An Intelligent Software Architecture for Semi-autonomous Robot Control

David Kortenkamp, Robert Burridge, R. Peter Bonasso, Debra Schreckenghost and Mary Beth Hudson

Metrica, Inc./TRACLabs
1012 Hercules
Houston, TX 77058
kortenkamp@jsc.nasa.gov

Abstract

This paper describes a software architecture that allows for traded control of a robot manipulator performing a pick-and-place task. The software architecture combines multi-agent planning, conditional sequencing, robot control and user interfaces to allow for a user to intervene at any level of robot task execution. At the heart of the software architecture is the 3T architecture, which allows for hybrid deliberative/reactive control of a robot. 3T is augmented with user interface tools that give the operators insight into the robot's actions and possible operator actions.

Introduction

Our long-term goal is to create intelligent, autonomous robots capable of coordinating and cooperating with each other and humans in order to accomplish complex tasks. In particular, we are interested in developing robots for environments that are inhospitable for humans, such as nuclear reactor cores, underwater, and in space. Unfortunately, due to sensor or actuator limitations, there are many useful tasks for which today's technology simply can not produce a fully autonomous robotic system. In such situations, the first solution is to design a system where a human in a safe location can tele-operate the robot, providing the "brains" of the control loop. Still, tele-operated systems can have unacceptable time delays between sensing and acting, making them inappropriate for more dexterous applications. Recent advances have led to *semi-autonomous systems*, which operate in the middle ground between full autonomy and tele-operation. The faster, reactive decisions are made on-board the robot, and the slower, deliberative decisions are made remotely by a human. Typically, such systems provide only the most basic "obstacle-avoidance" type of behavior on-board, and little or no support to aid the human with higher-level decisions. They also have no mechanism for changing the level of human involvement in the task, which may be desirable for training new operators, or when unusual situations occur.

We have begun developing a robot control architecture that can be used to create control systems operating anywhere in the continuum of semi-autonomous systems, from purely tele-operated to fully autonomous. These systems are capable of working cooperatively with humans to accomplish a task that neither is capable of doing alone. It is possible for the user to change the level of autonomy, and thus the level of human involvement, while the controller is actively running. As an important and integral part of our robot control architecture, we have developed user interface tools that can be used to create user interfaces that allow the user to interact with the control system at all levels of abstraction, from direct control of robot joints to cooperative long-term plan generation. Although designed for robot control, the architecture is flexible enough to be used with almost any system that has accessible input and output.

Issues in traded control

Effective traded control requires a robot system that can both perform routine operations autonomously and give control to a human to perform specialized or difficult operations. The advantage of a traded control system is that the unique (and expensive in space situations) capabilities of a human can be brought to bear when needed most and not during tedious, repetitive and routine operations. However, a problem with traded control is that the robot does not know what the state of the world or of the task will be when the human finishes his or her portion of the job. This can make it difficult and dangerous for the robot to resume autonomous operation. As an extreme example, the human may forget some crucial aspect of their portion of the task (such as securing a fastener) that the robot is expecting to be accomplished. Less drastic, but probably more commonplace would be subtle side effects of the human's performance such as putting a tool in a slightly different orientation than is expected by the robot. In either case, the problem is that the

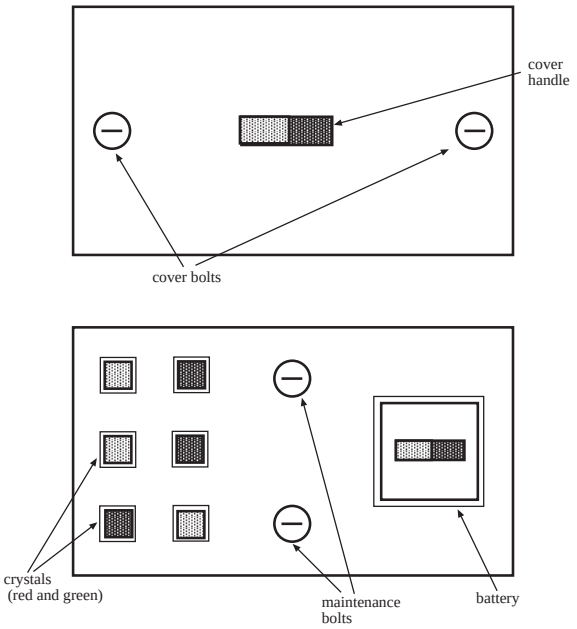

Figure 1: The experiment bay, or TaskBoard, used in our scenario. The top part of the figure shows the hard cover that sits on top of the bay. The bottom part of the figure shows the experimental area.

robot's model of the world and of the task are not consistent with the real state of the world and of the task.

Effective traded control also requires that the robot system know when it can safely perform a task, and when a human should be performing it. Ideally, the robot would plan both its own activities and those required of the human with whom it will be trading control. In this case, the robot will proceed autonomously until reaching the point where human intervention is required; the robot will then inform the human and safely wait until the human is ready to accept control. The robot system should also recognize situations which are off-nominal and stop to ask for human assistance even if human assistance was not originally required for this step of the plan. The robot system will then need to account for the human intervention and replan its task. Because our architecture has the ability to perform multi-agent planning and replanning with its top tier, and the ability to recognize off-nominal situations with its middle tier, it is well-suited for traded control situations.

Example task

We wanted to choose a task that would demonstrate our traded control philosophy and software architecture. The requirements were that the robot was capable of accomplishing most of the task, but that humans would need to be involved. The task we developed was to have a robot maintain an experiment bay, called the TaskBoard. The TaskBoard has several components. First, it has a shielding blanket, which can only be removed and replaced by a human at the TaskBoard location. Next, it has a hard cover with two bolts and a handle (see top of Figure 1). The robot needs to twist each bolt, then grab the handle and remove the cover. When the cover is removed the experimental area is exposed. There are three parts to the experimental area: 1) a battery; 2) two bolts; and 3) a set of "crystals." The bottom part of Figure 1 shows the experimental area. The robot has three tasks to do inside the bay. First, it needs to exchange the battery with a new one that it has resting by its side. The battery has a handle that allows the robot to grab it. The position and orientation of the battery handle is not known by the robot ahead of time and must be determined using vision. Second, the robot needs to tighten the two maintenance bolts to ensure that the experiment stays in place. Again, the robot does not know the orientation of these bolts ahead of time. Finally, the robot needs to deal with the crystals. The crystals come in two types – ready and not ready. Ready crystals are green, not ready crystals are red. The robot does not know ahead of time which crystals will be ready. The robot needs to pick up each ready crystal and place it before the cameras. At this point a human expert determines if the crystal should be placed back into the experiment bay or set aside and replaced with a new crystal. When the human's choice is transmitted to the robot it performs the appropriate actions. When all of the ready crystals have been dealt with, the robot replaces the hard cover and twists its bolts. Then a human replaces the shielding blanket and the task is done.

This task was chosen for several reasons. First, it is similar to a task that might be necessary in space. Second, it requires cooperation between a human and a robot and trading of control. Third, it offers opportunities for the human to take control of the robot to do certain parts of the task (i.e., adjust the viewing angle of the crystal or position the battery in a different location). Finally, it offers some *a priori* unknowns, such as the number and location of the crystals that are ready.

Hardware overview

For this project we built a six degree-of-freedom (DOF) manipulator. The robot arm has 5 DOF and can dexterously reach a table-top workspace of radius 0.6m. The final DOF is a parallel-jaw gripper. See Figure 2.

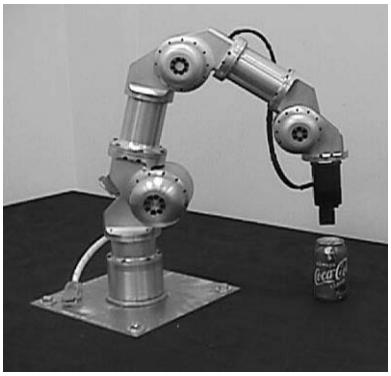

Figure 2: The six DOF manipulator designed for the project.

Figure 3: The vision system cameras and head.

The hardware set-up also includes a computer vision system consisting of four components: 1) Cameras; 2) Digitizers; 3) Pan-Tilt Head; and 4) Computer. The first three of these components are discussed in the next two subsections. The computer is discussed in the next section. Figure 3 shows the camera and the pan-tilt unit.

Related work

Related work for this research falls into several categories. First, there is work on tele-operation and traded control, which comes from the traditional robotics community. This includes work by Sheridan (Sheridan 1989) and work at NASA JPL (Backes *et al.* 1991). Second, there is work on intelligent control architectures from the AI community. This includes TCA (Simmons 1990), ATLANTIS (Gat 1992), Remote Agent (Muscettola *et al.* 1998), Cypress (Wilkins *et al.* 1995) and CIRCA (Musliner, Durfee, & Shin 1993). In this project we attempt to combine these two approaches in a single architecture.

Project Architecture

We have, over the last several years, developed an autonomous robot control architecture that approaches the general robot intelligence problem by separating the control into three interacting layers or tiers (and is thus known as 3T). The tiers are a Control Tier (with robot control skills), a Sequencing Tier and a Planning Tier. Each of these layers is described in the following subsections. This architecture has been applied to over a dozen robotic and non-robotic projects at NASA Johnson Space Center (Bonasso *et al.* 1997). For this project we added significantly to the 3T architecture in the area of user interfaces. The final software solution is shown in Figure 4.

Control

The Control Tier of 3T is used to produce the part of the Control System with the lowest level of abstraction and the fastest cycle time. It creates the only part of the Control System responsible for interaction with the physical mechanism, or robot. As such, it is expected to interpret all sensor input, relay it to the rest of the Control System as appropriate, interpret commands back, and generate output for the actuators.

The Control Tier is made up of *skills*. Each skill has input and output structures that, taken over all the skills collectively, define a directed graph of information flow among the skills. By activating, or **enabling** a subset of the skills, different behaviors can be evoked from the Skill Manager. Each skill may receive symbolic parameters from the Sequencer, which are a coding convenience for both ends. Thus, for instance, instead of having a different skill for moving the arm to every different symbolic location, a single skill can take "location" as a parameter and act accordingly. Each skill also has a frequency variable, which defines the number of times per second it is expected to run. It is important to remember, however, that the Skill Manager is a monolithic process, so only one skill is active at any given time, and if any skill takes too long or hangs, then the other skills will not run on time (or, possibly, ever). Nonetheless, skills are generally small and simple enough that dozens of them can run hundreds of times a second, even on today's desktop computers.

The skill set used for this project includes a core set of blocks that are designed to be enabled at startup and never disabled during normal operation. Most of these core blocks are interface skills, which establish communication with the GUI, Vision System, Simulation, and Robot. The other core blocks are devoted to safety and data logging. All of the core blocks have a *mode* parameter that can be changed by the Sequencer

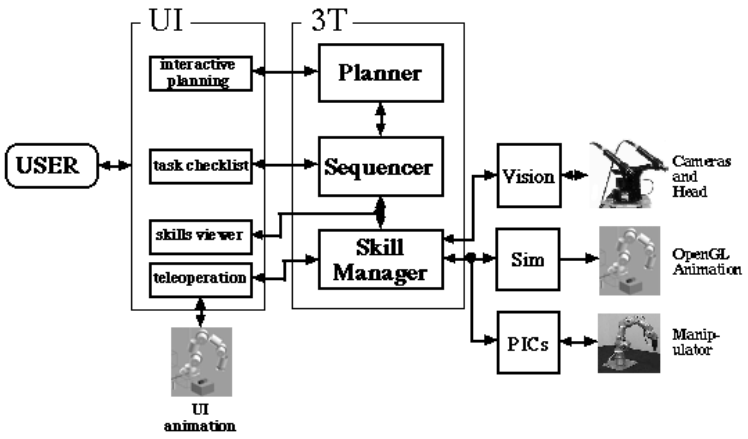

Figure 4: Overview of the software modules for traded control and for our specific TaskBoard scenario.

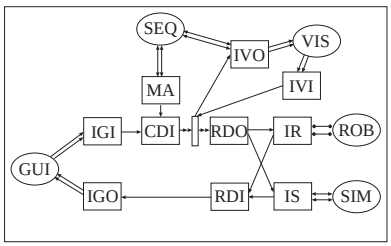

Figure 5: Topology of the interface blocks. In the interest of clarity, we show no events here.

while the block is enabled. Changes in the mode of the core skills are usually accompanied by changes in the topology of the active skill network.

Figure 5 shows the peripheral skills used in this project. Communication with both the user interface and the Vision System (VIS) is split between input and output skills to increase the round-trip communication rate. Because their full names are cumbersome to use, we generally refer to the various skills by their abbreviations.

The basic flow of data is clockwise around the diagram. Desired robot joint positions are either generated by the GUI and read in by IGI, or sent symbolically by the Sequencer and interpreted by MA. Both of these skills send the desired joint values to CDI, which sends them on to the filter blocks (not shown, but represented by the skinny block in the middle). The filter blocks interpret the data in different ways depending on the mode of the system, and send output to RDO. RDO sends the results to IS and/or IR, which interact with the Simulation and Robot respectively. Due to hardware constraints, it is impossible to have separate "in" and "out" skills at this end. These two skills interpret messages back from the Simulation and/or Robot and send them to RDI, which in turn makes the Robot and World data available for all skills that need them. In particular, IGO reads these values and sends them to the GUI, completing the loop. The Sequencer will not require real-time updates of the robot state, but can enable the *arm-move-done* event or the *arm-at* query when necessary to determine the current state of robot and world (symbolically).

Sequencing

The sequencing tier of 3T is responsible for context-dependent conditional execution of sequences of states of the Control Tier. This is where all the various real-life methods of accomplishing an abstract goal are encoded, and this tier is expected to try them all before giving up. In essence, it provides a differential between the two outer tiers so that the highly abstract Planner can actually interface with the physical world. We use the Reactive Action Packages (RAP) system (Firby 1987) as our sequencing tier.

The general adjustable autonomy approach for the TaskBoard scenario in this project is three fold. First, each primitive RAP has associated with it a level of autonomy which is set by the user at the outset, and defaults to "semi-autonomous". That is, each time the primitive is invoked, users are queried as to whether they want the operation done by the robot or by the human. There are also two other choices – the human can take over and do everything in a tele-operated mode or the remainder of the tasks are given to the robot to complete autonomously. The user can reset this primitive LOA mode at anytime.

The second area of RAPs development concerning

adjustable autonomy has to do with agent specification. If the agent specified for any RAP above the primitives is a human, the LOA for primitive RAPs is set to tele-operation, otherwise, the LOA is set to autonomous. Once a higher level RAP begins executing however, the user can change the LOA via the user interface. This kind of RAP is typically invoked by the planner or the user so the agent can be designated.

The third area to affect autonomy in RAPS concerns RAPs which can only be executed by a human or a robot because of the physical constraints of the task. For example, our robot is not dexterous enough to remove or replace the blanket over the work bay. The primitive concerning this RAP, as well as any higher level RAPs that invoke it, may have an agent argument for future use, but the primitive LOA query is not used.

Another issue arising from the use of adjustable autonomy is how the control system keeps itself in synch with the state of the world after the human has relinquished control. In Phase I we opted for periodic camera scans of the work area. Since these can be time-consuming and expensive, in this project we have chosen to use a locate RAP which is invoked whenever an item needs to be used in an operation. If it has been a while since the item's location was updated in memory, or if a lower-level operation fails (possibly because the item is not where it was a few minutes ago), the locate RAP is invoked on the item to re-establish its current location (and in some cases, its operational status).

Planning

The deliberative tier of the architecture is responsible for coming up with a plan of action to accomplish a high-level goal. In our TaskBoard scenario, where the human and robot cooperatively work together to perform specified tasks in the station bay, the goal is to perform a set of specified experiments. The plan is a list of actions that will ensure that the robot and human safely complete all of the experiments. Traditionally, once a planner devises a plan, the plan is published and the work of the planner is over. More realistically, once a plan is published, its execution must be monitored in order to ensure that at each step of the way no outside influences have adversely affected the world and the plan is still valid. In our research we take this a step further - during execution monitoring a user can interrupt the planner and make changes/recommendations to the planner, and the planner will replan respecting the user's wishes as much as possible. This is a form of "mixed-initiative" planning - allowing a user to interact with the planner to influence the resulting plan, yet letting the system create the plan making sure that hard constraints are not violated. The station bay scenario is an excellent example where this type of mixed-initiative planning is necessary. The hazardous environment requires that certain safety precautions be hard-coded. Yet, we want the astronauts performing the experiments to have the opportunity to modify the schedule to reflect preferences based on their knowledge and experience.

Our planner, AP (Elsaesser & MacMillan 1991), is able to coordinate multiple agents by making use of an operator's `:during-conditions` and the temporal relations specified in a plot. The following operator illustrates the use of the temporal relation "covers" to schedule a multiagent cooperative task. In this task the human views a display while the robot holds a crystal up to it.

```
(Operator  replace-crystals
  :purpose (state crystals replaced)
  :agents (?robot ?human)
  :constraints ((instance-of ?robot 'robot)
                (instance-of ?human 'human))
  :preconditions ((state bay opened))
  :plot (sequential
          (covers
            (monitor-crystals ?robot in-monitor)
            (display ?human monitored))
          (crystals ?robot are-replaced))
  :effects ((state crystals replaced)))
```

User interface

The user interface is a critical component of our approach. It is the only mechanism by which the user can interact with the control system, robot, and world. Our goal in designing the user interface was to abstract the user from the details of the underlying control system. There are five main components to the user interface. First, there is a viewer that shows the state of the skill network. Second, there are several panels for teleoperating the robot. Third, there is a checklist viewer, which peeks into the Sequencer and shows the current state of the tasks being performed by the robot. Fourth, there is an interactive interface to the Planner, which is used before execution of any tasks are started. Finally, there is a three-dimensional animation of the robot, which allows the user to see the current state of the robot and the world and to move a model of the robot to a desired location before moving the actual robot.

The following assumptions were made when developing the user interface:

- Users would receive significant amounts of training in the use of the user interface, the performance char-

acteristics of the robot arm, and the tasks to be performed.

- Screen space would be limited.

The design criteria for the user interface were:

- Allow users to rapidly detect and respond to errors.
- Clearly depict both the desired and actual states of the robot arm.
- Provide efficient and intuitive mechanisms for human tele-operation of the arm.

Checklist display Because of space limitations, in this paper we will focus on a single aspect of the complete user interface called the checklist display. The checklist displays information about control tasks executed in the Sequencer. Control tasks are hierarchically organized in the Sequencer. When a top-level task is scheduled for execution, selected subtasks are recursively put on the task agenda as well. Top-level RAPS, which correspond to bottom-level plan operators, are represented as *facets* in the checklist. A facet is a scrolling window containing a checklist item for each subtask of the top-level task on the agenda. Subtasks are represented as checklist items, sequentially ordered based on the time they are put on the task agenda. Checklists are generated automatically during the execution of control tasks. When a top-level task is put on the task agenda, an empty facet is displayed. As each subtask is put on the agenda, a checklist item is added to the facet. The information represented in a checklist item characterizes the control situation during task execution, including the control action, its execution status (pending, executing, successful, failed), the agent responsible to perform the task, the parameters of control, and the values of states affected by the action (both the predicted and actual values). Agreement between the predicted and actual values of these states provides confirmation that the action was successful. The time the task was executed and the duration of the execution are recorded also. As new information about task execution becomes available, the display of the task execution state on the checklist item is updated. When the task is complete, the user interface process stops updating the checklist item with new task information, effectively taking a snapshot of the control situation at task completion. Thus, the checklist display is only active in the context of the associated activity.

During this project, we implemented checklists for a set of joint human-robot tasks for remote maintenance at a space-based facility. We use a subset of these tasks to illustrate the use of the checklist. In the example below, a crew member and a robot jointly replace a battery. To replace a battery requires that a heat shielding blanket be removed, a protective cover opened, and the failed battery replaced by a new battery. To complete the task, the cover must be closed and the heat shielding replaced. The following example shows the checklist during an autonomous activity to replace the heat shielding after repair. In the figures shown in this section, a number of task steps have already been executed successfully, as indicated by the status GOAL-SUCCEEDED shown in the bars (facets) marked plan step. Plan steps are high-level control tasks that require multiple primitive actions for completion. For each primitive control action, there is a checklist item in the facet.

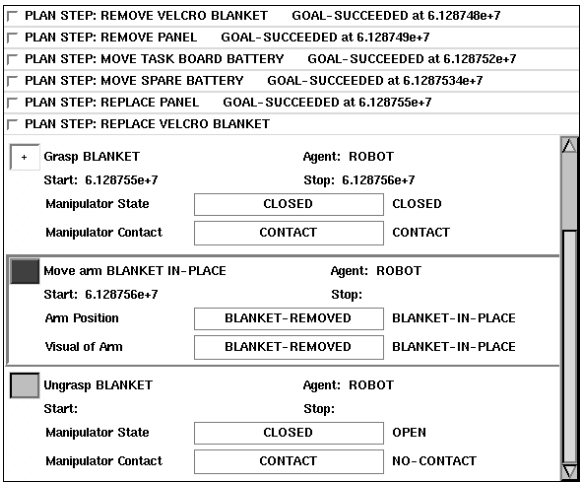

Figure 6: Subtasks can be viewed within the checklist display.

In Figure 6, the facet REPLACE-VELCRO-BLANKET has been toggled so the checklist items for that task can be viewed. The first checklist item (GRASP BLANKET) has been executed successfully. When tasks are completed normally, the status button displays a "+". The second checklist item (MOVE ARM BLANKET IN-PLACE) is being executed. Execution in progress is shown by changing the color of the status button to blue and drawing a box around the item. The third checklist item (UNGRASP BLANKET) is pending execution. Pending task execution is indicated by changing the color of the status button to gray.

Executing the task

We outlined the example task scenario in the introduction. In this section, we look at how all of the hardware

and software components come together to perform the task and what their roles are.

Initial situation

The robot and vision system are turned on and the start-up script starts up the different software modules. The user sees a top-level user interface and clicks the "Init" button. This brings up a second window, from which the user can initialize hardware (such as PICs), zero the robot, or zero the PTV head. After verifying that the zeroing process has been successful, the user can begin planning for the task.

Task planning

The user clicks on the "Task" button of the top-level interface and chooses which tasks to perform. This information is sent to the Planner, which constructs a plan based on default agent assignments and start/end times for each task. This plan is displayed in a window, in which the user can click on the "Edit Plan" button on the plan window and then change agents or change start/end times. The user signals satisfaction with the modified plan by clicking the "Replan" button. The Planner takes the new constraints and creates a new plan, which is re-displayed for the user. This process continues until the user is satisfied.

Plan execution

The user begins executing the plan by clicking on the "Go" button on the top level interface. At this point, the planner takes the first task and sends it to the Sequencer, which enables the appropriate skills in the Skill Manager. If the task is assigned to a human (as is the first task, Remove Blanket), then the Sequencer enables the appropriate events (`(blanket_at OFF)` in the case of the Remove Blanket task) to determine when the human has finished the operation. These events use the color vision system to determine the state of the world and any changes in that state. Once the operation is finished, the Sequencer informs the Planner, which moves to the next task.

In our scenario, the next task would be removing the battery. The Planner initiates execution of the "remove battery" RAP. First, the RAP uses a query to make sure that the battery is there. The query uses the color vision system to look for the battery handle and to return its location. If the battery is there, the RAP moves the arm to just above the battery handle. The location of the battery handle is constantly updated by the vision system, so even if the battery moves somehow, the robot will still get to it. Once the arm is above the battery handle, it moves down to engage the battery. Then the RAP commands the gripper to close. The gripper skill closes the gripper until the contact switches touch. The `gripper_done` event checks to see if the width of the gripper is the width of the handle. If so, then a good grasp is signalled. Otherwise, an error is returned and the RAP retries the task. When the handle is grasped, the RAP again moves the arm back to a position above the task bay, and then to a holding position on the side of the arm, where the battery is set down.

Monitoring task execution

While the robot is executing a task such as the one described in the previous section, the user can monitor the task in several ways. First, the progress of the plan is shown in the plan window of the user interface. The progress and state of the system can also be viewed through the checklist display (see the user interface section), which shows the low-level tasks (e.g., move-arm, grasp, ungrasp, etc.) being done by RAPS. The checklist shows those tasks that are finished, those tasks which are still pending, and the task that is currently being performed. For the current task, information about the task, including important sensor data and start and end times, is displayed so that the user can see at a glance how the task is progressing, and catch any problems. For more experienced users, the set of activated skills can be viewed by clicking on the "Show Skills" button at the top level of the user interface. Finally, users can see the health of all of the various subsystems by clicking on the "status" button at the top level of the user interface. This will bring up a window that shows all the subsystem processes and whether or not they are currently running.

Interacting with the robot

As the task progresses, the user can choose the level autonomy of the TaskBoard Control System (TBCS). At any point, the user can take control of the robot from the TBCS and use user interface elements designed to move the robot. To do this, the user clicks the "Level of Autonomy" button, then chooses "manual." The user interface informs RAPS, which safely stops whatever it is doing and activates the appropriate skills for tele-operation of the robot. At this point, the user can select from several windows to control the robot, moving a wireframe model of the robot in the user interface animation to a desired location, and then "send" that position to the TBCS for execution. The TBCS will still be ensuring that the robot does not harm itself by going past any joint limits. The user can relinquish control at any time and put the level of autonomy back to "autonomous." At this point the RAP system will begin executing tasks again, checking the state of the world beforehand to make sure that nothing significant has changed during user execution.

The user may want to take control for many different reasons. First, the robot may not be performing the task correctly. For example, the robot arm may have moved to the wrong place and the TBCS may not know it. The user can take over and nudge the robot into the right place and then allow the TBCS to return to autonomous execution. Second, the robot may be unable to perform the task because it doesn't have appropriate sensing. For example, it may not be able to find a particular object and it needs the user to move it to the object for it to continue. Finally, the user may want to gain some experience in using the manipulator so they will be competent in the future. Whatever the reason, our system allows for adjustable autonomy all the way down to the lowest level.

Conclusions

The goal of this project was to produce a software infrastructure that allows for traded control of a robot manipulator. This infrastructure includes automated, multi-agent planning, conditional sequencing, reactive control and user interfaces. All of the components tie to a manipulator that executes a pick-and-place task. During this research, we evolved the concept of traded control into a more general approach called *adjustable autonomy*. Adjustable autonomy allows systems to operate with dynamically varying levels of independence, intelligence, and control. A human user, another system, or the autonomous system itself may adjust the system's "level of autonomy" as required by the current situation.

Adjustable autonomy provides the flexibility to adapt to changing environments, user goals, and resources. This flexibility is important in order to successfully deploy autonomous system solutions for highly complex real-world problems. For example, advanced life support systems for space habitats will require sophisticated autonomous control that interacts closely with the crew members that depend on the life support system. Another example of adjustable autonomy is a team of robots supervised by a single operator in which the operator can provide input to the team or even take control of some team members for short periods of time. In each of these cases, the control system may be completely in control or it may be supervising manual control or it may be somewhere in between. A system that can shift amongst these control extremes in a safe and efficient manner is an adjustable autonomy system.

References

Backes, P. G.; Tso, K.; Lee, T.; Hayati, S.; and Phan, L. 1991. A local-remote telerobot system for time-delayed traded and shared control. In *Fifth International Conference on Advanced Robotics.*

Bonasso, R. P.; Firby, R. J.; Gat, E.; Kortenkamp, D.; Miller, D. P.; and Slack, M. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9(1).

Elsaesser, C., and MacMillan, R. 1991. Representation and algorithms for multiagent adversarial planning. Technical Report MTR-91W000207, The MITRE Corporation.

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI).*

Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI).*

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence* 103(1).

Musliner, D. J.; Durfee, E.; and Shin, K. 1993. CIRCA: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics* 23(6).

Sheridan, T. B. 1989. Telerobotics. *Automatica* 25(4).

Simmons, R. 1990. An architecture for coordinating planning, sensing and action. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control.*

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain dynamic environments. *Journal of Experimental an Theoretical AI* 7.