deviate too much from their nominal positions.

Our research differs from these related work in that we consider a more general problem with no prior knowledge about the speed, the feeding rate, the picking position, the picking arm, or the sequence of parts.

The planning system consists of two modules: the *Task Planner* and the *Path Planner*. The task planner is responsible for determining how to utilize the resources (in our case, two manipulator arms) to accomplish the user specified task. The result is a decomposition of the problem into subtasks, which are then sent to the path planner in order to find the necessary motion of the manipulators.

4.1. Task Planner

The role of the task planner is first determine how to solve the user-specified task (e.g. put object A at location P), then make use of the path planning algorithms to actually find the sequence of manipulator motions to complete the task and, finally to send the result to the robots in terms of the robot-command primitives listed in table 2.

The user-specified task may involve moving multiple objects to a set of specified goals. Because of this, the task-planner will break the task down into subtasks and, among other things, determine which object to manipulate first. The criteria for choosing the next object to manipulate is priority-based. The priority of an object is based on the expected time that the (moving) object will take before it leaves the work cell. For example, static objects have lower priorities than objects on the conveyer belt. Once the next object to be manipulated is identified, a manipulator arm(s) is selected to pick-up and deliver the object. The criteria used to select the arm(s) takes into account whether the object/task requires two-arm manipulation, which arm(s) is free (i.e. not currently involved in another task), which arm is closer to the object, the expected time it will take for the object to leave the workspace of each arm, etc. At this point, rather than solving the complete manipulation path, that is moving the arm to grasp the object, grasping it, carrying it to the goal, and then ungrasping it, the task planner further divides the problem into two subtasks. The first subtask is moving the arm to grasp the object, while the second subtask is to deliver the object to its goal.

The task planner runs in a loop selecting the highest priority subtask to plan according to the current state of the world (this state includes both user-specified tasks and subtasks that are already "in-progress"). For example, if there are objects that need to be moved within the workcell, the task planner will detect this at the start of the loop, select the object with highest priority, and invoke the path planner to find a trajectory for one (or both) of the arms to grasp the object. Each time around the loop only the subtask with the highest priority is sent to the path planner. In the event that the planner fails to solve the highest priority subtask, the subtask with the next highest priority is attempted. Once a plan is found, it is broken down into individual robot-command primitives and sent to the robot for execution. Once the motion of the robot (as a response to the command) is detected, the subtask becomes "in-progress" and the task planner returns to the top of the loop. In the event that the execution of this subtask become questionable (e.g. the obstacles have moved or the goal position has been changed) the task planner will first determine if the plan is still valid and if not it will replan accordingly.

4.2. Path Planner

The subtasks that are requested by the task planner to be solved are the following. Note that a free arm refers to a robot arm that is not committed to any subtask.

- Move one arm to grasp a static object while the other arm is free.
- Move one arm to grasp a static object while the other arm is moving.
- Move one arm to catch a moving object while the other arm is free.
- Move one arm to catch a moving object while the other arm is moving.
- Deliver the object that is grasped to its goal location while the other arm is free.
- Deliver the object that is grasped to its goal location while the other arm is holding another object.
- Move two arms to grasp a static object.
- Move two arms to catch a moving object.

Notice that we do not consider the case where both arms deliver two independent objects at the same time or the case where one arm is moving and the other arm delivers its object to the goal. For these cases we decouple the problem and plan the motion of the arms sequentially. Our reasoning is that though the *robot execution time* for this decoupled approach may be slightly longer than if the arms moved simultaneously, the *planning time* to find the sequential motion will be significantly shorter than the time to find the simultaneous motion of both arms. Indeed since efficiency is the key issue in on-line motion planning, we make several assumptions to simplify the path planning problem associated to each subtask and build a library of efficient primitives to solve them.

In our scenario, reasonable simplifications can be made to reduce the size of the search spaces implied by each subtask, and thus reducing the time required to solve them. Due to the fact that the first two links of the SCARA-type arms move in a plane and our assembly task only involves pick-and-place operations, we simplify the motion planning problem in the three dimensional workspace into a problem of two dimensions. This assumes that when the end-effectors of the arms are as high up as they can go, the arms can move in an unrestricted manner above the obstacles in the workspace. This reduces the search space of each arm from three dimensions to two[2]. Once the arm(s) are grasping an object, the size of the object is considered to be such that it can collide with the obstacles in the workspace–that is, the arms are unable to lift the object above the obstacles. The result is a three dimensional search space which fortunately still presents little difficulty for fast computation.

For each of the aforementioned search spaces, we build a path planning primitive to find a collision-free path connecting two configurations in the particular search space. These primitives are extremely efficient and can find paths in a fraction of a second. Some of them are based on existing algorithms [7, 6], while the others are completely new. We then associate a strategy with each subtask which utilizes these primitives to solve it. For example, the subtask of moving one arm to catch a moving object while the other arm is free has the following strategy. First the path of the moving arm to grasp the object is found while ignoring the presence of the free arm. The second step is to have the free arm comply with the motion of this moving arm. If the subtask is solved, the corresponding motion of the robots is returned to the task planner. Otherwise, the task planner is notified that the particular subtask could not be solved.

Our experiments demonstrate that this planning approach yields on-line performance.

5. Discussion and Conclusions

This paper has presented the overall design of an experimental test bed developed to explore the integration of planning in distributed control systems.

We have concentrated on several key aspects of this problem. First, we have developed a powerful distributed network communication architecture (NDDS), and implemented it within a generic real-time programming framework (ControlShell). We have defined clear interface specifications, and implemented simple-yet-powerful robotic system modules within that architecture.

In particular, we have developed and implemented a graphical user interface, a four-level dynamic control hierarchy, an on-line simulator, and an extremely fast on-line motion and task planning capability. The unique aspects of these modules are described briefly above.

This paper describes work in progress by the authors. Our experiments with *non-moving* objects have shown the power of our approach. Numerous tests in *simulation* have shown the system to be able to oper-

[2] the complexity of motion planning grows exponentially with the dimension of the search space

ate in the presence of moving objects. We will experimentally demonstrate assembly with moving parts in the immediate future.

References

[1] J.F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[2] A. Joubert D. Simon. The orccad: Towards an open robot controller computer aided design system. Research Report 1396, INRIA, February 1991.

[3] Chris Fedor. TCX task communications. School of computer science / robotics institute report, Carnegie Mellon University, 1993.

[4] Li-Chen Fu and Yung-Jen Hsu. Fully automated two-robot flexible assembly cell. In *IEEE International Conference on Robotics and Automation*, pages 332–338, Atlanta, Georgia, USA, May 1993.

[5] Real-Time Innovations Inc. *ControlShell: Object Oriented Framework for Real-Time Software User's Manual*. 954 Aster, Sunnyvale, California 94086, 4.2 edition, August 1993.

[6] Y. Koga and J.C. Latombe. Experiments in dual-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 2238–2245, Nice, France, May 1992.

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

[8] MBARI (Monterrey Bay Aquarium Research Institute. Data manager user's guide. Internal Documentation, 1991.

[9] Gerardo Pardo-Castellote and Stanley A. Schneider. The Network Data Delivery Service: Real-Time Data Connectivity for Distributed Control Applications. In *Proceedings of the International Conference on Robotics and Automation*, San Diego, CA, May 1994. IEEE, IEEE Computer Society.

[10] Lawrence E. Pfeffer. *The Design and Control of a Two-Armed, Cooperating, Flexible-Drivetrain Robot System*. PhD thesis, Stanford University, Stanford, CA 94305, December 1993. Also published as SUDAAR 644.

[11] T. Siméon R. Alami and J.P. Laumond. A geometrical approach to planning manipulation tasks: The case of discrete placements and grasps. In H. Miura and S. Arimoto, editors, *Robotics Research 5*, pages 453–459, Cambridge, MA, 1990. The MIT Press.

[12] S. Schneider and R. H. Cannon. Object Impedance Control for Cooperative Manipulation: Theory and Experimental Results. *IEEE Journal of Robotics and Automation*, 8(3), June 1992. Paper number B90145.

[13] Reid Simmons and Chris Fedor. Task control architecture programmer's guide. School of computer science / robotics institute report, Carnegie Mellon University, 1992.

[14] Sparta, Inc., 7926 Jones Branch Drive, McLean, VA 22102. *ARTSE product literature*.

[15] D. B. Stewart, D. E. Schmitz, and P.K. Khosla. Chimera ii: A real-time multiprocessing environment for sensor-based robot control. In *Proceedings of the IEEE International Symposium on Intelligent Control*, Albany, NY, September 1989.

[16] G. Wilfong. Motion planning in the presence of movable obstacles. In *4th ACM Symp. of Computational Geometry*, pages 279–288, Urbana-Champaign, Illinois, June 1988.

[17] J. D. Wise and Larry Ciscon. *TelRIP Distributed Applications Environment Operating Manual*. Rice University, Houston Texas, 1992. Technical Report 9103.