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

1. Introduction

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 happen automatically. To achieve these goals, sophisticated 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.

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 acquired by the robot *while in motion*. Several efficient path planning modules are also implemented 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 while others are manipulated by a single arm. The user monitors and issues task-level commands using a graphical user-interface.

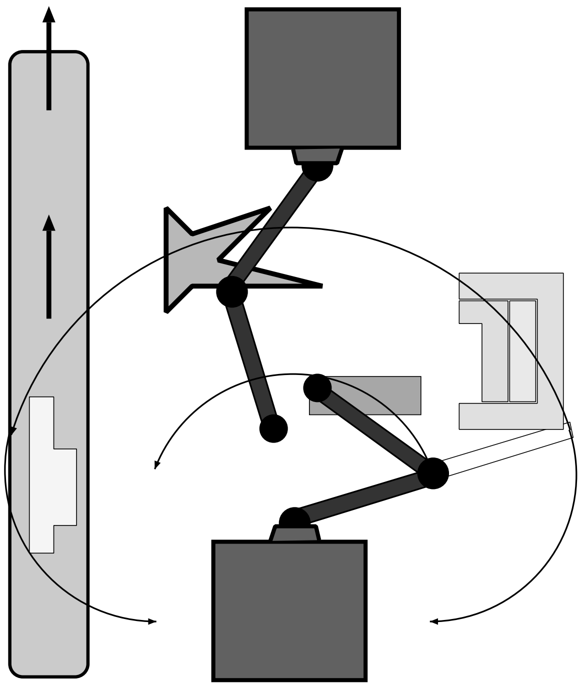

Figure 1. **Experimental Demonstration**

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.

2. System Architecture

Our experimental test-bed is composed of five 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 and the path planner receive continuous up-

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

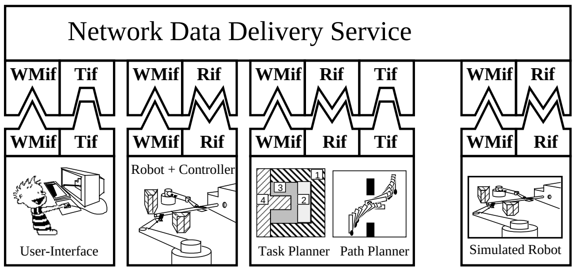

Figure 2. **System Architecture**

The overall system showing its five 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.

World Model information for objects	
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.

World Model information for robots	
location	robot location in the global reference frame.
joint values	value of each of the joint coordinates
joint limits	limits on the joint coordinates
Denabbitt-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

dates from the robot and sensors and produce primitive robot-commands. The robot controller executes these commands.

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 workstations and provides the necessary arbitration between data updates enabling multiple users to operate and monitor the system concurrently.

The NDDS system 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.

Using subscriptions allows us to drastically reduce the overhead required by a 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.

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.

All modules in the system communicate using one of the following three interfaces: 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. The Network Data Delivery Service plays the role of the physical interconnections and the three interfaces are 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 torque sensors, joint encoders and an end-point 6-axis force sensor. An overhead vision system provides global sensing. The robot is controlled from a VME-based real-time computer system. The controller enforces the Virtual Object Impedance Control policy [6]. This control policy is implemented using a four-layer control structure initially developed by [5]. This structure is shown in figure 3. Figure 4 illustrates the trajectory tracking performance of the system.

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. Commands available using the Robot interface.

All real-time software has been developed using the *ControlShell* framework [2]. *ControlShell* enables modular design and implementation of real-time software. Its object-oriented tool-set provides 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. Specifically *ControlShell* clearly 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 the graphical component editor, the finite state machine editor, the data-flow editor, the interactive menu facility etc. The structure of *ControlShell* is summarized in figure 5.

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 we aim 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, 7], we focus our effort on developing effective strategies and efficient path planning algorithms to achieve the on-line performance.

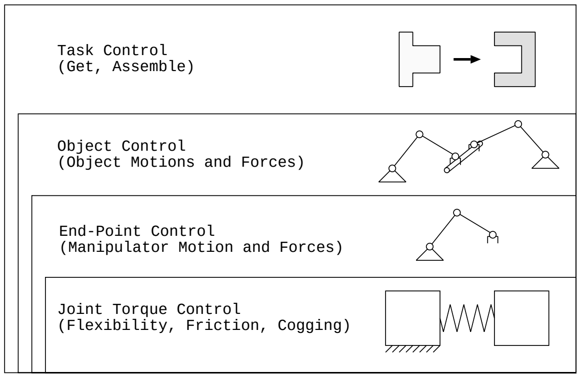
Task Control
(Get, Assemble)
Object Control
(Object Motions and Forces)
End-Point Control
(Manipulator Motion and Forces)
Joint Torque Control
(Flexibility, Friction, Cogging)

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

We use a three 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. On top 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 outer layer implements elementary tasks such as object acquisition and release, insertions etc.*

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 robot 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 for finding the necessary motion of the robots.

4.1. Task Planner

The role of the task planner is to 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 robot motions to complete the task, and finally to send the result to the robots in terms of primitive commands (e.g. move arm X along the trajectory, close the gripper of arm Y).

The task is broken down into parts. Since it may in-

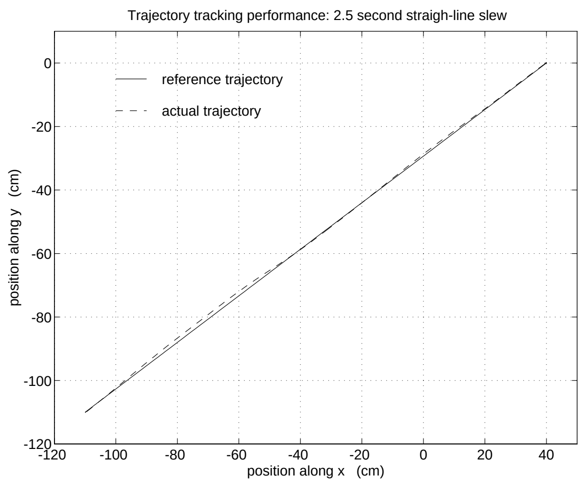

Figure 4. **Trajectory tracking performance for right arm.**

Illustration of the tracking response of right arm. The reference is a fifth 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 $4.3m/s^2$ *(close to* $1/2g$*). The maximum tracking error is 1.4 cm.*

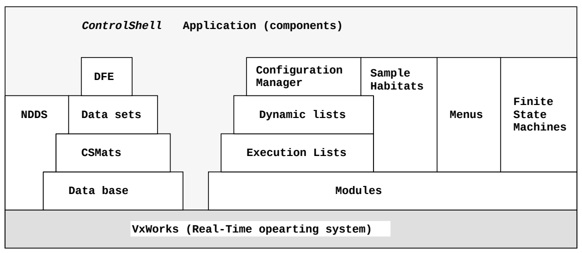

Figure 5. *ControlShell* **Structure.**

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.*

volve moving multiple objects to their specified goals, the task planner must determine which object to manipulate first. The criteria for choosing this object is based on the expected time that it will take for the object to leave the work cell. For example, static objects have lower priorities than the objects on the conveyer belt (i.e. objects on the conveyor belt will leave the workcell sooner). Once the object to be manipulated is identified, an arm is selected to grab and deliver the object. The criteria used to select the arm takes into account which arm is free, which arm is closer to the object, how long does the objects takes 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 and decides what to plan according to the current state of the world. For example, if there are objects in the workcell, the task planner will detect this at the start of the loop and will call the path planner to find a trajectory for one of the arms to go and grasp the object with the highest priority. In each 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 sent to the path planner - it may be the same object but this time using a different arm. Once a trajectory is found, it is sent to the robot for execution. After the motion of the robot is detected, the task planner returns to the top of the loop. In the event that the execution of this subtask become questionable - because the obstacles have moved or the goal position has been changed - the task planner will first determine if the path is still valid and if not it will replan accordingly and send the new path to the robot.

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.

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, efficiency is the key issue in on-line motion planning. We make some 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[1]. Once the arm(s) are grasping an object, the size of the object is 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 [4, 3], while the others are completely new. To each subtask we then associate a strategy that 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.

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

5. Current Status

We have completed the test-bed and performed several assembly experiments with *non-moving* obstacles. We are in the process of adding the conveyor and testing the dynamic aspect of our planners.

References

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

[2] Real-Time Innovations Inc. *ControlShell A Real-Time Software Framework User's Manual.* 954 Aster, Sunnyvale, California 94086, 1st edition, November 1992.

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

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

[5] 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. To be published.

[6] 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.

[7] G. Wilfong. Motion planning in the presence of movable obstacles. In *4th ACM Symp. of Computational Geometry*, pages 279–288, 1988.