January 1991

Model Based Teleoperation to Eliminate Feedback Delay NSF Grant BCS89-01352 First Report

Richard P. Paul
University of Pennsylvania

Janez Funda
University of Pennsylvania

Simeon Therry
University of Pennsylvania

Thomas Lindsay
University of Pennsylvania

Masahiko Hashimoto
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-02.

Model Based Teleoperation to Eliminate Feedback Delay NSF Grant BCS89-01352 First Report

Abstract

We are conducting research in the area of teleoperation with feedback delay. Delay occurs with earth-based teleoperation in space and with surface-based teleoperation with untethered submersibles when acoustic communication links are involved. the delay in obtaining position and force feedback from remote slave arms makes teleoperation extremely difficult. We are proposing a novel combination of graphics and manipulator programming to solve the problem by interfacing a teleoperator master arm to a graphics based simulator of the remote environment coupled with a robot manipulator at the remote, delayed site. the operator's actions will be monitored to provide both kinesthetic and visual feedback and to generate symbolic motion commands to the remote slave. the slave robot will then execute these symbolic commands delayed in time. While much of a task will proceed error free, when an error does occur the slave system will transmit data back to the master and the master environment will be "reset" to the error state.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-91-02.

Model Based Teleoperation To Eliminate Feedback Delay

NSF Grant BCS89-01352

(First Report)

MS-CIS-91-02
GRASP LAB 248

Richard P. Paul
Janez Funda
Simeon Thierry
Thomas Lindsay
Masahiko Hashimoto

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

January 1991

Model Based Teleoperation to Eliminate Feedback Delay NSF Grant BCS89-01352 First Report

Richard P. Paul
Janez Funda Thomas Lindsay
Simeon Thierry Masahiko Hashimoto

The University of Pennsylvania
GRASP Laboratory
Philadelphia PA 19104

ABSTRACT

We are conducting research in the area of teleoperation with feedback delay. Delay occurs with earth-based teleoperation in space and with surface-based teleoperation with untethered submersibles when acoustic communication links are involved. The delay in obtaining position and force feedback from remote slave arms makes teleoperation extremely difficult. We are proposing a novel combination of graphics and manipulator programming to solve the problem by interfacing a teleoperator master arm to a graphics based simulator of the remote environment coupled with a robot manipulator at the remote, delayed site. The operator's actions will be monitored to provide both kinesthetic and visual feedback and to generate symbolic motion commands to the remote slave. The slave robot will then execute these symbolic commands delayed in time. While much of a task will proceed error free, when an error does occur the slave system will transmit data back to the master and the master environment will be "reset" to the error state.

Contents

1 Introduction

We are conducting research in the area of teleoperation with feedback delay. Delay occurs with earth-based teleoperation in space and with surface-based teleoperation with untethered submersibles when acoustic communication links are involved. The delay in obtaining position and force feedback from remote slave arms makes teleoperation extremely difficult. We are proposing a novel combination of graphics and manipulator programming to solve the problem by interfacing a teleoperator master arm to a graphics based simulator of the remote environment coupled with a robot manipulator at the remote, delayed site. The operator will work without delay in this displayed environment. The operator's actions will be monitored to provide both kinesthetic and visual feedback and to generate symbolic motion commands to the remote slave. The slave robot will then execute these symbolic commands delayed in time. While much of a task will proceed error free, when an error does occur the slave system will transmit data back to the master and the master environment will be "reset" to the error state. From this point, the operator will correct the error by some alternative operations.

There is a clearly established need and corresponding capability to perform work remotely [1] [2]. Teleoperators were developed with the advent of the nuclear industry and have been used extensively since their development in 1945 [3]. A problem occurs, however, when teleoperation is attempted if there is delay between commands from the master arm and the response from the slave arm [4] [5]. Such problems occur when the operator, controlling the master arm and observing the slave (with a delay between command and response), attempts to bring the slave into contact with an object. When contact is perceived and motion of the master is stopped, the slave continues to move during the delay, causing a potentially damaging interaction. If a liquid is being poured and the master arm is lifted to stop the flow when the correct quantity is observed to have been poured, the slave continues to pour for the full extent of the time delay. Any delay in the feedback loop makes it very difficult to control such processes [3] [6].

Approaches to solving the delay problem involve slowing down the motion so as to minimize the effect of the delay [6] and on strengthening the slave arm and the objects with which it works in order to avoid damage. It has been shown that delays of the order of one second can produce instability and can dramatically slow down the performance of even the simplest tasks, such

as driving a vehicle [6]. With delays corresponding to 10 seconds, predicted for the Telerobotic Servicer working in synchronous orbit, for autonomous, deep underwater vehicles relying on low bandwidth acoustic communication, the control problem will become severe [7]. Proposed solutions to this problem include predictive displays [8] and the use of automated, sensor-driven programs [9] to accomplish the interactive parts of task performance.

We do not believe that either approach will prove satisfactory. While a totally autonomous manipulative capability would solve the problem, its realization is beyond the state of the art of robotics. We are combining computer graphics and sensing with teleoperation and robotics to solve the problem.

The use of sensors, such as vision or dense range, to produce a high level three-dimensional scene description consisting of such objects as edges, planes, vertices, is within the state of the art in computer vision [10] [11] [12]. It has also been demonstrated that an operator can interactively match such a description of a scene to a CAD model. We generate and display such a CAD image and have interfaced a six-degree-of-freedom input device to the display such that we can also display and move an image of a remote slave arm and any object that it is carrying.

We then monitor the position of the slave arm to detect penetration of any work objects by the slave arm or by any object it is carrying. When this occurs, we backdrive the input device (PUMA 250) so as to maintain positional and orientational correspondence between the input master device and the image.

With the first two capabilities in place, We can *feel*, kinesthetically, the objects represented in the display. When the inside of a box is displayed we can feel along a surface to a corner between two surfaces; we can slide along the edge into the corner of the box; we could can close our eyes and feel the displayed object. The combination of the visual display of a scene with kinesthetic feedback from the scene provides an extremely strong telepresence. The operator can really feel that she/he is "there."

Our current research is to detect motion of the input CAD manipulators and their contact with the image, together with forces the operator might exert on the objects in the image, and to turn these into symbolic motion commands [13]. For example, the operator might bring the slave arm into contact with a surface and then slide along it until the arm is brought into a corner between two surfaces. As we detect the collision with the surface

and detach motion in the surface normal direction we can issue symbolic instructions:

> move to <xyz, where contact occurred> while monitoring force in surface normal direction;
>
> on detection of force comply with surface in normal direction and exert force <observed to have been exerted by operator>
>
> move along surface to <xyz position where corner is detected> while monitoring force in direction of corner.

The above statements may be represented in any number of robot programming languages [14]. We believe that, given some general task description, and by monitoring the image and detecting interactions, we can generate such robot program statements. A task description will also simplify the image interactions, which must be monitored in real time. We propose to use input such as " moving into the corner of the box " to alert us that contact between the arm and the box is to be expected and that it will be in the vicinity of the corner. This type of mixed command interface is known as *hybrid analog/symbolic interface* [15]. This approach vastly reduces our task of detecting and monitoring image interactions. In the context of the box example, for instance, we would also expect sliding motions in the general direction of the corner. Such task information is a natural component of team procedures where the team leader announces intended operations before their execution is commenced such that other team members can take supportive action. We would see that this form of input would involve voice recognition, although we will limit ourselves to text string input corresponding to a reasonable transformation of potential speech input.

Notice that the robot programming commands do not include any conditionals such as "if a happened then do b else do c." These conditionals plague robot programming in which a situation must be anticipated and every possible outcome of an action predicted and programmed. As any robot programmer knows, it is impossible to account for everything that can happen during task execution, especially when one realizes that the corrective action for every error will itself involve errors – a hopeless situation [16].

The symbolic robot commands describing the operations that are being performed on the image can then be executed by a slave manipulator at a remote site. Notice that the time delay between the operator input and the

slave execution may be quite arbitrary; the slave is simply following along as if someone were sitting at a terminal writing a program and executing it line by line.

Of course, something might not work in the slave's world as it did in the image world. At this point we would alert the operator by "flashing" the image and then returning the display to the point at which the slave is "hung-up." We would also change the constraints on the input device to correspond to the situation the slave has detected. The operator would then try some other actions to move ahead in the task. Once again, these actions would be translated into symbolic robot commands and sent to the slave. We have not yet started on this final component of our proposed research: to provide feedback from the slave to the image modifying the state of the input device to represent the current situation.

We believe that our system will facilitate teleoperation with feedback time delay by providing a very natural interaction between the operator and an image of the task. This interaction involves immediate, simultaneous visual and kinesthetic feedback from a model of the task. The system allows for considerable feedback time delays limited only by the extent that the operator is allowed to move ahead of actual task execution. The system provides kinesthetic feedback, determined to be essential in the teleoperator industry back in 1948 [1], and eliminates the need to write elaborate robot manipulator programs attempting the impossible task taking account of any possible error.

Application of such technology to undersea manipulation would free us from the need to maintain wide bandwidth communications between an operator and the vehicle. While it appears possible to eliminate vehicle tethers based on energy considerations [17], it is still impossible to eliminate the tether based on manipulation control considerations due to the delays in bringing acoustic signals to the surface. Operators must either be in the vehicle or in a surface ship at the end of a tether. With the proposed technology it would be possible to drop a submersible from a plane together with an acoustic relay buoy and then to control operations at the ocean bottom remotely over a radio link from either the plane or the shore. The principal cost saving is, of course, the elimination of the need for a surface ship maintaining station during the entire underwater operation. Secondary cost savings relates to the elimination of the tether and the possibility of working in environments in which the tether might become tangled, also the possibil-

ity of using more than one submersible in the same working area when the control of tethers becomes impossible [18].

Cost justification for work in shallow space relate to the possibility of eliminating the need for an astronaut on EVA to perform the task, vastly reducing the cost involved.

2 Current Research

During the first year of the grant we have made substantial progress with the master station. We have built a data base using the Jack graphics system [19]. Simeon Thierry, representing the the Laboratoire d'Automatique et d'Analyse des Systemes with whom we are jointly conducting this research, developed the distance algorithm based on the approach of Gilbert [20] so that we could monitor collisions between objects. A small PUMA 250 robot manipulator was interfaced to the Sun control computer to act as the kinesthetic master input device. A Lord force/torque sensor, located at the wrist of the manipulator, was also interfaced to the Sun as part of the master. Programs have been written, running on both the Sun and the Jiffe processors [21] to control the image of the slave robot by means of the kinesthetic input device. See Appendix A.1. The kinesthetic input is quite dramatic providing a good sense of "telepresence," The operator can both *see* and *feel* what is going on in the simulation of the remote site.

We have also been working on the slave robot system. An initial interpreter was developed to run the robot fitted with the passively compliant wrist, developed here by Yangsheng Xu [22]. The wrist allows us to come into contact with the environment and to control forces of interaction. We are currently modifying this system to provide a communications link to the Master system and will provide a simple demonstration of the total system before the end of the summer. See Appendix A.2. This system is based on an intermediate language which will be generated at the master station as the task is performed in simulation and interpreted on receipt by the slave system. Masahiko Hashimoto, visiting from Japan, is working on the intermediate form and its parsing using yacc.

We have established connections with International Submarine in Vancouver, Canada and are discussing the possibility of a demonstration experiment on their autonomous vehicle in the third year of the grant. We have also

established connections with Monash University in Melborne, Australia and are jointly supervising a project to develop a low cost, low speed manipulator suitable for either AUV's of small ROV's.

3 Talks and Presentations

1. Richard P. Paul, February 1990, Carnegie-Mellon University, "Manipulation in Unstructured Environments," invited talk.

2. Richard P. Paul, March 1990, LAAS Toulouse, "Teleprogramming: A Basis for Programming Robots and Teleoperators," invited talk.

3. Richard P. Paul, March 1990, Univeristy of British Columbia, "Manipulation in Unstructured Environments," invited talk.

4. Richard P. Paul, Janez Funda, Simeon Thierry, Thomas Lindsay, June 1990, Intervention '90 Conference, Vancouver, Canada, "Teleprogramming for Autonomous Underwater Manipulation Systems," conference paper.

5. Richard P. Paul, July 1990, IROS '90 Conference, "Manipulation in Unstructured Environments," tutorial lecture.

References

[1] Ray C. Goertz. Manipulators used for handling radioactive materials. In E. M. Bennett, editor, *Human Factors in Technology*, chapter 27, McGraw Hill, 1963.

[2] Robert W. Corell. Closing summary. In *The Fifth International Symposium on Unmanned, Untethered Submersible Technology*, pages 618–625, Marine Systems Engineering Laboratory, University of New Hanpshire, June 1987.

[3] T. B. Sheridan. Telerobotics. In *Workshop on Shared Autonomous and Teleoperated Manipulator Control*, 1987. IEEE International Conference on Robotics and Automation.

[4] W. R. Ferrell. Delayed force feedback. *IEEE Trans. Human Factors in Electronics*, 449–455, October 1966.

[5] W. R. Ferrell and T. B. Sheridan. Supervisory control of remote manipulation. *IEEE Spectrum*, 81–88, October 1967. 4-10.

[6] W. R. Ferrell. Remote manipulation with transmission delay. *IEEE Trans. Human Factors in Electronics*, 1965. HFE-6, 1.

[7] Marine Systems Engineering Laboratory. *The Fifth International Symposium on Unmanned, Untethered Submersible Technology.* University of New Hanpshire, 1987.

[8] M. Noyes and T. B. Sheridan. A novel predictor for telemanipulation through a time delay. In *Proc. Annual Conf. on Manual Control*, Moffett Field, CA: NASA Ames Research Center, 1984.

[9] F. Schenker, R. French, and A. Sirota. The nasa/jpl telerobot testbed : an evolvable system demonstration. In *IEEE International Conference on Robotics and Automation*, March 1987.

[10] Martin Herman. Generating detailed scene descriptions from range images. In *IEEE International Conference on Robotics and Automation*, pages 426–431, 1984.

[11] C. I. Connolly, J. L. Mundy, J. R. Stenstrom, and D. W. Thompson. Matching from 3-d range models into 2-d intensity scenes. In *IEEE First International Conference on Computer Vision*, pages 65–72, 1987.

[12] David R. Smith and Takeo Kanade. Autonomous scene description with range imagery. *Computer Vision and Graphics Image Processing*, 31, September 1985.

[13] Richard P. Paul. Sensors and the off-line programming of robots. In *Proceedings of 1983 International Conference on Advanced Robotics*, pages 307–312, Tokyo, JAPAN, September 1983.

[14] Richard P. Paul. WAVE: a model-based language for manipulator control. *Industrial Robot*, 4(1):10–17, March 1977.

[15] Antal K. Bejczy. Data-driven automation in remote applications of robots. In *Workshop on Shared Autonomous and Teleoperated Manipulator Control*, 1988. IEEE International Conference on Robotics and Automation.

[16] Richard P. Paul. Programming languages for manipulation. In G. Saridis, editor, *Advances in Automation and Robotics: Theory and Applications*, JAI Press, 1983.

[17] Marilyn Niksa. Aluminum-oxygen batteries as power sources for submersibles. In *The Fifth International Symposium on Unmanned, Untethered Submersible Technology*, pages 121–127, Marine Systems Engineering Laboratory, University of New Hanpshire, June 1987.

[18] Richard P. Paul and Georges Giralt. An autonomous sensor-controlled manipulation capability for an untethered, unmanned, submersible. In *Proceedings of AUVS-88*, June 1988.

[19] Cary B. Phillips and Norman I. Badler. Jack: a toolkit for manipulating articulated figures. In *Proceedings of ACM/SIGGRAPH Symposium on User Interface Software*, Banff, Alberta, Canada,, 1988.

[20] E.G. Gilbert, D.W. Johnson, and S.S Keerthi. A fast procedure for computing the distance between objects in three space. In *IEEE International conference on Robotics and Automation*, 1987.

[21] R. L. Andersson. Computer architectures for robot control: a comparison and a new processor delivering 20 real Mflops. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1162–1167, 1989.

[22] Yangsheng Xu and Richard P. Paul. Hybrid position force control of robot manipulator with an instrumented compliant wrist. In V. Hayward and O. Khatib, editors, *Experimental Robotics 1, Lecture Notes in Control and Information Science*, pages 244–270, Springer-Verlag, 1990.

A APPENDICES

A.1 Teleprogramming: Overcomming Communication Delays in Remote Manipulation

Teleprogramming:

Overcoming communication delays in remote manipulation

Janez Funda
GRASP Laboratory
Computer and Information Science Department
University of Pennsylvania

Dissertation Proposal
Supervised by Dr. Richard P. Paul

June 14, 1990

Contents

List of Figures

1 Introduction

Modern industrial processes (nuclear, chemical industry), public service needs (fire-fighting, rescuing), and research interests (undersea, outer space exploration) have established a clear need to perform work remotely. Whereas a purely autonomous manipulative capability would solve the problem, its realization is beyond the state of the art in robotics [Stark *et al.*,1988]. Some of the problems plaguing the development of autonomous systems are: a) anticipation, detection, and correction of the multitude of possible error conditions arising during task execution, b) development of general strategy planning techniques transcending any particular limited task domain, c) providing the robot system with real-time adaptive behavior to accommodate changes in the remote environment, d) allowing for on-line learning and performance improvement through "experience", *etc.* The classical approach to tackle some of these problems has been to introduce problem solvers and expert systems as part of the remote robot workcell control system. However, such systems tend to be limited in scope (to remain intellectually and implementationally manageable), too slow to be useful in real-time robot task execution, and generally fail to adequately represent and model the complexities of the real world environment. These problems become particularly severe when only partial information about the remote environment is available.

Consequently, teleoperators remain the most reliable option for performing work remotely, in hazardous circumstances, and in unstructured (or partially structured) environments [Hayati *et al.*,1990]. Teleoperators were developed with the advent of nuclear industry in the mid 1940's and have since found applications in many other areas, such as undersea resource exploration, waste management, and pollution monitoring, as well as in outer space for sample acquisition, satellite deployment and repair, *etc.* From the early prototypes which provided for mechanical linking of kinematically similar master and slave arms [Goertz,1963], teleoperators have evolved into sophisticated systems, offering substantial dexterity of manipulation at maximum convenience to the operator [Ballard,1986], [NASA,1988], [Schenker,1987], [Hirtzinger,1989]. These systems feature dissimilar master and slave manipulators, where each is custom-designed to best perform its function, high bandwidth communication between the master and slave sites, high fidelity stereo visual feedback

from the remote site, as well as force-reflecting, bilateral servo control to allow the operator to kinesthetically "feel" the interactions of the slave arm with its environment. The combination of the above affords the operator an effective working environment and a good sense of *telepresence*, *i.e.*, the illusion that she is actively present in the remote environment.

This document is organized as follows: Section 2 defines the problem that we are addressing. Related work and the current state of the art in the field of remote manipulation are described in Section 3. Section 4 introduces the basic modules of the proposed conceptual architecture, whereas Sections 5—8 address separate components in more detail. The graphical simulation module is described in Section 5. The section presents the simulation technique and offers some detail on the manner in which distances between objects in the simulated (slave) environment are monitored and collisions, as well as contact types, are identified. In Section 6 we introduce the envisioned operator-machine interface and describe the manner in which we compute a real-time approximation to the (delayed) actual kinesthetic feedback. Section 7 proposes a simple 6 degree-of-freedom (DOF) filter to smooth the operator supplied motion trajectories, whereas Section 8 describes the methodology for partitioning the task in progress and generating a sequence of symbolic command strings to be executed by the remote slave.

2 Problem Statement

2.1 Communication delays

The above scenario of teleoperation assumes high-speed, high-bandwidth communication between the operator's station and the remote site. While this can be achieved for most land-based close proximity telerobotic applications, it becomes a problem when the master and slave sites are separated by a large distance (*e.g.*, earth – moon) and/or are forced to communicate over a limited bandwidth communication link (*e.g.*, acoustic link to an underwater manipulator) [Ferrell,1966], [Ferrell&Sheridan,1967]. Under such circumstances, both the instructions to the slave manipulator (desired velocities and forces) and the feedback from the slave back to the operator (visual and kinesthetic information) are delayed. This, clearly, will adversely affect the efficiency and "fluidity" of task performance, as the result of the operator's motion commands to the slave is not known to her until a communication delay later. A typical operator's response under such circumstances is to adopt a "move-and-wait" strategy ([Ferrell,1965], [Ferrell,1966]), where the operator repeatedly issues small motion commands and then waits for the feedback (resulting state) from the remote environment.

2.2 Communication delays and task performance

To illustrate the delay problem in more concrete terms, consider a situation where we are teleoperating in the presence of a (one-way) time delay τ, due to the combination of transmission delay and limited bandwidth. Let A denote a task, which takes T_{task} time to execute without delay, by executing elementary task commands, each of which takes on average t time to execute. Then it can be shown, that the total time to execute the task in the delayed environment by using the move-and-wait approach, is

$$T_{\text{total}} = (1 + 2\frac{\tau}{t})\, T_{\text{task}} \tag{1}$$

Figure 1 illustrates the effect of communication delays on the total task completion time using the move-and-wait strategy (t = 1 sec). Thus, if we consider a twenty minute task (T_{task} = 20 min) with an elementary task execution time of 1 second (t = 1 sec) and with a transmission delay time of 10 seconds (τ = 10

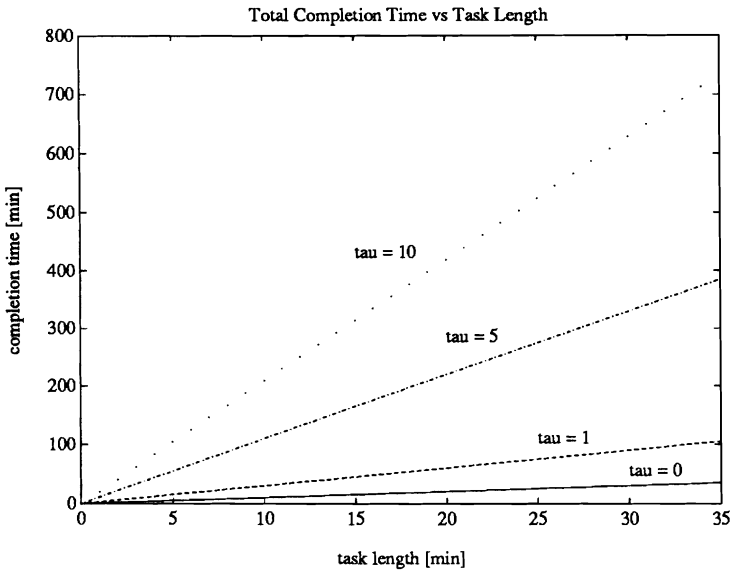

Figure 1: Total task completion time versus task length for elementary task length $t = 1$ sec and different values of the communication delay τ (in seconds).

sec), then the total time to execute task A would be 7 hours! Clearly this is not satisfactory.

2.3 Communication delays and telepresence

As we have seen, time delays can severely reduce the efficiency of task performance by forcing the operator to wait. Moreover, delays can also degrade or even destroy the sense of telepresence, which is essential for fluid and confident teleoperation. This is particularly true when the slave manipulator is in contact with the environment. Clearly, both the video signal and the information about the forces experienced by the slave arm are going to be delayed by 2τ. While delays in both cases cause a problem, the delay in the latter signal has been shown to be perceptually more significant in contact manipulation [Ouh-young,1989]. Physiological studies have shown that the neurological control of normal human movements operates at the rate of approximately 5 Hz [Stark,1987] and that a time delay of 300 ms ($\approx 1/3$ sec) is clearly perceptible to humans [Stark&Kim,1988]. Consequently, feedback delays approaching the one second level have severe destabilizing effects on the per-

formance of a human operator relying on this feedback information [Ferrell,1966], [Black,1971], [Bejczy&Kim,1990].

Unfortunately, in space and undersea applications, communication delays often exceed this threshold. Round-trip communication link delays between the ground station and a slave workcell in low earth orbit (*e.g.*, Space Shuttle) are normally in the 2 — 8 second range, depending on the number of intermediate geosynchronous satellite relay stations, the exact nature of the computer processing/buffering at the sending and receiving stations, *etc.* [Kim *et al.*,1990], [Sheridan,1990]. If teleoperation is to be performed in shallow space (*e.g.*, moon), then delays approaching or exceeding 10 seconds should be expected. Similarly, substantial delays arise when remote control of autonomous underwater vehicles (AUV) is employed to avoid the problems of dragging and tangling power/control tethers. Acoustic communication links are normally established between the AUV and the surface ship (or land-based operator station), and with the sound transmission underwater being limited to 1460 m/s, a round-trip time delay over a distance of 1 mile exceeds 2 seconds [Sheridan,1990].

2.4 Research goals

The goal of this research is to address the issue of communication delays in telerobotics and to propose, as well as test, some ideas which may help alleviate the problem. As basic control theory tells us, sustained, stable closed-loop control over a finite time delay is not possible [Sheridan,1990]. However, various control strategies and ways of sharing the necessary control functions between the remote site and the ground station in a remotely controlled robotic system are possible, which can dramatically improve our chances of being able to perform useful and effective work over large distances. We will in this work propose a possible solution to this problem, based on a notion of *teleprogramming* the remote robotic workcell. The basic components of the system will include a high-quality predictive display, real-time extraction of approximate kinesthetic feedback from the graphical simulation, and automatic generation of elementary task commands to be sent to the slave. The proposed solution is detailed in Section 4.

3 Related Work and State of the Art

3.1 Overcoming communication delays

Overcoming communication delays has been recognized as one of the central areas of research in telerobotics for some time [Stark,1987]. Among the proposed approaches to solve the problem are:

- slowing down the motion so as to minimize the effect of the delay [Ferrell,1965]
- strengthening the slave arm and the objects which it manipulates in order to avoid damage (*e.g.*, underwater remotely operated vehicles, ROV's, for off-shore oil exploration)
- adopting a "move-and-wait" strategy, where the operator proceeds through a sequence of incremental open-loop motions, each one followed by a wait of one round-trip delay to receive the correct feedback [Ferrell,1965]
- "supervisory control": limited autonomy at the remote site — sensory feedback loops are closed locally, the slave makes low-level decisions on its own, whereas the operator supervises the execution of tasks and supplies high-level goal information [Ferrell&Sheridan,1967]
- formally modelling up-link and down-link delays by augmenting the dynamic state-space model of the system (environment + slave) — delays are modelled as delay lines on the output and introduce (a potentially large number of) additional states [Hirtzinger *et al.*,1989]
- control theoretic approaches, such as introducing a control law which makes the communication link appear as a passive two-port lossless transmission line [Anderson&Spong,1988]
- using predictive visual (graphical) displays to allow the operator to "preview" the effects of her commands on the remote environment [Noyes&Sheridan,1984], [Bejczy&Kim,1990]
- full autonomy at the remote site: automatic on-line sensing, sensory data interpretation, strategy generation, task and motion planning, execution monitoring, error detection, replanning and error recovery

None of the above approaches to overcoming communication delays, by itself, has proven to be entirely satisfactory. In the presence of delays in excess of one second, simple move-and-wait strategies become impractical. On the other hand, full autonomy at the remote site is beyond the state of the art. At present, it seems that the integration of supervisory control, carefully designed control laws, and sophisticated operator station based predictive displays offers the best compromise between the desirable and the feasible.

In supervisory control, the sensory feedback from elementary actions of the slave is processed locally to make the necessary adjustments to the motion strategy in executing its current short-range goal. This partial remote autonomy reduces the sensitivity to the delays in communication with the operator's station and reduces the mental and physical burden on the operator. It also results in greater independence of the supervisory and the remote control loops, the two now being coupled only through the occasional asynchronous exchange of commands (from the operator to the remote workcell) and state information (from the remote workcell to the operator). In this type of remote control, the operator's station must make use of computer models of the remote environment and the remote workcell. These models are then graphically displayed to the operator, and the effects of operator's commands are computed in this simulated environment, offering the operator an immediate visual feedback of her actions.

The pioneering work in predictive display technology was done at the MIT Man-Machine Systems Laboratory [Noyes&Sheridan,1984], [Hashimoto *et al.*,1986], [Buzan,1989]. Other experiments have shown that 2-D perspective projections alone are not sufficient to represent 3-D information [Stark,1987], [Stark,1988]. Additional depth cues are needed to aid the operator in performing motions along the line of sight of the TV camera or along the graphical projection axis. Alternatively, stereoscopic displays can be used [Stark,1988]. [Pepper *et al.*,1981] have demonstrated the superiority of stereo displays over mono displays, and shown that the advantage of visual stereo increases with the complexity of the scene. State of the art predictive displays synchronize and overlay real-time computer graphics (complete with shading and a realistic lighting model) with the incoming delayed video camera signal on the same physical display [Bejczy *et al.*,1990].

3.2 Kinesthetic feedback

It is well established that force-reflection dramatically improves the sense of "telepresence" in teleoperation [Ferrell,1966], [Hannaford,1988], [Hannaford,1989]. Since visual and kinesthetic information can be supplied to the operator through different sensory input channels, they naturally integrate and augment each other. In fact, it has been shown that kinesthetic feedback can be at least as important as 3-D visual information [Kilpatrick,1976] and that, in some circumstances, force feedback alone can be more valuable than visual feedback alone [Ouh-young *et al.*,1989].

Communication delays preclude direct reflection of the reaction forces, experienced by the slave, to the operator's hand controller. Numerous studies have shown that delayed force feedback can destabilize the control loop. Moreover, experiments indicate that no force information at all may be better than delayed force feedback, since the perceived loss of the action/reaction causality tends to be confusing to the operator [Buzan,1989]. This confusion and disorientation arises regardless of whether the delayed force signal is fed to the active hand (*i.e.*, the one controlling the master arm) or the passive hand.

Consequently, delays in force information motivated research in generating synthetic, "quasi" kinesthetic feedback, which would approximate the expected actual force signal. Most of the effort concentrated on extracting force information from the predictive displays, *i.e.*, graphical simulations of the slave's interaction with the remote environment. Since too few physical parameters of the remote world and the objects therein are known for a full dynamic model to be useful and meaningful (even if there was time to compute it), remote environment simulations are almost invariably non-dynamic. Thus, the best one can do is to compute a reasonable approximation to the actual forces. A possible solution is to monitor contacts between objects in the graphical environment and compute the pseudo interaction force as an inverse function of decreasing distance between objects (beyond some proximity threshold). Both quadratic [Noll,1972] and linear laws [Fong *et al.*,1986] have been proposed.

An interesting application for extracting force information from a graphical display was proposed by [Ouh-young *et al.*,1988], [Ouh-young *et al.*,1989]. In this work, researchers simulate the interaction forces between a drug molecule and a specific

receptor site on a protein or nucleic acid molecule to find "good fits" by feel, rather than visualization alone. "Goodness of fit" is characterized by minimizing the interaction energy, which is a function of electric charges of the atoms and inter-atomic distances. The operator interacts with a magnified graphical display of the molecules and attempts to find, kinesthetically, the best geometric and electrostatic fit.

The state of the art telerobotic systems currently use sophisticated predictive displays but rarely attempt to generate pseudo-force information from the graphical slave/environment interactions. Normally, slave contact interactions are handled via local compliant control strategies at the slave site without generating kinesthetic feedback to the operator [Kim *et al.*,1990].

3.3 Programming robots

Robots can only perform useful work when they are in contact with the environment. However, interaction with the environment complicates control of robot manipulators due to sudden transitions between free-space and contact motion. Such transitions tend to excite high-frequency dynamics and cause control instabilities. Consequently, more sophisticated control strategies are needed to deal with contact manipulation and a variety of such control laws have been proposed: resolved acceleration control [Luh *et al.*,1980], operational space method [Khatib,1985], impedance control [Hogan,1980], stiffness control [Salisbury,1980], hybrid control [Raibert&Craig,1981] (see [Whitney,1987] or [An&Hollerbach,1989] for an overview of force control techniques).

The hybrid position/force approach separates the robot's Cartesian DOF's of motion into force and position (velocity) controlled directions. [Mason,1981] proposed a theoretical framework which allows us to analyze the geometry of contact(s) between the robot and the environment and define mutually orthogonal "naturally constrained" and "artificially constrained" directions. These directions can be though of as specifying a *task frame*, centered at the contact point, in which the robot's desired force and position trajectories can be conveniently specified. A task plan/strategy can thus be though of as a specification of a sequence of task frames and position/force trajectories along the artificially and naturally constrained DOF's, respectively.

While force control enables the robot to perform more skillful and more stable manipulation, programming such behavior is significantly more complex and intricate than programming simple positioning tasks. In order to facilitate easier and more convenient programming, a variety of programming languages has emerged: MANTRAN [Barber,1967], WAVE [Paul,1977], AL [Finkel *et al.*,1974], AUTOPASS [Lieberman&Wesley,1977], VAL [Shimano,1979], AML [Taylor *et al.*,1982], *etc.* The target application for most of these languages were assembly problems in manufacturing and automation, and programs were designed either off-line or interactively through a step-by-step interpretative process.

Recently, work has been done on at least partially automating the process of generating robot programs. [Grossman&Taylor,1978] used the manipulator itself as a three-dimensional pointing device to interactively generate object models and automatically produce the corresponding object declarations for the AL language. Asada *et al.* ([Asada&Izumi,1987], [Asada&Yang,1989]) have used a "teaching-by-showing" technique to automatically generate simple hybrid position/force control instructions for the robot. In this approach, the operator performs the task by holding on to the robot end-effector. During the teaching phase, the interaction forces and position trajectories are recorded and later processed off-line by using pattern matching techniques to map sensor signals to elementary motion commands. De Schutter *et al.* ([DeSchutter,1987], [DeSchutter&VanBrussel,1988]) proposed a method for automatically tracking and adjusting task frame position and orientation during task execution. The strategy consists of monitoring (on-line, through sensory readings) the evolution of the natural constraints and aligning the task frame with these dynamically determined constraints.

Most of the work on automatic robot program generation, to date, has concentrated in the area of automatic assembly task planning and strategy generation. Some of the major areas of research in this domain are:

- representational formalisms
  - representation of assembly parts (polyhedral models [Lozano-Perez *et al.*,1987], boundary representation models [Liu&Popplestone,1987], CSG models [Hoffman,1989])
  - representations of assembly sequences (AND/OR graphs [Sanderson,1988],

other types of graphs and trees),

- representations of part mating geometric constraints (Clifford algebra of projective 3-space [Ge&McCarthy,1990])

- formal frameworks for planing strategies

  - formal models for synthesizing compliant motion strategies from geometric descriptions of assembly operations and explicitly estimating errors in sensing and control [Lozano-Perez *et al.*,1983]
  - mathematical models for describing strategies which are guaranteed to succeed in the presence of sensory, control, and modeling errors [Donald,1986], [Jennings *et al.*,1989]
  - automatically generating assembly programs from design information by searching through a graph of contact formations [Desai&Volz,1989]

What emerges clearly from these efforts is that automatic generation of robot programs in the presence of significant modeling, sensory, and control errors is extremely difficult, and, in general, quite possibly unachievable [Desai&Volz,1989]. Typically, these methods analyze the problem of disassembly (a mathematically more constrained problem), produce a tree or a graph of all possible plans and call any reverse path through the graph a solution, *i.e.*, an assembly sequence. The search for a good (or at least feasible) solution in this graph may be guided by rule-based systems, heuristic data-bases, *etc.* Consequently, plan searching and selection must often be done off-line. In order to cope with the complexities of the problem, many simplifying assumptions are normally introduced into problem analysis (*e.g.*, planar surfaces only, translations only, *etc*), which limit the scope and usefulness of such schemes. Adaptive behavior and on-line learning techniques are needed for successful autonomous planning, error detection and replanning in the presence of uncertainties (*e.g.*, in unstructured environments).

4 Outline of the Proposed Solution

4.1 Approach

One of the guiding principles of our proposal is our belief that completely autonomous robotic systems are presently not feasible. A major problem with autonomous systems is the necessity to anticipate and provide corrective strategies for all possible error conditions [Brooks,1982]. This is difficult to do even when the robot is operating in a structured environment and hopeless in situations where the environment is only partially known and significant modeling, sensing and control uncertainties exist. Classical approaches to programming robots for such applications end up being bogged down with large amounts of error detection and recovery code. To make matters worse, recovery procedures are themselves subject to errors and the actual program may easily become dwarfed by the amount of error handling code.

Thus, we believe that there is a need for a human operator in the control loop of a remotely controlled manipulator system. Moreover, we propose that the human operator participate in the control and decision making process in a supervisory role. We will refer to the particular form of supervisory control, described in this document, as *teleprogramming*. Teleprogramming a remote manipulator essentially corresponds to visually and kinesthetically interacting with a virtual world (a graphical simulation of the remote environment), and generating, on-line, a sequence of symbolic instructions to the remote robotic system, based on the operator's interactions with the virtual environment.

In designing a system, which would implement these ideas, a key consideration was the notion that the most important long-term goal of remote manipulation research was improvement and "optimization" of the interface between human and artificial intelligence [Ferrell&Sheridan,1967]. Towards this aim, the system attempts to aid the operator by providing operational modes where the operator is expected to control only a few task parameters at a time, by "guessing" the operator's intent and making fine adjustments to her motions, *etc.* However, regardless of how the operator's motion might be modified, it is crucial that this be done in a manner which produces no surprises to the operator. This is because trust may well be the most important factor in a man-machine interaction — so much so, that it may be

often worth sacrificing efficiency and functionality for consistency [Boissiere,1988].

The following sections (4.2—4.9) introduce the major conceptual building blocks of the proposed solution. Section 4.10 evaluates the proposed approach in terms of task completion time and describes some applications for the proposed technology. Figure 4 at the end of this section (4) illustrates the place of these conceptual modules in the overall system hierarchy. The figure also shows the main hardware components of our laboratory experimental testbed, described in more detail in Section 10.

4.2 Modeling the environment

We will assume in this work that we are manipulating in an a priori unknown environment. The initial description of the environment is obtained through the use of sensors, such as vision or dense range data. The operator then interacts with the image segmentation process and aids the system in identifying objects and features in the remote environment, producing an unambiguous three-dimensional description of the scene in terms of object features such as planes, edges, and vertices. The process of extracting this information is within the state of the art of computer vision [Connolly *et al.*,1987], [Herman,1984], [Smith&Kanade,1985]. Moreover, it has been demonstrated that such descriptions can be converted to polyhedral CAD-type models [Hayati&Wilcox,1987], [Noyes&Sheridan,1984]. We propose to display such a CAD image of the environment (including the slave manipulator) and interface a 6 DOF input device (master) to the simulator, such that the images of the slave manipulator and any objects that it may be manipulating could be moved under the control of the operator.

Because the environment is assumed unstructured and we must rely on an idealized and simplified approximation of the actual environment, we can not predict all work situations (due to model incompleteness), nor can we predict the outcome of a particular action exactly (due to model inaccuracy). Therefore, we are unable to construct detailed, robust and reliable plans of action ahead of time. Instead, we propose to keep the operator in the control loop at all times, and let her define the plan incrementally as she interactively programs the slave robot actions by moving the master.

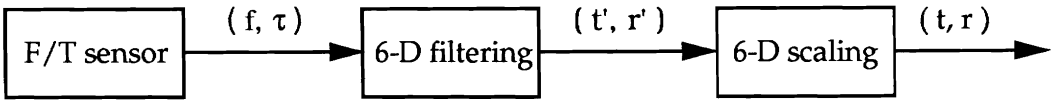

Figure 2: Interpretation of the force/torque sensor readings.

4.3 Controlling the motion of the slave

Motion parameters are specified to the (simulated) slave manipulator via the master input device, which can be controlled in a variety of ways. We propose to use a general 6 DOF force/torque sensor mounted at the tip of the master device, whose force and torque readings are interpreted (through a series of filters and amplifiers) as positional and orientational information, respectively (Figure 2). The pair $(\mathbf{f}, \tau)$ is the 6-vector of raw forces and torques as read from the sensor. This information is then filtered/smoothed and appropriately scaled to become the desired incremental positional/orientational displacement of the master (operator's hand). The rotation $\mathbf{r}$ is interpreted as roll/pitch/yaw (RPY) parameters.

The so obtained incremental displacement[1]

$$\Delta\mathbf{d} = (\mathbf{t}, \mathbf{r}) \tag{2}$$

is interpreted as *master handle* (sensor-based frame) displacement. The motion of the master manipulator is then computed by mapping this handle displacement into the master's end-effector frame ($T6_m$) and using it as an incremental Cartesian positional displacement in end-effector frame coordinates.

The motion of the slave simulator is coupled to the motion of the master by establishing a *correspondence of motion* between the *master's handle frame* and the *slave's end-effector frame* ($T6_s$). In general, due to the fact that the master and slave manipulators will be kinematically dissimilar (and will therefore have different workspace volumes), this correspondence will not be a straight-forward one-to-one positional/orientational equivalence of motion. Instead, another level of scaling for translational motions will be needed to account for the workspace volume differences.

[1]We use the term *incremental displacement* instead of *differential displacement*, since we deal with discrete rather than instantaneous changes in displacement.

The master manipulator is controlled in position mode. Successive Cartesian positions are computed from the incremental positional displacements and the master arm is servoed to these positions at a constant rate. A key consideration in controlling the master is ensuring that its particular kinematic properties do not affect the process of controlling the slave. The operator should not be concerned with the nature and implementation of the master device. Therefore, the master controller must ensure that the mechanism never reaches a singular configuration (where it becomes unstable) or approaches the boundary of its workspace volume. The control techniques, aimed at solving this problem, are normally referred to as *reindexing methods*. We propose to investigate three distinct reindexing schemes:

- offloading the reindexing responsibility to the operator: in this scenario, the operator needs to identify that the master is approaching a singular configuration and reindex (*i.e.*, reposition) the manipulator manually
- reindexing through a continuous drift back to the "home position": here, the magnitude of the restoring drift is a function of the distance (for translations) and twist amplitude (for rotations) from the home position
- automatic reindexing: the master device monitors its own motions and alerts the operator when it approaches a singular configuration (*e.g.*, by beeping); it then automatically returns to the home position and signals the operator that she may proceed with the task

All of the above approaches to reindexing imply that the display is decoupled from the motions of the master during reindexing. We propose to implement and evaluate relative advantages of the three methods.

4.4 Generating kinesthetic feedback

Having obtained an initial graphical description of the remote environment and being able to move the slave manipulator in this world, we now monitor the position of the slave arm (and any object it may be carrying) for contacts with the environment. This collision checking must be performed in real time and is used to prevent interpenetration of colliding objects. Penetrating motions are stopped on contact, thus modifying the intended motion of the (simulated) slave manipulator.

In order for the system to feel natural to the operator, the positional/orientational correspondence between the master device and the slave must be preserved at all times, including on contact with the environment, as well as while one or more contacts persist. We therefore need an input device, which is itself movable in space and backdrivable, such as a specially designed teleoperator master arm or a backdrivable general purpose robot manipulator. Such a device enables us to not only specify the desired positional/orientational displacement to the slave arm, but also gives the operator a sense of three-dimensional manipulation as it follows the operator's hand through space. More importantly, however, backdriving the master arm to correspond to the state of the simulated slave arm provides the operator with the ability to explicitly feel the constrained DOF of motion of the slave (and thus master) and therefore allows the operator to kinesthetically "feel" contacts between objects, examine shapes of objects, follow their contours, *etc.* This capability of combining graphical object interference detection with backdriving the master device represents a crucial feature of the proposed system. It provides the operator with a strong sense of telepresence (*i.e.*, a simulated sense of force reflection in real time), despite the communication delays, which cause the actual feedback to be delayed and therefore not usable for direct reflection to the operator.

4.5 Aiding the operator

The operator can now move the slave manipulator in the simulated world, come into contact with the environment and "feel" in a very natural way any constraints that the geometry of the task world may be imposing onto the motion of the slave. Moreover, we propose that the system provide a set of elementary *classes of motion*, which are natural, convenient and easy to perform, yet powerful enough to allow the operator sufficient flexibility in performing tasks. This is particularly crucial during contact motion, when the operator may wish to concentrate on a certain subset of motion parameters (*e.g.*, sliding, reorienting), and be aided by the system in keeping other parameters constant. The system can also assist the operator by biasing the interpretation of her motions towards preserving achieved contacts (for instance, to aid in feature tracking), while still allowing arbitrary changes of or departures from the current contact state. We will address these issues in more detail in Section 6.

4.6 Generating remote slave motion commands

We next attempt to interpret the information accumulated by the simulator to extract a stream of elementary motion commands that are to be commanded to the slave robot. In view of this, we first filter the gathered information and eliminate the "noise" in the data. We then analyze this filtered information of positional/orientational parameters and contact state changes to produce a sequence of symbolic instructions to the slave. Again, as our model of the slave world is only approximate, the nature of these instructions must reflect and accommodate possible discrepancies between the model and the actual world. While this is not critical during free space motion, it is vitally important when attempting to establish or maintain contact with the environment. Consequently, for the case of contact motion, we propose to generate instructions of the type "move along a given direction until contact" (guarded motion), or "move along a given feature while maintaining contact in some direction" (compliant motion). There will be also a class of motions (such as tight tolerance part mating, fine precision adjusting motions) which may be difficult to perform using an incomplete model and approximate kinesthetic feedback. Such motions are therefore best executed by the slave autonomously, under local sensor supervision and local high-bandwidth feedback processes. We will have more to say about symbolic command string generation in Section 8.

4.7 Using task information

The process of interpreting the operator's actions in the simulated world can be a difficult one in the absence of any other information about the nature of the task in progress. For instance, a sequence of rapid contact changes may be interpreted either as noisy data or a purposeful action, such as tapping, scraping, or rocking. Similarly, a highly irregular path of an object during a sliding motion could be taken as unintended (and therefore would be filtered out or smoothed) or it could correspond to a motion such as polishing or sanding (in which case it should be kept intact). In order to disambiguate between such interpretations, the system needs additional information about the task, such as a description of the type of expected primitive motions (*e.g.*, pick and place, polishing, pounding). Moreover, the graphical simulator should be supplied with some information as to which objects

are expected to come into contact during a given task to avoid having to monitor every pair of objects for a possible collision.

These are but a few examples of why high-level task information may be essential for correct interpretation of the operator's intent and for efficient internal computations. We feel that the design of the structure, organization, and content of such a task-level database is a significant research problem in itself. Consequently, we may not be able to address this aspect of the proposal fully in the preliminary stages of the project. However, we envision the task related information being gathered in the following manner:

- by loading and using a pre-existing task database
- by querying the user (operator) prior to the manipulation to extract the essential features of the task to be performed
- by maintaining an on-line dialogue with the operator to allow her to augment and modify the current task information while the task is in progress, as well as to allow the command stream generator to request additional information from the operator when her intent is still unclear

This would allow on-line refinement of the task description and should greatly expand the repertoire of tasks that the system could interpret correctly and thus issue appropriate motion commands to the remote slave.

4.8 The remote workcell

Although delayed communication with a human operator is available, the remote slave manipulator must operate with a certain degree of autonomy. The slave must be able to decide if an action is successful, and if not, it must decide what to do during the time when an error is detected and the human operator sends appropriate corrective instructions. When a command action is terminated successfully, the slave must be able to verify this and proceed to the next command.

Commands that are sent to the remote site are executed with caution – if the manipulator senses unexpected forces, it must respond correctly. The slave must first decide what action to take: to maintain the current position or to comply with the force. If the unexpected forces are small and static, the manipulator can

stop and maintain its current position until a response from the operator arrives. On the other hand, if the forces are large or dynamic, maintaining position could damage the manipulator. In this case, the manipulator should comply with the forces, trying to minimize the damage. Next, the slave must alert the operator and send enough information to the operator, so that the unexpected forces can be explained. If the resulting error state remains unclear, the operator may initiate some local exploratory motions or request that additional sensory information be gathered (*e.g.*, additional camera views). When the state of the slave and the remote environment has been determined, the graphical model at the operator's station is updated and the operator can proceed to take appropriate corrective actions.

In order to support its expected degree of autonomy, the remote robotic system needs to be equipped with sufficient sensing capability to carry out elementary motion commands robustly despite small errors in the command parameters. This sensory information from different sensors (TV cameras, range scanners, force/torque sensors, *etc.*) must be integrated into the low-level control algorithms to provide compliant and locally adaptive response in contact motion.

4.9 Error handling and model consistency

We now have a system where a human operator can *teleprogram* a remote slave robot, overcoming the communication delay problem by using real-time simulated visual and kinesthetic feedback. Of course, while all is well in the simulated world, various things may go wrong in the actual work environment. The slave can detect such error conditions by not reaching an expected motion-terminating condition, by hitting an obstacle, by sensing excessive or premature motor torques, *etc.* Upon detecting such a condition, the slave can signal the occurrence of an error state to the operator's station, which in turn can alert the user through a variety of visual or audio means (*e.g.*, flashing the display, synthesized voice warnings, *etc.*). It is then up to the operator to plan corrective actions. First, the operator's station based model of the world may need to be updated to properly reflect the current situation. This can be done through gathering and reconciling information from a variety of remote site based sensors (*e.g.*, video cameras, range finders, *etc.*) and/or purposeful exploratory motions on the part of the operator (if this is possible) to

find or correct certain model parameters. Then, the operator can attempt to recover from the error state and proceed with the task. Therefore, by keeping the human operator in the control loop, the system eliminates the need for elaborate exception and error handlers to be preprogrammed off-line.

It is important to note that discrepancies between the model and the world can also arise due to effects of external environmental agents, *i.e.*, other than slave's actions. Such changes may not be discovered through the actions of the slave, but may cause problems at a later stage in the manipulation. What is needed, therefore, is a rather sophisticated *environment updating mechanism*, which continuously (in reasonable intervals) checks at least the local portions of the environment model (*i.e.*, the immediate work area), but can also be brought into action by request from the operator. The latter facility is important not only for situations when the slave has entered an error state, but also when the operator wishes to verify poorly recovered or uncertain features of the workspace.

We believe that the problem of ensuring consistency between the model and the world is a very critical one for the successful operation of the proposed system and again represents a challenging research topic in its own right. We will in this work restrict ourselves to some general comments on how this problem may be solved and will not attempt to provide a detailed solution.

4.10 Summary and applications

The teleprogramming concept, outlined above, distributes decision-making and control between the human operator (who provides for high-level planning and error recovery) and the remote workcell control system (which provides for low-level autonomous task execution and control, as well as error state identification). Within this paradigm, commands may be sent from the operator's station one after another in a continuous stream, relying on the partial autonomy at the remote site to execute these commands under local sensory supervision a communication delay τ later. Therefore, the operator need not wait for explicit feedback from the remote site following each elementary command. When an error does occur, however, the remote control system stops the robot and alerts the operator. The operator then replans from this point, once again starting a stream of commands to be executed

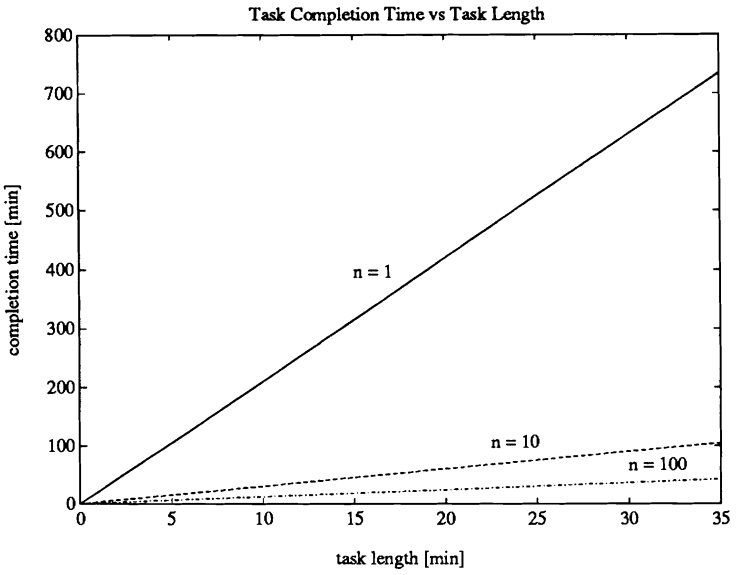

Figure 3: Total task completion times versus task length $\tau = 10$ sec, $t = 1$ sec, and three different values of n. Note that $n = 1$ corresponds to the move-and-wait strategy.

autonomously by the slave. In view of our earlier discussion of the total completion times using the move-and-wait strategy (Section 2.2), we can now compute the equivalent statistic for the teleprogramming paradigm. If n is the number of elementary commands that will be executed, on average, without the need for human supervisory intervention (*i.e.*, without an error occurring), then the time to perform a task will be given by

$$T_{\text{total}} = (1 + 2\frac{1}{n}\frac{\tau}{t})\, T_{\text{task}} \tag{3}$$

In view of Eq.(3), it is easy to see that the overall time efficiency of performing tasks will be improved by increasing either t (greater remote site autonomy) or n (greater system reliability and error-tolerance). Figure 3 illustrates the effect of increasing n on the total task completion time. It is evident from the figure that even a modest degree of autonomy and error-tolerance at the remote site improves the performance dramatically. In particular, for $n = 100$, the total completion time for the twenty minute task A of Section 2.2 becomes 24 minutes, which is a dramatic improvement over 7 hours.

We believe that a system, such as the one described above, will facilitate remote manipulation with time delay, allowing a very natural interaction between the operator and an image of the task involving both visual and kinesthetic feedback. The system will also allow for considerable time delays limited only by the extent that the operator is allowed to move ahead of actual execution.

Application of such technology to undersea manipulation would free us from the need to maintain wide bandwidth communications between an operator and the vehicle. While it appears possible to eliminate vehicle tethers based on energy considerations [Niksa,1987], it is still impossible to eliminate the tether based on manipulation control considerations due to the delays in bringing acoustic signals to the surface. Operators must either be in the vehicle or in a surface ship at the end of a tether. With the proposed technology it would be possible to drop a submersible from a plane together with an acoustic relay buoy and then to control operations at the ocean bottom remotely over a radio link from either the plane or the shore. The principal cost saving is, of course, the elimination of the need for a surface ship maintaining station during the entire underwater operation. Secondary cost savings relate to the elimination of the tether and the possibility of working in environments in which the tether might become tangled, as well as the possibility of using more than one submersible in the same working area when the control of tethers becomes impossible.

Cost justification for work in shallow space relate to the possibility of eliminating the need for an astronaut in performing "extra-vehicular activities" (EVA), vastly reducing the cost involved.

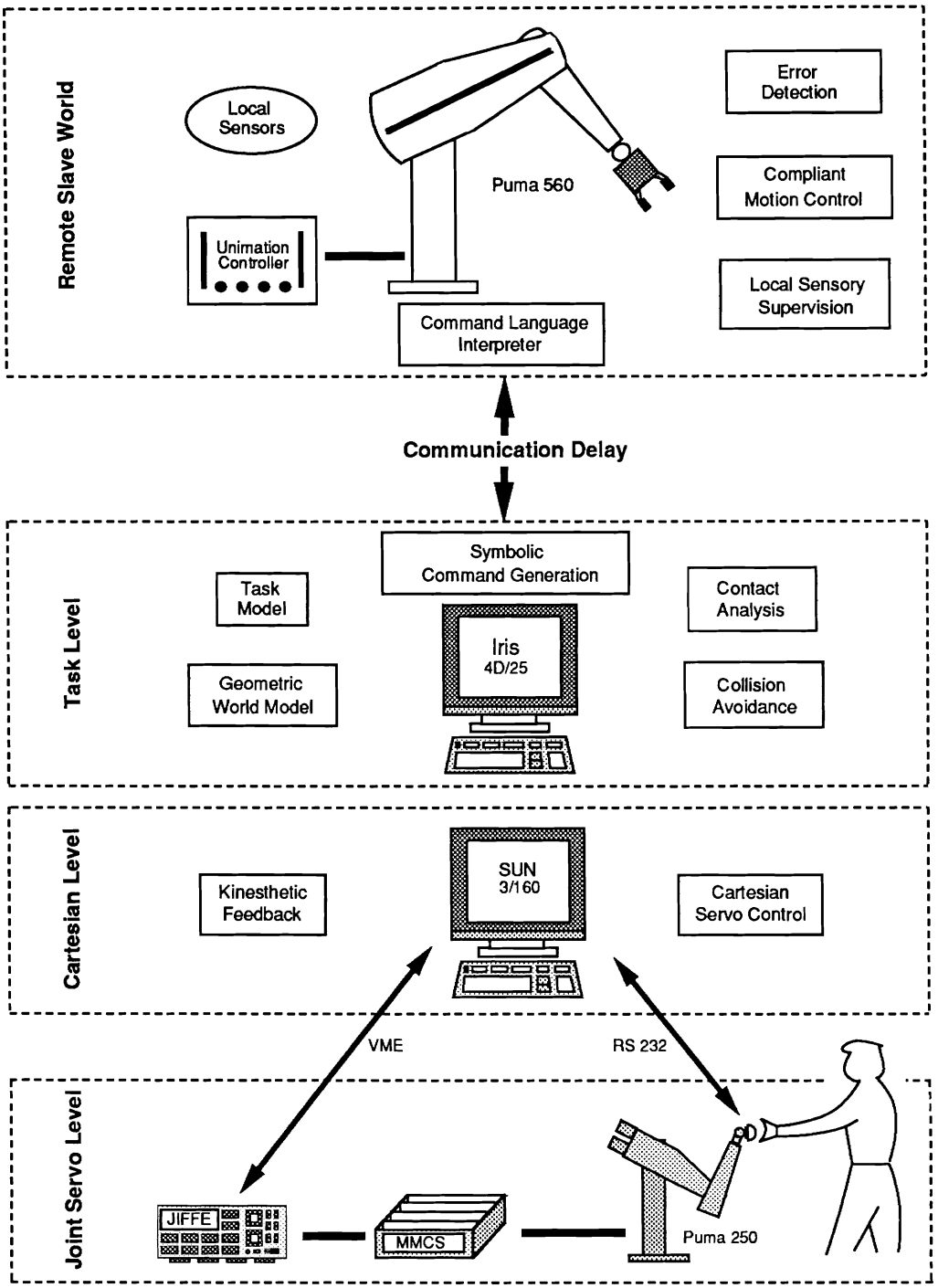

Figure 4: Overview of the proposed solution.

5 The Graphical Simulator

5.1 The polyhedral model

We propose to adopt a polyhedral, boundary-representation based graphical model of the world. While other representations are clearly possible (*e.g.*, constructive solid geometry (CSG), generalized cylinders), polyhedral models are widely used and consequently a variety of algorithms exist for polyhedral analysis. Perhaps the most important advantage, however, is the convenience of polyhedral models for contact analysis, which is a central requirement and feature of this work.

An important component of the graphical simulator is an exact kinematic model of the slave manipulator (and any attached equipment). This simulated slave robot must accurately reflect the kinematic limitations of the actual slave (*i.e.*, joint range and workspace limitations) and the simulator software must ensure this. Moreover, there should be no need for the slave and the master manipulator to bear any structural or kinematic resemblance to each other. While this significantly complicates the control of the system (space transformations, two sets of singular configurations, reindexing), it is an important feature of a general purpose teleprogramming software system.

5.2 The simulation technique

A key decision in this work has been to use a *kinematic* simulation of the motion of the slave and the manipulated objects. The simulation therefore does not account for the dynamic effects of either the slave robot or the environment. Moreover, the slave (plus any held object) are the only moving parts in the environment during each simulation time slice. Consequently, dynamic changes in the environment, other than the slave's state, must be related to the operator's station through the environment updating mechanism (Section 4.9), rather than direct simulation. This applies to the dynamic changes caused by the slave (*i.e.*, dropping or tipping an object), as well as those produced by external environmental agents (*i.e.*, winds, water currents). While the choice of a kinematic simulation may seem restrictive, we feel that it is the most practical approach for the following reasons:

- since only approximate information about the world is available, we can not expect to have complete information about the masses, centers of mass, inertias, frictional parameters, *etc.* about the objects in the environment; yet, these are essential parameters for a dynamic simulation

- in many environments and situations, a rigid-body dynamic model may not be adequate; we may be manipulating on a soft ocean bottom, or we may have erroneous confidence in the hardness of the objects in the slave world

- a dynamic simulation of both the robot and the environment represents a significant computational burden; in all but the simplest cases it, in fact, may not be computable in real time

- due to model uncertainty, only rough predictions based on dynamic computations are possible; such approximate, unreliable results do not justify the time spent in computation

- unmodelable and unpredictable external agents (water turbulence, buoyancy effects) may contribute to the dynamic state of the world, further diminishing the utility of a costly dynamic simulation

Clearly, a kinematic simulation leaves much to be desired, but under the circumstances we feel that it is a more reasonable and more practical choice than a full dynamic simulation of both the slave manipulator and the environment.

5.3 Distance computation

The kinesthetic feedback described in Section 6 relies heavily on the detection and analysis of the contacts which arise during the motion of the slave in the simulated environment. Expected contacts will normally occur between the slave's end-effector, tool, or an object it is currently holding, and some part of the slave world involved in the execution of the task. We will hereafter refer to the former as the *movable object* and will abbreviate it as MO. Moreover, the graphical simulator must also provide an aid to the operator by checking that undesired collisions between the slave arm and the environment do not occur during the motion.

Both desired and undesired collisions can be detected by monitoring the distances between pairs of objects. While the former requires precise models of the objects, simpler, approximate, yet conservative models suffice for the latter. Simplified models are preferred, whenever possible, in order to limit the computational cost of the collision checking module.

During the execution of a task, many pairs of objects may need to be monitored for contact at each step of the simulation. Consequently, there is a definite need for an efficient distance computation algorithm.

Several methods exist to compute distance[2] between polyhedral objects. Because of its efficiency, we chose to implement the distance algorithm between convex sets of points described in [Gilbert *et al.*,1987]. The aim of this section is to summarize the main features of this algorithm. For a more detailed description, the reader is referred to [Gilbert *et al.*,1987].

Let A and B denote the two polyhedral objects, whose distance (from each other) we are seeking. For the purposes of the algorithm the two objects need to be represented simply as the respective sets of vertices $S(A)$ and $S(B)$. The algorithm uses the following property of distance between the two sets

$$dist(A, B) = dist(\phi, C) \tag{4}$$

where ϕ denotes the origin of the space and $C = B \ominus A$ represents *Minkowsky's difference* between the sets A and B. Instead of first computing C[3], the algorithm is based on an iterative procedure which generates sequences of elementary sets C_k containing 1 to 4 vertices of $S(C)$. These C_k are such that their distance to the origin converges to the desired distance between the objects A and B.

An efficient procedure is used to compute the closest point $\mathbf{u}_k$ of the convex hull of these simple sets of points C_k (line segments, triangular faces, tetrahedrons) to the origin of the space. $\mathbf{u}_k$ is obtained from the computation of the coefficients λ_i of the set's barycentric representation, *i.e.*,

$$\mathbf{u}_k = \sum \lambda_i \cdot \mathbf{x}_i \qquad \text{with } \lambda_i \geq 0 \ , \ \sum \lambda_i = 1, \text{ and } \mathbf{x}_i \in S(C_k) \tag{5}$$

[2] *Distance* between two objects is defined as the smallest translation which will put them into contact.

[3] If A and B have n_A and n_B vertices, respectively, then C can have up to $n_A \cdot n_B$ vertices.

The points $\mathbf{x}_i$ of C_k whose $\lambda_i > 0$ define a $C_k^* \subset C_k$ containing $\mathbf{u}_k$ (for example, if C_k is a triangular face defined by three vertices, then C_k^* can be either one of the three line segments, or one of the three vertices of the face, depending on the number of positive λ_i computed). The sequence of $\mathbf{u}_k$ generated is such that $\|\mathbf{u}_{k+1}\| \leq \|\mathbf{u}_k\|$ and the norms converge to $dist(A, B)$.

The generation of the next C_{k+1} from the current C_k and $\mathbf{u}_k$ is based on the notion of a support function. The support function of a set of points X is defined by

$$h_X(\mathbf{n}) = \max_{\mathbf{x}_i \in S(X)} \{\mathbf{n} \cdot \mathbf{x}_i\} \tag{6}$$

and we will use $s_X(\mathbf{n})$ to denote one of the $\mathbf{x}_i$ which satisfies this maximum.[4]

It is shown in [Gilbert *et al.*,1987] that if $\|\mathbf{u}_k\| + h_C(-\mathbf{u}_k) = 0$, then $dist(A, B) = \|\mathbf{u}_k\|$. Otherwise, the C_{k+1} to be checked at the next iteration is obtained from the set of vertices $S(C_k^*) \cup \{s_C(-\mathbf{u}_k)\}$. The interest of using this support function for the generation of the vertices of C comes from the fact that $s_C(\mathbf{n})$ and $h_C(\mathbf{n})$ can both be computed in $O(n_A + n_B)$ time, *i.e.*,

$$\begin{aligned} h_C(\mathbf{n}) &= h_B(\mathbf{n}) + h_A(\mathbf{n}) \\ s_C(\mathbf{n}) &= s_B(\mathbf{n}) - s_A(\mathbf{n}) \end{aligned} \tag{7}$$

Each iteration is therefore performed in linear time in the total number of vertices and as only a few iterations are needed for the convergence, the distance algorithm is *quasi-linear* in the total number of vertices.

The overall structure of the algorithm also plays a important role in its efficiency:

- The algorithm relies exclusively on simple computations (dot products and vector additions). Moreover, the procedure used for the computation of $\mathbf{u}_k$ reuses many of the values already computed during the previous step. These values are stored and each iteration needs to perform only a few additional computations.

- An extra speedup is obtained by providing an initial estimation of $S(C_0)$ to the algorithm. This feature turns out to be particularly interesting when only

[4] In fact, for a given direction $\mathbf{n}$, this function defines a plane $\mathbf{x} \cdot \mathbf{n} = h_X(\mathbf{n})$, such that all the points of X lie on the same side of this plane.

small positional changes occur between two successive distance computations. In this case, the set $S(C_k)$ computed at the last iteration of the previous distance computation can be used for this initial estimation. While the closest point of C to the origin stays inside the convex hull of this set, only one iteration will be needed to compute the new distance. Whenever changes occur, a couple of iterations will be generally sufficient to update the new sets of points and compute the distance.

5.4 Collision detection

Let $\mathbf{x}_A$ and $\mathbf{x}_B$ denote the closest points between two convex objects A and B. Their distance is then given by $d = \|\mathbf{x}_B - \mathbf{x}_A\|$. If an incremental displacement $(\Delta\mathbf{p}, \Delta\mathbf{r})$ is applied to A, it can be shown [Faverjon&Tournassoud,1987] that the distance variation Δd can be expressed as

$$\Delta d = -\mathbf{n} \cdot \Delta\mathbf{x}_A \tag{8}$$

where $\mathbf{n} = (\mathbf{x}_B - \mathbf{x}_A)/d$ and $\Delta\mathbf{x}_A$ is the positional displacement of the point $\mathbf{x}_A$ due to the displacement $(\Delta\mathbf{p}, \Delta\mathbf{r})$. For technical reasons (the distance computation algorithm returns reliable information only when the distance between objects is positive), we define the objects A and B to be *in contact* whenever $d < \epsilon$. The constant ϵ is a small positive distance that is imperceptible to the human eye, but keeps the mathematics of collision computation well behaved.

Clearly, a positive Δd indicates that the motion causes the objects to be separated further apart. Even if Δd is negative, there is no danger of collision as long as $|\Delta d| < (d - \epsilon)$. Otherwise, the intended incremental motion $(\Delta\mathbf{p}, \Delta\mathbf{r})$ will cause a collision and must thus be modified to apply only the allowable portion of the motion, *i.e.*, to stop the offending motion in a non-penetrating contact configuration. The allowed fraction of the motion is given by the *contact coefficient*

$$t = \frac{-(d - \epsilon)}{\Delta d} \tag{9}$$

It should be noted that this distance variation computation is only valid for *strictly convex* sets of points[5]. Consequently, special steps are needed to handle

[5]Strictly convex sets exhibit a continuous tangent along the surface.

situations where the *nearest point* on (the surface of) either object crosses a local surface tangent discontinuity. In practical terms, this corresponds to a sudden dramatic shift of the nearest point along the object's surface, such as during a *face/face* to *edge/face* contact transition. We will address this problem in detail in Section 5.7.

5.5 Contact type determination

So far we are able to detect impending collisions and stop the offending motion precisely in contact. Once in contact, the motion computation module of Section 6 *filters*[6] the operator's motion in such a way as to aid her in maintaining contacts, following environment features, *etc.* This module thus requires precise and noise-tolerant information about the nature of the current contact, *i.e.*, the *contact type* and the *contact feature centroids* for both contacting objects. The following paragraphs outline the manner in which this information is obtained.

The collision detection algorithm described above (Section 5.4) returns the following information:

- the *contact coefficient* t, where $t \in [0..1]$
- the two *nearest points*, $\mathbf{p}_1$ and $\mathbf{p}_2$, on the surfaces of the two objects
- the two *contact features*, f_1 and f_2, where $f_i \in \{vertex, edge, face\}$

If a new contact occurred during the last incremental motion, then $t < 1$ and $|\mathbf{p}_1 - \mathbf{p}_2| = \epsilon$. Moreover, the pair of the returned contact features identify the contact type (*e.g.*, *vertex/face*, *edge/face*, *etc.*) and it seems that we have all the information about the contact that we need.

However, the nearest points returned by the distance estimator may not necessarily correspond to the contact feature centroids. More importantly, as contacting objects slide and pivot with respect to each other, small numerical errors in computing their successive locations (Section 6) accumulate and cause small misalignments of contacting features. These errors are negligible on the scale of the task world parameters, but are sufficient to affect the mathematics of the distance estimator.

[6] For lack of a better term, we use *filter(ing)* here to denote a *transformation* of positional data. Six DOF data is examined and altered component-wise, rather than time-smoothed.

Thus, an incremental motion that was intended (*i.e.*, generated by the motion computation module of Section 6) to place two objects into an *edge/face* contact, may appear, due to small alignment errors, to the distance estimator as an *vertex/face* contact. Consequently, an additional step is necessary to correct for this "numerical noise". This is accomplished by establishing *tolerance bounds* on the relative orientation of pairs of contacting features and upgrading the contact to a *higher-order contact type* whenever the error lies within the tolerance interval. To improve the numerical stability of the following computational steps, the misaligned features are also physically adjusted in the simulator to remove the misalignment.

Finally, the exact *contact feature centroids* are computed for both contacting objects. Once the final contact features are known, this is a relatively trivial matter of a few vectorial operations on the internal polyhedral data structures of the respective objects.

5.6 Constraint information

As already mentioned, two types of collisions can occur in the system – *wanted* and *unwanted* collisions. Wanted collisions are those that the operator intended to achieve and will normally involve a part of the environment and the movable object. Unwanted collisions, on the other hand, are all other collisions. Because the slave (plus the held object, if any) is the only moving object in the environment, these collisions will normally involve a part of the slave robot accidentally coming into contact with some part of the environment (obstacle).

Corresponding to the two types of collisions we will define two lists of object pairs (wanted and unwanted *collision list*). As we saw in Section 4.7, this information must be supplied to the system either by the user or a task description module prior to the execution of the task. At each simulation step, while the task is in progress, the collision detection module then checks both lists for possible new or persistent contacts. In the case of an unwanted collision, the system refuses to perform the offending motion that would cause the collision and alerts the operator by "freezing" the motion of the master arm and any other means necessary to unambiguously communicate the problem to the operator (*e.g.*, sound, altering display, console messages, *etc.*). The operator can then adjust her intended motion to avoid the

collision or adopt a different strategy to accomplish the same task. Note that this feature in a sense offers a rudimentary *collision avoidance* facility, where motion adjustment and/or replanning are left to the operator.

In the case of a wanted collision, the system stops the motion short of causing the collision, *i.e.*, the system allows the two objects to come into contact but not interpenetrate (see Section 5.4). Moreover, as we saw in Section 5.5, the system extracts the relevant information about the contact. In particular, it records what type of a geometric constraint this contact imposes on the motion of MO and adds this information to the list of already active constraints. This information is then used to *filter* commanded incremental motions to the master (and thus indirectly to the slave), such that the resulting (filtered) motion does not violate *any* of the currently active constraints on the motion of MO (Section 6).

A *constraint* can be defined as a pair of contacting features along with a set of parameters that uniquely define the geometry of the given constraint. This information will be needed both in the motion filtering process, where it will be used to define a filtering coordinate frame (Section 6), as well as in the command string generation process, where it will be used to define a task frame (Section 8). As we will see, the following three parameters suffice to uniquely describe the geometry of a constraint in all cases (*i.e.*, regardless of the types of contacting features)

- the vector $\mathbf{p}$ connecting the *slave wrist center* (where the commanded motions are applied) and the *contact point* (feature centroid, associated with the constraint)
- the *constraint normal* $\mathbf{n}$ (see Section 6.3.2 for the definition of constraint normal)
- *edge direction* $\mathbf{e}$, if the contact involves an edge

For convenience, all of the above vector quantities are computed *w.r.t.* the common global reference frame $\mathcal{F}_B$. Therefore, a constraint c_i can be encoded as the quintuple

$$c_i = \{f_1\ ,\ f_2\ ,\ {}^B\mathbf{p}\ ,\ {}^B\mathbf{n}\ ,\ {}^B\mathbf{e}\} \tag{10}$$

where f_1 and f_2 belong to the set $\{vertex, edge, face\}$ and correspond to the contact features of MO and the environment, respectively. The list of all (N) currently

active constraints can thus be encoded as

$$\mathcal{C} = \bigcup_{i=1}^{N} c_i \tag{11}$$

Depending on what types of motions the system allows and how the filtering process is carried out, not all of the above information may be needed in all cases. Therefore, for reasons of compactness and efficiency, an actual implementation may condense the information contained in $\mathcal{C}$ to optimize run-time performance.

5.7 Contact type transitions

Let $\mathcal{L}$ denote the *wanted collision list* of all object pairs $p = (O_1, O_2)$ which are currently being monitored for mutual collisions and let $\Delta\mathbf{d}$ denote the current commanded incremental displacement of the slave manipulator. Moreover, let O_1 in each pair be the movable object, *i.e.*, O_1 is rigidly attached to the slave, whereas O_2 belongs to the environment. The skeleton of the collision checking algorithm is as follows:

for each $p \in \mathcal{L}$

1. see if $\Delta\mathbf{d}$ causes the nearest point $\mathbf{p}_1 \in O_1$ to violate the ϵ envelope of O_2
2. if so, perform only the allowable fraction of the motion t, $t < 1$
3. otherwise, perform the entire motion, $t = 1$
4. recompute the distance $dist(O_1, O_2)$ for the next step

The final constraint on the current motion of the slave is then derived from the smallest t_i over all p_i in $\mathcal{L}$.

Observe that only the current nearest point $\mathbf{p}_1$ is being checked for penetration in step 1 above. Still, all is well as long as the nearest point travels slowly and continuously along the surface of O_1. However, if the nearest point changes significantly in a single simulation step (*e.g.*, from one edge to another), then the motion may seem acceptable based on the resulting motion of the old nearest point, but nevertheless cause penetration of O_2's ϵ envelope. The nearest point $\mathbf{p}_1'$ following the motion belongs to the penetrating portion of O_1 and corresponds to the deepest

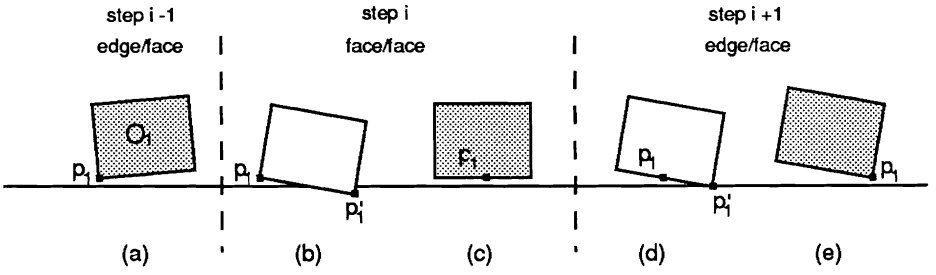

Figure 5: *edge/face* → *face/face* → *edge/face* contact type transition.

penetration point. Therefore, it is this point that the collision estimator should have monitored for contact instead of $\mathbf{p}_1$.

Figure 5 illustrates the side view of a typical discontinuity in the location of the nearest point on the movable object. The block in the figure is being pivoted about its bottom left edge in the clockwise direction and it is the operator's intent to tumble the block through the *face/face* contact into an *edge/face* contact, where the edge now is the bottom right edge. Suppose that an incremental motion in the $(i-1)^{th}$ step left the block as shown in Figure 5-a. Then, in the i^{th} step, the intended motion will be checked to ensure that $\mathbf{p}_1$ does not penetrate O_2's ϵ envelope. Since the operator's commanded motion has been filtered such as to leave the contact point fixed (see Section 6), it will pass the check and the motion will be applied in full. This may result in the configuration of Figure 5-b, which, of course, constitutes a collision.

Step 4 of the contact monitoring procedure above allows us to handle such situations. The call to the distance estimator will reveal that $dist(O_1, O_2) < \epsilon$ and that $\mathbf{p}_1' \neq \mathbf{p}_1$.[7] Having determined the new contact point, we now set the block back into its original position (Figure 5-a), set $\mathbf{p}_1 = \mathbf{p}_1'$ and repeat steps 1–4. This time, the motion will be found to be only partly realizable and only the corresponding fraction

[7]Because the results of the distance computation are reliable only when the distance between the two polyhedra is positive, we must perform an extra step of separating the objects along the direction of smallest translational distance, issue a call to the distance estimator while they are separated, and subsequently return them to their original (penetrating) locations.

t ($t < 1$) of $\Delta\mathbf{d}$ will be applied, bringing the block into a *face/face* contact. The post-processing realigning step of Section 5.5 will compute the new contact feature centroids for the two objects as shown in Figure 5-c. Assuming that the pivoting motion persists, the $(i+1)^{th}$ step will similarly produce the situation of Figure 5-d, where the contact point $\mathbf{p}_1$ again moves discontinuously to the right edge ($\mathbf{p}_1'$). As before, this is detected by the call to the distance estimator, the block is reset to its *face/face* configuration and the same motion is reapplied with $\mathbf{p}_1'$ serving as the contact point. Clearly, this motion is allowable and the block transitions to the *edge/face* contact of Figure 5-e.

This mechanism therefore allows us to transition between contact types smoothly, with no burden to the operator.

6 Computing Kinesthetic Feedback

6.1 Classification of allowable motions

A teleoperation system must provide a wide range of motions both in free space (while approaching/leaving the work area) and in contact with the surroundings (while performing the work). At the same time the allowed motions should be carefully partitioned and restricted to aid the operator in performing the type of motion intended. A natural way to simplify general motion (both for the operator and for the system) is to separate rotations and translations whenever possible. This is particularly crucial in contact motion, as the contact point is physically removed from the wrist center, where motion is commanded and rotations and translations are kinematically decoupled.[8] This separation gives rise to a remote compliance center and consequently introduces complex and potentially confusing coupling between rotational and translational parameters of the wrist and contact frames. The choice of elementary motions should strive to eliminate such coupling effects without compromising the flexibility and power of the system.

Another important consideration in deciding on the most convenient and effective set of motion modes is the class of tasks that the system is expected to handle. In view of the intended applications of our system (Section 4.10), the operator will need to be able to perform a relatively wide range of tasks. Representative examples are : accurate free-space motion, standard pick and place operations, basic exploratory procedures (*i.e.*, surface or feature following), simple assembly/disassembly tasks, *etc.*

Therefore, in view of the above considerations, we propose the following set of elementary classes of motions:

1. **Free Space Motion**
    - **general motion** (both rotations and translations)
    - **freeze position** (rotations + fixed position)
    - **freeze orientation** (translations + fixed orientation)

[8] Technically, this is only true for manipulators with Euler wrists, but most modern (single-chain) manipulator designs tend to separate rotational and translational degrees of freedom at the end-effector by serially connecting a positional linkage and a rotational wrist.

2. **Contact Motion**

   - **freeze** (no motion)
   - **slide** (translation along constraint features, fixed orientation)
   - **pivot** (rotational motion about contact point, fixed position)

3. **Pushing**

Given a set of elementary motion modes, the operator then specifies to the system which mode she currently desires. To minimize the burden on the operator, the motion mode selection information can be supplied to the system via a hand-held push-button device.

The following sections elaborate on each type of elementary class of motions.

6.2 Free space motion

In free space the system should offer the operator the maximum possible maneuverability. At the same time it should aid the operator preserve positional/orientational parameters that she wishes to keep constant during a significant portion of a manipulation task. For instance, if the operator has achieved the desired approach orientation, then the system should allow her to *freeze* (lock) it and subsequently concentrate on translational motion of the slave robot (and MO) only. Similarly, situations may arise (*e.g.*, screwing, valve adjusting), where the operator has positioned the slave end-effector and wishes to freeze the position and concentrate on grasping or turning the desired feature. Therefore, we provide three corresponding elementary free space modes of motion. One could proceed further and introduce single DOF motion modes restricting the operator's motion to translations along a single direction at a time or rotations about a single axis. However, we have decided against such facilities as they increase the burden on the operator of having to mentally keep track of some task-based coordinate frame in which these restrictions would be specified, all at a dubious benefit to the operator's ability to perform tasks more easily or more efficiently.

Therefore we feel that the above free space motions provide a reasonable compromise between convenience (for the operator) and functionality. Finally, in view of Eq.(2), the three motion modes are realized in a straightforward fashion as follows:

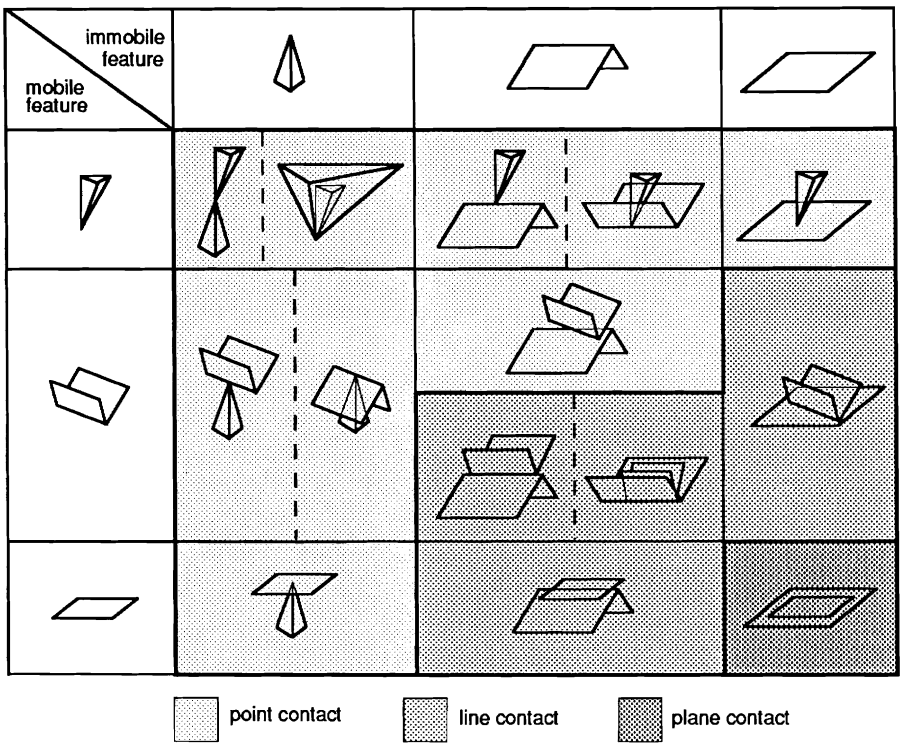

Figure 6: Types of polyhedral contacts.

- general motion: $\Delta\mathbf{d} = (\mathbf{t}, \mathbf{r})$
- freeze position: $\Delta\mathbf{d} = (\mathbf{0}, \mathbf{r})$
- freeze orientation: $\Delta\mathbf{d} = (\mathbf{t}, \mathbf{0})$

6.3 Contact motion

6.3.1 Types of contact

When the movable object (MO) is in contact with the (simulated) environment, its motion (and therefore the motion of the slave manipulator) is restricted, depending on the type of contact. Figure 6 lists the types of contacts that we will consider in this work. Let us emphasize again that we are concerned with rigid polyhedral contacts only. A few notes about Figure 6 are in order. It is easy to see that convex *vertex/vertex* and *vertex/edge* contacts are highly transient contact types and will

rarely occur in practice . However, as pointed out in [Sawada *et al.*,1989], the two types of contacts can be significant and persistent when one of the contacting features is concave. Following this work and recognizing that vertices and edges can be either convex or concave, we generalize the contacts involving these two features to include both cases. This is reflected in Figure 6 by juxtaposing the two cases, separating them with a vertical dashed line.

We will in the following sections have the occasion of referring to *adjacent*, as well as *high* or *low* order contacts. All of these terms are to be interpreted in view of Figure 6. We will define an adjacent contact to be one which can be reached in one contact change from the current state. Also, we will say that a contact c_i is higher (of higher order) than contact c_j, if c_i offers fewer remaining DOF of motion than c_j.

6.3.2 Constraint normals

In Section 5.6 we discussed the nature of the constraint information maintained by the graphical simulator and passed to the master's Cartesian level servo module. Recall that for each active constraint this information includes an associated *unit normal direction*. We now offer a convention to unambiguously define this constraint normal in each contact type.

We will let the constraint normal in each case be directed away from the environment contact feature and towards the movable object (MO), *i.e.*, the normal specifies the direction *against* which MO can *not* move. Referring to Figure 6, it seems natural to consider the geometry of both contacting features in determining the direction of this normal. Still, different conventions may prove to be equally reasonable and practical. We will choose to let the higher-order feature in each case dominate the choice and will break the ties in favor of the environment feature. The only exception to this rule will be the *edge/edge* point contact (see Figure 6), where the normal is most naturally defined by the cross-product of the two edge directions.

In keeping with the above convention, then, the constraint normal direction for a *face/face* planar contact is given by the face normal of the environment plane. Similarly, for the two line contacts involving only edges, as well as for the *vertex/vertex* point contact, the environment feature determines the normal. In all remaining

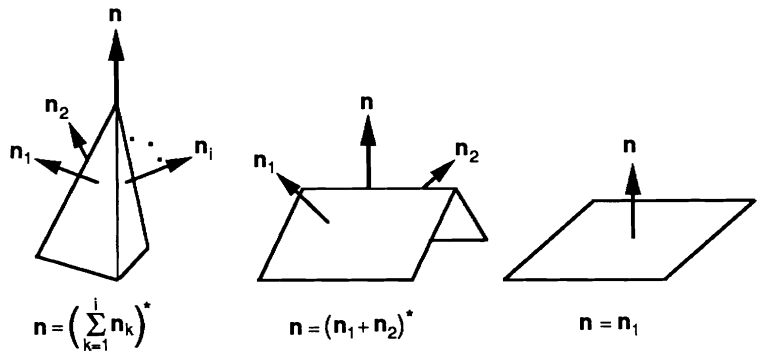

Figure 7: Constraint normals for the three types of polyhedral features.

contact types (except the already mentioned *edge/edge* contact), the higher-order feature (regardless of which object it belongs to) determines the axis (but not necessarily the direction) of the constraint normal.

Finally, the normals for each of the three elementary polyhedral features are defined in a straightforward fashion as illustrated in Figure 7.[9] Note that this definition assumes that all face normals in our polyhedral models are outward pointing.

6.3.3 Kinesthetic feedback and graphics

As we saw in Section 5, the graphical simulator maintains the current constraint information on the motion of the movable object. Thus, following the initial motion that caused a particular contact (and caused the new constraint to be reflected in the constraint information) the intended (*i.e.*, operator specified) motion of the movable object (MO) can be checked against the active constraints and appropriately modified. Therefore, in the context of a purely kinematic simulation, we propose to provide pseudo *kinesthetic feedback* to the operator by filtering the intended motion of MO, bringing it into compliance with the existing geometric constraints. By applying this filtered motion to the master manipulator as well (*i.e.*, backdriving the master manipulator appropriately), the operator holding the master *feels* these constraints as resistance to motion.

[9]The asterisk (*) in Figure 7 denotes that the corresponding vector is of *unit magnitude*.

The filtering must be relatively simple, intuitively natural to the operator, fast to compute and as general as possible, given the above requirements. Simplicity and computational speed are necessitated by the requirement that the kinesthetic feedback be provided to the operator in real time.

6.3.4 Contact motion modes — overview

As indicated in Section 6.1, we propose three basic types of contact motion. For the case of fine precision motions, where even slight unintended changes in position/orientation of MO caused by an impact (contact with a new constraint) are unacceptable, we provide the trivial *freeze* mode (no motion at all). In other words, all commanded motion of MO following a new contact is ignored until the operator selects a higher-order contact motion mode. Two such modes are provided.

In *slide* mode, the operator can slide MO along the constraining feature(s) (surfaces, edges) in the permissible directions (*i.e.*, the directions not violating *any* of the constraints). The orientation of MO remains fixed for the duration of motion in this mode. The system attempts to help the operator maintain contact with the environment but will allow the operator to break the contact if she clearly indicates such intent. A crucial feature of the way we propose to handle contact motion is to require *decisive* actions on the part of the operator to transition to a lower-level contact. This aids the operator in preserving high-order contacts (which are presumed preferred), while still allowing her to transition to an arbitrary adjacent contact. We will analyze this class of motions in the case of a single constraint, as well as in a situation where multiple constraints are acting on MO simultaneously.

Alternatively, the operator can adjust the orientation of MO or transition between adjacent contacts by rotating or pivoting about the contact point (*pivot* mode). In this mode the contact point is not allowed to translate (slide) along or depart from the constraint feature. As the contact type (between MO and the environment) changes, the contact point moves on the surface of MO and with it the pivoting point about which rotational motions are computed. This allows a variety of reorienting and contact changing motions of MO. Again, motion analysis will be performed on the commanded displacements so as to aid the operator perform the desired changes of orientation. We will provide a restricted version of this motion

modality to the operator also in situations where multiple constraints are restricting the motion of MO.

In the following sections we detail the proposed approach to contact motion analysis in free space as well as in contact.

6.3.5 'Freeze' mode

This trivial mode ($\Delta\mathbf{d} = (\mathbf{0},\mathbf{0})$) is included solely to prevent unwanted slippage and twists of MO *w.r.t.* the environment upon the initial (or new) contact. This mode is thus the default contact mode, entered automatically when a new contact is detected.

6.3.6 'Slide' mode — single contact

In the case of a single contact, the constraint information, as defined in Section 5.6, specifies the unit constraint normal ${}^B\mathbf{n}$. Given the desired motion of the slave wrist $\left(\Delta^B\mathbf{d} = \left({}^B\mathbf{t}, {}^B\mathbf{r}\right)\right)$, we compute the corresponding allowable subset of translational motion $\Delta^B\mathbf{d}'$ as follows[10]

$$\Delta^B\mathbf{d}' = ({}^B\mathbf{t}' \ , \ \mathbf{0}) \tag{12}$$

where

$$\mathbf{t}' = \begin{cases} \mathbf{t} - (\mathbf{t}\cdot\mathbf{n})\mathbf{n} & , \text{ if } (\mathbf{t}\cdot\mathbf{n}) < \epsilon \\ \mathbf{t} & , \text{ otherwise} \end{cases} \tag{13}$$

Figure 8 illustrates a typical situation for single-contact sliding, where w.p. and c.p. denote the slave wrist center and the contact point, respectively. Note that choosing ϵ to be a positive value, the operation of Eq.(13) above will filter out not only the component of the commanded translation against the constraint normal $\mathbf{n}_i$, but also the component along $\mathbf{n}_i$ (*i.e.*, away from the contact) if its magnitude is smaller than ϵ (Figure 8). This, in effect, provides an illusion of *contact surface tension*, *i.e.*, with a proper choice of ϵ the operator is forced to exert a decisive, deliberate pull away from the contact in order to break it.

[10]Note that the translational displacement of MO is the same as the commanded translational displacement of the slave wrist, despite the offset between the two.

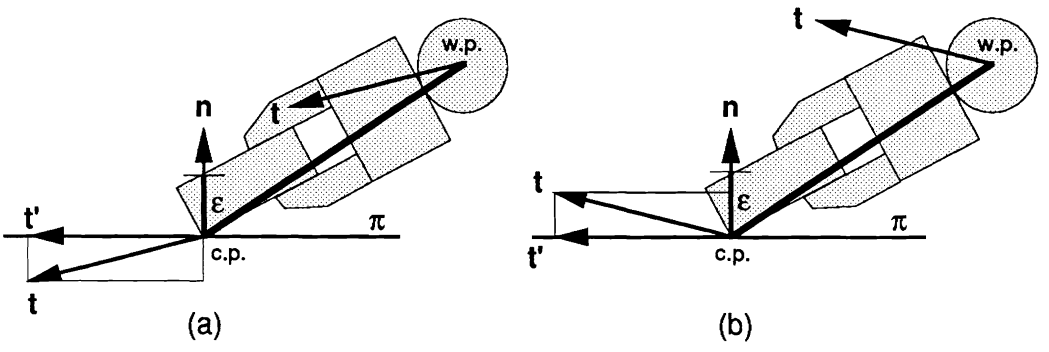

Figure 8: Single-contact sliding.

6.3.7 'Slide' mode — multiple contacts

In case of multiple contacts, the constraint information contains a list of constraint normals ${}^B\mathbf{n}_i$, which are currently restricting the motion of the movable object (MO). In general, these constraint normals will *not* be mutually orthogonal and we must approach the filtering process with caution. We will in the following development refer to a *constrained direction* as the negative of the corresponding constraint normal $\mathbf{n}_i$, as defined in Figure 7, and denote it as $\tilde{\mathbf{n}}_i$.

Figure 9 illustrates a typical situation, where MO is in contact with two non-orthogonal constraints.[11] In this situation the operator should be able to slide MO along both constraining surfaces, break either contact and slide along the other contact's environment feature (surface), or even break both contacts and transition to free-space motion.

Again we will assume that the commanded incremental slave wrist motion is given as $\Delta^B\mathbf{d} = \left({}^B\mathbf{t}, {}^B\mathbf{r}\right)$. The analysis of the multi-contact case centers on identifying *the primary constrained direction* $\tilde{\mathbf{n}}_p$, *i.e.*, the one which is "closest to" the desired translational motion $\mathbf{t}$. The measure of closeness is the projection of $\mathbf{t}$ along a unit direction $\tilde{\mathbf{n}}_i$. Given this closest $\tilde{\mathbf{n}}_i$ (*i.e.*, $\tilde{\mathbf{n}}_p$), we then construct an orthogonal filtering frame $\mathcal{F}_F$, such that $\tilde{\mathbf{n}}_p$ is one of its axes, and the cross product with any other constrained direction $\tilde{\mathbf{n}}_j$ gives its second orthogonal axis. This choice of

[11] A two-constraint example has been chosen for illustrative convenience. The discussion and results of this section apply to higher-multiplicity contacts as well.

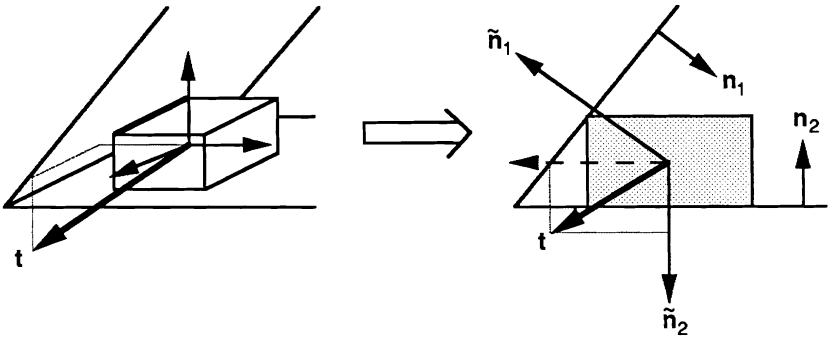

Figure 9: Multiple-contact sliding.

a filtering coordinate frame is adopted because a commanded translational motion $\mathbf{t}$ in a multi-constraint case will normally give rise to a sliding motion along the constraint feature, whose associated constrained direction is closest to $\mathbf{t}$.

Having constructed the filtering frame, we then express both the commanded motion ${}^{B}\mathbf{t}$ and the constrained directions ${}^{B}\tilde{\mathbf{n}}_k$ in this frame (*i.e.*, ${}^{F}\mathbf{t}, {}^{F}\tilde{\mathbf{n}}_k$) and filter the commanded slave wrist motion accordingly. The sequence of steps below formalizes the filtering procedure and supplies the necessary details.

1. for all $c_i \in \mathcal{C}$, compute the projections $p_i = \left({}^{B}\mathbf{t} \cdot {}^{B}\tilde{\mathbf{n}}_i\right)$
2. let ${}^{B}\tilde{\mathbf{n}}_p = {}^{B}\tilde{\mathbf{n}}_i$, for which p_i is most positive over $\mathcal{C}$
3. construct the filtering frame $\mathcal{F}_F$,

$$\mathcal{F}_F = \left\{\left(\left({}^{B}\tilde{\mathbf{n}}_p \times {}^{B}\tilde{\mathbf{n}}_j\right) \times {}^{B}\tilde{\mathbf{n}}_p\right)^{*} \ , \ \left({}^{B}\tilde{\mathbf{n}}_p \times {}^{B}\tilde{\mathbf{n}}_j\right)^{*} \ , \ {}^{B}\tilde{\mathbf{n}}_p\right\}$$

   where $c_j \in \{\mathcal{C} - \{c_p\}\}$, *i.e.*, ${}^{B}\tilde{\mathbf{n}}_j \neq {}^{B}\tilde{\mathbf{n}}_p$;
   construct the rotational matrix ${}^{B}\mathbf{R}_F$ from $\mathcal{F}_F$ (see Section A.2)
4. map ${}^{B}\mathbf{t}$ into $\mathcal{F}_F$, *i.e.*, ${}^{F}\mathbf{t} = \left({}^{B}\mathbf{R}_F\right)^{-1} * {}^{B}\mathbf{t}$
5. for each $c \in \mathcal{C}$, filter ${}^{F}\mathbf{t}$ *w.r.t.* c,
   - map ${}^{B}\tilde{\mathbf{n}}$ into $\mathcal{F}_F$, *i.e.*, ${}^{F}\tilde{\mathbf{n}} = \left({}^{B}\mathbf{R}_F\right)^{-1} * {}^{B}\tilde{\mathbf{n}}$
   - filter each component of ${}^{F}\mathbf{t}$ in turn, *i.e.*,
     $\Lambda\left({}^{F}\mathbf{t}, {}^{F}\tilde{\mathbf{n}}, x\right)$, $\Lambda\left({}^{F}\mathbf{t}, {}^{F}\tilde{\mathbf{n}}, y\right)$, $\Lambda\left({}^{F}\mathbf{t}, {}^{F}\tilde{\mathbf{n}}, z\right)$

6. map filtered ${}^F\mathbf{t}$ back into $\mathcal{F}_B$, *i.e.*, ${}^B\mathbf{t}' = {}^B\mathbf{R}_F * {}^F\mathbf{t}$

Procedure 1: Multi-constraint sliding motion filter.

The core of the filtering process is Step 5, where each constrained direction $\tilde{\mathbf{n}}$ is in turn rotated into the filtering frame and the components of the commanded motion are filtered according to the Λ operator. This operator is defined as follows

$$\Lambda(\mathbf{t}, \tilde{\mathbf{n}}, x) : t_x = \begin{cases} t_x & , \text{ if } (\tilde{n}_x = 0) \text{ or } (t_x \cdot \text{sgn}(\tilde{n}_x)) \leq -\epsilon \\ 0 & , \text{ otherwise} \end{cases} \tag{14}$$

Therefore, any constrained components of the commanded motion are zeroed. Also, small components away from the constrained orthogonal directions are zeroed as well, providing a sense of surface tension as in the single-contact case above.[12] Having performed the filtering operation on ${}^F\mathbf{t}$, we then rotate the filtered commanded displacement back into the reference frame (Step 6) and assemble the final filtered motion of the slave wrist as $\Delta^B\mathbf{d}' = \left({}^B\mathbf{t}', \mathbf{0}\right)$.

Observe that a filtering frame is constructed even in the case where the original commanded motion does not violate any constraints, *i.e.*, when all p_i in Step 1 are negative. This is done so that the filtering of small components away from the constraint features in Step 5 (which must be done in this case as well) is performed in an orthogonal frame. The requirement that filtering be done only *w.r.t.* orthogonal axes is crucial.

Finally, for clarity, various optimizations of the above procedure have been omitted (in particular, in Step 5). Any implementation must consider these carefully.

6.3.8 'Pivot' mode — single contact

Computing the motion

As mentioned before, in this single contact mode the contact point is stuck in contact and can not be moved (*i.e.*, slid along a contact feature or pulled away from the contact). Only rotations of MO about the contact point are allowed. The class of allowed motions and the nature in which these motions are computed are intended to give the operator the feel of manipulating in a "sticky" environment, as

[12]The same ϵ value may be used both in single and multiple-contact situations.

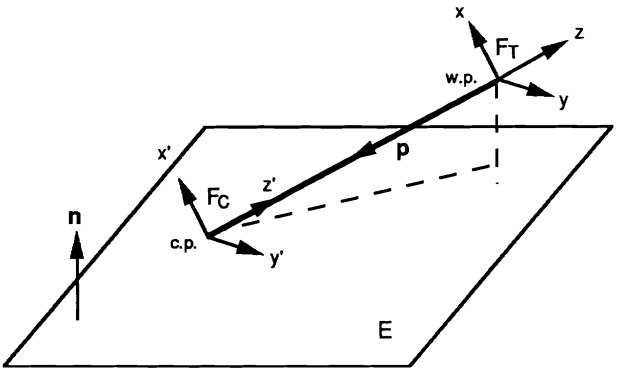

Figure 10: Tangential and contact frames.

well as allowing the operator to concern herself with the orientational parameters of MO alone, while keeping the contact point position fixed.

The input to the filtering module are the commanded (operator supplied) motion of the slave wrist ($\Delta^B\mathbf{d}$) and the current constraint information $\mathcal{C}$ (Section 5.6). Let the commanded motion be given as a displacement/RPY pair. Our task is to compute the rotational motion of the contact frame (centered at the contact point), based on the supplied slave wrist motion and subject to the above assumptions. Toward this aim we will define two coordinate frames (with the same orientation) as illustrated in Figure 10. In the figure, ${}^B\mathbf{n}$ is the constraint normal, the vector ${}^B\mathbf{p}$ denotes the (directed) distance between the slave wrist center point (w.p.) and the contact point (c.p.), and E labels the constraint feature (plane in this case). The first frame $\mathcal{F}_T$ (*tangential frame*) is defined such that its x-y plane is tangential to the surface of the sphere centered at c.p. and having radius $|\mathbf{p}|$. For convenience, we will define a second frame $\mathcal{F}_C$ (*contact frame*) with the same orientation as $\mathcal{F}_T$, but slid along the $\mathbf{p}$ vector, such that its origin coincides with the contact point, *i.e.*, [13]

$$\mathcal{F}_T = \mathcal{F}_C = \left\{ {}^B((\mathbf{p} \times \mathbf{n}) \times \mathbf{p})^* \, , \, {}^B(-\mathbf{p} \times \mathbf{n})^* \, , \, {}^B(-\mathbf{p})^* \right\} \tag{15}$$

The rotational matrix ${}^B\mathbf{R}_T$, specifying the orientation of the frame $\mathcal{F}_T$ *w.r.t.* $\mathcal{F}_B$, is again derived directly from the above definition of the two frames (see Section A.2). In fact, since the orientations of the frames $\mathcal{F}_T$ and $\mathcal{F}_C$ are identical, we have

[13] If ($\mathbf{n} \parallel \mathbf{p}$), then any non-parallel vector $\mathbf{v}$ can be used instead of $\mathbf{n}$.

${}^B\mathbf{R}_T = {}^B\mathbf{R}_C$.

We will describe the (rotational) motion of the contact point in terms of the motion of the contact frame $\mathcal{F}_C$ due to the (operator supplied) motion of the wrist-based tangential frame $\mathcal{F}_T$. In an attempt to kinematically simulate the rotational motion of MO, whose contact point is stuck in contact, and at the same time minimize the complexity of motion analysis, we propose to compute the rotational motion of $\mathcal{F}_C$ as follows:

(a) rotational motion of w.p. about the z-axis of $\mathcal{F}_T$ corresponds directly to the rotational motion of c.p. about the z-axis of $\mathcal{F}_C$

(b) translational motion of w.p. (along the x-y plane of $\mathcal{F}_T$) is used to compute the remaining two orthogonal rotational displacements of $\mathcal{F}_C$

In (b), the rotational displacement of $\mathcal{F}_C$ (about its x and y axes) is approximated by considering the components of the commanded translational vector ${}^T\mathbf{t}$ (*i.e.*, ${}^B\mathbf{t}$ rotated into the $\mathcal{F}_T$ frame) projected onto the x, y axes of $\mathcal{F}_T$. For the case of computing the incremental rotation $\Delta\theta_y$ about the y-axis of $\mathcal{F}_C$, we have

$$\Delta\theta_y = \frac{{}^T t_x}{|\mathbf{p}|} \tag{16}$$

Figure 11 illustrates the situation.[14,15]

An important detail that must be noticed here is that the translational vector ${}^B\mathbf{t}$ will only cause pivoting (rotation about c.p.) if it lies below the x-y plane of the *tangential frame* $\mathcal{F}_T$, *i.e.*, if

$$\left({}^B\mathbf{t} \cdot {}^B\mathbf{p}\right) \geq 0 \tag{17}$$

Therefore, the RPY rotation of $\mathcal{F}_C$ due to the (rotational and translational) motion of $\mathcal{F}_T$, under the assumption of stiction, is

$${}^C\mathbf{r}' = {}^T\mathbf{r}' = \left(-\frac{{}^T t_y}{|\mathbf{p}|}\ ,\ \frac{{}^T t_x}{|\mathbf{p}|}\ ,\ {}^T r_z \right) \tag{18}$$

[14]The y-axes of both $\mathcal{F}_T$ and $\mathcal{F}_C$ frames are directed out of the page.

[15]Note that the approximation of equating the tangential projections of the displacement vector $\mathbf{t}$ with the corresponding great arc segments along the sphere surface is equivalent to assuming that $\sin(\Delta\theta) = \Delta\theta$, as $\sin(\Delta\theta) = \Delta\theta = {}^T t_x/|\mathbf{p}|$ in Figure 11. It is easy to verify that this approximation is quite good for $-\pi/6 < \Delta\theta < \pi/6$, which is more than sufficient for our purposes.

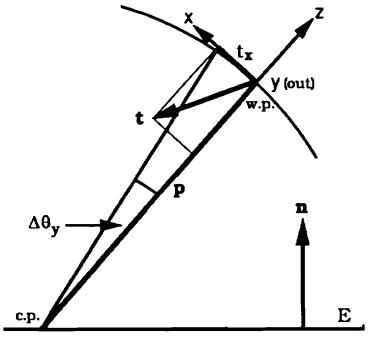

Figure 11: Computing $\Delta\theta_y$ of the contact frame.

The superscripts on the right hand side of the above equation indicate that the corresponding displacement and RPY parameters have been rotated into the $\mathcal{F}_T$ coordinates. See Appendix A for details.

The computed rotational motion of the contact point (and thus MO), as given by Eq.(18), is designed to provide a natural feel to the operator, as she is forced to introduce translational motion at the slave wrist to achieve rotational (pivoting) motion at the contact point. In the absence of a full dynamic model, the generated model is only approximate, of course, but nevertheless it has an intuitive basis and should feel natural to the operator.

Filtering

Having computed the rotational motion of the contact point based frame $\mathcal{F}_C$, we now filter this motion on the basis of the contact type. The filtering is done primarily to discard small (presumably unintended) rotational components and has the effect of biasing (the interpretation of) operator's motions towards higher order contact types. In the following paragraphs we will outline the filtering procedure.

In order to filter the rotational motion of $\mathcal{F}_C$, we will first define a contact point based *filtering frame* $\mathcal{F}_F$, which is particularly convenient for the given constraint type. We will then express the intended motion of the contact point in this frame ($\mathcal{F}_F$) and perform the filtering *w.r.t.* its coordinates. In each case the filtering frame will be constructed in terms of the geometric parameters supplied by the constraint information, *i.e.*, the constraint normal ($^B\mathbf{n}$), wrist-to-contact vector ($^B\mathbf{p}$), and

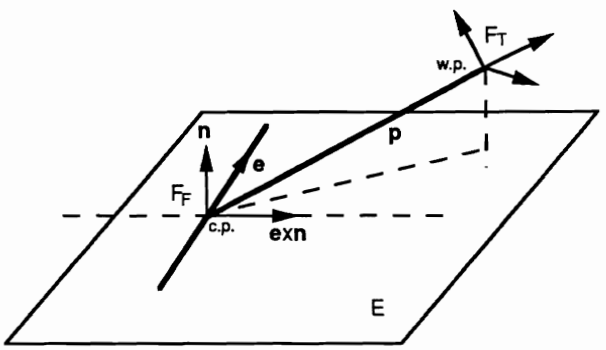

Figure 12: Single-contact pivoting — line contact.

the edge direction (${}^B\mathbf{e}$) (see Section 5.6). The input motion of the collision point $\Delta^C\mathbf{d}' = (\mathbf{0}, {}^C\mathbf{r}')$ is as computed in Eq.(18) above.

(a) Point Contacts: A filtering frame need not be specified in this case as all three orthogonal rotations are permissible in all point contacts (see Figure 6). Therefore, no filtering is necessary.

(b) Line Contacts: A line contact always involves an edge (at least one, see Figure 6), and it is this edge direction (${}^B\mathbf{e}$), together with the constraint normal (${}^B\mathbf{n}$), that defines the most convenient filtering frame, *i.e.*,

$$\mathcal{F}_F = \left\{ {}^B(\mathbf{e} \times \mathbf{n}) \, , \, {}^B\mathbf{e} \, , \, {}^B\mathbf{n} \right\} \tag{19}$$

where ${}^B\mathbf{e}$ and ${}^B\mathbf{n}$ are assumed to be of unit magnitude. The specification of the rotational matrix ${}^B\mathbf{R}_F$ follows immediately (see Section A.2). Figure 12 illustrates the case of an *edge/face* line contact.

Filtering of the contact point motion $\Delta^C\mathbf{d}'$ can now be achieved as a two-stage process:

1. map the motion (RPY rotation) of the contact point from $\mathcal{F}_C$ (${}^C\mathbf{r}'$) into $\mathcal{F}_F$ (${}^F\mathbf{r}'$), using ${}^C\mathbf{R}_F = \left({}^B\mathbf{R}_C\right)^{-1} * {}^B\mathbf{R}_F$ (see Section A.3)

2. filter out small rotations about ${}^B(\mathbf{e} \times \mathbf{n})$ tending to destroy the edge contact, *i.e.*,

$$ {}^F\mathbf{r}'' = \left(\Upsilon({}^F r'_x) \, , \, {}^F r'_y \, , \, {}^F r'_z \right) \tag{20}$$

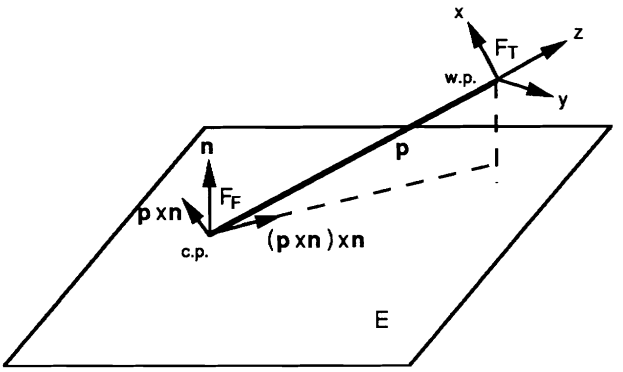

Figure 13: Single-contact pivoting — plane contact.

where the Υ operator is defined as follows[16]

$$\Upsilon(x) = \begin{cases} 0 & , \text{ if } |x| < \xi \\ x & , \text{ otherwise} \end{cases} \tag{21}$$

(c) Plane Contacts: The only representative of this class of contacts is the *face/face* contact (see Figure 6). Here, the filtering frame can be defined as follows

$$\mathcal{F}_F = \left\{ {}^B((\mathbf{p} \times \mathbf{n}) \times \mathbf{n})^* \, , \, {}^B(\mathbf{p} \times \mathbf{n})^* \, , \, {}^B\mathbf{n} \right\} \tag{22}$$

and the rotational matrix ${}^B\mathbf{R}_F$ can be constructed as before. Figure 13 illustrates the situation.

Again, a two-stage filtering procedure is employed. The given rotational motion of the contact point is mapped from $\mathcal{F}_C$ into $\mathcal{F}_F$ (via the rotational matrix ${}^C\mathbf{R}_F$). The second filtering stage in this case attempts to remove from ${}^F\mathbf{r}'$ small destabilizing rotations about the x and y-axes of the filtering frame, *i.e.*,

$${}^F\mathbf{r}'' = \left(\Upsilon({}^Fr'_x) \, , \, \Upsilon({}^Fr'_y) \, , \, {}^Fr'_z \right) \tag{23}$$

Postprocesing

[16]The Υ operator is a simple bidirectional threshold filter zeroing out rotations whose magnitude is smaller than ξ ($\xi > 0$). A good candidate value for ξ may be a third or even a half of the maximum magnitude of an incremental rotational displacement normally experienced by the system. This forces the operator to indicate a decisive rotation about the edge in order to break the edge contact.

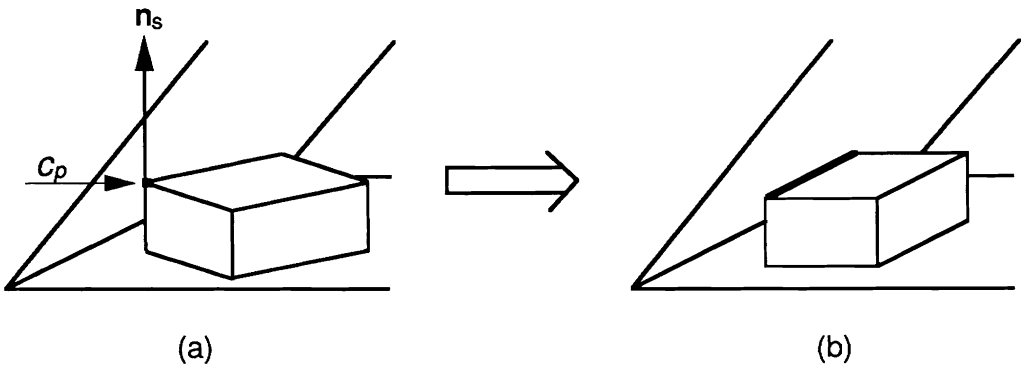

Figure 14: Multiple-contact pivoting.

Having computed the filtered motion of the pivoting contact point, we must now produce the corresponding motion of the slave wrist in the reference ($\mathcal{F}_B$) coordinates, as this is the motion ultimately commanded to the slave manipulator. This is accomplished by mapping the filtered contact point motion $\Delta^F\mathbf{d}'' = (\mathbf{0}, {}^F\mathbf{r}'')$ into $\mathcal{F}_B$ coordinates $\Delta^B\mathbf{d}''$ (see Section A.3) and computing the corresponding $\mathcal{F}_B$ displacement of the slave wrist as described in Section A.4.

6.3.9 'Pivot' mode — multiple contacts

In this section we extend the results of Section 6.3.8 to accommodate a restricted, but useful subset of multiple-constraint pivoting motions. The restrictions are imposed both to aid the operator in performing simple and intuitive multi-contact rotations, as well as to keep the geometrical and numerical complexity of the motion analysis low.

A typical situation that this motion mode is intended to address is one where the operator has brought the movable object into a multiple contact and wishes to align MO *w.r.t.* the environment so as to obtain a higher order (*i.e.*, more stable) contact type. Figure 14-a illustrates an example, where MO has been slid along a surface (*face/face* contact) against a wall (*vertex/face* contact). This mode will allow the operator to rotate the object into a stable configuration *w.r.t.* the environment (*i.e.*, *edge/face* wall contact, Figure 14-b) and align MO for subsequent sliding along either or both of the constraining surfaces.

It is clear, that in view of the intended applications of this motion mode, the only practical situations will involve two constraints. Also, we will assume that realigning motions either preserve or raise the order of existing contacts. Finally, as any pivoting multi-constraint motion will involve sliding of the moving object along one of the constraints, we will require that one of the contacts be a *face/face* contact.

While the imposed conditions may seem restrictive, the allowed motions still span a sizable set of useful realignment motions that may be needed in a practical application. For instance, most two-constraint situations will arise by sliding the movable object against a second constraint, where the single-constraint sliding motion will be performed in a *face/face* contact state for obvious reasons of convenience and stability. Similarly, upon encountering a second constraint, the most likely subsequent motion (if any) is one where the object is pivoted about this new contact into a higher order multiple contact state.

In order to compute the allowed motion of MO in a two-contact situation, we will again make use of the notion of a *primary constraint*, and label the two contacts as *primary* (c_p) and *secondary* (c_s) contact. By convention, we will refer to the mandatory *face/face* contact as the secondary contact. The motion of MO will then be computed as a pure rotation about the contact point associated with the primary contact, and filtered such that it will not violate the secondary constraint. Clearly, if any rotation is to take place, the primary contact must be of a lower order (*e.g.*, *vertex/face*, *edge/face*, *face/edge*, *etc*) than the secondary contact. Moreover, if the primary constraint forms a line contact (see Figure 6), then motion will only be possible if the corresponding edge direction is parallel to the secondary contact normal $\mathbf{n}_s$ (see Figure 14).

Once again, let the original commanded motion of the slave wrist be given by $\Delta^B\mathbf{d} = ({}^B\mathbf{t}, {}^B\mathbf{r})$. Assuming that the above set of conditions is satisfied, we identify the primary constraint c_p and compute rotational motion ${}^C\mathbf{r}'$ about its associated contact point as in Section 6.3.8 (Eq.(18)). This contact-frame based RPY rotation must then be filtered so as to retain only the rotation about the axis parallel to the normal of the secondary constraint. We therefore define a filtering frame $\mathcal{F}_F$, such

that one of its axes (*e.g.*, z) coincides with this normal direction, *i.e.*,

$$\mathcal{F}_F = \left\{ {}^B((\mathbf{n}_p \times \mathbf{n}_s) \times \mathbf{n}_s)^* \, , \, {}^B(\mathbf{n}_p \times \mathbf{n}_s)^* \, , \, {}^B\mathbf{n}_s \right\} \tag{24}$$

and map the rotation ${}^C\mathbf{r}'$ into this frame to obtain ${}^F\mathbf{r}'$ (see Section A.3). The filtered rotation is then obtained trivially as

$${}^F\mathbf{r}'' = \left(0, 0, {}^F r_z'\right) \tag{25}$$

The remaining task is to compute the corresponding motion $\Delta^B\mathbf{d}'$ of the slave wrist in the reference frame coordinates. This is accomplished in a straightforward fashion as described at the end of the previous section.

6.4 Pushing

6.4.1 Single-contact pushing

It has been established that pushing motions are difficult to analyze and predict accurately [Peshkin&Sanderson,1987], [Mason,1985], [Mason,1986]. This is due primarily to the fact that the motion of a pushed object depends critically on the complex interaction between the microscopic features of the two sliding surfaces. This in turn accounts for continuously changing frictional properties of the sliding contact, making reliable predictions of the resulting motions impossible without a detailed knowledge of the surface textures and the distribution of the support forces.

In order to facilitate rudimentary pushing operations and yet generate instructions which can be executed successfully and reliably under the slave's local supervision, we provide a simple pushing mode, where the operator can indicate to the system that she wishes to push an object through a certain distance along a *straight-line* trajectory. We require that the object to be pushed be in a planar (*face/face*) contact with some supporting surface and that the task information (Section 4.7) indicate that this object is in fact *pushable*. We also require that the slave establish a planar contact with the *pushed object* (PO). The requirements of a straight-line pushing motion and a planar *pushing contact* (between PO and the slave) minimize the possibility of slippage along the pushing contact or unexpected twists of the pushed object in the actual environment.

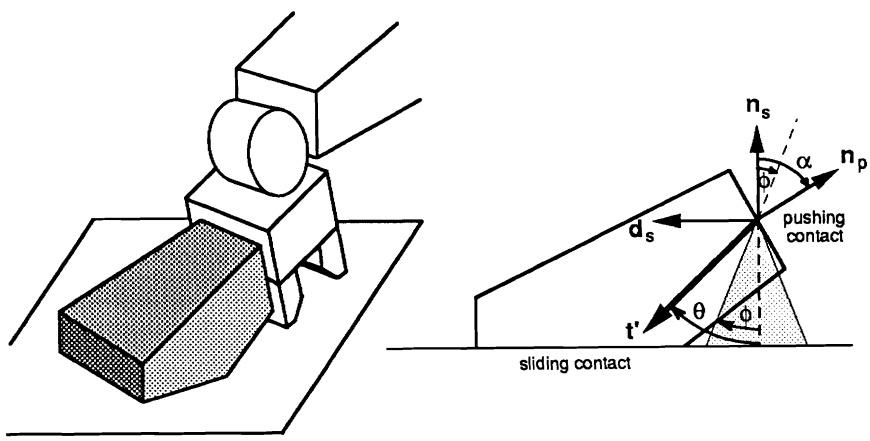

Figure 15: Single-contact pushing.

Another requirement aimed at avoiding slippage along the pushing contact is that the pushing contact plane have a "reasonable" orientation *w.r.t.* the sliding surface. We quantify this condition by introducing a *pushing frame*

$$\mathcal{F}_P = \left\{ {}^B((\mathbf{n}_p \times \mathbf{n}_s) \times \mathbf{n}_s)^* \, , \, {}^B(\mathbf{n}_p \times \mathbf{n}_s)^* \, , \, {}^B\mathbf{n}_s \right\} \tag{26}$$

centered at the contact point associated with the pushing contact, and requiring that the pushing and sliding contact normals ($\mathbf{n}_p$ and $\mathbf{n}_s$) form a sufficiently large angle α, so as to prevent slippage (see Figure 15)[17]:

$$\alpha = \arccos(\mathbf{n}_p \cdot \mathbf{n}_s) > \alpha_{\min} \tag{27}$$

Moreover, in order to give the operator a sense of the frictional effects during pushing, we filter the operator's motion such that no sliding motion occurs, unless the pushing direction (${}^B\mathbf{t}$) lies outside of the *friction cone* [Mason,1984]. The angle of the friction cone is given by $\phi = \arctan(\mu)$, where μ is the frictional coefficient of the sliding contact (see Figure 15).[18] Likewise, no sliding motion should be generated

[17]The vector labeled $\mathbf{t}'$ in the figure is the projection of the commanded translation vector ${}^B\mathbf{t}$ onto the x-z plane of $\mathcal{F}_P$.

[18]Technically, the apex of the friction cone should be located at the center of mass of the pushed object. Within the context of a non-dynamic simulation, we use the approximation of locating it at the pushing contact centroid.

unless the intended displacement vector ${}^B\mathbf{t}$ lies below and has a positive component along the *sliding direction* $\mathbf{d}_s$ (see Figure 15). Given a commanded motion $\Delta^B\mathbf{d}$ of the slave wrist, we can therefore compute the straight-line sliding motion of the pushed object as follows

$$\Delta^B\mathbf{d}' = \left({}^B\mathbf{t}' , \mathbf{0}\right) \tag{28}$$

where

$${}^B\mathbf{t}' = \begin{cases} {}^Pt_x & , \text{ if } (\phi < \theta < \frac{\pi}{2}) \\ \mathbf{0} & , \text{ otherwise} \end{cases} \tag{29}$$

In Eq.(29), ${}^P\mathbf{t}$ denotes the operator-supplied translational displacement (${}^B\mathbf{t}$) rotated into the pushing frame $\mathcal{F}_P$ (see Section A.3), ${}^Pt_x$ is the component of ${}^B\mathbf{t}$ along the sliding direction ${}^B\mathbf{d}_s$, and $\theta = \arctan({}^Pt_x, -{}^Pt_z)$ (see Figure 15). Moreover, in light of our use of the friction cone and Figure 15, we may set $\alpha_{\min} = \phi$.

In order for pushing motion to take place, the operator must first establish a planar contact with some environment object. We propose that the operator signal her intent to push the object by exerting a significant (and therefore easily identifiable) force against it. If this object is identified as pushable, the system then enters the *pushing mode*. In this mode, the graphical simulator rigidly attaches the pushed object to the slave at the point of pushing contact and filters commanded slave wrist motions so as to move in a straight line along the sliding surface (Eq. 29). Similarly, a decisive pull away from the pushing contact can be made to terminate the pushing mode.

Whereas every precaution has been taken to ensure that pushing motion commands generated at the operator's station are simple and easily executable by the slave, things can still go wrong. In particular, as the operator's station relies on a kinematic simulation of the slave world, error conditions such as the pushed object tipping over in the remote world can not be predicted and detected ahead of time. Avoiding such situations is thus left to the operator who can draw on her approximate knowledge of the relevant dynamic parameters or simply on her intuition in choosing a reasonable pushing contact.

6.4.2 Multiple-contact pushing

In order to enhance the versatility of the system, we again extend the single-constraint pushing motion mode to multi-contact situations. We envision this class of motions being used primarily to push and align an object with respect to two simultaneously active environmental constraints or to slide the object along an edge by pushing it. The analysis of such aligning and sliding pushing motions is therefore analogous to the analysis of double-constraint pivoting and sliding motion cases, respectively, with the movable object in this case being the pushed object together with the (rigidly attached) slave's end-effector or tool, if any.

7 Filtering Operator's Motions

In this section we describe a simple filtering procedure, which is applied to the positional data generated by the graphical simulator. The aim of this filtering stage is to smooth the observed slave trajectories and eliminate the undesired noise in the data.

The input to this module is the motion of the slave as computed in Section 6. As we have seen, various filtering steps have already been applied to the operator-generated motions so as to avoid object penetration and to force the operator to clearly indicate her intent to break (or reduce the order of) an existing contact. We will therefore assume that all the contact changes contained in the incoming data were intended by the operator and that there is no further need to detect and to eliminate transient changes of contact type.

During the same *contact state* (*i.e.*, the same set of elementary contacts), the information available from the graphical simulator is the trajectory of the slave end-effector along the unconstrained degrees of freedom defined by this contact state. This trajectory $\mathcal{T}$ is initially represented by the discrete set $\{\mathbf{p}_i : 0 \leq i \leq n\}$, where $\mathbf{p}_i = (\mathbf{t}_i, \mathbf{r}_i)$ describes the position and orientation of the frame $\mathcal{F}_{SW}$ (the frame attached to the slave wrist) at the i-th step of the simulation, and n is the number of discrete positional data acquired since the generation of the last command stream.[19]

This trajectory needs to be filtered for two reasons:

- The positional data will be inherently noisy due to the way in which this information is acquired, *i.e.*, operator-guided motions of the master. The filtering will eliminate small oscillations and deviations introduced by the operator and the sensor readings.

- More importantly, this trajectory has to be represented in a more compact fashion in order to reduce the number of motion commands to be sent to the remote slave.

Given the set describing $\mathcal{T}$ and two thresholds $\epsilon_{\mathbf{t}}$, $\epsilon_{\mathbf{r}}$, the filtering algorithm produces an approximate trajectory $\mathcal{T}_\epsilon$, composed of straight-line translations and

[19] Generation and partitioning of the command streams will be addressed in Section 8.

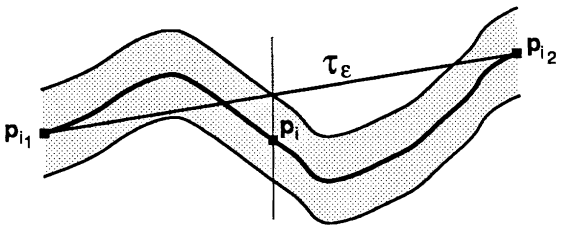

Figure 16: Trajectory filter — the "closeness test".

rotations of $\mathcal{F}_{SW}$, such that $\mathcal{T}_\epsilon$ stays inside the *space tunnel* defined by $\mathcal{T}$ and by the radii $\epsilon_{\mathbf{t}}$ and $\epsilon_{\mathbf{r}}$ (for the translational and rotational components, respectively).

The algorithm starts with the simplest approximation of $\mathcal{T}$, *i.e.*, the straight-line segment between the initial generalized position[20] $\mathbf{p}_0$ and the final one $\mathbf{p}_n$. If this approximation is "close enough" to $\mathcal{T}$, the algorithm simply returns this straight-line motion. Otherwise, an intermediate position $\mathbf{p}_j$ in $\mathcal{T}$ is added to the representation of $\mathcal{T}_\epsilon$ and the two line segments $Seg(\mathbf{p}_0, \mathbf{p}_j)$ and $Seg(\mathbf{p}_j, \mathbf{p}_n)$ are respectively checked against the corresponding portions $\{\mathbf{p}_i \ : \ 0 \leq i \leq j\}$ and $\{\mathbf{p}_i \ : \ j \leq i \leq n\}$ of the original trajectory $\mathcal{T}$. The same process is iteratively applied to each segment which needs to be refined and the algorithm converges to an approximation of $\mathcal{T}$ by a polygonal path including generally only a few intermediate points. Clearly, the larger the space tunnel defined by the radii $\epsilon_{\mathbf{t}}$ and $\epsilon_{\mathbf{r}}$ around $\mathcal{T}$, the fewer intermediate positions will be returned.

A line segment $Seg(\mathbf{p}_{i_1}, \mathbf{p}_{i_2})$ of $\mathcal{T}_\epsilon$ is considered to be a good approximation of the corresponding part of $\mathcal{T}$ defined by the set $\{\mathbf{p}_i \ : \ i_1 \leq i \leq i_2\}$, if all the $\mathbf{p}_i$ satisfy

$$\max\left(\frac{\|\mathbf{t} - \mathbf{t}_i\|}{\epsilon_{\mathbf{t}}}, \frac{\|\mathbf{r} - \mathbf{r}_i\|}{\epsilon_{\mathbf{r}}}\right) < 1 \tag{30}$$

where $\mathbf{t}$ (resp. $\mathbf{r}$) denotes the closest point on $Seg(\mathbf{t}_{i_1}, \mathbf{t}_{i_2})$ (resp. $Seg(\mathbf{r}_{i_1}, \mathbf{r}_{i_2})$) to $\mathbf{t}_i \in \mathcal{T}$ (resp. $\mathbf{r}_i$). Figure 16 illustrates the process.

Several approaches can be adopted for the selection of the intermediate position to be introduced after each non-terminal iteration of the algorithm. The point on $\mathcal{T}$ which is farthest from the current approximation $\mathcal{T}_\epsilon$ is in general a good candidate.

[20] We use the term *generalized position* to denote the 6-vector of positional and orientational parameters.

However, the drawback of this method is that it requires the computation of all distances between the points $\mathbf{p}_i \in \mathcal{T}$ and the line segment $Seg(\mathbf{p}_{i_1}, \mathbf{p}_{i_2})$, $i_1 < i < i_2$.

Consequently, a *binary subdivision* method offers a much more efficient approach: as soon as the algorithm finds a $\mathbf{p}_i$ which does not satisfy the "closeness test" of Eq.(30) for a given line segment of $\mathcal{T}_\epsilon$, it immediately introduces a new generalized position vector $\mathbf{p}_j$, where $j = \max\left(\frac{i_1+i_2}{2}, i\right)$, and cuts this segment into $Seg(\mathbf{p}_{i_1}, \mathbf{p}_j)$ and $Seg(\mathbf{p}_j, \mathbf{p}_{i_2})$.

Clearly, this method will sometimes produce a slightly larger number of intermediate positions than the former approach. Notice, however, that the algorithm will at each step at least halve the complexity of the problem.

This filtering procedure must be applied to all six components of the positional information in the case of a general motion in free space. However, both in the case of free-space motion with frozen orientation (resp. position), as well as in the case of sliding (resp. pivoting) contact motion, only positional (resp. orientational) motion parameters need to be filtered. Moreover, in each motion mode, only the components corresponding to the free degrees of freedom defined by the contact type need this filtering stage. For example, during a sliding motion along a plane whose normal coincides with the $\mathbf{z}$ axis of the reference frame $\mathcal{F}_R$, only the components of translational motion along ${}^R\mathbf{x}$ and ${}^R\mathbf{y}$ will need to be filtered.

8 Generating Symbolic Slave Commands

In this section we detail our approach to using the sequence of contact state changes (Section 6) and the filtered slave trajectory information within each contact state (Section 7) to extract a stream of symbolic commands to the remote slave. The resulting symbolic command language constructs are described in Appendix B.

The commands which will be issued to the slave by the system can be classified into two groups. The first group is composed of *low-level* commands, essentially encompassing *guarded* and *compliant* motions. These commands will be generated to execute simple tasks such as free-space navigation, pick and place operations, motion into contact with the environment, contour following, *etc.*

The *high-level* class of motions, on the other hand, contains more specific special-purpose operations such as tight tolerance part mating, fine-precision motions, camera repositioning *etc.* Even if the operator were able to perform a complex insertion in the simulated world, the observed sequences of contacts clearly would not be reproducible by the slave, due to the environment modeling errors. Therefore, such tasks can not be decomposed into elementary motions and must be executed autonomously by the slave under local sensory supervision. In this case, the graphical simulator need only identify that the operator wishes to perform a high-level operation (either by using the information provided by the task model or by interpreting the operator's motion information directly). The system then gathers the relevant parameters of the task and sends this information to the remote slave, where the information is used to instantiate a local special-purpose procedure.

A new stream of commands is issued after each addition or deletion of a new contact. However, there is also a maximum time (*e.g.*, on the order of the transmission delay) after which a new stream is automatically generated even if the same contact state persists. This is done to avoid increasing the delay and to prevent accumulation of the positional information to be processed.

In this section, we restrict our analysis to the generation of the low-level commands and discuss the algorithms used to transform the contact-state and positional information provided by the graphical simulator and by the kinesthetic feedback module in order to produce a stream of guarded and compliant motion commands to be executed by the slave.

8.1 Types of motion commands

An important issue that must be addressed when generating these commands results from the presence of uncertainties in the world model used by the graphical simulator. During free space motion, simple positioning commands will generally be sufficient to be executed safely by the slave. However, as soon as the task involves interactions between the robot and its environment, these discrepancies may cause a failure during the command execution. This problem has been studied extensively during the last decade [Mason,1981], [Whitney,1987] and various methods of using the forces and torques occurring during the contact motion to suitably adapt the robot's trajectory have been proposed. We will in our work make use of the hybrid force-position approach [Inoue,1971], [Paul,1976], [Raibert&Craig,1981], where the *free* directions of the motion are controlled in position (or velocity), while the directions *constrained* by the contacts are controlled in force. Contact motions will thus consist of two main types of commands: *guarded motions* and *compliant motions*. A guarded motion is generally used when approaching a surface to avoid excessive forces after the contact is established. A compliant motion is then required to move along one or more constrained surfaces while maintaining a given force (or torque) constraint in the directions normal to constraining surfaces.

The following section describes how the positioning and contact information provided by the graphical simulator can be translated into a stream of such hybrid control motions.

8.2 Task frame specification

In order to facilitate convenient specification of guarded and compliant motions of the slave manipulator, we will define a *task frame* $\mathcal{F}_T$, such that its position and orientation is closely related to the constraints imposed by the geometry of the current contacts. For each type of elementary contact, the task frame $\mathcal{F}_T = \{\mathbf{p}\,;\,\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z\}$ is defined in the following manner:

- Its origin $\mathbf{p}$ coincides with the centroid of the *contact feature* (see Section 5.5).
- $\mathbf{n}_z$ is aligned with the *constraint normal* (see Section 6.3.2).

- For the three types of contact where an edge is involved (see Figure 6), $\mathbf{n}_y$ is aligned with the direction of this edge. For the other cases, an arbitrary direction lying in the contact plane is chosen.

- $\mathbf{n}_x$ is obtained by $\mathbf{n}_y \times \mathbf{n}_z$.

More work needs to be done to identify the optimal choice of task frame coordinates for the case of multiple-constraint motions!

Whenever a new task frame needs to be specified, an assignment command is sent to the slave. This command must specify the 3-dimensional vectors $\mathbf{p}, \mathbf{n}_x, \mathbf{n}_y$ and $\mathbf{n}_z$. In general, this task frame will not have a fixed relation with respect to the global reference frame $\mathcal{F}_B$ or to the end-effector frame $\mathcal{F}_{SW}$. Depending of the contact type, each of these vectors can be defined with respect to any of the currently defined coordinate frames.

We propose to use the following syntax to specify task frame axes:

CreateFrame (<name>:<ref-fm>; <origin>:<ref-fm>;
<x-axis>:<ref-fm>; <y-axis>:<ref-fm>, <z-axis>:<ref-fm>)
AssignFrame (<name>)

where the angle-bracketed expressions denote symbolic labels for the corresponding entities. See Appendix B for more detail on the syntax and semantics of the language.

8.3 Motions to keep contact

Motions, tending to maintain the current contact state, are compliant motions. Several types of commands are issued to specify such motions. First, the Cartesian hybrid control axes must be designated either as position or force controlled directions. The next step is to specify compliance forces and torques along force controlled axes. Finally, a motion command must be issued, giving the desired displacements along position controlled directions. Because the task frame has been chosen to be aligned with the constraints imposed by the contact geometry, the specification of the compliant commands becomes relatively straightforward.

For the case of sliding motions, regardless of the contact type, the translational motion along the x and y directions of $\mathcal{F}_T$, will be position controlled while a force will be specified along the z-axis to maintain the contact.

In point-contact pivoting mode (see Figure 6), any rotational motion around the contact point is allowed and the three axes are therefore position controlled. Line contacts will require that zero torque be maintained about the contact-plane axis perpendicular to the edge direction. Finally, the only allowed rotation in a planar contact is the rotation about the constraint normal direction (task frame z-axis) and zero torques must therefore be commanded about the other two axes. In all cases a force must also be maintained along the z-axis to maintain contact.

The force to be exerted will be specified by a symbolic value in order to indicate what the intended result of this force is (for example F_{stick} or F_{slide}). The actual values of these forces will depend on the physical parameters of the task (*e.g.*, contact surface friction, *etc.*) and will be determined by the slave manipulator control software.

For example, during an *edge/face* contact, the following sequence of commands will be generated to execute a simple translational motion through a distance d in the direction of this edge (task frame y-direction):

AssignMode (P, P, F, F, P, P)
Force ($< 0, 0, -$ F$_{slide}$ $>$, **0**)
Slide ($< 0, d, 0 >$)

where F$_{slide}$ is a positive force, sufficient to ensure sustained contact during the sliding motion.

8.4 Motions to change contact

Both sliding and pivoting motions can cause a change of contact. Sliding motions can result only in the introduction of a new contact or deletion of a current one. Pivoting motions, on the other hand, will generally cause a change of the current contact type (for example, a transition from a *vertex/face* to an *edge/face* contact).

Whenever such changes are observed in the simulated world, the command generator must specify one (or more) terminating conditions for each of the corresponding

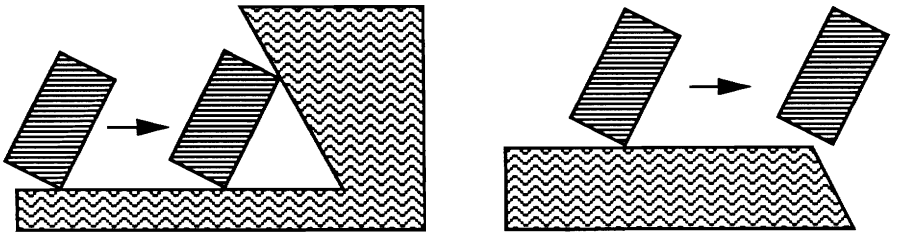

Figure 17: Changes of contact during a sliding motion.

motions.

8.4.1 Sliding case

When sliding motion along a given direction encounters a new contact (see Figure 17-a), it has to be stopped when a force discontinuity occurs along this direction. Because of the modeling uncertainties, the location of the environment feature to be contacted may not be known precisely. Therefore, in the interest of safety, the slave should also be given the maximum allowable displacement (function of estimated modeling uncertainties) — if the contact has not been encountered within this distance, then the motion should be terminated in an error state. Figure 17-b illustrates another situation where explicit terminating conditions need to be specified. The movable object is being slid along a surface towards the boundary of the sliding surface. In this and similar situations, termination of the motion corresponds to the occurrence of an acceleration discontinuity on the axis which was controlled in force during sliding.

We provide the following language constructs to alert the slave to the possible occurrence of the various terminating conditions during the upcoming motion:

GuardPosition (**p**, **o**)
GuardForce (**f**, τ)
GuardAcceleration (**a**, α)

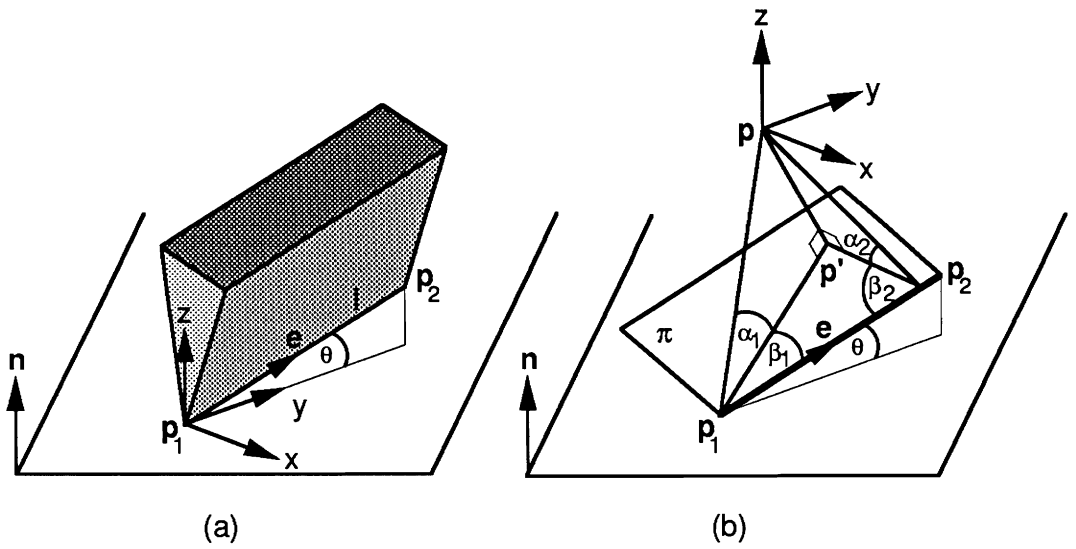

Figure 18: Transition between two *vertex/face* contacts.

8.4.2 Pivoting case

When a change of the contact type occurs, this transition can be characterized by a discontinuity of the torques about the contact edges. For example, figure 18-a illustrates a situation where a vertex of the mobile object is in contact with a planar surface of the environment. A rotational motion around the contact point $\mathbf{p}_1$ is then applied to put the edge $\mathbf{e}$ in contact with this face, while exerting a positive force f along $-\mathbf{n}$. In the frame defined by $\{\mathbf{p}_1\ ;\ (\mathbf{e}\times\mathbf{n})^*, (\mathbf{e}\times\mathbf{n})^*\times\mathbf{n}, \mathbf{n}\}$, the component τ_x of the torque acting on $\mathbf{p}_1$ remains null while this point remains in contact with the surface. However, when the transition occurs and the vertex $\mathbf{p}_2$ comes into contact with the supporting plane, this torque τ_x will suddenly increase to $f \cdot l$ (where l is the length of the edge) and the contact can thus be detected.

In fact, we show in Appendix C that this variation of torque remains constant, independently of the position of the coordinate frame in which the torques are expressed. This provides an easy way to detect such transitions directly from the torques measured in the frame of the F/T sensor, mounted at the slave manipulator's wrist.

The proposed symbolic encoding of the above pivoting motion is as follows:

AssignMode (P, P, F, P, P, P)
Force ($< 0, 0, -$ $F_{stick} >$, **0**)
GuardForce (**0**, $< l \cdot F_{stick}, 0, 0 >$)
Pivot (*edge*, *face*; l; $< -\theta, 0, 0 >$)

where $l = |\mathbf{e}|$, *edge/face* is the target contact type, and $l \cdot F_{stick}$ is the expected motion termination torque about the x-axis of the task frame (see Figure 18).

9 Contribution of This Work

The contribution of this research is the development and implementation of the *teleprogramming* concept for remote control of robotic systems in the presence of substantial communication delays. The essence of the this supervisory control technique is a high-fidelity kinesthetic and visual interaction with a graphically displayed virtual world, allowing for continuous and fluid on-line task-level programming of a remote robotic system. The operator is interrupted only when the remote control system was unable to carry out the specified sequence of elementary instructions or could not unambiguously verify the slave's resulting state. The two specific contributions of this work are

1. producing natural, real-time kinesthetic feedback to the human operator despite significant communication delays

2. on-line analysis of operator's motions and automatic generation of task-oriented symbolic instructions to the remote robotic workcell

The proposed methodology of extracting pseudo-force information from a non-dynamic graphical simulation represents a novel approach to providing the human operator with a sense of kinesthetic telepresence. Most existing systems do not offer this feature at all and instead rely on the local control of the slave to execute compliant motions autonomously without providing the operator with a kinesthetic "feel" of the resistive forces encountered by the slave. On the other hand, the systems which do attempt to provide this facility, are normally limited to simple linear or quadratic repulsion rules, where the resistive force is computed from the graphical simulation as an inverse linear (or quadratic) function of the decreasing distance d between objects, *i.e.*,

$$\mathbf{f} = -\frac{1}{d}\mathbf{v} \qquad \text{or} \qquad \mathbf{f} = -\frac{1}{d^2}\mathbf{v} \qquad ; \qquad |\mathbf{v}| < \epsilon \tag{31}$$

where $\mathbf{v}$ denotes the unit length vector along the shortest distance between objects and ϵ represents the contact threshold.

In our work we propose to generate pseudo-force reflection to the operator by analyzing polyhedral contact types and transitions between them. The kinesthetic feedback is generated as a consequence of enforcing lost translational and rotational

degrees of freedom both in the graphical world and in the low-level control loop of the master manipulator. A crucial component of this approach is a detailed model of polyhedral contacts and interaction between polyhedral features, which has been developed and implemented as part of the graphical simulation package. Both force and torque information to the operator is generated in a uniform and consistent manner.

The second area of contribution pertains to the automatic on-line generation of a stream of elementary instructions to the slave. Toward this aim, we are developing a symbolic language, which will encode sufficient information about the task geometry and modeling, sensory, and control uncertainties, to facilitate reliable execution of the elementary actions by the slave under its own local sensory supervision. Again, a prerequisite for generating this instruction stream is a consistent model of polyhedral feature interactions and the corresponding control issues (*e.g.*, how much information must the operator's station communicate to the slave for the latter to be able to safely and accurately transition from a vertex/face to an edge/face contact).

Finally, this research has addressed a variety of important subproblems, such as selection of a suitable control methodology for the master manipulator, ensuring that the operator need not be concerned with encountering workspace volume limitations or kinematic singularities on the master arm (reindexing techniques), automatically maintaining a natural and convenient view and projection of the graphical environment to the operator, *etc.* We feel that our solutions, results, and experience gained in exploring these issues will likewise contribute to the general body of knowledge in remote control of robotic systems.

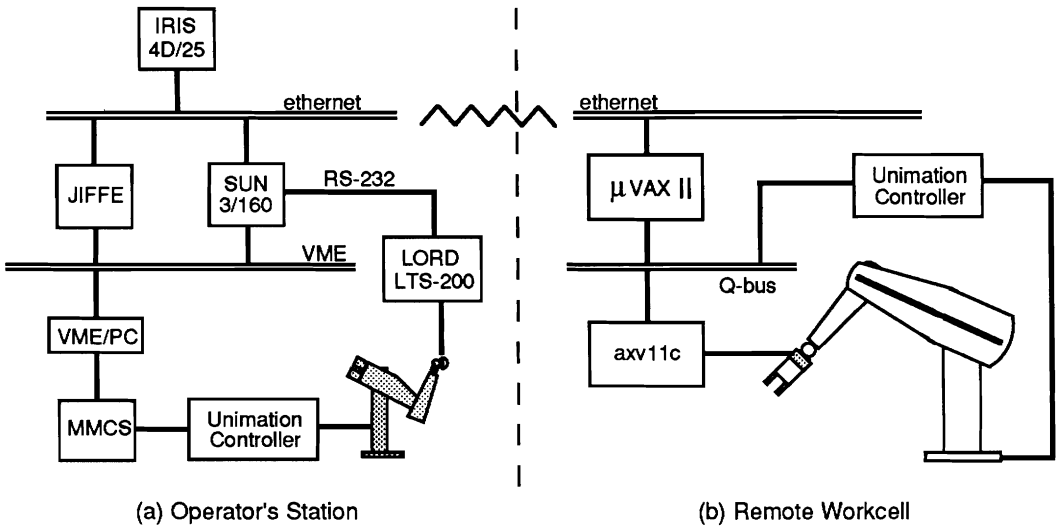

Figure 19: The hardware architecture of the experimental testbed.

10 Current Status

10.1 The experimental hardware/software testbed

The hardware architecture of our experimental operator's station setup is illustrated in Figure 19. The master manipulator in our scenario is a Unimation Puma 250 manipulator. It provides a backdrivable 6 DOF "joystick" with a sufficient operating volume to afford the operator a true sense of spatial positioning and orienting. Digital hardware control for the master is provided by the Modular Motor Control System (MMCS) [Corke,1989]. This system was designed and built at the laboratory as an experimental PC-bus based general purpose digital motor controller capable of controlling up to 16 independent actuators simultaneously. The MMCS hardware is interfaced to the original (factory-supplied) controller, whose sole remaining function is to provide power and the front panel interface. Finally, a custom-designed PC/VME adaptor connects MMCS's backbone to the VME bus.

Mounted at the wrist of the master is a 6 DOF force/torque sensor (LORD Corp., LTS-200) enclosed within a "whiffle-ball" handle for convenient grasping by the operator (see Figure 20). The sensor is read over a serial line (RS-232) and provides

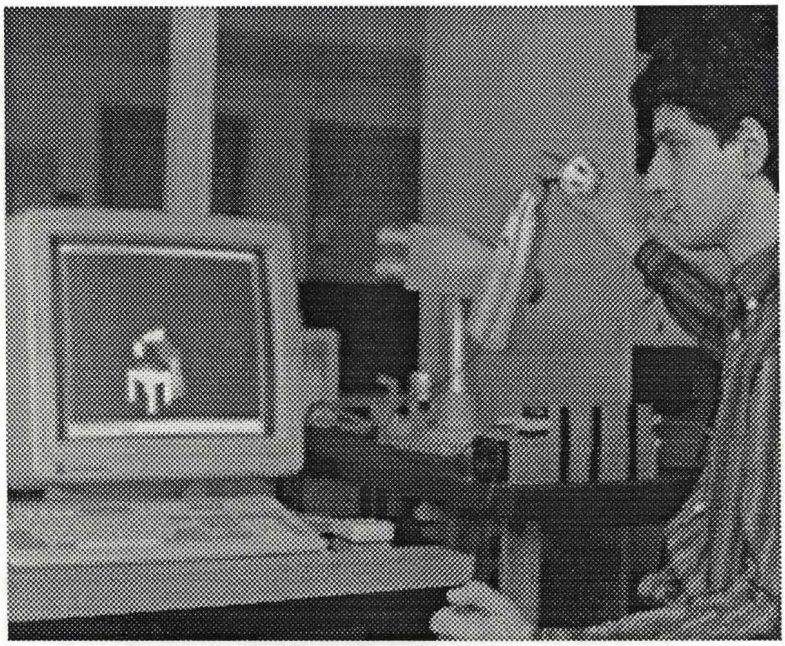

Figure 20: The operator's station.

information at a rate of approximately 30 Hz[21]. These readings are interpreted as incremental displacement/RPY Cartesian motion parameters of the sensor/handle assembly, and thus (through a transformation) of the master manipulator.

The computational engine of the system is JIFFE – a very fast, very-long-instruction-word floating point scalar processor delivering 20 real Mflops of computational power [Andersson,1989]. The processor has a standard VME interface and physically resides inside the Sun cage. It is fully C-programmable and supports most of the essential UNIX operating system facilities. JIFFE runs both the low-level joint servo code for the master at 500 Hz (PD control loop + gravity feedforward), as well as the Cartesian level servo code, which runs at 30 Hz (Cartesian setpoint computation and filtering as described in Section 6)[22]. It communicates with the Sun (model 3/160) via JIFFE-resident shared memory and (via the Sun and ethernet connection) with the Iris graphical workstation. The Sun currently serves

[21]There is a substantial variation about this nominal bandwidth, largely due to the unpredictable UNIX-incurred delays in servicing the serial port accumulating incoming data.

[22]The Cartesian servo loop bandwidth is limited only by the rate at which force/torque sensor can provide new information, and not by the JIFFE's computational capacity.

mostly as the accumulator and processor of the force/torque information from the sensor and as an intermediary between JIFFE and the Iris. In later stages of the system design and implementation, the Sun will provide a console for an on-line task-level dialogue with the operator (see Section 4.7).

The incremental Cartesian displacements are appropriately scaled into the remote slave's workspace and sent (via ethernet) to the Iris, which tries to realize them in the simulated slave environment. In case of a collision (see Section 5.4), the offending motion is appropriately modified so as to stop colliding objects in a contact but non-penetrating configuration. The new constraint information is added to the existing set of constraints and communicated back to JIFFE, which in turn filters subsequent operator-supplied motion demands so as to not violate any of the current constraints on the motion of the slave (see Section 6). This filtered motion is then applied both to the graphical model of the slave and the master manipulator, thus providing a sense of kinesthetic feedback to the operator.

The link between JIFFE and the Iris is a bidirectional communication channel conveying filtered incremental Cartesian motions one way and new/updated constraint information the other way. The link is implemented as a standard UNIX socket communication channel (between the Sun and the Iris) and has a round-trip latency of only a few miliseconds. The graphical workstation is a 16 MIPS Personal Iris 4D-25 with a hardware turbo graphics option to boost its drawing speed. Even so, its ability to render shaded graphical images of modest complexity (*e.g.*, the slave manipulator plus an object) lags far behind its scalar number crunching capacity. We are able to obtain refresh rates of about 7 Hz for low complexity environments and only partial shading. However, it is now within the realm of possibility to obtain fully shaded graphic displays of relatively complex scenes at video rates using the latest Silicon Graphics hardware [Bejczy&Kim,1990].

The software modeling environment for 3-D manipulation of articulated figures was provided by the Computer Graphics Laboratory at the University of Pennsylvania [Phillips&Badler,1988].

The remote manipulator in our experimental system is a PUMA 560, which is controlled using a Unimation controller, interfaced to a Microvax II. The robot is programmed using RCI and RCCL commands [Hayward,1983], [Lloyd,1985]. Information sent to this remote site will be parsed with a command language inter-

preter, which will translate the symbolic task-level instructions into a sequence of RCI/RCCL commands.

A six DOF instrumented compliant wrist, mounted at the slave's end-effector, is used as the remote force sensing device [Xu&Paul,1989]. With this wrist, a hybrid force/position control algorithm allows the manipulator to move in free space and in contact with surfaces. The passive compliance of the wrist and the active compliance of the control algorithm eliminate the problems associated with transitions from free space movement to constrained movement. Within the control loop, unexpected forces are monitored by limits on the wrist deflection.

10.2 Preliminary results and discussion

The current implementation of the system allows the operator to move the master and control the motion of the graphical model of the slave. The simulated slave can be brought into contact with the environment and the master is appropriately backdriven to provide a kinesthetic sense of contact to the operator. Recent experiments have shown that purely translational and sliding tasks can be performed with confidence and ease both for single and multiple constraining surfaces. The kinesthetic feedback to the operator feels natural and allows her to easily identify motion constraints and the shape of the constraining surfaces without looking at the display.

We are currently implementing the rotational (pivoting) contact motion mode. This should be completed in the near future and the resulting system should offer a versatile 3-dimensional 6 DOF input device that will allow the operator to perform a variety of probing tasks, exploratory procedures, surface following and identification tasks, *etc.*

Preliminary experiments with the system showed that reindexing is an important issue in control of the master arm. This is perhaps all the more true of our particular implementation, where a general purpose manipulator is employed as the master device, and as such is not designed to meet the requirements of a versatile master. In particular, we found that due to a large number of kinematic motion singularities, a relatively small workspace volume around any given initial "home" position can be used for maneuvering. Of the three reindexing schemes described in Section 4.3,

we have implemented the first two.

With the first method, the operator initiated reindexing by depressing a mouse button, which in turn put the arm in a free, gravity compensated mode and allowed the operator to reposition the master to an arbitrary new (presumably singularity-free) configuration before resuming position servo mode. As expected, the drawback of this approach lies in burdening the operator with having to be concerned with the kinematics and the current state of the master. This is especially unacceptable as the operator's full attention is often required to control the task in progress.

The second "drift-back" method was implemented using an exponential relationship between the displacement/twist away from the home position and the magnitude of the restoring drift. Whereas this eliminated the need for operator's intervention in the reindexing process, it significantly impaired the spatial resolution of the master's motion (work volume), which in turn obscured the kinesthetic feedback effects during contact motion. Since producing this kinesthetic feedback is a central feature of the proposed system, we chose not to adopt this approach.

The third approach of reindexing automatically seems the most promising, but it has not yet been experimentally verified. We intend to, in fact, offer a hybrid reindexing scheme, using automatic reindexing, as well as allow the operator to at any time reindex manually (the first approach).

Our current goal is to complete the implementation of the kinesthetic feedback features as described in Section 6, implement a satisfactory reindexing scheme, and concentrate our efforts on the problem of automatically partitioning the task in progress and extracting the relevant parameters to generate a stream of robust elementary task-level instructions to the remote slave.

11 Proposed Work Plan

I plan to complete the kinesthetic feedback portion of the graphical interface and its integration into the overall system by the beginning of September, 1990. By this time, we should have also completed the design of the symbolic language interface between the master and slave sites, and as well as resolved the issues of filtering the operator's motion trajectories as a preprocessing step to the symbolic command generation module. In September, I plan to visit our collaborating laboratory in Toulouse, France, and report on our progress, as well as ensure that our efforts remain compatible and coordinated.

I anticipate that the fall will be dedicated to the implementation of the language interface and the integration of both the operator's station modules (my work) and the remote workcell modules (Tom Lindsay's work) into a working system. I hope to complete the implementation of the proposed system by the end of December, 1990.

I would then like to spend some time investigating the performance issues, future directions and expansions of the system, as well as documenting the experience that I will have gained through this work.

I hope to submit my dissertation and graduate by May, 1991.

A Notation and Coordinate Transformations

A.1 Notation

Both 3 and 6-dimensional vector quantities are denoted as boldface (lower-case) characters with an optional preceding superscript indicating the coordinate frame with respect to which they are given, *i.e.*, $\mathbf{a}, {}^{B}\mathbf{n}$, *etc.*

A coordinate frame is specified by a triple of mutually orthogonal unit vectors, with an optional indication of the frame's origin, *i.e.*,

$$\mathcal{F} = \{\mathbf{x}, \mathbf{y}, \mathbf{z}\} \qquad \text{or} \qquad \mathcal{F} = \{\mathbf{p}\,;\, \mathbf{x}, \mathbf{y}, \mathbf{z}\} \tag{1}$$

Rotational matrices are denoted by upper-case boldface letters with optional superscripts and subscripts indicating which two coordinate frames they relate, *e.g.*, the matrix ${}^{B}\mathbf{R}_{F}$ describes the orientation of frame $\mathcal{F}_F$ *w.r.t.* $\mathcal{F}_B$.

Finally, we occasionally use the following non-standard vector notation

$$\begin{aligned} \mathbf{a}^{*} &= \frac{\mathbf{a}}{\|\mathbf{a}\|} \\ \tilde{\mathbf{a}} &= -\mathbf{a} \end{aligned} \tag{2}$$

A.2 Coordinate frames and rotational matrices

Let $\mathcal{F}_A$ be a coordinate frame and let ${}^{A}\mathbf{y}$ and ${}^{A}\mathbf{z}$ be two mutually orthogonal unit vectors, expressed in $\mathcal{F}_A$'s coordinates. Then the two vectors can be thought of as defining a second coordinate frame

$$\mathcal{F}_B = \left\{ \left({}^{A}\mathbf{y} \times {}^{A}\mathbf{z} \right), {}^{A}\mathbf{y}, {}^{A}\mathbf{z} \right\} \tag{3}$$

whose origin is coincident with $\mathcal{F}_A$'s and whose orientation *w.r.t.* $\mathcal{F}_A$ is given by the rotational matrix

$${}^{A}\mathbf{R}_{B} = \begin{bmatrix} | & | & | \\ \left({}^{A}\mathbf{y} \times {}^{A}\mathbf{z} \right) & {}^{A}\mathbf{y} & {}^{A}\mathbf{z} \\ | & | & | \end{bmatrix} \tag{4}$$

Moreover, the rotational matrix ${}^{A}\mathbf{R}_{B}$ can be used to map (rotate) an arbitrary vector ${}^{B}\mathbf{r}$ expressed in $\mathcal{F}_B$'s coordinates into its corresponding description in $\mathcal{F}_A$ coordinates, *i.e.*,

$${}^{A}\mathbf{v} = {}^{A}\mathbf{R}_{B} * {}^{B}\mathbf{v} \tag{5}$$

Likewise,

$$^B\mathbf{v} = {}^B\mathbf{R}_A * {}^A\mathbf{v} \tag{6}$$

where ${}^B\mathbf{R}_A = \left({}^A\mathbf{R}_B\right)^{-1}$.

A.3 Mapping rotations between frames

Let $\mathcal{F}_A$ and $\mathcal{F}_B$ be two arbitrary coordinate frames and let ${}^A\mathbf{r} = \theta \cdot {}^A\mathbf{k}^*$ denote a rotation expressed in $\mathcal{F}_A$'s coordinates. The same rotation can be expressed in frame $\mathcal{F}_B$ as

$$^B\mathbf{r} = \theta \cdot {}^B\mathbf{k}^* = \theta \cdot \left({}^B\mathbf{R}_A * {}^A\mathbf{k}^*\right) = {}^B\mathbf{R}_A * {}^A\mathbf{r} \tag{7}$$

Alternatively, if the rotation ${}^A\mathbf{r}$ is expressed as a triple of roll/pitch/yaw parameters, *i.e.*, ${}^A\mathbf{r} = (\theta_x, \theta_y, \theta_z)$, the equivalent rotation expressed *w.r.t.* $\mathcal{F}_B$'s coordinates is obtained by

- assembling a rotational matrix representing ${}^A\mathbf{r}$

$$^A\mathbf{R} = \text{RPYtoM}\left({}^A\mathbf{r}\right) \tag{8}$$

- transforming this matrix to $\mathcal{F}_B$'s coordinates

$$^B\mathbf{R} = \left({}^A\mathbf{R}_B\right)^{-1} * {}^A\mathbf{R} * {}^A\mathbf{R}_B \tag{9}$$

- extracting the new triple of RPY parameters

$$^B\mathbf{r} = \text{MtoRPY}\left({}^B\mathbf{R}\right) \tag{10}$$

See [Paul,1981] for a detailed discussion of the RPYtoM and MtoRPY conversion operators. For the linear-algebraic basis of these operations, the reader is referred to [Nering,1970].

A.4 Displacement of a point due to motion of the frame

Let $\mathcal{F}$ be a coordinate frame undergoing a translational and rotational motion $\Delta\mathbf{d}_{\mathcal{F}} = (\mathbf{t}, \mathbf{r})$. Then the resulting displacement of a point located at $\mathbf{p}$ *w.r.t.* the origin of $\mathcal{F}$ is

$$\Delta\mathbf{d}_{\mathbf{p}} = (\mathbf{t} + (\mathbf{R} * \mathbf{p}) - \mathbf{p}, \mathbf{r}) \tag{11}$$

where $\mathbf{R} = \text{RPYtoM}(\mathbf{r})$, and $\Delta\mathbf{d}_{\mathbf{p}}$ is given *w.r.t.* to the original frame $\mathcal{F}$.

B The symbolic command language

This section of the appendix briefly summarizes the main constructs of the low-level language interface between the operator's station and the remote workcell. The details of the scope, syntax, and semantics of the language are still under development — the following is the current conception of the low-level command language.

In what follows, syntactic constructs are given informally. When the language design is completed, we will derive the corresponding context free grammar description and produce the corresponding parser/interpreter. A brief semantic clarification follows the statement which are not self-explanatory.

B.1 Task frame management

Basic entities, such as vectors and coordinate frames, are given symbolic labels, *e.g.*, <ref-fm> (reference frame). All subsequent higher order constructs refer to these labels, instead of the actual numeric quantities. Besides labeling vectors and frames, the slave controlling software can be asked to track the current contact point(s), assemble right-handed orthogonal coordinate frames from discrete axis information, and assign an arbitrary (defined) coordinate frame as the current task frame. Note that in creation of a task frame, each of the component axes can be specified *w.r.t.* an arbitrary known coordinate frame. Moreover, these axes are functions of the task geometry and may, in general, not be mutually orthogonal. The slave site controlling module will thus need to orthogonalize and normalize the specified coordinate frames and associated transformations. An assigned task frame remains in effect until overriden.

LabelVector (<label>, **v**)
LabelNormal (<label>)
LabelContactPt (<label>)
TrackContactPt (<label>)
CreateFrame (<name>:<ref-fm>; <origin>:<ref-fm>;
<x-axis>:<ref-fm>; <y-axis>:<ref-fm>, <z-axis>:<ref-fm>)
AssignFrame (<name>)

B.2 Force control commands

The following statements are designed to support (at a task level) the hybrid force/position control paradigm.

AssignMode (X, X, X, X, X, X) ; X $\in$ { F, P }

Specifies force (X=F) and position (X=P) controlled directions and thus defines the selection matrix **S**. A force-controlled direction is assumed to require 0 force compliance (default), unless otherwise specified by a subsequent **Force** statement.

Force (**f**, τ)

Specifies force preload. If preload is specified on a force-controlled axis, it is interpreted as the compliance force. If preload is specified on a position-controlled direction, it is interpreted as a preload force for operations like pushing, screw or valve tightening, *etc.*

Velocity (**v**, ω)

Specifies velocity preload for tracking moving parts of the environment.

Time (t)

Specifies the amount of time in which to accomplish the forthcoming motion command. Note that velocity information can be derived from the motion displacement parameters and timing info. If and argument t=0 is given, the system is expected to compute the necessary timing/velocity information, based on maximum allowed joint/Cartesian rates and accelerations. Default velocities should be used if no **Time** statement appears prior to the motion command.

GuardPosition (**p**, **o**)
GuardForce (**f**, τ)
GuardAcceleration (**a**, α)

The guards are filtered through the inverse of the selection matrix **S'**. Task level force guards only make sense along position controlled directions. Similarly, task level position guards are only relevant along the force controlled directions. Low-level safety force guards should be active at all times (task independent).

B.3 Motion commands

All motion commands are subject to velocity constraints imposed on the motion by the preceding **Time** statement.

Move (**p**, **o**)

Free-space motions. **p** and **o** give the *incremental* translation and rotation of the slave's end-effector frame (T6) *w.r.t.* the current task frame.

Slide (**p**)

Contact sliding — the slave must be in contact, at least one axis should be force controlled, and a force preload should be given along that axis. If any are missing (or don't have defaults), the interpreter should complain as we have an inconsistent motion request. This allows some cross-checking for consistency.

Pivot (<feature1>, <feature2>; <dim>; **o**)

Perform a pivoting motion about the contact point. <feature(i)> $\in$ { *vertex*, *edge*, *face*}, <feature1> belongs to the moving (i.e., held) object. <feature1> and <feature2> specify the two feature types denoting the target contact, *e.g.*, *edge/face*, *face/face*, *etc.* (Note: these are **not** labels of specific features, only feature **types**) <dim> gives the dimension (size) of the critical target contact feature to help the slave monitor the torques and decide when it has reached the desired contact. Normally this will be an edge length (note that for both *vertex/edge* and *edge/face* transitions, edge length is the critical parameter). **o** is the string label of the rotation parameters about the contact point (origin of the current Task Frame).

B.4 Effector commands

Grasp (<feature1>, <feature2>)

Two-finger grasping — close the gripper such that finger(i) contacts <feature(i)>.

Release ()

Release the grasp, i.e., open the gripper as wide as necessary to clear the object by a reasonable tolerance.

B.5 Issues

- **Syntax/Semantics**: The scope of the statements must be clearly defined.
- **Labels**: Certain labels can be assumed to be predefined (e.g., KB for slave's Kinematic Base frame, WST for the slave's wrist position, *etc*), others are defined for later reference as the task proceeds.
- **Tolerances**: Maximum uncertainty bounds on modeling errors (ϵ_p, ϵ_o) must be estimated — maximum displacements (tra/rot) during guarded moves are then computed as functions of these tolerances.
- **Symbolic Nature of Commands**: The command stream should be mostly symbolic in nature — the numeric values supplied by the operator station are primarily: a) the wanted displacements (tra/rot) of the manipulated object, and b) certain task frame axes derived from graphics. On the other hand (since the simulation is kinematic), force/torque parameters are supplied symbolically, *i.e.*, F_{slide}, F_{push}, $F_{contact}$ and must be determined empirically by the slave as a function of locally determined masses, inertias, and frictional parameters of the actual objects.

C Independence of the torque measurement site

We have seen in Section 8.4.2 that the motion terminating torque in a *vertex/face* to a *edge/face* transition (Figure 18-a), as measured at the contact point $\mathbf{p}_1$, is

given by $\tau = (l \cdot \mathbf{e}) \times \mathbf{f}$. The force $\mathbf{f}$ corresponds to the stiction compliance force during the pivoting motion. Because of the choice of the task frame orientation, the terminating torque reduces to $\tau =< f \cdot l, 0, 0 >$.

Consider now a situation, where the termination torque is to be measured in a coordinate frame with the same orientation as the task frame, but whose origin has been displaced from $\mathbf{p}_1$ (task frame origin) to $\mathbf{p}$ (Figure 18-b).

The torque acting at $\mathbf{p}$ due to the reaction force $\mathbf{f} = (0, 0, f)^T$, applied at the point of contact, is expressed in the frame $\{\mathbf{p}\,;\, (\mathbf{e} \times \mathbf{n})^*, (\mathbf{e} \times \mathbf{n})^* \times \mathbf{n}, \mathbf{n}\}$ as follows

$$\tau = \mathbf{r} \times \mathbf{f} = (r_y \cdot f, -r_x \cdot f,\, 0)^T \tag{12}$$

where $\mathbf{r}$ is the vector from $\mathbf{p}$ to the point of contact. During a contact with $\mathbf{p}_1$, the components of this vector $\mathbf{r}_1 = \overline{\mathbf{p}\mathbf{p}_1}$ are

$$\mathbf{r}_1 = \begin{pmatrix} l_1 \cdot \cos\alpha_1 \cdot \sin\beta_1 \\ l_1 \cdot (\sin\alpha_1 \cdot \sin\theta - \cos\alpha_1 \cdot \cos\beta_1 \cdot \cos\theta) \\ -l_1 \cdot (\cos\alpha_1 \cdot \cos\theta + \cos\alpha_1 \cdot \cos\beta_1 \cdot \sin\theta) \end{pmatrix} \tag{13}$$

where α_1 is the angle between $\overline{\mathbf{p}_1\mathbf{p}}$ and its projection $\overline{\mathbf{p}_1\mathbf{p}'}$ onto the plane π whose normal is obtained by a rotating $\mathbf{n}$ through $Rot(\mathbf{x}, \theta)$ (see Figure 18-b). β_1 denotes the angle between $\overline{\mathbf{p}_1\mathbf{p}'}$ and the edge $\mathbf{e}$.

Similarly, during a contact with the point $\mathbf{p}_2$, the vector $\mathbf{r}_2 = \overline{\mathbf{p}\mathbf{p}_2}$ is given by

$$\mathbf{r}_2 = \begin{pmatrix} l_2 \cdot \cos\alpha_2 \cdot \sin\beta_2 \\ l_2 \cdot (\sin\alpha_2 \cdot \sin\theta + \cos\alpha_2 \cdot \cos\beta_2 \cdot \cos\theta) \\ -l_2 \cdot (-\cos\alpha_2 \cdot \cos\theta + \cos\alpha_2 \cdot \cos\beta_2 \cdot \sin\theta) \end{pmatrix} \tag{14}$$

The transition from contact $\mathbf{p}_1$ to contact $\mathbf{p}_2$ occurs for $\theta = 0$. Computing the values of the two torques just before and after the *edge/face* contact gives

$$\begin{aligned} \tau_1 &= \lim_{\theta \to +0} (\mathbf{r}_1 \times \mathbf{f}) = f \cdot (-l_1 \cdot \cos\alpha_1 \cdot \cos\beta_1, -l_1 \cdot \cos\alpha_1 \cdot \sin\beta_1, 0)^T \\ \tau_2 &= \lim_{\theta \to -0} (\mathbf{r}_2 \times \mathbf{f}) = f \cdot (l_2 \cdot \cos\alpha_2 \cdot \cos\beta_2, -l_2 \cdot \cos\alpha_2 \cdot \sin\beta_2, 0)^T \end{aligned} \tag{15}$$

The variation of the torque across the contact then is

$$\Delta\tau = \tau_2 - \tau_1 = f \cdot \begin{pmatrix} l_1 \cdot \cos\alpha_1 \cdot \cos\beta_1 + l_2 \cdot \cos\alpha_2 \cdot \cos\beta_2 \\ l_1 \cdot \cos\alpha_1 \cdot \sin\beta_1 - l_2 \cdot \cos\alpha_2 \cdot \sin\beta_2 \\ 0 \end{pmatrix} \tag{16}$$

It is easy to see from Figure 18 that

$$\begin{aligned} l_1 \cdot \cos\alpha_1 \cdot \cos\beta_1 + l_2 \cdot \cos\alpha_2 \cdot \cos\beta_2 = l\,, \quad &\text{and} \\ l_1 \cdot \cos\alpha_1 \cdot \sin\beta_1 = l_2 \cdot \cos\alpha_2 \cdot \sin\beta_2 \quad &(17) \end{aligned}$$

and the change of contact therefore introduces a discontinuity on τ_x only. Moreover, the magnitude of this discontinuity is again given by $f \cdot l$. Since the torque measuring site was chosen arbitrarily, we conclude that the reaction torques will be the same regardless of the location of the sensing device.

A.2 Robot Slave System

Slave Robot System

Tom Lindsay

January 8, 1991

Abstract

This research centers around the slave-side operations of teleoperation with significant communication delays [1]. The tasks include dissassembly/salvage/repairs in an unstructured environment. In order to speed the process, and overcome communication time delays, the slave runs semi-autonomously – it receives commands from a command queue, tries to complete each command, and if successful, continues with the next command. However, if there is a problem, the slave communicates back to the master with the information the human operator needs to correct the error.

The study of the remote slave operation can be divided into two areas: hardware and software. Hardware for our research includes a PUMA 560 industrial robot controlled with RCCL and RCI software through a MicroVax II and the Unimation controller. A force/torque sensing compliant wrist, based upon work by Xu [3], is the current end effector. This wrist is being redesigned to improve its performance and usefulness as the major sensing device used in the remote world. Two levels of software are being developed for use at the remote site. A low level control program is used for basic manipulator movement, and includes simple error detection algorithms. A higher level communication language parser converts messages received from the master site into information used by the control program. It also returns information to the master about error states encountered.

1 Hardware

1.1 Robot

At the remote site, a PUMA 560 is used as the manipulator. It is dissimilar to the PUMA 260 used as the master, to illustrate the fact that with the methods we are using for teleoperation, the master robot can be designed for better interaction with the human operator, while the slave robot can be designed to be more useful in the environment it will be used in. Thus, the operator would not have to move a large,heavy robot, even though the remote site requires one.

The puma is equipped with a compliant instrumented wrist, described below. Currently, there are no tools for the robot to use, although research is being conducted to determine the feasibility of using a box end wrench [2], and an impact wrench. The impact wrench is a useful tool for dissassembly tasks, where bolts may be frozen or rusted in place. However, the vibrations caused by the wrench may present control problems for the robot. We are working with an air impact wrench, but for underwater applications, an electric impact wrench will probably be needed.

1.2 Wrist

The end effector is based upon Xu's compliant instrumented wrist [3]. The existing wrist is compliant, yet has a serial linkage with potentiometers at the joints which determines the deflection in the wrist. Thus position errors can be sensed, and forces/torques that the end effector is subjected to can be calculated. The wrist has all of the benefits of compliance, coupled with the accuracy of a much stiffer force/torque sensor.

For useful work at the slave site, there are several improvements being made. The compliant element structure and the serial sensing linkage, which currently are placed in series between the robot and the tool, have been redesigned as surrounding elements, so as to reduce the distance between the end of the robot and the end of the tool. Two other improvements come immediately from the new design. First, the ratio of translational to rotational stiffness can be improved. Second, the serial linkage structure is made with longer links, which increases the sensitivity. A further improvement that came about from the redesign was an improved mechanical system, with simpler parts and more protection for the electronics.

Improvements can also be made to the electrical system to increase the sensitivity of the potentiometers, and to reduce noise. The working range of the potentiometers about their home position is less than 30 degrees, while the potentiometer's full range is 270 degrees. By rescaling the voltage drop over the working range to the full range of the A/D board we are using, a potential increase of nine times the sensitivity is possible. Also, by using analog low pass filters, the need for digital filters can hopefully be eliminated.

1.2.1 Compliant Structure

The compliant structure of the new wrist is composed of 12 rubber elements, which provide compliance and some degree of damping. Figure 1 shows the design, with the bottom plate (attached to the robot) fixed to the four aluminum blocks at the corners, and the top plate (where the tool is attached) fixed to the four compliant elements (cylinders) at the top. The tool can then be partially enclosed in the middle of this structure.

The stiffness in each direction can be approximated as follows:

$$K_z = (\frac{1}{4K_a} + \frac{1}{8K_r})^{-1}$$

$$K_x = K_y = (\frac{1}{4K_r} + \frac{1}{4K_a + 4K_r})^{-1}$$

$$K_\psi = (\frac{1}{4K_r L_1^2} + \frac{1}{8K_a L_1^2})^{-1}$$

$$K_\phi = K_\theta = (\frac{1}{2K_a L_1^2} + \frac{1}{4K_r L_1^2 + 4K_r L_2^2})^{-1}$$

where K_a and K_r are the angular and radial stiffnesses of a single element. A tabular comparison of the old and new wrists is shown below.

	K_x	K_y	K_z	K_ϕ	K_θ	K_ψ
	lb/in	lb/in	lb/in	in-lb	in-lb	in-lb
old wrist [3]	32.30	32.30	64.86	6.65	6.65	3.97
new wrist	41.65	41.65	70.59	61.37	61.37	68.81

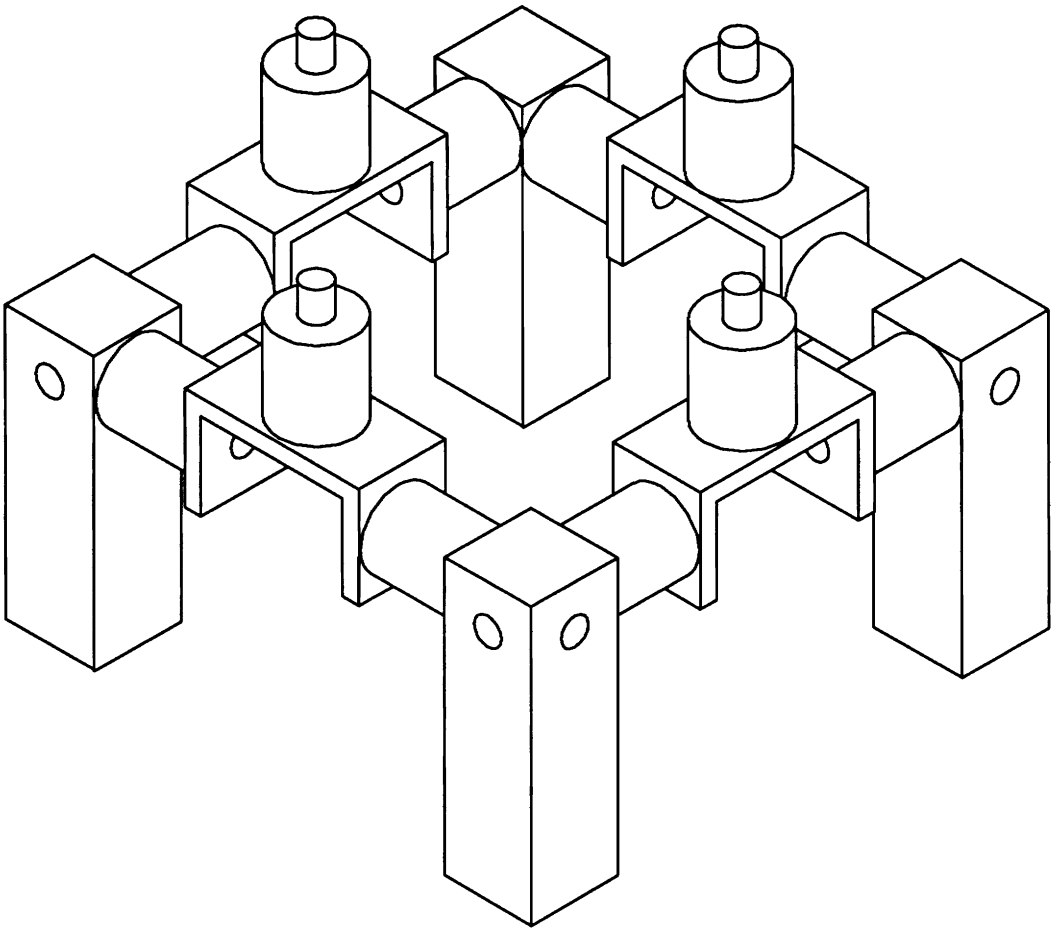

Figure 1: Compliant Structure

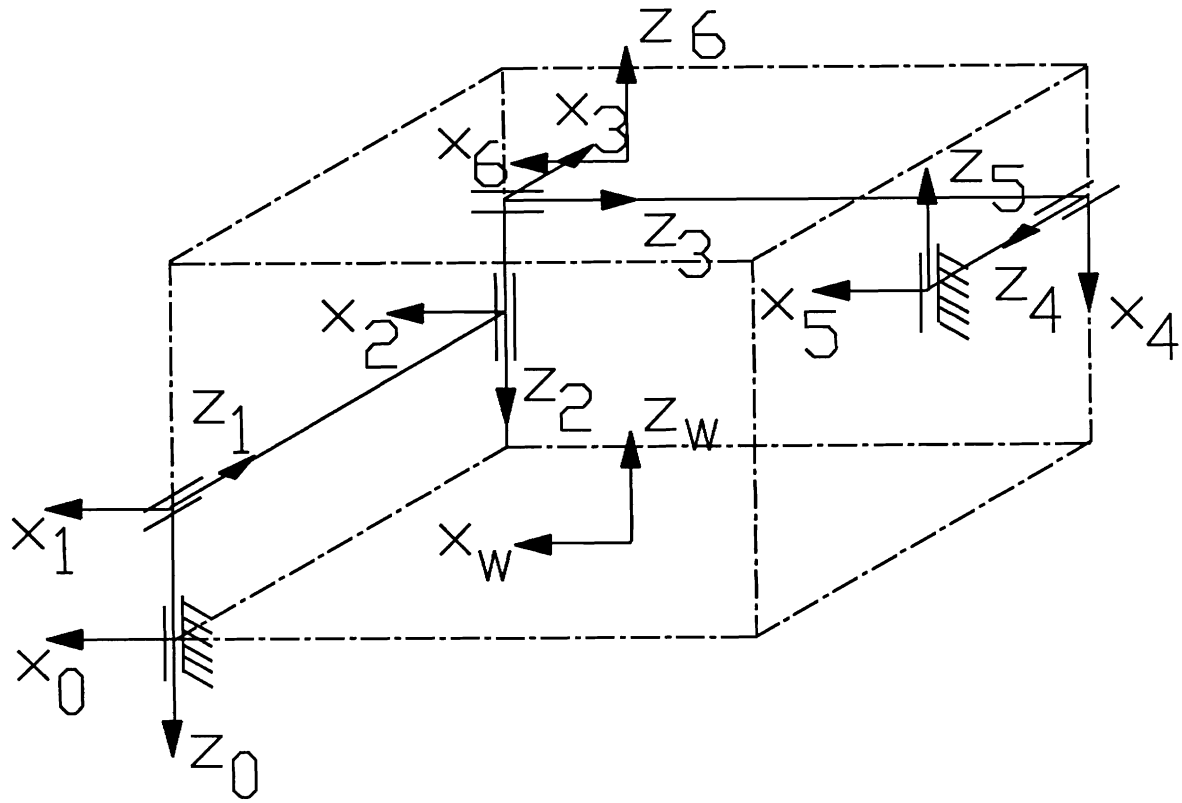

Figure 2: Kinematic Skeleton of Sensing Mechanism

Notice that while there is a modest increase in stiffness for the translational directions, the new wrist is much stiffer in the rotational directions.

1.2.2 Sensing Mechanism

The sensing mechanism is composed of six links, with potentiometers at each joint. Figure 2 roughly shows the kinematic skeleton of the sensing mechanism. From the change in resistance across the potentiometers, the joint angles of the six links can be found, and the position of the top plate relative to the bottom plate can be determined from the kinematics of the linkage.

The D-H parameters for the wrist are:

	a	d	α	θ
joint	inches	inches	deg.	deg.
1	0	-0.875	-90	0
2	0	3.875	90	0
3	0	-0.750	-90	90
4	0	3.875	90	-90
5	0	1.750	90	90
6	1.9375	-0.875	0	0

Also needed to define the transform between robot and tool is:

$$A_W^0 = \begin{bmatrix} 1 & 0 & 0 & 1.9375 \\ 0 & -1 & 0 & 1.9375 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

With this information, a transform from the end of the robot to the end of the wrist is formed. A further transform from the end of the wrist to the end of the tool will complete the transformation from the end of the robot to the tip of the tool.

The Jacobian matrix for the sensing mechanism in the home position is found to be:

$$J = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0.875 & 0 & 1.925 & 1.625 & -1.925 \\ 1.925 & 0 & 0.875 & 1.925 & 0 & 1.925 \\ 0 & -1.925 & -1.925 & 0 & -1.925 & 0 \end{bmatrix}$$

The inverse Jacobian can be calculated:

$$J^{-1} = \begin{bmatrix} 0.0162 & 0.2110 & 0.5000 & 0 & 0.2597 & 0.1096 \\ 0.5000 & 0.5000 & 0 & 0 & 0 & -0.2597 \\ -1.0000 & 0 & 0 & 0 & 0 & 0 \\ -0.1055 & -0.0081 & -0.2500 & 0.2597 & 0.1299 & 0.1138 \\ 0.5000 & -0.5000 & 0 & 0 & 0 & -0.2597 \\ 0.1218 & 0.2192 & -0.2500 & -0.2597 & 0.1299 & -0.0042 \end{bmatrix}$$

Thus, the eigenvalues of the rotational and translational kinematic sensitivity matrices [3] are:

$$\lambda = (.36, .60, 1.53)$$

$$\mu = (.07, .13, .20)$$

Because the largest eigenvalue for each set is less than five times the smallest, we can assume that the relative sensitivity for each direction in translation and rotation is fairly similar. Although the new design is more sensitive because of longer link lengths, it is not as isotropic in sensitivity.

1.2.3 Dynamics of Wrist

The dynamics of the new wrist have not yet been explored. The natural frequencies of the wrist may become important when using tools, such as the impact wrench, that have their own driving frequency. If the driving frequency matches a natural frequency of the wrist, the resulting motion is likely to cause control problems. In such a case, the tool would have to be mounted on vibration absorbers.

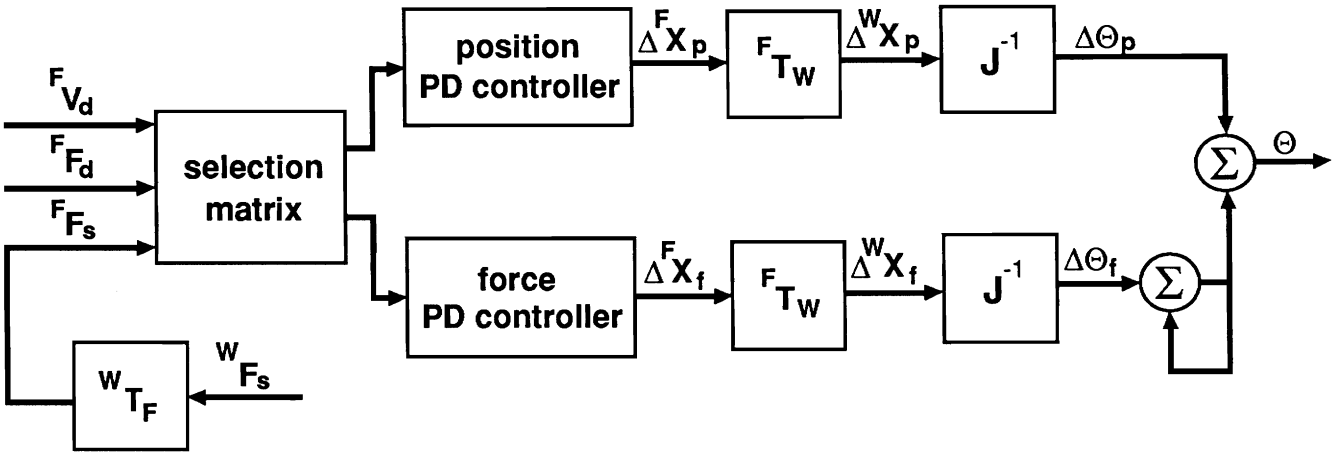

Figure 3: Control Structure

1.3 Vision system

It appears that some of the slave side motions will have to rely to some extent on a vision system. Tasks such as pin-in-hole insertions, operations on bolts, and hook-in-eye operations are much more suited to autonomous control with feedback from a vision system than the semi-blind control from the master. Some research is being conducted in this area.

2 Software

2.1 Robot control

The robot is controlled via RCI commands. RCI sends commands to the robot at 28 ms intervals, which is a limitation in the current system. Above the RCI control is a PD controller which uses information from the instrumented wrist as feedback. Figure 3 shows a simplified schematic for the controller. Basically, the controller receives cartesian force and motion commands, where the cartesian frame is known, but can be arbitrary. Feedback is input as cartesian positional errors in the wrist frame. The control program, using a hybrid force/position algorithm, converts motion commands to velocities in the wrist frame, computes differential motion, converts the cartesian motion to joint space, and sends the updated joint positions to the robot. There are also checks for deflection limits in the wrist. If any constraint is violated, motion stops and the operator is alerted.

2.2 Communication Language

The communications language defines how the master and the slave interact. Because of the communication limitations that are implicit in this research, the interaction must be minimal. Therefore, a simple set of commands is being developed that must be sufficient to perform undersea salvage/exploration/etc. tasks.

The language contains commands for task frame management, which basically determines what transform goes in the "T_F^W" and "T_W^F" boxes in Figure 3. Also, there

are force and motion commands, which specify $x_{c,F}$ and $F_{c,F}$. The c subscript means commanded, and the F denotes the frame of reference, which is arbitrary but known. The AssignMode command specifies which directions are force controlled and which are position controlled. Finally, there are commands for specific actions, such as Grasp and Release. Actions which involve the vision system have not yet been investigated.

The slave works through a queue of commands, which are conveniently numbered by the master. If and when an error occurs, the state of the slave manipulator and the command number are sent back to the master operator. At this point, the queue of commands is flushed, up to the point where a correcting command is issued. If commands are successful, the task can be completed without direct feedback from the slave to the master, and thus much time can be saved. When errors occur, the human operator is available to make error corrections.

3 Conclusion

Within the slave framework discussed above, teleoperation with significant time delays becomes possible. The human operator is able to perform tasks in the local world with kinesthetic feedback, and the remote slave operates in a semi-autonomous mode, following the queued instructions. However, when an error occurs, the human operator intervenes and is able to use human reasoning powers to correct the error.

References

[1] Richard P. Paul, Janez Funda, Thierry Simeon, and Thomas Lindsay. Teleprogramming for autonomous underwater manipulation systems. In *Intervention '90*, pages 91–95, The Marine Technology Society, June 1990.

[2] Walter Santarelli. Wrench end effector project. 1990. To be published as a tech. report.

[3] Yangsheng Xu. *Compliant wrist design and hybrid position/force control of robot manipulators.* PhD thesis, University of Pennsylvania, 1989.