Journal of Intelligent and Robotic Systems **13**: 247–262, 1995.

Real-time Shared Control System for Space Telerobotics

ALEXANDER DOUGLAS and YANGSHENG XU
The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

(Received: 14 June 1993; in final form: 10 December 1993)

ISSN 0921-0296

Journal of Intelligent and Robotic Systems

Theory and Applications
(Incorporating *Mechatronic Systems Engineering)*

Editor-In-Chief:
Spyros G. Tzafestas
Dept. of Electrical Engineering, National Technical University of Athens, Greece

The theory and practice of intelligent and robotic systems is currently one of the most intensely studied and promising areas in computer and systems science and engineering, and one that will certainly play a primary role in future life. The *Journal of Intelligent and Robotic Systems: Theory and Applications* provides a source linking all fields in which the system intelligence and/or intelligent control plays a dominant role, and stimulates interaction between workers performing both theoretical and applied research.

The papers published are all highly original, novel, of acute interest and, above all, clearly presented. A particular feature of the journal is the number of papers that suggest new avenues of research and new developments in the field.

Topics of particular relevance to the *Journal of Intelligent and Robotic Systems* are: computer integrated manufacturing systems, computer vision, diagnostic systems, expert systems, intelligent systems, learning

Kluwer Academic Publishers, P.O. Box 17, 3300 AA Dordrecht, The Netherlands

rnal of Intelligent and Robotic Systems **13**: 247–262, 1995.
Printed in the Netherlands.

eal-time Shared Control System for Space elerobotics

LEXANDER DOUGLAS and YANGSHENG XU
he Robotics Institute, Carnegie Mellon Univer sity, Pittsburgh, PA 15213, U.S.A.

eceived: 14 June 1993; in final form: 10 De cember 1993)

bstract. The shared control system is a m odular real-time system designed to execute omplex tasks through the intelligent coordinat ion of task modules. A state machine is used control task sequencing, and due to the aut omatic switching, the accuracy and reliability ith which tasks are executed, is greatly imp roved. Tasks consist of sets of independent, odular, and reusable subtasks whose outputs a re combined to create the control. This system as proved itself useful for rapid developmen t of reliable high level multiple sensor based anipulation and control tasks. Additionally, we have integrated a neural network based isual servoing system, a semi-compliant Ca rtesian trajectory following heuristics, and a eal-time graphical user interface in the system. The shared control system was implemented or the Self-Mobile Space Manipulator (SM^2) to handle a range of tasks associated with ocomotion, manipulation, and material transp ortation on Space-Station Freedom.

Key words. Telerobotics, Space robotics, R obot control, Shared control, Mobile manipulator, Robot programming.

1. Telerobotic System: SM^2

In future space exploration, robotic technology will be vital in helping or replacing astronauts for various tedious and dangerous extra-vehicle-activities (EVA). For a space-robot system, because of the complexity of tasks and uncertainly of environment, the control system must incorporate both model-based autonomy and human-supervised teleoperation. The software architecture of such a control system is the so-called *shared control* of teleoperation and autonomy. Considerable research and development efforts have been directed to this area, such as [5–8].

Since 1989, we have been developing the Self-Mobile Manipulator (SM^2) which is a walking robot to assist astronauts on the Space-Station Freedom and other space structures in performing construction, maintenance and inspection tasks. It has end-effectors for attachment, and can step from point to point to move freely around the exterior of space structures. SM^2 can replace EVA astronauts in performing tedious or dangerous tasks, and can be deployed quickly to investigate emergency situations. It is simple and modular in construction to maximize reliability, simplify repairs and minimize development time. SM^2

is lightweight, so it can operate with minimum energy and disturbance to the structures.

Over the past four years, SM^2 has progressed from concept, through hardware design and construction, to software development and experiments with several versions of the robot. During the first year, we developed a concept for robot mobility on the space-station trusswork, and experimentally tested a variety of control algorithms for simple one-, two- and three-joint robots. During the second year, we developed a simple, five-joint robot that walked on the tubular-strut-and-node structure of the original Space Station Freedom design, and a *gravity compensation* system that allowed realistic testing in a simulated zero-gravity environment. The third-year work focused on development of the manipulation function; we added a part-gripper and extra joint at each end of the robot, and developed control software [1–3]. Photographs of SM^2 and its configuration in carring a long strut on the tubular-strut-and-node structure are shown in Figures 1 and 2.

During the fourth year of the project, in response to the changing design and needs of the Space Station, our focus has shifted to adapting SM^2 to new Space-Station trusswork and specializing it as a mobile inspection robot to augment the fixed video cameras planned for the Space-Station Freedom. System changes, to enable this function on the new I-beam-truss Space-Station Freedom design, encompass the truss mockup; robot configuration and end-effectors; gravity compensation system; and control software, including shared-control, neural-network-based vision and graphics simulation.

The robot's size and configuration have been adjusted to a accommodate the new truss structure. SM^2 now has seven joints (Figure 3) one at the elbow and three at each end (Figure 4) to allow out-of-plane motions needed for stepping from one truss face to another, while preserving the robot's symmetry to simplify control. Joints are self-contained and modular, so a minimum inventory of parts is needed for joint repair or replacement. The robot links are long enough to permit stepping between adjacent longerons. At each end of the robot is a three-fingered gripper for grasping the truss I-beam flanges. Finger separation is measured by a linear potentiometer, while motor current is sensed to indicate grasp force. Contact switches on the three fingers verify the grasp. Capacitive proximity sensors at the base of the fingers are used to sense beam proximity, and can be used for aligning the gripper with the beam.

To facilitate a variety of tasks that SM^2 is designed to perform, we developed a real-time shared control system. The *shared control system* shares the capabilities of teleoperation and autonomous control. From the real-time control point of view, the shared control function needs an architecture that allows multiple tasks to be executed simultaneously. By combining task model-based control with sensor-based control, the system enables the robot to perform dynamic sequences of tasks that range from teleoperational to semi-autonomous to fully autonomous such as trajectory tracking, ORU placement, beam grasping, beam following

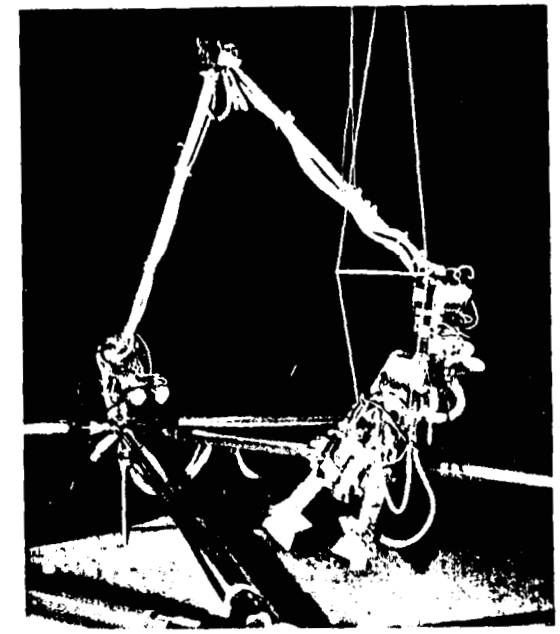

Fig. 1. Photograph of seven-joint SM^2 on the tubular-strut-and-node space station structure.

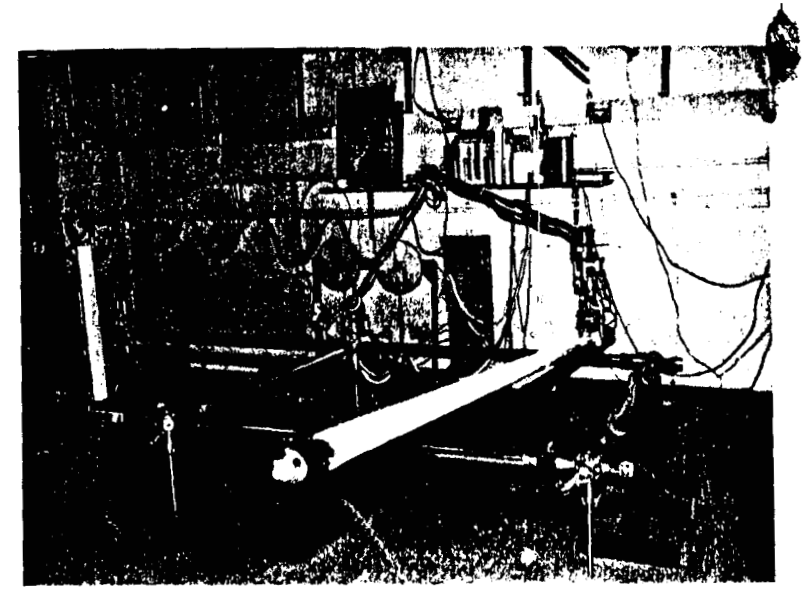

Fig. 2. Photograph of SM^2 carrying 3.1-meter-long strut on the tubular-strut-and-node structure.

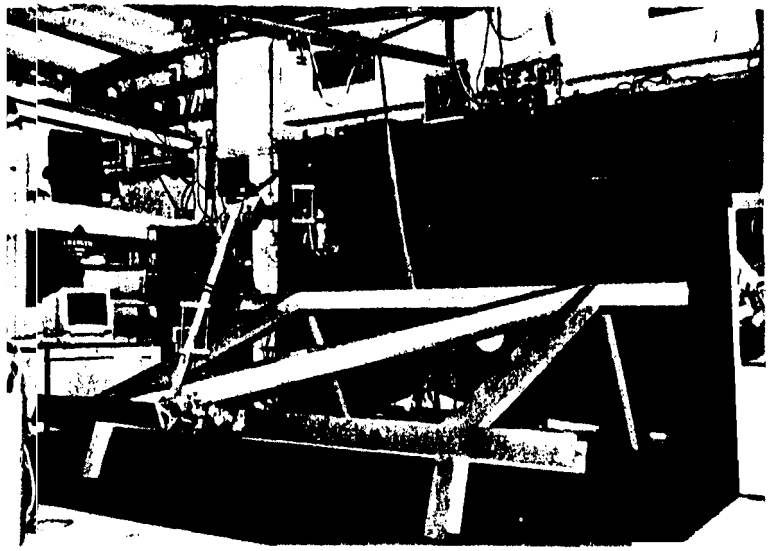

Fig. . Photograph of the SM^2 testbed with wood/aluminum I-beam prein tegrm truss mock

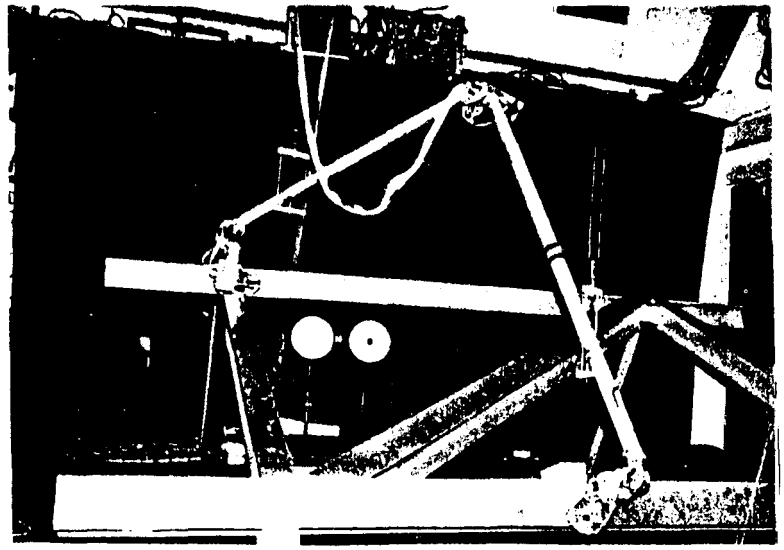

Fig. 4. Photograph of SM^2 standing on the I-beam integrated truss mockup.

2. Shared Control System

2.1. SYSTEM OVERVIEW

The shared control system is designed to achieve robust task execution and error handling while still being easily extensible. Because tasks are defined as a group of subtasks–modules–which run concurrently, it is possible to execute several tasks at once. Upon task execution, a module can generate two types of output; for example, a module could give robot control information, i.e., a differential motion in base-relative Cartesian coordinates, and could also send status messages to the state machine.

To elaborate on this example: in order to enable the robot to grab a beam, the operator could run the trajectory task module simultaneously with the contact sensor module. The trajectory module would take into account the robot's current position and designed trajectory; then, rather than returning an actual goal point, the module would return the difference between the goal point and the current position as the robot control motion. Upon completion of this task, the module would send a *done* message to the state machine. Meanwhile, the contact-sensor module, also taking into account the robot's current position and designed trajectory, would initiate gripper contact with the beam. If the contact sensor module would, indeed, sense contact with the beam, the module would return a *contact* message to the state machine. If, and only if, both the trajectory and the contact sensor modules return the messages *done* and *contact*, respectively, the state machine would then perform a transition to a new state such as closing the gripper on the beam. Should this attempt fail, the state machine would try again, this time using the visual servo to aid in the approach.

The state machine is parsed at runtime, i.e., no re-compilation is needed to change the task execution. Consequently, much less time is needed for designing and testing tasks. Also, since these task modules can be used within any task, little code needs to be written. In the state machine, tasks are represented as states; transitions occur based on the messages sent from the task modules. In order to make the system more useful and flexible, task modules are also designed to receive commands from the state machine. For example, a trajectory task module might have several preloaded trajectories to choose from; it is the state machine that sends a message telling the module which trajectory to execute.

The shared control system comprises four types of modules: task modules, remote task modules, coordination modules, and combination modules. The *task modules* generate differential movements based on inputs from the remote task modules, hardware devices, and robot position information. The *remote task modules* run on high speed processors or workstations and perform heavy computation or user interface tasks. They then send their results to task modules running on the realtime system via UNIX sockets, VME bus or serial line. The *coordination module* contains the state machine and handles all of the actions involved with the state machine, including switching tasks and sending messages

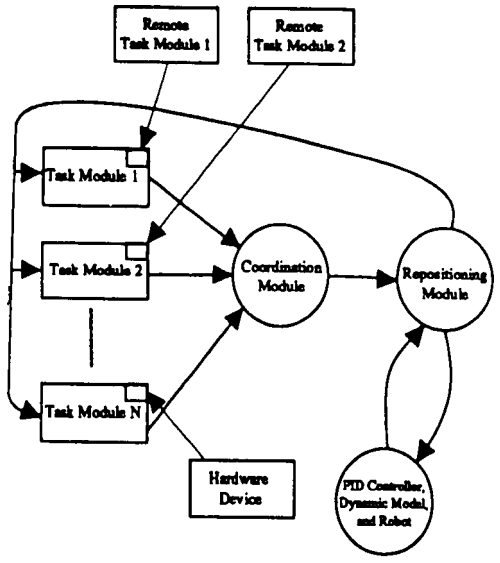

Fig. 5. Shared control concept model.

to the task modules. The *combination module* intelligently combines the control outputs from the running task modules into one coherent control command and then determines that the command is a reasonable motion for the manipulator to perform (see Fig. 5).

2.2. TASK MODULES

We have developed several reusable task modules for the SM^2 control software. In each control cycle, the task modules perform four basic functions:

- Read messages from the state machine and respond in appropriate fashion.
- Read sensor devices, global variables, or receive input from remote tasks.
- Generate desirable control motion based on local inputs.
- Send appropriate messages to the state machine.

Such a data-processing structure facilitates the design of task modules that can operate independently of each other, thus increasing real-time concurrency. Ideally, each sensor group would constitute its own module which would not only generate useful messages about the state of the sensors but would also provide various types of control based on these sensors. For example, on the SM^2 each

gripper has three proximity sensors which are used by one task module. The task module receives commands telling it from which gripper to read sensors and also what type of control output to produce. If at least one sensor senses something in close proximity, the module sends the message *partial contact* to the state machine; if all three senses something in close proximity the module sends the *contact* message to the state machine; if none senses an object in close proximity, the module sends the *no contact* message. These messages provide useful information about the completion of a task or a state of error.

In our shared control system, the task module also provides compliant control to aid in grasping operations. Based on the values of the proximity sensors, it is easy to compensate for orientation errors for the tip. Since this module can run simultaneously with many other modules, it can aid in the grasping of beams for either autonomous motion or teleoperation.

Each module can produce only a *desired* motion which is combined with the desired motions from other task modules, thus, there is no guarantee that this motion will be what the robot executes. Therefore, task modules must be robust. For example, consider a Cartesian trajectory task module. During each cycle, the module must find the closest via point to the robot position and generate a desired motion based on this difference. To address this problem, we have designed a trajectory-tracking algorithm which can be robust in light disturbances or compliant to the control from other modules which are running simultaneously.

The task module serves two purposes: to generate a desired movement for the robot and to notify the task coordination modules of interesting events. A desired movement for the robot is generated based on real-time information such as global variables, e.g., position, hardware devices, and remote task modules. Interesting events include completion of a goal or recognizing that an important change has occurred somewhere; for example, a task module whose job is to follow a trajectory generates a desired direction for movement based on the

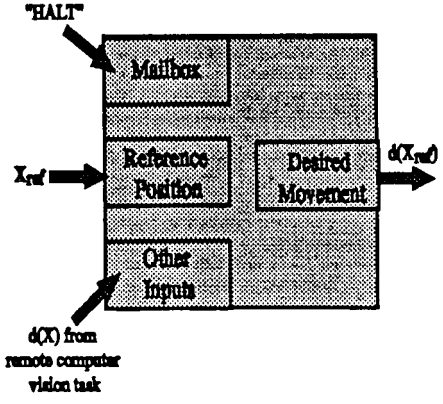

Fig. 6. Task

current position of the robot every cycle until the goal is reached. The module then notifies the task-control module that execution is complete. All task modules can have multiple instances running in the shared control system. Each instance gets its parameters from a different configuration file at startup and, based on messages sent to the module, may be changed dynamically while running. Thus, through reparameterization, very different and dynamic effects can be achieved within a relatively small piece of code.

2.3. REMOTE TASK MODULES

In a complex robot-control system, many tasks are CPU-intensive, require special hardware or have user interfaces. These types of tasks are not suitable for slow real-time processing boards, but usually run well on workstations or specialized processors. For this reason, the shared control system includes the concept of a remote task. A remote task running as a process on a remote workstation functions in exactly the same way as a task module running on the real-time system. The remote task module receives the same inputs, robot position information and control messages as task modules receive and can also give the same outputs, desired motion and messages to the state machine. The SM^2 system uses one workstation equipped with digitizer boards to run neural network-based image processing as a remote task. In addition, the system includes several graphic interfaces that interact with the state machine and display real-time information. For example, a graphic interface can allow a dynamic change in scaling parameters for the teleoperation input device or can provide a real-time CAD model display of the robot's position.

Through the use of UNIX sockets, remote task modules open up a communication line with a server process running on the host workstation for the real-time

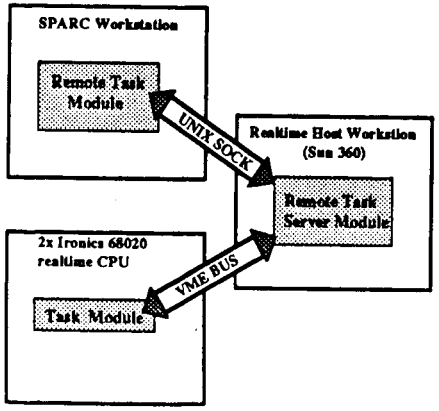

Fig. 7. Remote task communications.

system. Using the VME bus, the server process handles sending messages to and from the state machine as well as passing the control information to the combination module. The server also provides data from the realtime system and synchronization signals for the remote task modules (see Fig. 7).

2.4. COMBINATION MODULE

The combination module serves two purposes: combining the movements of the task modules into one movement, and ensuring that the movement is not too drastic for the robot. There are many ways to combine the output vectors from the task modules. We have tested several methods, both in simulation and real-time experiments: a simple summation; a simple average; weighted sum and average; voting on angle and velocity; and some unusual variations. In practice, the weighted average performs well since it is not computationally expensive and its performance is predictable. The weighted average used in the coordination module is as follows:

$$\dot{X}_{\text{ref}_t} = n^{-1}(w_1 \ldots w_n) \begin{pmatrix} \dot{X}_{\text{task}_1} \\ \vdots \\ \dot{X}_{\text{task}_n} \end{pmatrix}$$

$$X_{\text{ref}_t} = X_{\text{ref}_{t-1}} + \dot{X}_{\text{ref}_t}$$

The reference position does not just accumulate the output of the task modules, but rather is bounded by the robot's position;

$$|X_{\text{ref}} - X_{\text{mez}}| <= X_{\text{Err}_{\max}}$$
$$|\dot{X}_{\text{ref}} - \dot{X}_{\text{mez}}| <= \dot{X}_{\text{Err}_{\max}}$$
$$|\Theta_{\text{ref}} - \Theta_{\text{mez}}| <= \Theta_{\text{Err}_{\max}}$$
$$|\dot{\Theta}_{\text{ref}} - \dot{\Theta}_{\text{mez}}| <= \dot{\Theta}_{\text{Err}_{\max}}$$

This ensures that the robot never lags too far behind the reference position, and thus implies several things: the information for each cycle is still relative to the robot's actual position; and the reduction in position errors may cause an increase in controller gains. This also gives the robot a natural compliance, which is very important in space manipulation tasks. Once a new X_{ref} and $\dot{X}_{\text{ref}}$ are computed by the inverse kinematics, the joint angles, Θ_{ref}, are thresholded to stay in the robot's workspace.

$$\Theta_{\text{ref}} = \text{invKin}(X_{\text{ref}}, \dot{X}_{\text{ref}}, \Theta_{\text{mez}}, \dot{\Theta}_{\text{mez}})$$
$$\dot{\Theta}_{\text{ref}} = \text{Frequency}(\Theta_{\text{ref}} - \Theta_{\text{ref}_{\text{old}}})$$

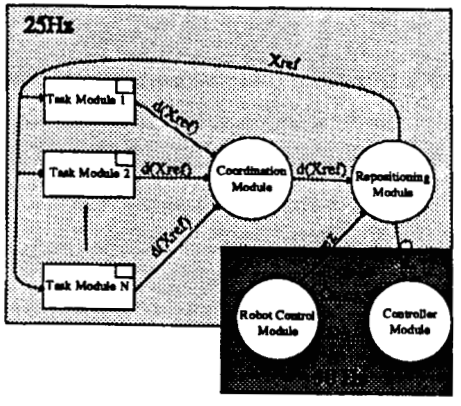

Fig. 8. Data flow and cycle times. Lower left box is controller loop running at 50 Hz and the larger box is the shared control system running at 25 Hz. The transitions are labeled with the variables that pass along them.

such that

$$\Theta_{\min} <= \mathrm{inv}Kin(X_{\mathrm{ref}}, \dot{X}_{\mathrm{ref}}, \Theta_{\mathrm{mez}}, \dot{\Theta}_{\mathrm{mez}})$$
$$\mathrm{inv}Kin(X_{\mathrm{ref}}, \dot{X}_{\mathrm{ref}}, \Theta_{\mathrm{mez}}, \dot{\Theta}_{\mathrm{mez}}) <= \Theta_{\max}$$

The combination module is the key to linking the shared control system to the manipulator, as illustrated in Figure 8. Because the reference position is never allowed to get too far away from the robot, the slow manipulator is always able to keep up with the reference point. If the robot is bumped, the reference point moves with it. This forms a natural compliance and avoids large position errors in the control loop.

2.5. COORDINATION MODULE AND STATE MACHINE

One of the major concepts behind this shared control system is that the task modules have no knowledge of what other task modules they are interacting with. They simply operate in whatever mode they are told to, and send out messages whenever an event that might be important occurs. It is the coordination module's responsibility to make the task modules perform in an intelligent manner. Currently, we are using a state machine to coordinate the task modules and sequencing of tasks; however, the machine could easily be replaced by some intelligent decision-making system. Based on messages returned from the task modules, the state machine determines the end of an event, or the need to change states. Each state in the state machine knows what task modules should be running during that state. When a new state is entered, the correct set of task

modules is started up and all others are turned off. In addition, these modules receive a set of initialization commands which tell the modules how to perform in this new state.

The state machine is programmed via a simple language in which the programmer describes: the commands and outputs of the task modules; the states, including the initial state and what task modules should be running; the initialization command sequence; and transitions. Since all of this data is in a file that is parsed and checked for accuracy at runtime, development of new high level tasks is quick. No recompilation and source code are required. To enable our system to perform elegantly, we have found it necessary to modify the concept of the state machine. Some of the additions we have made include: pattern matching to transitions and task module commands, counter-type variables, and the ability to reparameterize task modules during transitions.

When we first implemented the state machine, it was obvious that there was a need to somehow compress the seemingly endless, geometric, enumeration of states. This problem has to do with the fact that, while there are many aspects of a task and a sequence of tasks that can be logically generalized, this is not so in a state machine. Our solution was the inclusion of wildcards in the description of task module commands and transitions. For example, a trajectory task module might have any number of preloaded trajectories and the state machine must be able to tell the module what trajectory to execute, i.e. the machine sends a message such as *traj3*. To avoid enumerating a possible trajectory as a command, *traj1...trajn*, the operator can use a wildcard, *traj?*. Any string that matches this pattern is then considered an appropriate command.

A similar idea is applied to the concept of state transitions, which is described in the following way. When a message is received from a task module, move from *Source-State* to *Destination-State*. Through the use of wildcards, these transitions become more rules for changing states rather than transitions from one state to another. The first use of wildcards appears in the description of the message that actually creates the transition. For example, once a trajectory has been executed, the state machine receives a message, *traj:done*, which means that the trajectory module has completed the trajectory. Rather than making a separate transition for each trajectory that could have been executed, we can simplify the description by stating that if the state machine receives a message matching *done?* from task *traj*, then the machine should the perform a transition to another state.

Often we have found that many states are simply variations on a theme, e.g., where the only difference between states is parameterization of the task modules, as in the state machine for teleoperation where the group of states differs only in the scaling of the teleoperation input and the use of position or velocity control. Command sequences performed during a transition can be eliminated since this allows reparameterization during a transition which might lead back to the same state. The measure for dealing with this is the use of generalized transitions.

Through the use of wildcards, transitions from one type of state to another may be performed upon the receipt of a specific message in the source state. The characters of the completed wildcard are used to describe the destination state. An example of this would be switching from position mode, *TelePos?*, to velocity mode, *TeleVel?* in teleoperation, while maintaining the same scaling. If the current state were *TelePos3*, the result of receiving the message to switch to velocity mode would be a transition to state *TeleVel3*.

The addition of counter variables was necessary to allow different branches from same state to be based on seemingly identical situations. Since state machines can not elegantly express temporal knowledge without maintaining a separate state for each possibility, we have added a task module that can manage a set of counter variables which can be set, incremented, decremented and read. This allows transitions to be based on data such as how many times an event has occurred or how many times sequences of high-level tasks have been executed, starting from the same initial state.

3. System Implementation

In this section we will present the actual system we have developed, based on the shared control concept in the previous section, for the SM^2 manipulator. As described in the first section, locomotion and inspection tasks need a robot with the capability to align the gripper with the beam so as to allow the robot to approach and grab the beam. Slightly modification of this function will enable the robot to keep a certain height and orientation with respect to the beam while following it. This capability will be needed for trusswork inspection tasks.

In our development of the shared control system, we have found it necessary to develop some secondary concepts to make the system useful yet flexible. Among these developments, are two that might be of particular interest: a retrainable generic, neural network-based visual servoing system and a semi-compliant trajectory algorithm.

3.1. TRAJECTORY CONTROL

In the shared control environment, conventional trajectory following does not work particularly well, since one can not make any assumptions about the position or speed of the robot from one cycle to the next. This means that it is impractical to ramp velocities using precomputed values. Also in the shared control environment, it is impossible to tell how far off course one may be. To remedy this problem, we have designed a trajectory-following task which handles this problem in an elegant manner. Using a graphic software simulation, we have developed a control strategy that generates a more desirable motion. Supposing one considers a trajectory as a curve in 3-dimensional Cartesian space, and on this curve there is some closest via-point. At this point, there is a vector tangent

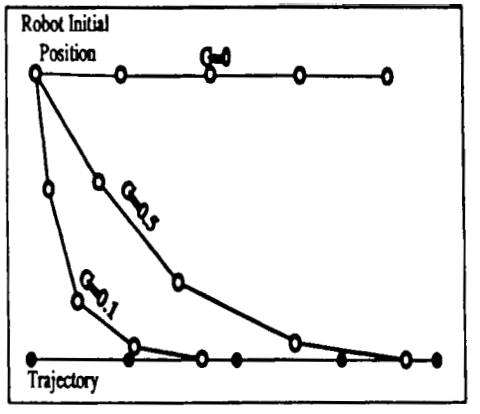

Fig. 9. Trajectories with different attraction constants.

to the curve and a normal vector which passes through the current reference point of the robot. Movement along the tangent vector is movement in along the *trajectory path*, while movement along the normal vector is movement along the *recovery path*. The desired motion constructed is a linear combination of these two vectors. Every trajectory has an attraction variable, G, which determines the degree of attraction of the reference point to the trajectory. As G approaches 1, the recovery path becomes dominant, i.e., there is a strict traversal of the path through the via points. As G approaches 0, the shape of the path becomes more important (see Fig. 9).

$$\dot{\mathbf{X}}_{\text{task}} = g * \mathbf{N} + (1 - g) * \mathbf{T}$$

3.2. NEURAL NETWORK BASED VISUAL SERVOING

Neural networks have demonstrated their ability to track and recognize objects. In Carnegie Mellon, we designed a generic neural network-based vision system [4]. This system is composed of three parts: data collection, network training and network execution. We have automated all three parts in order to facilitate rapid development of visual servoing tasks. By plotting the scene in a known position, it is possible to execute random trajectories, or to move the robot manually, and to collect training data. Then, using a neural network description language, we can create and train a network on this data. Once a network is trained, its configuration and weights are stored. Using a generic remote task module, any of these neural networks can be used to control the robot. The remote module sends robot control values just as a task module does, and also sends a message signifying the stabilization of the network, i.e., the completion of the servo task.

Rather than modifying the control software to compute the desired output during the data collection process, the system tags each collected image with the measured tip and camera position. Either a simple post-processing program or the UNIX program *awk*, enables computation of the desired outputs. This ability is quite useful in that several networks can be trained to behave differently, based on the same collected data; e.g., with regard to the task of centering over a beam, one network can be trained to center itself directly over the beam; another can be trained to center the manipulator with a constant offset; yet another can be trained to approximate the distance of the beam from the camera.

The actual architecture of the network, including number of layers, number of nodes, connections, transfer functions, etc., is described in a file which is fed along with the exemplar file to the training system. The training system uses an adaptive training supervisor algorithm to perform pruning and avoid local minima. A backpropagation network is trained to reliably generate the desired response. Once adequate performance is achieved, the network is saved and is ready to run with the shared control system.

A generic visual servoing task runs on a remote SPARC workstation equipped with two digitizer boards. This program continually digitizes images on both boards. While its images on one board are digitized, the images on the other board are being read into main memory. This procedure provides the fastest possible update of the images. Since the idea behind shared control is to allow multiple tasks to control the robot simultaneously, several visual servoing tasks are able to run simultaneously. Due to the relatively simple computation performed by the neural network, the main problem is with acquiring the images. After the images are acquired, several networks then perform their computations and have their control values sent to the real-time system at approximately 10 Hz.

3.3. REAL-TIME GRAPHICAL USER INTERFACE

The real-time graphical user interface (RGUI) is a PHIGS- and XView-based application that acts as a remote task module. This GUI was designed to provide the following capabilities:

- Real-time 3D display of the robot position and configuration in a model of its environment.
- Interface manually controlled task sequencing, i.e., interaction with the state machine.
- Teleoperation input device for directly controlling the robot motion.
- Programming and testing trajectories.
- Collision detection and obstacle avoidance using CAD model

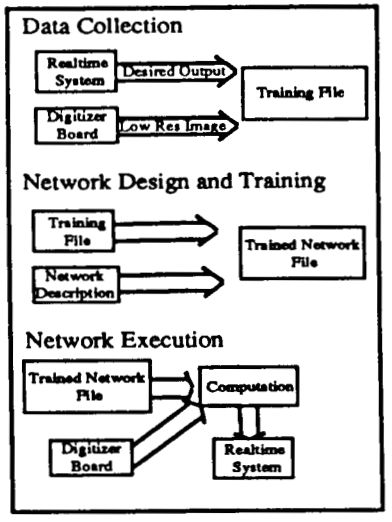

Fig. 10. Overview of general neural network vision system.

- Simulation: graphic display to preview the robot motion with respect to Space Station Freedom before execution.

This graphical interface demonstrates that the modular structure of the shared control system does not limit the functionality of a control system, but rather simplifies it. The GUI has two different 3D views of the robot in its environment. In teleoperation mode, the GUI transforms mouse movements and button clicks into robot movement relative to the view in which the action is happening. The movement is then converted to be relative to the robot's base and sent to the combination module on the real-time system. The real-time display draws the robot's configuration according the location of the base in world coordinates and joint angles which are passed to it from the real-time system. Collision detection is done by checking for intersections of the robot with the CAD model. If a collision is detected, a message is sent to the state machine.

4. Conclusion

The shared control system is a modular system designed to execute complex tasks through the intelligent coordination of task modules. The system has been implemented in the Self-Mobile Space Manipulator for execution of various tasks associated with locomotion, manipulation, and material transportation on Space-Station Freedom. The unique aspect of this system lies in the design and combination of the task modules. The state machine is a valuable method for

controlling the flow of tasks, and, due to the automatic switching, the machine greatly improves the accuracy of task execution. In addition, the inclusion of the generalized visual servoing and other control techniques contributes to making this system truly reusable, modular and efficient. The shared control system has proven itself to be useful for rapid development of high level tasks. Furthermore, this system seems to reliably solve many manipulation problems we have faced in the past.

References

1. Xu, Y., Brown, B., Friedman, M. and Kanade, T.: Control System of Self-Mobile Space Manipulator, *Proc. IEEE Int. Conf. Robotics and Automation*, 1991.
2. Ueno, M., Ross, W. and Friedman, M.: TORCS: a teleoperated robot control system for the Self-Mobile Space Manipulator, Technical Report CMU-RI-TR-91-07, Carnegie Mellon University, Pittsburgh, PA, 1991.
3. Xu, Y., Brown, B., Aoki, S. and Kanade, T.: Mobility and Manipulation of a Light-Weight Space Robot, *Proc. Int. Workshop on Intelligence Robots and Systems*, 1992.
4. Pomerleau, D.: Neural Network-Based Road Navigation, PhD Thesis, Carnegie Mellon University, Dept. of Computer Science, 1992.
5. Kim, W. S.: Graphical Operator Interface for Space Telerobotics, *Proc. IEEE Int. Conf. Robotics and Automation*, 1993.
6. Lawn, C. A. and Hannaford, B.: Performance Testing of a Passive Communication and Control in Teleoperation with Time Delay, *Proc. IEEE Int. Conf. Robotics and Automation*, 1993.
7. Desrochers, A.: *Intelligent Robotic Systems for Space Exploration*, Kluwer Academic Publishers, Dordrecht, 1992.
8. Brunner, B., Hirzinger, G., Landzettel, K. and Heindl, J.: Multisensory Shared Autonomy and Tele-sensor-programming–Key Issues in the Space Robot Technology Experiment ROTEX, *Proc. Int. Workshop on Intelligence Robots and Systems*, 1993.