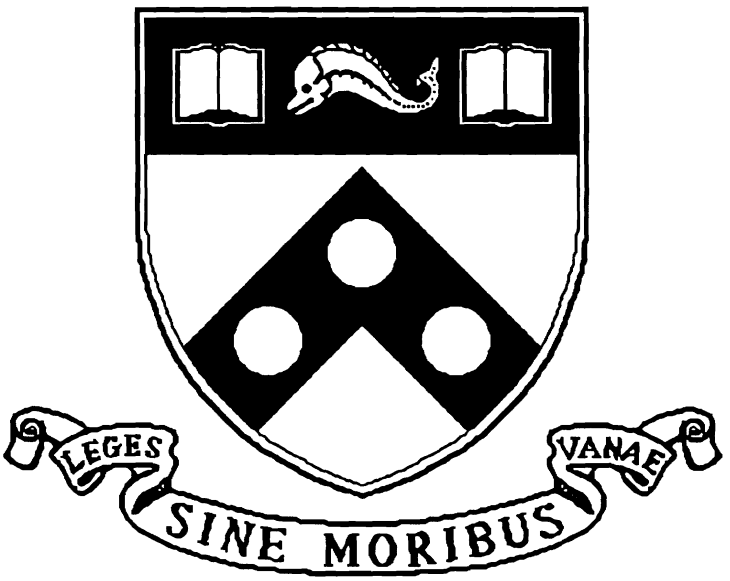
Model Based Teleoperation To Eliminate Feedback Delay
NSF Grant BCS89-01352
Second Report

MS-CIS-92-75
GRASP LAB 333

Richard P. Paul
Janez Funda
Thomas Lindsay
Craig Sayers
Masahiko Hashimot



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

October 1992

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

August 1991

Richard P. Paul
Janez Funda Thomas Lindsay
Craig Sayers 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 leading to very low productivity.

We have combined computer graphics with manipulator programming to provide a solution to the problem. A teleoperator master arm is interfaced to a graphics based simulator of the remote environment. This system is then coupled with a robot manipulator at the remote, delayed site. The operator's actions are monitored to provide both kinesthetic and visual feedback and to generate symbolic motion

commands to the remote slave. The slave robot then executes these symbolic commands delayed in time. While much of a task proceeds error free, when an error does occur, the slave system transmits data back to the master environment which is then "reset" to the error state from which the operator continues the task.

Contents

1 Introduction

Teleoperation of a remote manipulator system is severely hampered by the presence of any significant delays in the communication channel between the two sites. The presence of geosynchronous satellites in the communication path causes delays during earth-based teleoperation of robotic systems in space. Similarly, acoustic communication links result in significant delays in information propagation during surface-based teleoperation of untethered submersibles. Such delays destabilize the feedback control loop between the master and the slave sites, forcing the operator to slow down and thus resulting in low productivity.

We have developed a novel combination of computer graphics and manipulator programming to solve the problem. In our system a teleoperator master arm is interfaced to a graphics based simulator of the remote environment in which the operator can perform a task without delay. The operator's actions are monitored to provide immediate kinesthetic as well as visual feedback and to generate symbolic motion commands to the remote slave. These instructions are based on the hybrid position/force model of robot's interaction with its environment and are generated so as to account for the estimated uncertainty in the graphical model of the remote environment. These instructions are then sent to the remote slave in real time and executed remotely delayed in time.

The graphical model of the remote environment is constructed on the basis of range, video, or sonar scans of the actual environment. Interfaced to the geometric modeling system is a six-degree-of-freedom input device which controls the Cartesian motion of the immediately reacting simulated slave robot and any object that it might be holding or carrying. The system monitors the position of the slave arm, in the geometric model, to detect penetration of any work objects by the slave arm or by any object it is carrying. When this occurs, the input device (PUMA 250) is backdriven so as to maintain positional and orientational correspondence between the input master device and the image. This provides the operator with a good approximation to real-time force feedback, which has been found cruicial for natural and efficient teleoperation.

With these capabilities, the operator can not only see what is going on but can also *feel*, kinesthetically, the objects represented in the display. When the inside of a box is displayed the operator can feel along a surface to a corner

between two surfaces; the operator can slide along the edge into the corner of the box. 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."

The robot commands automatically generated for execution at the remote site are very simple consisting of free space moves and guarded and compliant moves. Whenever the operator brings two objects together, so that a collision is detected, a guarded move is sent to the remote site. The system does not generate any conditionals, such as "if a happened then do b else do c". Therefore, if an execution error occurs at the remote site, the system waits for the operator to interpret the situation and to generate the appropriate corrective actions. Conditionals have plagued robot programming as every 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 [1].

The symbolic robot commands describing the operations that are being performed on the image are executed by a slave manipulator at a remote site a communication delay later in time. 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, due to unmodelled dynamics the slave may fail to carry out a commanded motion. At this point the remote site communicates the state of the remote system back to the operator's station and the operator is alerted of the error condition through an audio interface and the displayed state of the slave robot and the environment is reset back to the state in which the slave is "hung up". The constraints on the motion of the input device are also changed to correspond to the current situation at the remote site. The operator then resumes task execution from this new state. Once again, these actions are translated into symbolic robot commands and sent to the slave, thus recovering from the error condition and proceeding with the task.

2 Past Research

During the first year of the grant we made substantial progress with the master station. We built a data base using the Jack graphics system [3]. 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 [4] 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 [5] 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 also began 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 [6] and Tom Lindsay. The wrist allows us to come into contact with the environment and to control forces of interaction.

3 Current Research

During the last year the master station was completed, to the extent that the operator could perform a task in the modeled world and automatically generate robot manipulator instructions for the slave, see Appendix A.1. The slave station was also completed, to the extent that robot instructions could be received and then executed after introducing an appropriate delay see Appendix A.2. The task we choose was the exploration of a box with the operator finding the box, its sides, bottom and corners, etc. The slave was delayed by about 3 seconds. Only primitive error recovery was developed, but, by the end of the year the system was quite reliable and we could sustain operation for up to an hour at a time.

3.1 Master

Research at the master station involved the robot instruction generation based on contact information and the rate of change of the operator's inputs. A low level language was developed and a parser written for the slave manipulator. Delay in parsing instructions by the slave were solved by a double buffering scheme so that the slave manipulator was kept in synchronization with the master but with a constant delay. The kinesthetic feedback was used to confirm, to the operator, the motions the slave was to perform, and to maintain kinematic correspondence between the master and the slave arms. We also developed a system of automatic re-indexing of the master as it approached kinematic singularities.

3.2 Slave

The slave was interfaced to the master using ethernet and sockets. We introduced an artificial delay into the communications to approximate an acoustic link. A double buffered parser was written for the command language to translate the kinematic commands received from the master into instructions that the slave could execute. All dynamic and frictional effects were taken care of by the slave system. Research was undertaken into estimating stopping conditions by statistically modeling the environment as the slave moved.

4 Talks and Presentations

1. Janez Funda, November 1990, IEEE Conference on Systems, Man and Cybernetics, "Teleprogramming: Overcoming Communication Delays in Remote Manipulation," presenter.

2. Richard P. Paul, July 1990, DOE/Industry/University/Lab forum on Robotics for Environmental Restoration, Waste Management and Waste Minimization, "Teleprogramming: A Basis for Waste Handling," invited presentation.

3. Richard P. Paul, October 1990, ICAR, First Workshop on Mobile Robots for Subsea Environments, "Teleprogramming: Overcoming Communications Delays in Remote Manipulation," workshop paper.

4. Richard P. Paul, January 1991, The Second Workshop on Architectures for Real-Time Intelligent Control of Unmanned Vehicle Systems, "Teleprogramming: Overcoming Communication Delays in Remote Manipulation," invited presentation.

5. Janez Funda, April 1991, 1991 IEEE International Conference on Robotics and Automation, "Remote Control of a Robotic System by Teleprogramming," presenter.

6. Richard P. Paul, April 1991, 1991 IEEE International Conference on Robotics and Automation, "The Application of Robotics to the Handling of Hazardous Wastes, Materials, and Equipment," workshop organizer and presenter.

7. Thomas Lindsay, June 1991, Second International Symposium on Experimental Robotics, "Contact Operations Using an Instrumented Compliant Wrist," presenter.

8. Richard P. Paul, September 1991, Seventh International Symposium on Unmanned, Untethered, Submersible Technology, "Teleprogramming for Manipulation by Autonomous Underwater Vehicles," presenter.

9. Janez Funda, October 1991, Oceans '91 Conference, "A Symbolic Teleoperator Interface for Time-Delayed Underwater Robot Manipulation," presenter.

5 Documentation

1. J. Funda, T. Lindsay and R. P. Paul, "Teleprogramming: Towards delay-invariant remote manipulation". To appear in *Presence: Teleoperators and Virtual Environments.*
2. J. Funda and R. P. Paul, "Efficient control of a robotic system for time-delayed environments". In *Proceedings of the Fifth International Conference on Advanced Robotics*, pp. 219–224, Pisa, Italy, June 1991.
3. T. Lindsay, J. Funda and R. P. Paul, "Contact operations using an instrumented compliant wrist". Presented at the *Second International Symposium On Experimental Robotics*, Toulouse, France, June 1991.
4. J. Funda and R. P. Paul, "Model-based, delay-tolerant teleoperation in unstructured environments". In *Proceedings of the IEEE Melecon '91 Mediterranean Electrotechnical Conference*, Ljubljana, Yugoslavia, May 1991.
5. J. Funda and R. P. Paul, "Remote control of a robotic system by teleprogramming". Presented at the *IEEE International Conference on Robotics and Automation*, Sacramento, CA, April 1991.
6. J. Funda and R. P. Paul, "Teleoperation for remote or hazardous environments". Presented at the *Workshop on the Application of Robotics to Handling of Hazardous Wastes, Materials, and Equipment*, IEEE International Conference on Robotics and Automation, Sacramento, CA, April 1991.
7. J. Funda and R. P. Paul, "Teleprogramming: overcoming communication delays in remote manipulation". In *Proceedings of the First Workshop on Mobile Robots for Subsea Environments*, Monterey, CA, October 1990.
8. R. Paul, J. Funda and T. Lindsay, "Teleprogramming for autonomous underwater manipulation system". In *Proceedings of the Eighth Annual Intervention '90 Conference*, pp. 91–95, Vancouver, Canada, June 1990.
9. J. Funda and R. P. Paul, "Teleprogramming: overcoming communication delays in remote manipulation". In *Proceedings of the*

IEEE International Conference on Systems, Man and Cybernetics, pp. 873–875, Los Angeles, CA, November 1990.

10. J. Funda and R. P. Paul, "Teleprogramming for manipulation by autonomous underwater vehicles". To appear in *7th International Symposium on Unmanned Untethered Submersible Technology*, University of New Hampshire, September 1991.

11. J. Funda and R. P. Paul, "A symbolic teleoperator interface for time-delayed underwater robot manipulation". To appear in *Proceedings of Oceans 1991 Conference*, Honolulu, Hawaii, October 1991.

6 References

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

[2] 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 Hampshire, June 1987.

[3] 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.

[4] 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.

[5] 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.

[6] 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: Towards Delay-Invariant Remote Manipulation

Teleprogramming : Towards Delay-Invariant Remote Manipulation

Janez Funda
General Robotics and Sensory Perception Laboratory
University of Pennsylvania, Philadelphia, PA 19104

August 15, 1991

Abstract

This dissertation addresses the problem of remote manipulation in the presence of communication delays. Delays occur with earth-based control of a robotic system in space or when an untethered submersible system is controled from the surface via an acoustic communication channel. The resulting delay in obtaining position and force feedback from the remote slave arm(s) makes direct teleoperation infeasible.

We propose a new control methodology, called *teleprogramming*, which allows for efficient control of a robotic system in the presence of significant feedback delays without substantial degradation in the overall system performance. A teleprogramming system allows the operator to kinesthetically, as well as visually, interact with a graphical simulation of the remote environment and to interactively, on-line teleprogram the remote manipulator through a sequence of elementary symbolic instructions. These instructions are generated automatically by the operator's station software in real time as the task progresses. The slave robot executes these symbolic commands delayed in time and, should an error occur, allows the operator to specify the necessary corrective actions and continue with the task.

Teleprogramming offers a practical compromise between the ultimate and the feasible, and provides an effective and time-efficient approach to remote manipulation. Advantages of teleprogramming over existing control methodologies include a relatively modest required level of remote site autonomy, and the absence of the need for complex automatic task planners and preprogrammed error recovery modules.

This document describes the overall conceptual architecture of teleprogramming and presents a detailed treatment of all major components a teleprogramming system. An operational prototype system is described and preliminary experimental results are reported. Experimental results have confirmed the validity and feasibility of the teleprogramming control methodology. Sustained and efficient remote control of a robot manipulator in the presence of a five second feedback delay was successfully accomplished for simple contact tasks.

Contents

Chapter 1

Introduction and Problem Statement

1.1 Introduction

While robots have failed to become the logical conclusion of the industrial revolution, robotics has had good success in industrial environments. This is due to the fact that factory automation is characterized by a structured, well-controlled, and static work environment and that the tasks to be automated are often relatively simple, repetitive, and do not require sophisticated environmental interaction on the part of the robot. The work pieces are presented to the robot in precise position and orientation at precise time intervals and the robot blindly and tirelessly executes its task. No attempt at understanding its actions or even recovering from accidental errors is usually made in these situations.

Factory automation exemplifies an application where robots have been brought into the production process to relieve people of repetitive work as well as to increase productivity, efficiency, and in some cases quality of labor. A parallel application of robotics has been in environments where people can not perform work themselves. Examples of such applications are performing work in areas of biological, chemical, or nuclear contamination, which is hazardous or detrimental to humans, clean room facilities, where people may disturb the precisely controlled production environment, etc. Similarly, robotic technology has been introduced in applications, such as space and undersea exploration, where the cost and risk of manned missions is often prohibitive.

The latter class of applications is characterized by unstructured and often *a priori* unknown working environments, as well as non-repetitive tasks, where the robotic system is required to interact with the environment and react intelligently to the dynamically changing environmental circumstances. Consequently, if robotic devices are to be effective in

such situations, they must possess a much greater degree of sophistication than their less ambitious industrial counterparts. The development of such autonomous robotic devices has proved to be a great challenge. Some of the major difficulties relate to the need to (a) adequately model the complexities or real world physics and dynamics, (b) develop general strategy planning techniques, transcending any particular limited application domain, (c) anticipate and correct the multitude of possible error conditions arising during task execution, (d) provide the robot system with real-time reactive and adaptive behavior to accommodate the changes in the surrounding environment, (e) allow for on-line learning and performance improvement through "experience", etc. The classical approach to tackle these problems has been to introduce problem solvers and expert systems as part of the remote robot workcell control system.[1] However, such systems tend to be limited in scope and application domain in order to remain intellectually and implementationally manageable. They are normally too slow to be useful in real-time robot task execution, and by virtue of their limited and discretized knowledge of the world and a predetermined set of inference rules generally fail to adequately model the complexity and generality of real world interactions. Likewise, detecting, and correcting all possible run-time error conditions poses a major obstacle in the development of autonomous robotic systems. This is a difficult problem even in well-structured industrial environments and becomes hopeless in situations where the environment is only partially known and significant modeling, sensing, and control errors exist. These error conditions must be anticipated ahead of time and appropriate detection and recovery routines must be programmed prior to deployment of the system. This of course implies that error handling is only as complete as the programmer's mental model of the application environment. Finally, recovery procedures are themselves subject to errors and the error handling process thus suffers from a combinatorial explosion in both volume and complexity.

Consequently, teleoperation remains the most reliable option for performing work in situations where people are forced to be physically separated from the actual work environment. 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. The early prototypes were essentially mechanical pantographic linkages of kinematically similar master and slave arms [Goertz,1954].

[1] We will in this document refer to a robot manipulator, along with its onboard sensors and the supporting (possibly movable) platform, as a "robotic workcell".

Despite their simplicity, they provided for good kinesthetic control of remote manipulation. However, as the spectrum of tasks to be performed under teleoperated control expanded, the need for kinematically dissimilar masters and slaves became increasingly more apparent. This was necessitated by applications where the operator's actions (displacements and forces) needed to be scaled upward or downward into the task domain. Similarly, the mechanical linking precluded indexed or relative-motion control which would often be more natural to the operator. This led to the introduction of electrically actuated teleoperators under computer control [Goertz,1954], which removed the limitations of the mechanical linkages, but which were unable to provide kinesthetic feedback to the operator [Goertz,1963]. The development of bilateral, force-reflecting systems once again allowed the operator to "feel" the remote environment through the teleoperator. Since then, sophisticated teleoperated systems have been designed and built, offering high dexterity of manipulation and low fatigue on the part of the operator [Ballard,1986], [NASA,1988], [Schenker,1987], [Hirzinger,1989], [Hatamura,1990]. These systems feature dissimilar master and slave manipulators, coordinated two-arm telemanipulation, 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 for target tasks ranging from molecular docking to mining. 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.

1.2 Problem Statement

1.2.1 Communication Delays

Direct 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) 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) as well as the feedback from the slave back to the operator (visual and kinesthetic information) are delayed. This adversely affects the efficiency 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, when the feedback arrives. A

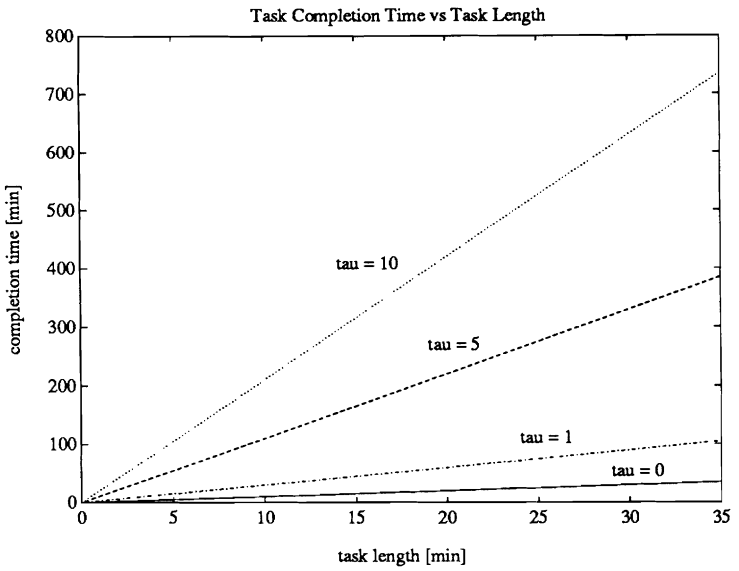


Figure 1.1: Total task completion time versus task length for $t = 1$ second and different values of the communication delay τ (in seconds).

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 feedback (resulting state) from the remote environment to determine the effect of each motion.

1.2.2 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 and other delays in the system.[2] Let A denote a task, which takes T_{task} time to execute without delay, by executing elementary commands, each of which takes on average t time to execute. Then the total time to execute the task in the delayed environment by using the move-and-wait approach, is

$$T_{\text{total}} = (1 + 2\frac{\tau}{t})T_{\text{task}} \tag{1.1}$$

Figure 1.1 illustrates the effect of communication delays on the total task completion time using the move-and-wait strategy for $t = 1$ second and three different values of the

[2]We will throughout this work refer to this lumped delay as the *communication* delay or the *feedback* delay.

communication delay τ. The bottom-most line ($\tau = 0$) in Figure 1.1 corresponds to the case where there are no delays in the control loop at all, i.e., $T_{\text{total}} = T_{\text{task}}$.

Thus, in view of Eq. (1.1), if we consider a twenty minute task ($T_{\text{task}} = 20$ min), with an elementary command execution time of 1 second ($t = 1$ sec) and with a feedback delay time of 10 seconds ($\tau = 10$ sec), the total time to execute the task would be 7 hours! Clearly this is not satisfactory for most applications.

1.2.3 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 severely degrade or even destroy the sense of telepresence during contact manipulation. This is a direct consequence of the fact that both the video signal, as well as the information about the forces experienced by the slave arm, are delayed by 2τ. While delays in receiving both visual and kinesthetic information cause a problem, the delay in receiving force information has been shown to be perceptually more significant [Ouh-young,1989]. Physiological studies have shown that the neurological control of human musculoskeletal movements operates at the rate of approximately 5 Hz ([Stark,1987], [Brooks,1990]), and that a time delay of approximately 300 ms ($\approx 1/3$ sec) is clearly perceptible and distracting to humans [Stark&Kim,1988]. Consequently, feedback delays approaching one second severely destabilize the performance of a human operator relying on real-time feedback information [Ferrell,1966], [Black,1971], [Bejczy&Kim,1990].

Unfortunately, in space and undersea applications, communication delays often exceed this threshold. Round-trip communication delays between the ground station and a slave workcell in low earth orbit (e.g., Space Shuttle) are normally in the range of 2 to 8 seconds, 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 teleoperated work is to be performed in shallow space (e.g., moon), then delays approaching or exceeding 10 seconds should be expected. Similarly, substantial delays arise during remote control of autonomous underwater vehicles (AUV) and their on-board manipulator arms. Acoustic communication links are normally established between the AUV and the surface ship (or a land-based operator's station), and with the sound transmission underwater being limited to 1460 m/s, the round-trip time delay over a distance of 1 mile therefore exceeds 2 seconds [Sheridan,1990].

1.3 Research Goals

The goal of this research is to address the issue of communication delays in remote manipulation and to design, as well as experimentally verify, a new control methodology, capable of controlling a remote robotic workcell in the presence of significant feedback delays without a substantial degradation of the overall system performance. In particular, we will develop a delay-tolerant control strategy, which will allow for continuous and efficient control for feedback delays up to approximately 20 seconds. In light of the discussion in Section 1.2.3 above, this delay interval should include most, if not all, earth-based, ocean-based, as well as shallow space telerobotic applications.

According to basic control theory, sustained, stable closed-loop control in the presence of a significant 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 ability to perform useful and effective work over large distances. We will in this work present and demonstrate a solution to this problem, based on the concept of *teleprogramming* the remote robotic workcell. The basic components of the *teleprogramming* control methodology include a real-time graphical simulation of the remote environment, real-time extraction of approximate kinesthetic feedback from the graphical simulation, and on-line automatic generation of elementary task commands to be sent to the slave.

1.4 Outline of the Dissertation

The remainder of this document is organized as follows. Chapter 2 reviews related work by other researchers and describes the state of the art in time-delayed manipulation technology. In Chapter 3 we offer a brief description of our proposed control methodology and its major conceptual components. Subsequent chapters address individual building blocks of a *teleprogramming* system in detail. Issues related to the master input device and control of the simulated slave are discussed in Chapter 4. The graphical simulation of the remote environment is addressed in Chapter 5. Geometric constraint enforcement and computation of the kinesthetic feedback to the operator is detailed in Chapter 6. In Chapter 7 we detail the symbolic command generation, and Chapter 8 addresses the issues of command translation and execution at the remote site. Experimental results are reported in Chapter 9. The concluding remarks, the contribution of this work, and the future research directions are summarized in Chapter 10.

The first of the four appendices (Appendix A) reviews some of the basic results in kinematics, which are used in the text. Next, Appendix B gives a brief description of the syntax and semantics of the low-level symbolic command language. Appendix C describes the details of the hardware and software architecture of the prototype *teleprogramming* system, which was used to verify the conceptual design and ideas. Finally, Appendix D lists a portion of a program, generated by our experimental system during one of the test runs.

Chapter 2

Background and Related Work

The following sections offer a brief review of related work in three main areas, addressed by this dissertation: approaches to overcoming communication delays in remote manipulation, research in providing the operator with a real-time approximation to the delayed force information, and automatic generation of robot control programs.

2.1 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)
- 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 modeling up-link and down-link delays by augmenting the dynamic state-space model of the system (environment + slave) — delays are modeled as delay lines on the output and introduce (a potentially large number of) additional states [Hirzinger *et al.*,1989]

- control theoretic approaches, such as modeling a teleoperated system as a two-ported network, and devising control laws which attempt to cancel the effects of feedback delays [Anderson&Spong,1988], [Raju,1988], [Hannaford,1989]

- 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 recovery

Unfortunately, none of the existing approaches to overcoming communication delays in remote manipulation has proven to be entirely satisfactory. In the presence of delays in excess of one second, simple move-and-wait strategies become impractical for most applications. At the other extreme, full autonomy at the remote site is beyond the state of the art. At present, it seems that the integration of the available, however limited, remote site autonomy, carefully designed control laws, and sophisticated operator station based predictive displays offers the best compromise between the desirable and the feasible.

Supervisory control [Ferrell&Sheridan,1967] provides a broad conceptual framework for the design of effective telerobotic systems despite communication delays. The central idea, as outlined above, is to distribute decision making and control between the operator's station and the slave workcell in favor of the remote site, to the extent possible. This results in greater independence of the supervisory and the remote control loops, the two now being coupled only through a low-bandwidth, asynchronous exchange of commands (from the operator to the remote workcell) and state information (from the remote workcell to the operator). However, the realization of the full promise of supervisory control methodology in remote manipulation has been hampered by the difficulty of adequately automating the low-level environmental interaction at the remote site. This relates primarily to the difficulty of incorporating sufficiently sophisticated knowledge of the world and models of contact physics into on-board reasoning systems, as well as designing corresponding manipulator control algorithms, capable of operating reliably in unstructured and *a priori*

unknown environments. A related problem, as discussed in Section 1.1, is the need to anticipate, detect and provide preprogrammed corrective actions for the multitude of possible error conditions arising during subtask execution in order to support the necessary level of autonomy at the remote site. With error handlers themselves being subject to errors, the error handling code can easily come to dominate an application program as well as the programming effort itself.

Many approaches to time-delayed remote manipulation rely on predictive displays as a means of providing approximate, partial feedback to the operator in real time. In such systems, the operator's station makes 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 MIT [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 can 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].

2.2 Providing Kinesthetic Feedback

It is well established that force-reflection dramatically improves the sense of *teleperception* 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 artificial 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 for one-dimensional force reflection.

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 force information from the interaction between the simulated slave and its environment. 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].

2.3 Automatic Robot Programming

While robots can be used to perform non-contact tasks, such as inspection or surveilance, the majority of robotic manipulation tasks require that the robot physically interact with its environment. However, this interaction with the environment complicates control of robot manipulators, due to oscillatory dynamic effects on contact and time-varying, high-frequency interaction between the manipulator's end-effector and the environment during contact manipulation. These effects are difficult to model accurately and can result in control instabilities. Consequently, sophisticated control strategies are needed to deal with contact manipulation and a variety of 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], and others (see [Whitney,1987] or [An&Hollerbach,1989] for an overview of force control techniques).

Perhaps the most popular of these control strategies is the hybrid position/force control method. The hybrid position/force approach separates the robot's Cartesian d.o.f. 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 thought 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 can thus be defined as a sequence of task frame specifications and position/force trajectories along the artificially and naturally constrained d.o.f., respectively, in the current task frame.

While force control enables the robot to perform contact manipulation more stably and reliably, programming such applications 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.

More 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 automati-

cally 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).

Chapter 3

The Teleprogramming Solution

3.1 General Approach

There is a clearly established and growing need to perform work in remote environments, unreachable or unsafe for humans. Of course, a purely autonomous manipulative capability would provide the solution to the problem. However, as discussed in Section 1.1, its realization remains beyond the state of the art in robotics. On the other hand, as we saw in Section 1.2, direct teleoperation degrades to move-and-wait control in the presence of any significant feedback delays in the control loop. Consequently, intermediate solutions must be explored and Chapter 2 outlined some of the approaches proposed by other researchers.

We propose to solve the problem of time-delayed remote manipulation by a new control methodology, based on incremental *teleprogramming* of the remote workcell. Figure 3.1 illustrates the high-level view of the *teleprogramming* concept. As in supervisory control, the low-bandwidth human-master-slave control loop is separated into two locally closed, high-bandwidth control loops, which exchange information over the low-bandwidth, delayed communication link. However, unlike supervisory control, the *teleprogramming* control paradigm requires a relatively modest amount of autonomy at the remote site and relies on a different type of information exchange between the operator's station and the remote site, as explained below.

Teleprogramming provides a practical solution to time-delayed remote manipulation by combining the power of a graphical previewing display with the provision of real-time kinesthetic feedback, to allow the operator to interactively, through a bilateral kinesthetic coupling with a virtual environment, define the task to be performed remotely. The locally closed, high-bandwidth feedback loop at the operator's station allows for stable interaction

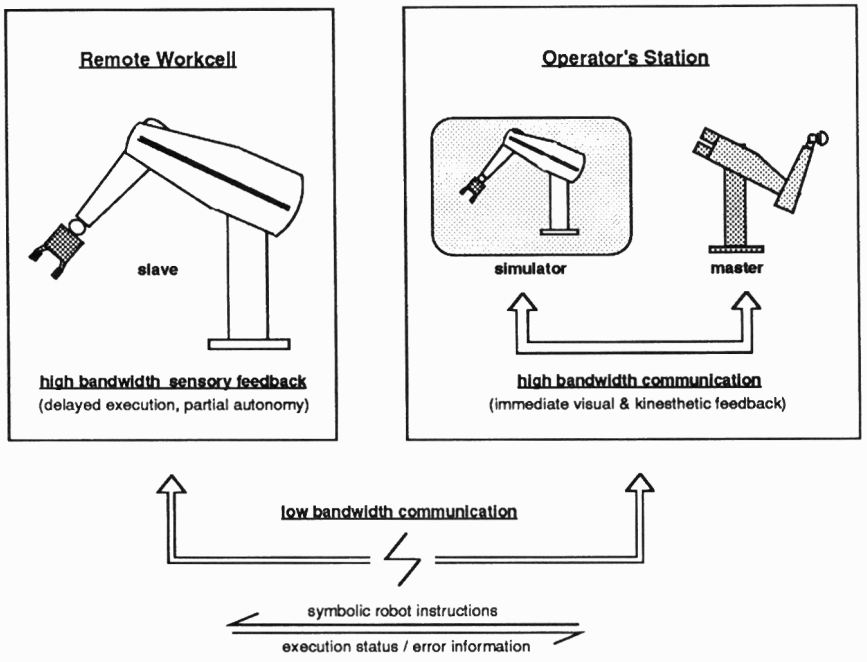


Figure 3.1: A high-level view of *teleprogramming*.

between the operator and the simulated task environment, and the immediate visual and kinesthetic feedback provide for a strong sense of *teleperception*. As the operator performs the task in the simulated environment, the system continuously monitors the operator's actions and generates a stream of symbolic robot instructions, capturing all essential features of the task in progress. The action interpretation and command stream generation process is guided by *a priori* information about the nature and goals of the task. The resulting instructions are symbolic in nature and at the level of guarded and compliant motion primitives to allow for discrepancies between the world and the operator's station based model. The instructions are generated automatically, on-line, as the task progresses and are sent to the remote workcell incrementally, as they become available. The remote site receives them a transmission delay later, translates them into the local control language, and executes them (delayed in time) under the control of a local high-bandwidth sensory feedback controller. Due to modeling, sensing, and control errors, execution failures will inevitably occur in the remote environment. On detecting an error, the slave sends all relevant information about the error state to the operator's station. There this information is used to alert the operator to the error condition, properly adjust and update the graphical world model, and allow the operator to specify the necessary corrective actions and proceed with the task.

A more detailed overview of the *teleprogramming* control paradigm is given in Figure 3.2, which illustrates all the major components of the conceptual system architecture and indicates the basic inter-relationships. In the following sections we will outline the functionality of these components in order to present a comprehensive view of the proposed control methodology. Detailed treatment of each component is presented in Chapters 4 to 8.

3.2 The World Model

We assume in this work that we are manipulating in an *a priori* unknown environment. Upon arrival to the designated work area, the remote workcell obtains the initial description of the environment through the use of its on-board sensors, such as vision cameras, sonars, or range scanners. This sensor information is then sent to the operator's station, where it is used to construct an initial geometric model of the remote area. This is a difficult problem in general, involving sensor fusion, multi-stage image processing, and segmentation of the final regions into three-dimensional objects [Besl&Jain,1985], [Bolle&Vemuri,1991]. While automating many of the stages of this process is within the state of the art of computer vision and image processing, it may be difficult to obtain a high-level segmentation, consistent with the operator's mental model of the scene, in a purely automatic fashion. This is particularly true if the original data is noisy and of poor quality, which may very well be the case with data, such as undersea vision images. Likewise, occlusions in cluttered environments result in incomplete data, further complicating the segmentation process.

As this geometric model is constructed only once at the beginning of a *teleprogramming* session, we propose that the operator interact with the segmentation process and aid the system in constructing a model of the environment, that is consistent with the operator's best estimate of the nature and relationships of objects in the images. Therefore, the output of this stage is an unambiguous description of the environment in terms of identifiable objects, which in turn are described in terms of faces, edges, and vertices. Such descriptions can then be converted into standard representations, such as polyhedral models, constructive solid geometrical (CSG) models, generalized cylinder models, etc. [Srihari,1981], [Aristides&Requicha,1980], [Badler&Bajcsy,1978]. By augmenting this graphical world with a corresponding model of the manipulator workcell itself, we obtain a complete geometric representation of the remote environment, which can be displayed, animated and manipulated in real time using standard computer graphics techniques [Pentland,1986].

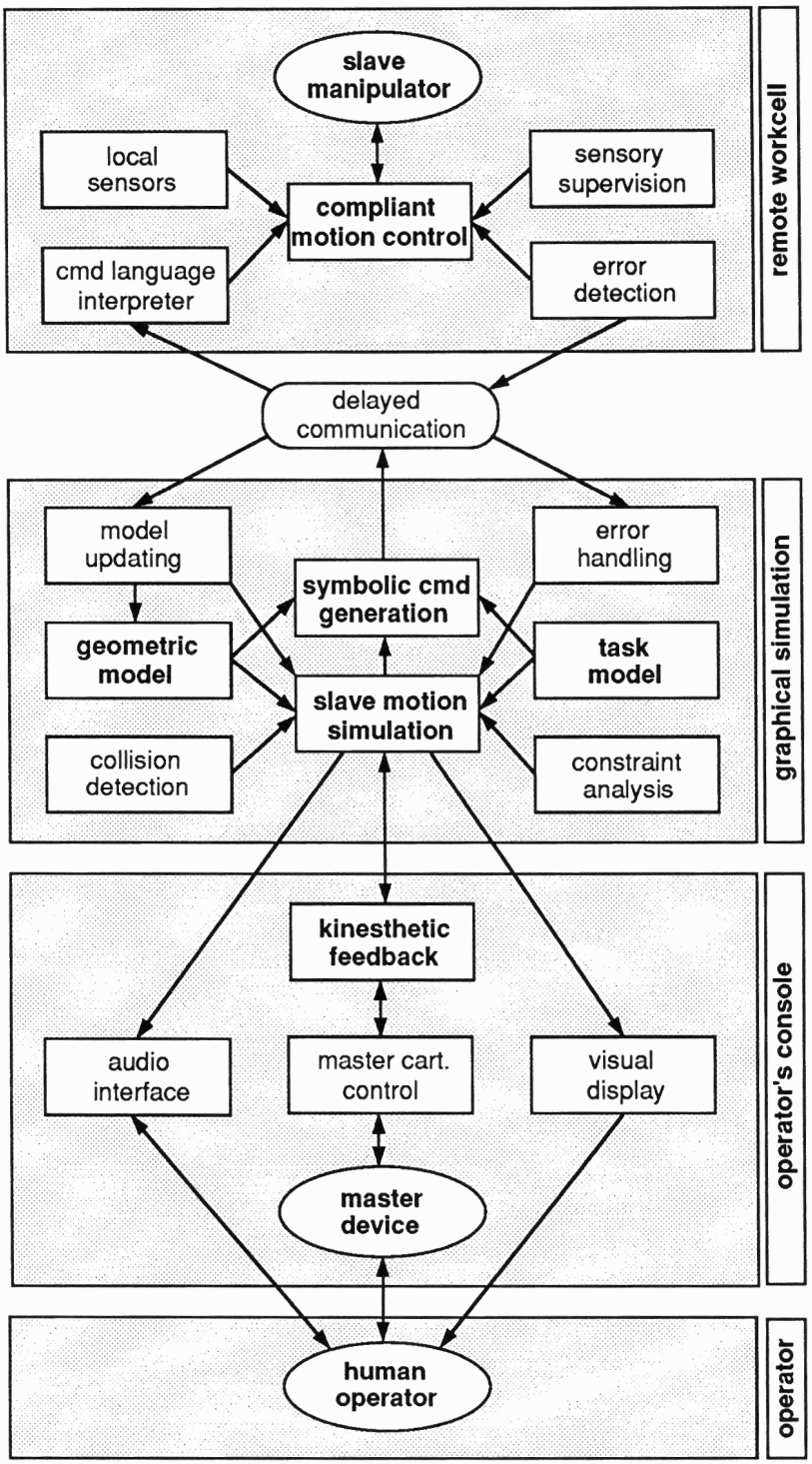


Figure 3.2: The conceptual organization of *teleprogramming*.

Sensor imperfections will invariably introduce errors into the initial data and consequently the resulting model. It is important that adequate models of sensor characteristics exist to estimate and at least bracket the positional and orientational uncertainties in the resulting world model. This information will later be used both by the symbolic command generation module (Section 3.5, Chapter 7), as well as by the on-line model refinement process (Section 3.8).

3.3 The Graphical Simulator

Having constructed a three-dimensional geometrical model of the remote environment and the workcell, we now need to allow the operator to interact with this simulated world and specify tasks to be performed by the actual workcell. Toward this end, a 6 d.o.f. master input device is interfaced to the graphical display, allowing the operator to control positional and orientational parameters of the simulated workcell.

Because we are presumably operating in unstructured and largely unknown surroundings, many of the dynamic parameters of the remote environment, such as masses, inertial parameters, and frictional properties, will not be known *a priori*. This, along with the difficulty of adequately modeling effects, such as hydrodynamics and buoyancy in underwater applications, suggests that we employ a non-dynamic, kinematic simulation of the remote environment, including the slave robot.

The primary task of the graphical simulator in a *teleprogramming* system is to provide a real-time, realistic graphical animation of the slave workcell operating in the simulated environment under operator's control. Secondly, the simulator software continuously monitors the slave robot and any object in its grasp for collisions or contacts with the environment. The system distinguishes between desired and undesired contacts. Desired contacts will normally occur between the slave's end-effector or an object it is currently holding, and some part of the remote environment involved in the execution of the task. Undesired collisions, on the other hand, are all other collisions and will normally involve some non-effector part of the slave robot and an environmental obstacle. The information as to which object pairs are expected to come into contact during the execution of a given task is part of the task model, discussed in Section 3.6.

Each commanded incremental positional displacement to the simulated slave is checked to see if it causes a collision between any of the object pairs. If so, the offending motion is modified by computing the fraction of the commanded displacement, which results in a

non-penetrating configuration, placing the most deeply penetrating object pair exactly in contact. For each new contact the system records the necessary information to uniquely and unambiguously describe the contact geometry. This information is updated at each simulation step and is used by the motion restriction and kinesthetic feedback computation modules (Section 3.4, Chapter 6), as well as by the command generation process (Section 3.5, Chapter 7).

3.4 Motion Restriction and Kinesthetic Feedback

When the operator brings the simulated workcell into contact with the environment, the commanded motion of the slave manipulator is appropriately modified to prevent penetration of environmental surfaces, and the geometric information describing the contact is added to the list of all currently active contacts (Section 3.3, Chapter 5). While the operator remains in contact with the environment, the motion constraints resulting from these contacts must be enforced on subsequent commanded motions to the slave manipulator in order to produce correct and realistic motion of the simulated slave. This is accomplished by computing the set of independent, orthogonal constraints on the motion of the slave workcell corresponding to the current contact set and restricting (i.e., modifying) the operator's commanded motions of the slave manipulator with respect to this constraint set. This allows the simulated slave to slide along surfaces, follow edges, reach corners, reorient its end-effector or the grasped object while in contact, etc.

Secondly, this constraint set is also used to provide the operator with a real-time sense of kinesthetic interaction with the environment. This is accomplished by using the same constraint set, derived from the graphical simulation, to restrict the motion of the master device as well. The operator-supplied commanded motions are therefore modified in accordance with the currently active Cartesian motion constraints and the master arm is actively servoed to resist attempted motion in the constrained directions. This allows the operator holding the master device to *kinesthetically* feel the impact of contacting a surface, reaching a corner, pivoting about an edge, etc. This feature is very important as it affords the operator a sense of kinesthetic teleperception despite the fact that the actual force information is delayed, and thus not available for real-time reflection to the operator.

3.5 Generating Symbolic Motion Commands

So far the operator can perform tasks in a virtual environment by visually and kinesthetically interacting with the simulation of the remote site. The next important feature of the *teleprogramming* system is that the operator's station software is capable of monitoring the operator's activity in this simulated environment and extracting from it a stream of elementary robot instructions that capture all essential features of the task in progress. This action interpretation process is guided globally by the *a priori* information about the nature and goals of the task (Section 3.6). At a more immediate level, the system monitors the elapsed time, the motion and force trajectories of the simulated slave and manipulated objects, as well as the contact state information, to generate a stream of instructions, describing the activity in the simulated environment. As the model of the remote environment 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, the system generates instructions of the type "move along a given direction until contact" (guarded motion) or "move along a given feature while maintaining contact" (compliant motion). These instructions are based on the hybrid position/force model of robot's interaction with the environment and have model error tolerances built into the motion parameters. Due to the kinematic nature of the simulation, the necessary dynamic parameters, such as frictional coefficients or compliance forces, are supplied symbolically, rather than numerically.

Aside from these low-level instructions, the system should also recognize and correctly interpret the operator's intent to initiate special-purpose subtasks, such as for example a grasping action. Similarly, the system should allow the operator at any point during the execution of a task to specify (kinesthetically and orally) a sequence of actions to be encapsulated as an unparameterized, unnamed, one-time "procedure", and be executed repeatedly until some terminating condition is reached. The decision-level support, allowing the command generation module to correctly disambiguate and interpret operator's input, is provided by the task model, discussed next. We will address command generation in more detail in Chapter 7.

3.6 The Task Model

In order for the simulator software to correctly interpret the operator's actions, it may require some knowledge of the general goal of the task in progress or be aware of special characteristics of the task. For instance, a sequence of rapid contact changes may be interpreted either as noisy data (and thus be smoothed) or a purposeful action, such as tapping, scraping, or rocking (in which case it should be kept intact). Similarly, a highly irregular path of an object during a sliding motion could be taken as unintended or it could correspond to a motion such as polishing or sanding. In order to disambiguate between such interpretations, the system needs additional information about the task, and in particular, the types of expected primitive motions (e.g., pick and place, polishing, pounding), which are to be expected during execution of the upcoming task. Other relevant information includes a list of environmental objects and features, which are expected to come into contact with the slave arm during the task. This information can be used by the graphical simulator (Section 3.3, Chapter 5) to efficiently manage the collision computation load. Similarly, an indication of the relevant relationships between environmental objects, involved in the execution of the task (e.g., which objects are rigidly attached to their support, which ones are detachable, pushable, etc.), can be used by the simulator and the command generation process (Section 3.5, Chapter 7).

At a more sophisticated level, the task model should encode the knowledge of the special-purpose actions and iterative procedures, mentioned in Section 3.5 above, as well as their associated terminating conditions. This information can then guide the command generation process to correctly detect and interpret such actions, when they appear in the input stream. The audio interface can be used in conjunction with this feature to ensure proper interpretation and facilitate on-line adjustments in the definition and execution of these actions, if necessary. The task model may be also used by the system to automatically and dynamically adjust the viewing angle, zoom, and other viewing parameters of the simulated remote environment. Based on the geometry of the task environment and basic knowledge of the current subtask, the system can select the viewing parameters so as to provide the operator an unoccluded and intuitive view of the work area throughout the execution of the task. Similarly, drawing on the knowledge of the dexterity of the basic subtasks to be expected during a given task, the system can automatically adjust the scaling of the commanded master device displacements or exerted forces into the slave task space. Naturally, the operator should be able to override or disable the automatic view-adjustment module, as well as the automatic displacement/force scaling, if so desired.

The task information discussed above can be gathered either by using a pre-prepared database, by querying the operator prior to the task, or by maintaining an on-line dialogue with the operator. Any combination of the above methods may also be used. In particular, the last approach can be used as a run-time supplement allowing the operator to augment and modify the current task information while the task is in progress. Likewise, run-time on-line dialogue with the operator may be used by the command stream generator to request additional information from the operator when her intent is unclear.

3.7 The Remote Workcell

The operator's station sends the symbolic instructions to the remote workcell continuously as the task progresses. These instructions are received at the slave site a transmission delay later. The slave high-level control software then parses incoming command strings, substitutes numerical values for the symbolically specified dynamic parameters, translates them into the local control language, and passes them to the low-level controller for execution.

It is crucial that the remote workcell be capable of some autonomy in executing the commanded motion primitives. 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, as well as local sensory and control errors. In view of the hybrid force/position control paradigm, external forces and torques, acting on the slave manipulator, must be available to the local control algorithm to provide for compliant and locally adaptive response in contact motion. Additionally, sensory information from other external sensors (such as TV cameras, sonars, or range scanners) may be gathered, fused into a consistent representation of the state of the system and the environment, and integrated with the control algorithm. This is particularly crucial as the commanded motions are derived from imperfect operator's station based model of the remote environment. Consequently, the control of the slave workcell must exhibit sufficient flexibility to accommodate the majority of such discrepancies without execution failures. A good part of this flexibility has already been designed into the control language describing the actions to be performed (Section 3.5, Chapter 7). However, additional mechanisms, such as robust, low-level, sensor-based controllers, smart end-effectors, local sensory reflex loops, passive end-effector compliance, etc., may further enhance the performance and reliability of the workcell.

During task performance, the slave workcell must monitor its execution status, verifying

that elementary motions terminate correctly or identifying that an execution error has occurred. In either case, this information should be propagated to the operator's station to report the status of the remote workcell. We will discuss the remote workcell command execution management and controller design in more detail in Chapter 8.

3.8 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 still 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 signals the occurrence of an error state to the operator's station, which in turn alerts the operator and interrupts the task. Alerting the operator can be done through a variety of visual or auditory means, such as flashing the display, issuing synthesized voice warnings, etc. (see Figure 3.2). If the error state is not clear from the information supplied to the operator by the slave workcell, the operator may initiate various exploratory procedures and maneuvers at the remote site to clarify the resulting state of the workcell and the environment. Both contact (force based) and non-contact (vision or sonar based) exploratory actions can be invoked in order to gather additional information about the error configuration. When the state of the slave and the remote environment has been determined, the graphical model at the operator's station is updated to reflect the error configuration and the operator can proceed by taking appropriate corrective actions and continue 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.

Note that the above facility of executing local exploratory procedures to determine the state of the environment is useful 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. Similarly, in order to ensure that dynamic changes in the environment, caused by external agents or influences (e.g., winds, water currents, etc.) are properly reflected in the operator's station based world model, such exploratory procedures could be invoked periodically or whenever there is reason to suspect that unmodeled dynamic changes have occurred in the remote environment.

Going a step further, we may choose to update the operator's station model continuously as the task progresses. While interacting with the environment, the slave manipulator comes into contact with various objects and features in the environment and can thus precisely determine their position and orientation relative to its local reference coordinates. As the task progresses, this information can be accumulated and used to provide for on-line refinement of the operator's station based model of the remote environment. However, as this information will normally be discretized and local, care must be taken in propagating this local corrective information through the model.

3.9 Summary and Evaluation

The *teleprogramming* concept, outlined above, distributes decision-making and control between the human operator (who provides for task planning and error recovery) and the remote workcell control system (which provides for low-level autonomous 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 autonomously by the slave. In view of our earlier discussion of the total task completion times using the move-and-wait strategy (Section 1.2.2), we can now compute the corresponding behavior for the teleprogramming paradigm. If n is the number of elementary commands that are executed by the slave workcell, on average, without entering an error state, then the time to perform a task is given by

$$T_{\text{total}} = (1 + 2\frac{1}{n}\frac{\tau}{t})T_{\text{task}} \tag{3.1}$$

Clearly, in the interest of minimizing the overall completion time, the desired behavior of the system is

$$nt \gg \tau \tag{3.2}$$

Figure 3.3 illustrates total completion times (T_{total}) versus task length (T_{task}) for $\tau = 10$ seconds, $t = 1$ second, and three different values of n. Note that the case of $n = 1$ corresponds to the move-and-wait strategy (Eq. (1.1)) and the solid line on the very bottom

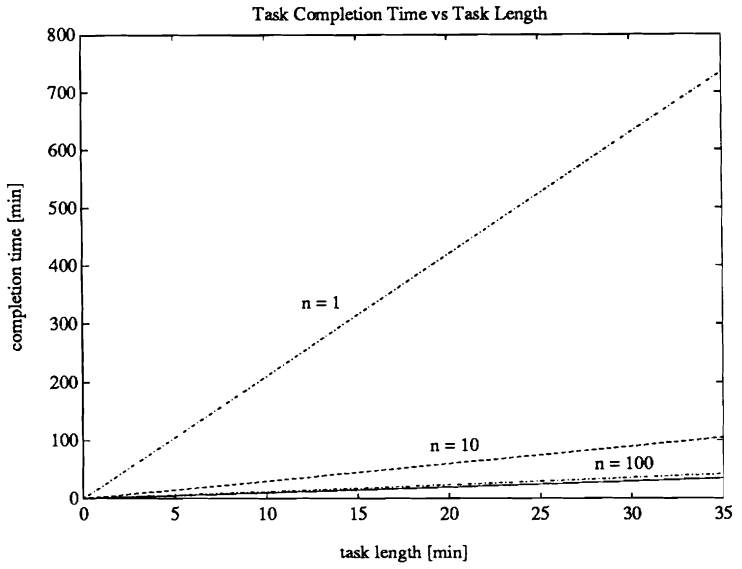


Figure 3.3: Total task completion times versus task length for $\tau = 10$ sec, $t = 1$ sec, and three different values of n.

corresponds to direct teleoperation with no delay ($T_{\text{total}} = T_{\text{task}}$). The figure suggests that even a relatively modest amount of remote site autonomy, e.g., $nt = \tau$, dramatically improves the system's throughput (task completion times), whereas autonomy at the level of $nt = 10\tau$ results in completion times which are only slightly longer than the times obtained with direct teleoperation when there is no delay in the control loop at all. For shallow space and underwater applications we normally have $\tau < 10$ sec, and so

$$nt < 100 \text{ sec} \tag{3.3}$$

which is clearly within the state of the art of modern robot control strategies. This suggests that the *teleprogramming* control paradigm can be successfully applied in shallow space and underwater environments, effectively eliminating the adverse effects of transmission delays, and allowing for near-optimal remote control of robotic workcells in these environments.

Chapter 4

Master and Control of the Simulated Slave

4.1 The Master Device

The choice of the master device is a crucial factor in operator's comfort, her physical and mental load, and dexterity of the achievable manipulation. In selecting a master device, the following core requirements should be considered (see [Fischer,1990] for a more comprehensive description of desired master device specifications):

1. the master should be spatially mobile, affording the operator unconstrained maneuverability within a workspace volume that is comfortable to a human operator

2. the master should allow for specification of arbitrary positional and orientational parameters, and should thus possess at least 6 d.o.f. of motion

3. the master should allow for bilateral control, i.e., force reflection back to the operator, and should thus be actively servoed during operation

4. the kinematic structure of the master should be transparent to the operator, and should be free of control singularities within the normal workspace

5. the kinematic structure of the master should in no way constrain or dictate the kinematics of the slave

6. the master should allow for relative or indexed motion, so that the operator can detach and relocate the master arbitrarily during operation and resume control of the

slave from a new, presumably more comfortable or natural master configuration (see Section 4.2)

7. the master should allow for controllable (i.e., programmable) impedance and input/output displacement/force scaling; it is generally considered best to cancel (passively or actively) the inertial effects of the master and slave arms while retaining a fraction of the inertial effect of the load [Fischer,1990]

8. the master should afford feed-forward (power) bandwidth of at least 10 Hz, and feedback (information) bandwidth in the KHz range [Brooks,1990]

9. the master should possess sufficient velocity and acceleration response to not frustrate the operator, regardless of the corresponding characteristics of the slave (Section 10.3)

State-of-the-art master devices (or hand controllers, as they are commonly referred to) are specifically designed to meet the above requirements and typically feature low inertia, as well as high-fidelity motion input and force reflection [Bejczy *et al.*,1988] [Fancello *et al.*,1988] [Hatamura *et al.*,1990].

4.2 Reindexing Techniques

A particularly crucial 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 or implementation of the master device. Moreover, if the master device does possess kinematic singularities in its workspace, the master controller must ensure that the corresponding configurations are never reached. The control techniques, aimed at solving this problem, are often referred to as *reindexing methods*.

In our work we have used a small, standard 6 d.o.f. industrial manipulator as the master device (see Appendix C). While suboptimal in some respects (in particular on the issue of kinematic singularities), this choice meets most of the stated requirements for a good master input device. The problem of numerous orientational, as well as positional singularities led us to address the issue in some more detail. In particular, we have formulated and evaluated three different reindexing schemes

- manual reindexing,

- continuous drift-back, and

- automatic reindexing

We will discuss their advantages and disadvantages in turn. Note that all reindexing methods imply that the display is decoupled from the motions of the master during reindexing.

Manual reindexing: This approach offloads the reindexing responsibility to the operator. The operator must monitor the master's motions and identify that it is approaching a singular configuration. She can then detach the master from the simulation and manually reposition the device to a convenient, non-singular configuration.

This can be easily implemented by allowing the operator to signal the system (via a push button, step pedal, voice command, etc.) that she wishes to reindex the master. The master servo controller is then switched from position (or velocity) mode to torque mode with only gravity compensation torques being applied to the actuators. The master can be then freely repositioned to an arbitrary configuration and the normal servo mode is resumed.

The obvious disadvantage of this reindexing scheme is that it requires the operator to be aware of the master kinematics and pay attention to its configuration. This is clearly an unacceptable additional mental load on the operator, particularly as her visual and mental attention is already focused on the display of the remote task environment.

Continuous drift-back: Here, reindexing is accomplished via a continuous drift back to the master's "home position". The home position can be taken to be the master's configuration on start-up and can be dynamically changed during task execution by using a mechanism, similar to the manual reindexing above. In the continuous drift-back method, the magnitude of the restoring drift velocity is a function of the master's distance (for translations) and twist amplitude (for rotations) from the home position. Thus, denoting the current Cartesian location of the master's kinematic tip (wrist)[1] by ${}^{B}\mathbf{T}_{W}$ and the home position by ${}^{B}\mathbf{T}_{H}$ (both with respect to the base coordinates, $\mathcal{F}_B$), we have

$$ {}^{W}\mathbf{T}_{H} = ({}^{B}\mathbf{T}_{W})^{-1} * {}^{B}\mathbf{T}_{H} \tag{4.1} $$

and the positional and rotational drift velocities can be computed (in master's wrist coordinates) as

$$ \mathbf{v} = \frac{\mathbf{p}^*}{t} f_p(\|\mathbf{p}\|) \quad ; \quad \boldsymbol{\omega} = \frac{\mathbf{k}}{t} f_r(\theta) \tag{4.2} $$

[1]For lack of an established term, we will in the remainder of this document refer to the tip of the kinematic chain of the master arm (i.e., $\mathbf{TG}_m$) as its "wrist", even though this nomenclature may not apply to all kinematic designs.

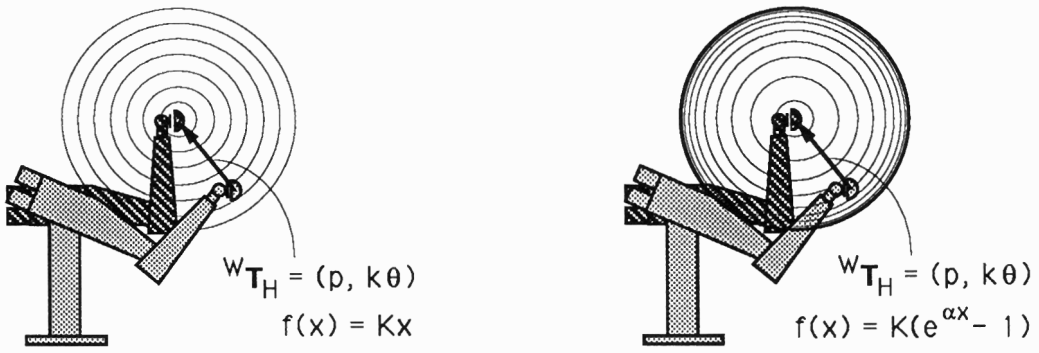


Figure 4.1: Linear and exponential drift-back reindexing schemes.

In Eq. (4.2), $\mathbf{p} = \text{Trans}({}^{W}\mathbf{T}_{H})$, $\mathbf{p}^{*} = \mathbf{p}/\|\mathbf{p}\|$, $\mathbf{k}\theta = \text{Rot}({}^{W}\mathbf{T}_{H})$ with $\theta > 0$[2], and t denotes the Cartesian control sampling interval. f_p, f_r are scalar functions, determining the magnitude of the drift-back rate. Both linear ($f(x) = Kx$) and exponential ($f(x) = K(e^{\alpha x} - 1)$) drift-back rates have been investigated, as shown in Figure 4.1.

This reindexing method relieves the operator of being concerned with the kinematics of the master and, especially for the case of the exponential drift-back, produces a smooth, exponentially decaying drift back to the home position. By choosing α an K appropriately, the master can be constrained to remain within the desired singularity-free radius of its Cartesian origin and produce a "comfortable" rate of return home. However, this in most cases reduces the available workspace of the master, which is typically not spherical. Moreover, the configuration-dependent drift-back interferes with force reflection during contact manipulation. This reduces the fidelity of the kinesthetic teleperception as the reflected force information becomes intertwined with the effort exerted against the operator due to the drift-back. Small force information can be entirely lost while manipulating near the outer boundary of the master's spherical workspace envelope.

Automatic reindexing: In this reindexing mode, the master device monitors its own motions and alerts the operator when it approaches a singular configuration. The operator can be alerted through visual (e.g., flashing the display) or auditory means (e.g., synthesized or prerecorded message playback). The master then automatically returns to the home position and signals to the operator that she may proceed with the task.

[2]For any rotational displacement, two corresponding angle/axis descriptions can be obtained, namely $\mathbf{k}\theta$ and $-\mathbf{k}(-\theta)$. We have here stipulated that the desired solution be the one which yields a positive twist angle θ.

This method combines the advantages of the above two approaches. It relieves the operator of the need to pay attention to the master device while allowing unconstrained motion and high-fidelity force reflection within the singularity-free workspace of the master. Should the master approach a singular configuration, the operator is informed of this and asked to interrupt task specification for the duration of the automatic reindexing maneuver. We have in our experimental system adopted this reindexing strategy (see Appendix C).

4.3 Control of the Simulated Slave

If the master device is mechanically backdriveable, then the desired motion parameters can be specified to it by simply exerting forces against it. The direction and magnitude of the motion command can be derived from the servo position and/or torque errors. However, it is important to note that stiction and friction effects will limit the force threshold (resolution), as well as accuracy, that can be achieved.

Alternatively, a sensor of operator's exerted effort, such as a force/torque sensor, can be integrated into the master device. This provides the commanded motions directly and is insensitive to frictional characteristics of the master device drive train.

We will in this work adopt the latter method and propose a sensor based approach to deriving the input motion command from the operator's exerted effort as illustrated in Figure 4.2. The six-vector of Cartesian raw sensor measurements ${}^{S}\mathbf{F}$ is first low-pass filtered to smooth the data. Despite relatively low minimum requirements on the feed-forward bandwidth (approximately 10 Hz, as discussed in Section 4.1 above), it is critical to sample the data sufficiently fast, so that noisy data can be adequately smoothed without introducing a significant phase lag. To illustrate this important point, consider the first order digital low-pass filter

$$y_k = K u_k + (1 - K)\, y_{k-1} \tag{4.3}$$

whose time constant τ, for a given sampling frequency $1/T$, is given by [Palm,1983]

$$\tau = -\frac{1}{\ln(1 - K)}\, T \tag{4.4}$$

Consequently, in order to properly smooth a noisy signal, a small filter gain K must be offset by a high sampling frequency $1/T$ in order to control the lag, introduced by filtering.

This filtered force information (${}^{S}\mathbf{F}'$) is then scaled into the Cartesian velocity (incremental displacement ${}^{S}\mathbf{D}_m$) of the sensor frame ($\mathcal{F}_S$). The scaling factors correspond to the ratio of the operator's exerted effort (force) versus the resulting flow (master velocity) and

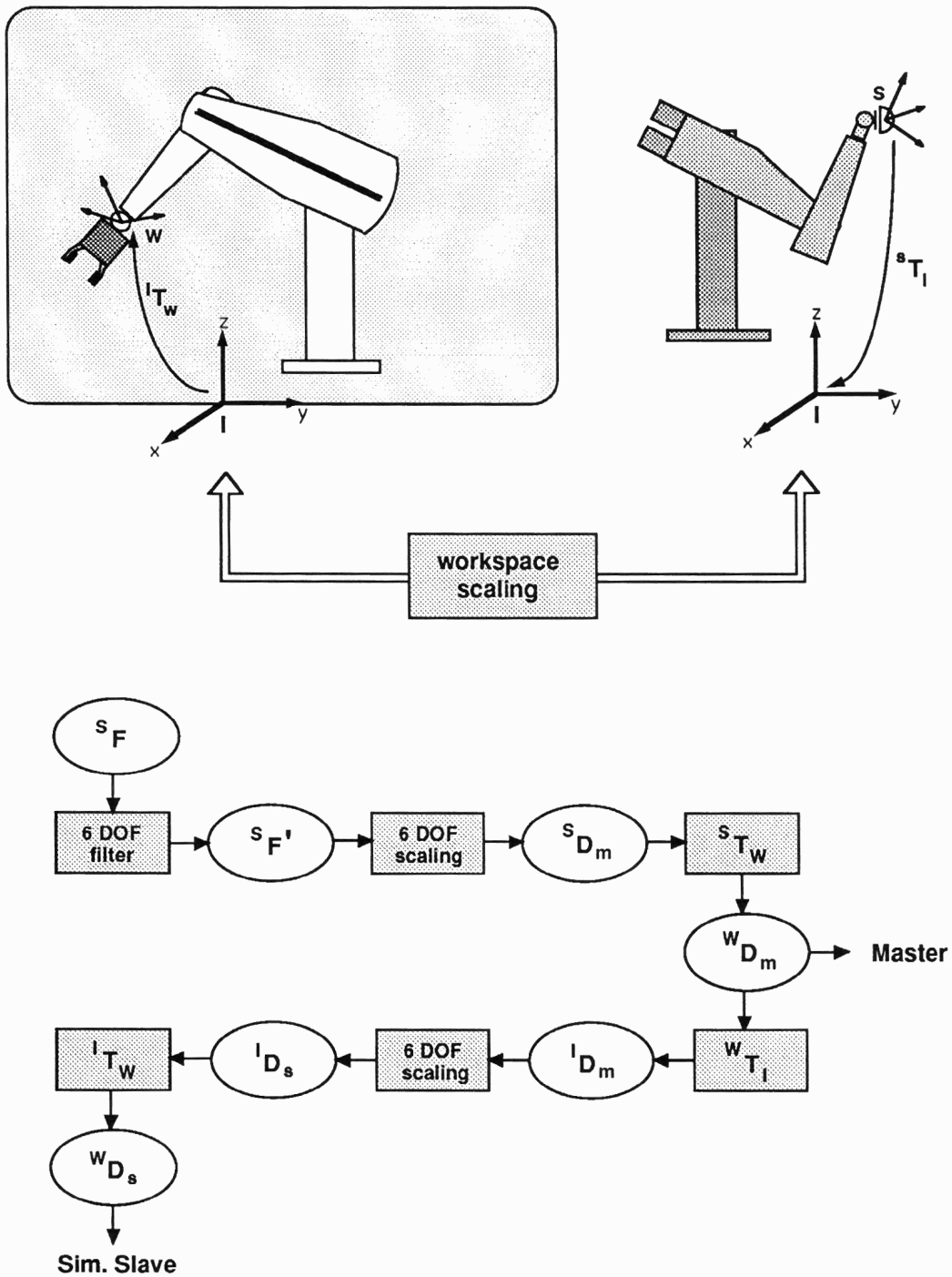


Figure 4.2: Transforming operator's effort into slave's motion.

thus determine the effective *impedance* of the master arm as perceived by the operator. By changing these parameters, different master, as well as reflected slave and load inertias can be simulated.

This incremental Cartesian displacement ${}^{S}\mathbf{D}_m$ is further transformed through the kinematic structure of the master device into the *view independent* coordinate frame $\mathcal{F}_I$. This frame serves as the common orientational reference between the master device and the viewpoint-dependent display of the remote slave. In other words, as the view of the simulated remote environment changes (either under operator's explicit control, or under the automatic guidance of the task model), this reference frame remains unchanged. This affords the operator an intuitive, constant reference frame, with a left-to-right movement of the master device always corresponding to the left-to-right movement of the slave, regardless of the display parameters, determining the view of the remote environment. This correspondence between telekinesthesis and visual feedback is an important special case of a broader issue of consistency of information channels in man-machine interfaces [Fischer,1990].

The view-independent displacement ${}^{I}\mathbf{D}_m$ of the master is then appropriately scaled into the slave workspace. The corresponding scalars determine the *magnification* or *reduction* of the operator's motions into the task space. Research has shown that scaling orientational parameters causes confusion on the part of the operator and is therefore normally not done [Fischer,1990]. Finally, the magnified slave motion ${}^{I}\mathbf{D}_S$ is transformed through the kinematics of the slave arm to become the incremental Cartesian displacement of the slave wrist[3] ${}^{W}\mathbf{D}_s$.

Through this chain of transformations we have thus established a *Cartesian correspondence of motion* between the master sensor frame $\mathcal{F}_S$ and the slave wrist frame $\mathcal{F}_W$. Moreover, this correspondence is view independent, intuitive, and allows for easy, dynamically adjustable master-to-slave displacement and/or force scaling for a broad range of target tasks.

[3] As in the case of the master arm, we will refer to the kinematic tip of the slave, i.e., $\mathbf{T6}_s$, manipulator as the slave's "wrist".

Chapter 5

The Graphical Simulation

5.1 The Simulation Technique

We have adopted a polyhedral, boundary-representation based graphical model of the world. While other representations are clearly possible (e.g., constructive solid geometry (CSG), generalized cylinders etc.), 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 a *teleprogramming* system.

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 grasped or manipulated object are assumed to be the only moving parts in the environment. We will hereafter refer to the movable portions of the remote environment, which are directly under operator's control, collectively as the *movable object* and abbreviate them as MO. Because of the kinematic nature of the simulation, dynamic changes in the environment, other than the state of the workcell and the object(s) being directly manipulated, need to be relayed to the operator's station and incorporated into the world model through the available environment updating mechanisms, as discussed in Section 3.8, rather than direct simulation. This applies to the dynamic changes caused by the slave (e.g., dropping or breaking an object), as well as those produced by external environmental agents (e.g., winds, water currents). While the choice of a kinematic simulation may seem restrictive, we believe 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, inertias, frictional properties, etc. about the objects in the environment; yet, these parameters are essential 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 or stress resistance 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

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

Clearly, a kinematic simulation leaves much to be desired. However, while nothing in the conceptual design of the *teleprogramming* methodology precludes incorporation of partial or full dynamic modeling into the system, it is not clear whether this is the proper direction in which to extend the system [Bejczy,1990]. This is to a large extent due to the fact that we are designing a control methodology, which is to be used in unstructured environments and which should not require a detailed knowledge of the dynamic properties of the environment to be useful and effective. The latter issue will be further clarified when we discuss the types of instructions, in terms of which the system describes operator's activity in the simulated world, and which in turn are sent to the remote workcell for execution.

5.2 Polyhedral Contact Types

Motion simulation, constraint analysis, as well as interpretation of operator's actions in the simulated environment critically depends on a detailed knowledge of the nature of polyhedral interactions between the slave workcell (MO) and the environment. Figure 5.1 lists the types of polyhedral contacts that we will consider in this work. Contact types are defined in terms of the *elementary polyhedral features*, i.e., vertex, edge, and face. The basic contact types (e.g., vertex/edge, edge/face, etc.) are further classified as either *point*, *line*, or *plane* contacts, as shown in Figure 5.1. We will refer to the latter categories as *contact classes*.

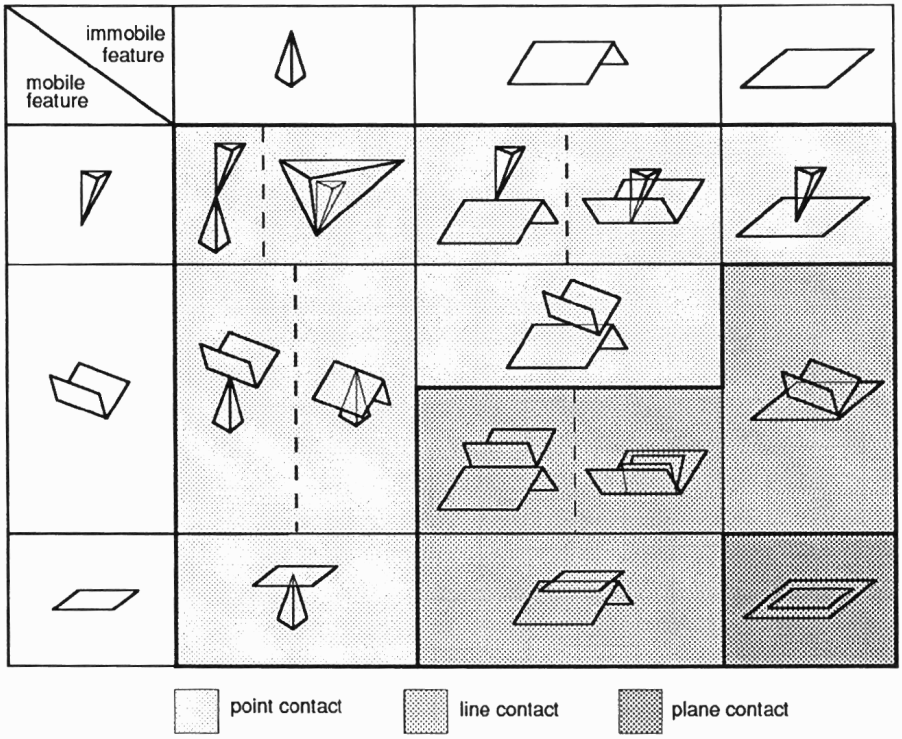


Figure 5.1: Types of polyhedral contacts.

It is easy to see that convex vertex/vertex and vertex/edge contacts represent 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 persistent and stable when one of the contacting features is concave. Following this work in 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 5.1 by juxtaposing the two cases, separating them with a vertical dashed line.

In the following sections we will 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 5.1. We will define an adjacent contact to be one which can be reached in one contact change from the current contact 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 d.o.f. of motion than c_j.

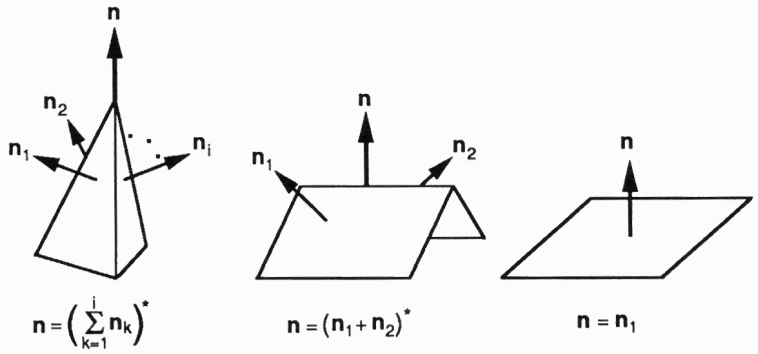


Figure 5.2: Contact normals for the three types of polyhedral features.

5.3 Contact Normals

An important parameter of every polyhedral contact is the corresponding *contact normal*. Clearly, the contact normal will be a function of the contact type and the *feature normals*, associated with the two polyhedral features, defining the contact. These feature normals are obtained in a straightforward manner and are illustrated in Figure 5.2.[1] Note that this definition assumes that all face normals of a convex polyhedral object are directed outward. The following paragraphs offer a convention which uses feature normal information to unambiguously define the contact normal direction for each contact type.

We will let the contact 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 5.1, 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 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 5.1), where the normal is most naturally defined by the cross-product of the two edge directions.

In keeping with the above convention, the contact normal direction for a face/face planar contact is then 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 contact types (except the

[1]The asterisk (*) in Figure 5.2 denotes that the corresponding vector is of unit magnitude.

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

5.4 Desired and Undesired Collisions

In Section 3.3 we made a distinction between *desired* and *undesired* collisions. We defined desired collisions as those, resulting from the intentional operator-specified interaction between the slave workcell and its environment. Such collisions will normally involve the slave manipulator's end effector or grasped object, and some part of the environment, involved in the execution of the task. As we saw in Section 3.6, the task model contains the information as to which objects in the environment are to be directly manipulated. The system therefore possesses all the necessary information to construct a list of all object pairs that are expected to come into contact during task execution. We will refer to this list as the *desired collision list*. In order to provide for an accurate simulation of these intentional slave-environment interactions, the simulator must be supplied with the corresponding polyhedral models that closely approximate the geometric description of the real objects.

Undesired or accidental collisions, on the other hand, were defined as all other collisions in the environment. These will normally involve some non-effector part of the slave manipulator linkage (such as the slave arm's "elbow") and a part of the environment, not directly involved in the execution of the task, according to the task model. In principal, we could therefore construct and *undesired collision list* by including all potentially colliding object pairs, which are not part of the desired collision list. However, practical considerations (e.g., computational efficiency) necessitate, that we restrict this list to a small number of object pairs, most likely to be involved in an unintended collision. This will be a function of the environment (obstacle distribution), the slave workcell kinematics, as well as the nature of the task itself, and should be specified as part of the task model. In order to further reduce the computational load of the graphical simulator, the obstacles and the slave manipulator features, which constitute the undesired collision list, can be described by simple and conservative polyhedral envelopes, as precise distance information is not necessary in this case.

During task execution, the graphical simulator thus checks commanded incremental motions for inter-object collisions. In the case of an undesired 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 informs her of the condition via the audio subsystem (Section 3.1). The operator can then adjust her intended motion to avoid the collision or back up and replan the last portion of the task. Note that this feature offers a rudimentary *collision avoidance* facility, where motion adjustment and/or replanning are left to the operator.

In the case of a desired collision, on the other hand, the system stops the motion of the object pair precisely in contact and records the necessary information to unambiguously describe the geometry of the contact. We will address this process in detail in the following sections.

5.5 Distance Computation

In order to support contact manipulation, the graphical simulator of a *teleprogramming* system must be able to monitor distances between objects in the simulated environment, detect collisions between them, and appropriately modify subsequent commanded motions of the slave workcell so as to not violate any of the geometric motion constraints.

As we saw in Section 5.4 above, the graphical simulator is supplied with the initial model of the remote environment and the desired and undesired collision lists. As the task progresses, the graphical simulator must then continuously monitor distances between object pairs on both lists and provide complete information about the contact geometry in the simulated environment at any instant in time. For complex environments or manipulation intensive tasks, a large number of inter-object distances may thus need to be computed at each simulation step. The basic software component, necessary to support this facility, is a fast distance estimator for polyhedral objects.

Several methods exist to compute distances between polyhedral objects.[2] In our work we have chosen to adopt the distance computation algorithm for convex polyhedra described in [Gilbert&Johnson,1987]. While many distance computation algorithms exhibit quadratic or even cubic complexity, this algorithm is near-linear in the total number of vertices of the two objects, whose inter-distance is being computed. The algorithm operates on a pair of convex sets of points and returns two points, one belonging to each set, which define the endpoints of the shortest straight line segment between the two sets. We will in the remainder of this chapter refer to these points as *nearest points*. For a pair of contacting

[2] In this context, *distance* between two objects is defined as the magnitude of the shortest translation that will put them precisely into contact.

objects, we will also use the term *contact point* to denote the pair of closely spaced nearest points on the surfaces of the respective objects.

In order to facilitate higher-level contact type determination and management (Section 5.7), we have extended the algorithm to also return the closest polyhedral *features* (e.g., vertex, edge, face). The result is a fast and reliable distance computation module which is used extensively throughout the task simulation. Section 5.6 bellow uses this module to detect collisions between objects and prevent inter-penetration of contacting objects.

5.6 Collision Detection

We now make use of the distance computation algorithm to implement a general polyhedral collision detection module.

Let $\mathbf{x}_A$ and $\mathbf{x}_B$ denote the closest points between two convex objects A and B. The distance between them is then given by $d = \|\mathbf{x}_B - \mathbf{x}_A\|$. If incremental displacements $\Delta\mathbf{d}_A = (\Delta\mathbf{t}_A, \Delta\mathbf{r}_A)$ and $\Delta\mathbf{d}_B = (\Delta\mathbf{t}_B, \Delta\mathbf{r}_B)$ are applied to A and B, respectively, it can be shown [Faverjon *et al.*,1987] that the *distance variation* between the two objects (Δd) can be expressed as

$$\Delta d = \mathbf{n} \cdot (\Delta\mathbf{x}_B - \Delta\mathbf{x}_A) \tag{5.1}$$

where $\mathbf{n} = (\mathbf{x}_B - \mathbf{x}_A)/d$ and $\Delta\mathbf{x}_A$, $\Delta\mathbf{x}_B$ are the positional displacements of the points $\mathbf{x}_A$ and $\mathbf{x}_B$, due to the object displacements $\Delta\mathbf{d}_A$ and $\Delta\mathbf{d}_B$, respectively. We define the objects A and B to be *in contact* whenever $d < \epsilon$, where ϵ is a small positive distance which is imperceptible to the operator's 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 commanded incremental motion 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*

$$t = \frac{-(d - \epsilon)}{\Delta d} \tag{5.2}$$

Note that the computational load of the graphical simulator can be significantly reduced if a bound on the maximum displacements per simulation step ($(\Delta d)_{max}$) for any object in the environment is available. This information may be task dependent and as such derived from the task model. Alternatively, a conservative estimate can always be made based on

the properties of the master device (Section 4.1). A *lazy* collision detection scheme can then be implemented by only monitoring an object pair when their inter-object distance approaches or becomes less that $(\Delta d)_{max}$. Therefore, if a call to the distance computation module for an object pair (O_i, O_j) returns

$$d > k\,(\Delta d)_{max} \quad ; \quad k \in \mathbb{N} \tag{5.3}$$

then the distance between these two objects need not be recomputed for the next k simulation steps.

It should also be noted that the distance variation computation of Eq. (5.2) is only valid for *strictly convex* sets of points[3]. Consequently, special steps are needed to handle 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 contact point along the object's surface, such as during an edge/face to face/face contact transition. We will address this problem in Section 5.9.

5.7 Contact Information Management

So far we are able to detect impending collisions and stop the offending motion precisely in contact. In order for the system to compute the motion constraints, corresponding to the current contact set, and enforce these constraints on subsequent commanded motions to the slave workcell (Chapter 6), the system must first analyze and record the exact nature of each existing contact. This information should include reliable and noise-tolerant indication of the *contact type* and the *contact feature centroids* for both contacting objects. As described in Section 5.6, the collision detection algorithm returns the following information:

- the *contact coefficient* t, where $0 \leq t \leq 1$
- 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.

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

However, the nearest points returned by the distance computation algorithm (Section 5.5) 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 (Chapter 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. Thus, an incremental motion that was intended (i.e., generated by the motion computation module of Chapter 6) to place two objects into an face/face contact, may appear, due to small alignment errors, to the distance estimator as an edge/face or even a 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* (Section 5.2) whenever the amount of misalignment 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. Once the final contact features are determined and realigned, the exact contact feature centroids are computed for both contacting objects in straightforward manner.

Having obtained this information, a *contact* is then defined as a pair of contacting features along with a set of parameters that uniquely define the geometry of the contact. It is easy to verify that the following parameters suffice to uniquely and unambiguously describe the geometry of a contact, regardless of the type of contacting features

- the *contact vector* $\mathbf{p}$ connecting the slave wrist ($\mathcal{F}_W$, Section 8.3), where the commanded motions are applied, and the contact point (feature centroid, associated with the contact)

- the *contact normal* $\mathbf{n}$ (see Section 5.3)

- the *edge direction* $\mathbf{e}$, in case of a line contact (see Section 5.2)

For convenience, all of the above vector quantities are computed with respect to the common global reference frame $\mathcal{F}_B$. Therefore, a contact c_i is encoded as the quintuple

$$c_i = \{f_1\ ,\ f_2\ ,\ {}^B\mathbf{p}\ ,\ {}^B\mathbf{n}\ ,\ {}^B\mathbf{e}\} \tag{5.4}$$

1. for each $p_{i,j} \in \mathcal{L}$
   - simulate the effect of $\Delta\mathbf{d}$ on the motion of O_i
   - compute $dist(O_i, O_j)$ and the contact coefficient $t_{i,j}$
2. $t \leftarrow \min\{t_{i,j}\}$
3. perform the motion $t(\Delta\mathbf{d})$
4. update the contact information in $\mathcal{C}$
5. for each $p_{i,j} \in \mathcal{L}$, recompute $dist(O_i, O_j)$

Algorithm 5.1: Outline of the collision checking algorithm.

where f_1 and f_2 correspond to the contact features of MO and the environment, respectively. The list of all (N) currently active contacts is stored as the *contact set* $\mathcal{C}$, where

$$\mathcal{C} = \bigcup_{i=1}^{N} c_i \tag{5.5}$$

This information is then used to restrict subsequent commanded motions of the simulated workcell so as to not violate any of the environmental motion constraints, as well as to provide real-time kinesthetic feedback to the operator (Chapter 6). As we will see in Chapter 7, this information will also play a vital role in generation the symbolic command strings, describing the operator's activity in the simulated environment.

5.8 The Algorithm

Let $\mathcal{L}$ denote the desired collision list of all object pairs $p_{i,j} = (O_i, O_j)$ which are currently being monitored for collisions and let $\Delta\mathbf{d} = (\Delta\mathbf{t}, \Delta\mathbf{r})$ denote the current commanded incremental displacement of the slave manipulator. Moreover, let O_i in each pair belong to the movable object (i.e., O_i is rigidly attached to the slave), and let O_j belong to the environment. The skeleton of the collision checking algorithm is given in Algorithm 5.1 below.
Step 1 above examines the effect of the commanded motion $\Delta\mathbf{d}$ on each object pair without actually performing the motion. For each pair, the system checks to see if the motion causes the nearest point $\mathbf{p}_i \in O_i$ to penetrate the ϵ-envelope of O_j. In either case, the contact coefficient $t_{i,j} \leq 1$ (Section 5.6) is recorded. In step 2 the minimum over all $t_{i,j}$ is taken as the overall contact coefficient and the corresponding fraction of the commanded motion is performed (step 3). Step 4 ensures that any resulting new or persistent contacts are

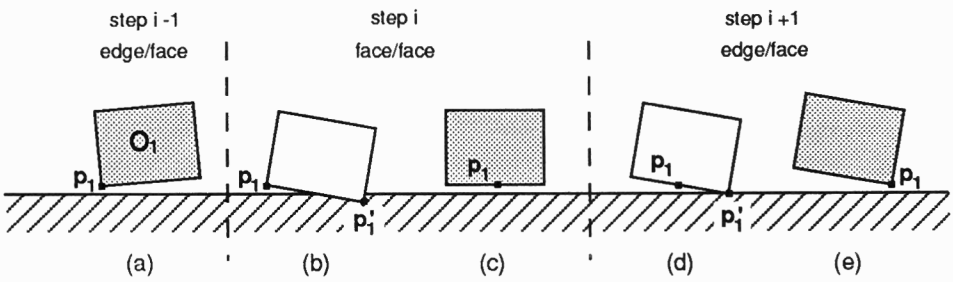


Figure 5.3: edge/face → face/face → edge/face contact type transition.

reflected in the updated contact set $\mathcal{C}$. Finally, in step 5, inter-object distances are recomputed for all object pairs in $\mathcal{L}$. This is done to ensure that any contact point discontinuities are detected (see Section 5.9) and that object locations, contact information (type, features, centroids), and other configuration-dependent information is properly updated with respect to the final environment configuration.

5.9 Contact Type Transitions

Observe that only the current nearest point $\mathbf{p}_i$ 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_i. However, if this 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_j's ϵ-envelope. The nearest point $\mathbf{p}_i'$ following the motion belongs to the penetrating portion of O_i and in fact corresponds to the deepest point of penetration. Therefore, it is this point that the collision estimator should have monitored for contact instead of $\mathbf{p}_i$.

Figure 5.3 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 5.3-a. Then, in the i^{th} step, the intended motion will be checked to ensure that $\mathbf{p}_i$ does not penetrate O_j's ϵ-envelope. Since the operator's commanded motion has been restricted such as to leave the contact point fixed (see Chapter 6), it will pass the check and the motion will be applied in full. This may result in the configuration of Figure 5.3-b,

which, of course, constitutes a collision.

Step 5 of the contact monitoring algorithm in Section 5.8 above allows us to handle such situations. The corresponding call to the distance estimator will reveal that $dist(O_i, O_j) < \epsilon$ and that $\mathbf{p}_i' \neq \mathbf{p}_i$.[4] Having determined the new contact point, we now set the block back into its original position (Figure 5.3-a), set $\mathbf{p}_i = \mathbf{p}_i'$ and repeat steps 1–5. This time, the motion will be found to be only partly realizable and only the corresponding fraction 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 5.7 will compute the new contact feature centroids for the two objects as shown in Figure 5.3-c. Assuming that the pivoting motion persists, the $(i+1)^{th}$ step will similarly produce the situation of Figure 5.3-d, where the contact point $\mathbf{p}_i$ again moves discontinuously to the right edge ($\mathbf{p}_i'$). As before, this is detected by the call to the distance estimator in step 5, the block is reset to its face/face configuration and the same motion is reapplied with $\mathbf{p}_i'$ serving as the contact point. Clearly, this motion is allowable and the block transitions to the edge/face contact of Figure 5.3-e.

The collision detection and contact management algorithm of Section 5.8 thus allows for smooth transitions between contact types and offers the operator a wide repertoire of pivoting contact motions.

[4]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.

Chapter 6

Motion Restriction and Kinesthetic Feedback

6.1 Motion Mode Classification

A *teleprogramming* system should offer the operator control over a wide range of slave workcell 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 operator should be able to select different subsets of the physically realizable motion, which would allow her to concentrate on only those motion parameters that are relevant to the current subtask. This can be accomplished by defining a set of elementary *motion modes*, which provide a collection of basic and intuitive motion modalities.

A natural way to simplify general motion (both for the operator and the slave robot) is to separate rotations and translations whenever possible. This is particularly crucial in contact motion, as the contact point is normally physically removed from the wrist-based reference location ($\mathcal{F}_W$, Figure 4.2), where motion is commanded. This separation gives rise to a remote compliance center and consequently introduces complex and dynamically changing coupling between rotational and translational parameters of the wrist and contact frames. This coupling may lead to control instabilities at the slave workcell ([Whitney,1982], [Zhang,1986], [An&Hollerbach]) and may result in confusing reflected motion applied to the master device and perceived by the operator.

The choice of elementary motion modes should strive to eliminate such coupling effects without compromising the flexibility and power of the *teleprogramming* system. Therefore, in view of the above considerations, we propose the following set of elementary classes of

motions:

1. **Free Space Motion**
    - **free motion** (both rotations and translations)
    - **translation** (fixed orientation)
    - **rotation** (fixed position)
2. **Contact Motion**
    - **sliding** (translation along constraint features, fixed orientation)
    - **pivoting** (rotational motion about contact point, fixed position)
    - **pushing**

Given a set of elementary motion modes, a mechanism to switch between them needs to be designed. In order to minimize the burden on the operator, mode switching should be done automatically whenever possible. In particular, in contact motion, the motion mode can be inferred automatically from the current contact state (between MO and the environment) and the commanded force and motion input from the operator. In free space, on the other hand, kinesthetic information can not be used for mode selection. In this case, as well as when the operator wishes to override the automatically inferred mode, the operator can use the audio interface to communicate the desired motion mode to the system.

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 grasped feature. Therefore, we provide three corresponding elementary free space modes of motion. One could proceed further and introduce single d.o.f. 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 and view-dependent 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.

Given the operator supplied commanded motion $\Delta\mathbf{d} = (\mathbf{t}, \mathbf{r})$, the restricted motion $\Delta\mathbf{d}'$, corresponding to each of the three free-space motion modes, is trivially computed as follows

- free motion: $\Delta\mathbf{d}' = (\mathbf{t}, \mathbf{r})$
- translation: $\Delta\mathbf{d}' = (\mathbf{t}, \mathbf{0})$
- rotation: $\Delta\mathbf{d}' = (\mathbf{0}, \mathbf{r})$

6.3 Contact Motion

The *teleprogramming* system provides three contact motion modes, as indicated in Section 6.1.

In *sliding* mode, the operator can slide MO along the constraining feature(s) (surfaces, edges) in the permissible directions, i.e., such that none of the geometric motion constraints are violated. The orientation of MO remains fixed for the duration of motion in this mode. The system can be asked to help the operator maintain contact with the environment by providing a small amount of surface adhesion, if desired, but will allow the operator to break existing contact(s) if she clearly indicates such intent. 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 contacts are restricting the motion of MO.

Alternatively, the operator can adjust the orientation of MO or transition between adjacent contacts by rotating or pivoting about the contact point (*pivoting* mode). In this mode the contact point is not allowed to slide along or depart from the supporting environment contact 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, the system performs motion analysis on the commanded displacements in order to aid the operator in achieving the desired changes of orientation.

The system also provides a restricted version of this motion modality for multiple-contact configurations.

A third contact motion mode (*pushing*) is provided to support a rudimentary pushing capability. Predicting the exact outcome of pushing motions in actual situations is extremely difficult, because the motion of a pushed object critically depends on the complex interactions between the microscopic features of the two sliding surfaces [Peshkin&Sanderson,1987], [Mason,1985], [Mason,1986]. Consequently, in order to generate instructions, which can be executed successfully and reliably under slave's local sensory supervision, the system offers only a restricted, straight-line pushing mode. In Section 6.4.4 we will address both single-contact and multiple-contact pushing.

6.4 Restriction Operators

6.4.1 Contacts and Constraints

In Chapter 5 we described how the *teleprogramming* system detects collisions, examines contacts, and maintains the resulting collection of all currently active contacts as the contact set $\mathcal{C}$. Associated with each contact are one or more mutually orthogonal Cartesian constraints on the relative motion of the contacting objects [Mason&Salisbury,1985]. The number of resulting motion constraints is a function of the geometric contact type (Section 5.2) and the physical properties of the contacting surfaces (e.g., friction). It is crucial to note that constraints in this context are defined with respect to Cartesian reference coordinates and are thus mutually orthogonal. In contrast, the collection of contacts in $\mathcal{C}$, and in particular their associated contact normals, bear no particular relationship with each other and in general will not be mutually orthogonal. For situations where multiple contacts define the current contact state between two object, we must therefore orthogonalize the associated constraints with respect to a set of reference coordinates in order to obtain a meaningful description of the constraints, restricting the relative motion of the two bodies. In the trivial case of a single contact, an orthogonal frame can be aligned with the only contact normal and the resulting constraints can be defined in this frame. In the case of multiple contacts, however, such a frame in general does not exist (unless all contact normals happen to be mutually orthogonal). In this case, then, we need to define a set of orthogonal reference coordinates and project the constraints associated with each contact into this common reference coordinate frame to obtain the *constraint set* $\mathcal{S}$. Any subsequent motion imparted on an object, whose contact set is given by $\mathcal{C}$, is thus constrained by $\mathcal{S}$. Specifically, effort

exerted on the object along any of the orthogonal constrained directions in $\mathcal{S}$ will not result in motion.

In our case, the objects are MO and the environment. For each contact configuration (regardless of its contact multiplicity) we will define a *restriction frame* $\mathcal{F}_R$, which will be aligned with the dominant contact features and chosen so as to facilitate easy and intuitive restriction of commanded motion with respect to the current constraint set. For single contact configurations, the commanded motion, appropriately mapped into the restriction frame, will be restricted with respect to the only constraint $\{c_1\} = \mathcal{C}$. For MO/environment configurations with contact multiplicity greater than 1, on the other hand, the constraints associated with the contacts $c_i \in \mathcal{C}$ will be mapped into this restriction frame and the commanded motion will be restricted based on the resulting orthogonal set of motion constraints.

We will in the upcoming discussion use the terms *contact normal* and *constraint normal* interchangeably for single contact configurations. We will also refer to a *constrained direction* $\tilde{\mathbf{n}}_i$ as the negative of the corresponding contact normal $\mathbf{n}_i$, i.e., $\tilde{\mathbf{n}}_i = -\mathbf{n}_i$.

In the following sections we will present the details of contact motion computation and restriction for all three contact motion modes.

6.4.2 Sliding

Single contact sliding

Given the desired motion of the slave wrist ($\Delta^B\mathbf{d} = ({}^B\mathbf{t}, {}^B\mathbf{r})$), we compute the corresponding contact point translational motion as[1]

$$\Delta^B\mathbf{d}' = \left({}^B\mathbf{t}, \mathbf{0}\right) \tag{6.1}$$

The contact information $\mathcal{C}$, as defined in Section 5.7, specifies the single unit constraint normal as ${}^B\mathbf{n}$. The resulting restricted contact (as well as wrist) motion $\Delta^B\mathbf{d}''$ is therefore given by

$$\Delta^B\mathbf{d}'' = ({}^B\mathbf{t}'' \ , \ \mathbf{0}) \tag{6.2}$$

where

$$\mathbf{t}'' = \begin{cases} \mathbf{t}' - (\mathbf{t}' \cdot \mathbf{n})\mathbf{n} & , \text{ if } (\mathbf{t} \cdot \mathbf{n}) < \epsilon \\ \mathbf{t} & , \text{ otherwise} \end{cases} \tag{6.3}$$

[1]Note that the incremental translational displacement of MO's contact point is the same as the commanded translational displacement of the slave's wrist frame $\mathcal{F}_W$, despite the offset between them.

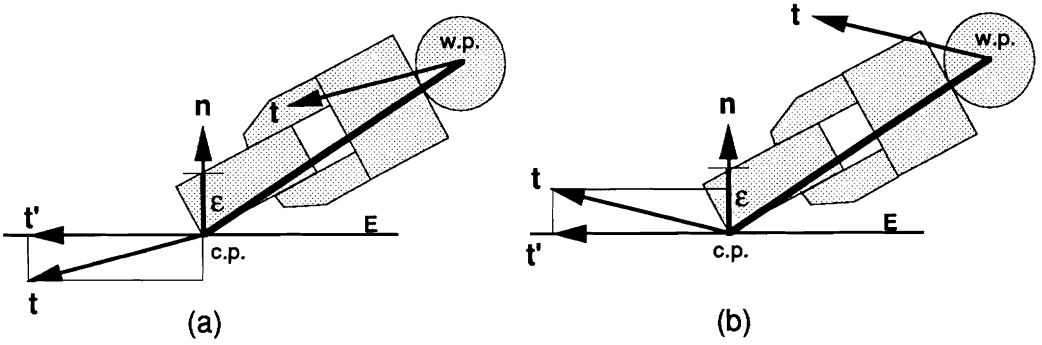


Figure 6.1: Single contact sliding.

Figure 6.1 illustrates a typical situation for single-contact sliding, where w.p. and c.p. denote the slave wrist center, where motion is commanded, and the contact point, respectively. Note that for $\epsilon > 0$, the operation of Eq. (6.3) above will remove 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 6.1). This, in effect, provides a programmable amount of *contact surface adhesion*. Clearly, ϵ can be set to 0 and the effect disappears. Alternatively, ϵ can be set to be small positive value, in which case the system will aid the operator in preserving attained contacts, while still allowing her to break a sliding contact if she indicates such intent by commanding a decisive motion away from the contact surface.

Multiple contact sliding

Figure 6.2 illustrates a typical situation, where the motion of MO is being restricted by two contacts, with the corresponding constraints forming a non-orthogonal constraint set.[2] 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 $\Delta^B\mathbf{d} = ({}^B\mathbf{t}, {}^B\mathbf{r})$. The analysis of the multi-contact case centers on identifying *the primary constrained direction* $\tilde{\mathbf{n}}_p$ (Section 6.4.1). The primary constrained direction is defined to be the one which absorbs the largest component of the operator's exerted translational effort.

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

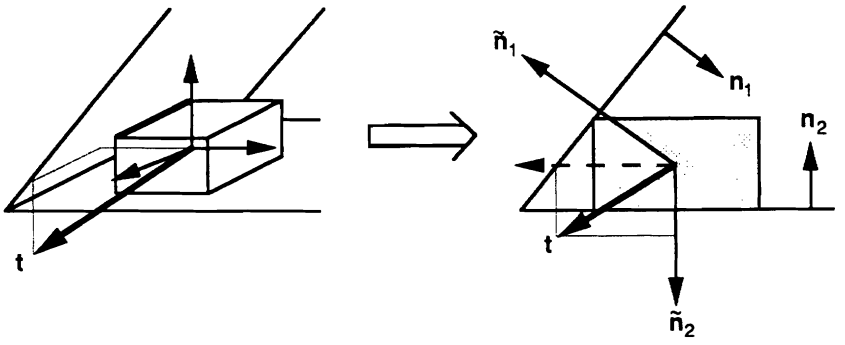


Figure 6.2: Multiple contact sliding.

Computationally, it is taken to be the one with the largest (positive) projection of $\mathbf{t}$ along its unit direction (see Algorithm 6.1). Given the primary constrained direction $\tilde{\mathbf{n}}_p$, we then construct an orthogonal restriction frame $\mathcal{F}_R$, such that $\tilde{\mathbf{n}}_p$ is one of its axes, and the cross product with any other constrained direction $\tilde{\mathbf{n}}_j$, $j \neq i$, gives its second orthogonal axis, i.e.,

$$\mathcal{F}_R = \left\{ \left(\left({}^B\tilde{\mathbf{n}}_p \times {}^B\tilde{\mathbf{n}}_j \right) \times {}^B\tilde{\mathbf{n}}_p \right)^* \ , \ \left({}^B\tilde{\mathbf{n}}_p \times {}^B\tilde{\mathbf{n}}_j \right)^* \ , \ {}^B\tilde{\mathbf{n}}_p \right\} \tag{6.4}$$

This choice of a restriction coordinate frame is adopted because a commanded translational motion $\mathbf{t}$ in a multi-contact case will normally give rise to a sliding motion along the constraint feature, whose associated constrained direction is closest to $\mathbf{t}$. The goal of this computation is not to produce dynamically, or even quasi-statically, correct motion predictions, but merely to produce a resulting motion of MO in the simulated environment, which appears intuitively correct with respect to the effort exerted on the master device by the operator.

Having constructed the restriction 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., ${}^R\mathbf{t}, {}^R\tilde{\mathbf{n}}_k$) and restrict the commanded slave wrist motion accordingly. Algorithm 6.1 below formalizes the restriction procedure and supplies the necessary details. Steps 1 and 2 of the algorithm identify the primary constrained direction $\tilde{\mathbf{n}}_p$. In step 3 we construct the restriction frame $\mathcal{F}_R$. The commanded displacement ${}^B\mathbf{t}$ is mapped into this frame in step 4. The core of the restriction process is step 5, where each constrained direction $\tilde{\mathbf{n}}$ is in turn rotated into the restriction frame to produce the corresponding constraint set $\mathcal{S}$. The components of the commanded motion ${}^R\mathbf{t}$ are then restricted with respect to the Λ restriction operator, operating in $\mathcal{F}_R$.

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 $p_k = \max\{p_j\}$ and let ${}^B\tilde{\mathbf{n}}_p = {}^B\tilde{\mathbf{n}}_k$
3. construct the restriction frame $\mathcal{F}_R$, according to Eq. (6.4) and define the rotational matrix ${}^B\mathbf{R}_R$ from $\mathcal{F}_R$ (see Appendix A.2)
4. map ${}^B\mathbf{t}$ into $\mathcal{F}_R$, i.e., ${}^R\mathbf{t} = \left({}^B\mathbf{R}_R\right)^{-1} * {}^B\mathbf{t}$
5. for each $c \in \mathcal{C}$, map c into $\mathcal{F}_R$ and restrict ${}^R\mathbf{t}$ accordingly,
   - map ${}^B\tilde{\mathbf{n}}$ into $\mathcal{F}_R$, i.e., ${}^R\tilde{\mathbf{n}} = \left({}^B\mathbf{R}_R\right)^{-1} * {}^B\tilde{\mathbf{n}}$
   - restrict each component of ${}^R\mathbf{t}$ in turn, i.e., $\Lambda\left({}^Rt_x, {}^R\tilde{n}_x\right)$, $\Lambda\left({}^Rt_y, {}^R\tilde{n}_y\right)$, $\Lambda\left({}^Rt_z, {}^R\tilde{n}_z\right)$
6. map restricted ${}^R\mathbf{t}$ back into $\mathcal{F}_B$, i.e., ${}^B\mathbf{t}' = {}^B\mathbf{R}_R * {}^R\mathbf{t}$

Algorithm 6.1: Multi-contact sliding motion restriction.

This operator is defined as follows

$$\Lambda(a,b) : a = \begin{cases} a & , \text{ if } (b = 0) \text{ or } (a \cdot \text{sgn}(b)) \leq -\epsilon \\ 0 & , \text{ otherwise} \end{cases} \tag{6.5}$$

where $a, b \in \mathbb{R}$. Therefore, in view of step 5 of Algorithm 6.1, 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 adhesion as in the single contact case above.[3] Having performed the restriction operation on ${}^R\mathbf{t}$, the restricted commanded displacement ${}^R\mathbf{t}'$ is then rotated back into the base reference frame (step 6). The restricted motion of the slave wrist is then assembled as $\Delta^B\mathbf{d}' = \left({}^B\mathbf{t}', \mathbf{0}\right)$.

Observe that a restriction frame is constructed even in the case where the original commanded motion does not violate any of the active constraints, i.e., when all p_i in step 1 are negative. This is done so that the removal of small components away from the contact features in step 5 (which must be done in this case as well) is performed in an orthogonal frame. Finally, for clarity, various optimizations of the above procedure have been omitted (in particular, in step 5). Any implementation must consider these carefully.

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

6.4.3 Pivoting

Single contact pivoting

Computing the contact motion: As in the case of sliding, the input to the restriction module are the commanded (operator supplied) incremental motion of the slave wrist ($\Delta^B\mathbf{d}$) and the current contact information $\mathcal{C}$ (Section 5.7). The first step in the restriction process is to compute the rotational motion of MO about the current contact point, based on the supplied slave wrist motion and subject to the requirement that the contact point remain fixed. In view of Section 4.3, the commanded incremental wrist displacement $\Delta^B\mathbf{d}$ can be written as

$$\left({}^B\mathbf{t}, {}^B\mathbf{r}\right) = \left(\left(\frac{1}{\alpha}\right){}^B\mathbf{f}, \left(\frac{1}{\beta}\right){}^B\boldsymbol{\tau}\right) \tag{6.6}$$

where $(\mathbf{f}, \boldsymbol{\tau})$ corresponds to the original operator's effort (force), exerted at the master device. The scalar parameters α and β represent the product of the effective master impedance and the magnification factors for translational and rotational motions, respectively (Section 4.3). Under the assumption of contact point stiction, the operator's effort $(\mathbf{f}, \boldsymbol{\tau})$, applied at the slave wrist, results in a moment about the contact point equal to

$$\boldsymbol{\tau}' = \boldsymbol{\tau} + (\mathbf{p} \times \mathbf{f}) \tag{6.7}$$

Using the relationship between exerted forces and incremental displacements of Eq. (6.6), we obtain the equivalent rotational motion about the contact point as

$$\begin{aligned} \mathbf{r}' &= \frac{1}{\beta}(\boldsymbol{\tau} + (\mathbf{p} \times \mathbf{f})) \\ &= \frac{1}{\beta}(\beta\mathbf{r} + \alpha(\mathbf{p} \times \mathbf{t})) \\ &= \mathbf{r} + \frac{\alpha}{\beta}(\mathbf{p} \times \mathbf{t}) \end{aligned} \tag{6.8}$$

where all vector quantities in Eq. (6.7) and Eq. (6.8) are given in base coordinates (i.e., with respect to $\mathcal{F}_B$). Therefore, the contact displacement $\Delta^B\mathbf{d}'$ is

$$\Delta^B\mathbf{d}' = \left(\mathbf{0}\,,\, {}^B\mathbf{r}'\right) \tag{6.9}$$

with ${}^B\mathbf{r}'$ given by Eq. (6.8). Note that the dimension of the term α/β is

$$\left[\frac{\alpha}{\beta}\right] = \frac{rad}{m^2} \tag{6.10}$$

and so the dimension of the resulting incremental rotational displacement is

$$\left[{}^B\mathbf{r}'\right] = rad \tag{6.11}$$

as expected. Also note that the magnitude of the resulting contact point rotation includes contributions from both the rotation and translation at the slave wrist, i.e.,

$$\|\mathbf{r}'\| = \|\mathbf{r}\| + \frac{\alpha}{\beta}\|\mathbf{p}\|\|\mathbf{t}\| \sin\phi \tag{6.12}$$

where ϕ is the angle between $\mathbf{p}$ and $\mathbf{t}$. In particular, the magnitude of the contribution, due to the exerted pure force $\mathbf{f}$, is a function of the angle ϕ, as well as the offset between the slave wrist and the contact point ($\|\mathbf{p}\|$). As the direction of the force $\mathbf{f}$ approaches the direction of the offset vector $\mathbf{p}$ (i.e., $\mathbf{p} \parallel \mathbf{t}$), the contribution vanishes. Similarly, assuming that $\mathbf{p} \times \mathbf{t} \neq \mathbf{0}$, the larger the offset $\|\mathbf{p}\|$, the more significant the corresponding contribution. Conversely, as this offset approaches zero, the corresponding contribution to the contact rotation also vanishes, as expected.

Restricting the contact motion: Having computed the rotational motion of MO's contact point, we now restrict this motion with respect to the constraints imposed by the particular contact type. The restriction 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 describe the restriction procedure for each contact type.

In order to perform the restriction, we will first define a *restriction frame* $\mathcal{F}_R$, centered at the contact point, and aligned with the dominant contact features. We will then express the computed contact rotation of MO in this frame and perform the restriction with respect to its coordinates. In each case the restriction frame will be defined in terms of the geometric parameters supplied by the contact information $\mathcal{C}$. (see Section 5.7). The input to this restriction process is the contact point displacement $\Delta^B\mathbf{d}'$ as computed above.

(a) Point Contacts: A restriction frame need not be specified in this case as all three orthogonal rotations are permissible in all point contacts (see Figure 6.3-a). Therefore, no restriction is necessary and we have

$$\Delta^B\mathbf{d}'' = \Delta^B\mathbf{d}' \tag{6.13}$$

(b) Line Contacts: A line contact always involves an edge (see Figure 5.1), and it is this edge direction ($^B\mathbf{e}$), together with the constraint normal ($^B\mathbf{n}$), that defines the most convenient restriction frame, i.e.,

$$\mathcal{F}_R = \left\{ {}^B(\mathbf{e} \times \mathbf{n}) \, , \, {}^B\mathbf{e} \, , \, {}^B\mathbf{n} \right\} \tag{6.14}$$

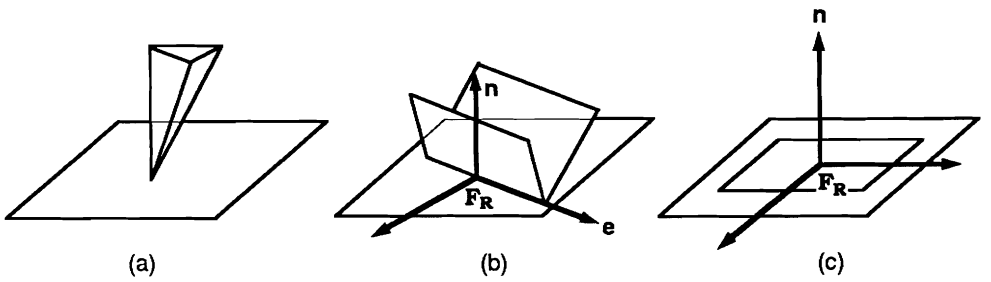


Figure 6.3: Single contact pivoting.

where ${}^B\mathbf{e}$ and ${}^B\mathbf{n}$ are assumed to be of unit magnitude. The specification of the rotational matrix ${}^B\mathbf{R}_R$ follows immediately (see Appendix A.2). Figure 6.3-b illustrates the case of an edge/face line contact.

The restriction operation is now performed by first rotating the contact point motion $\Delta^B\mathbf{d}'$, using ${}^B\mathbf{R}_R$, into the restriction frame to obtain $\Delta^R\mathbf{d}'$ (see Appendix A.3). Removal of small rotational components about ${}^B(\mathbf{e} \times \mathbf{n})$, tending to destabilize the line contact is then accomplished by applying the Υ restriction operator as follows

$$ {}^R\mathbf{r}'' = \left(\Upsilon({}^Rr'_x) , {}^Rr'_y , 0 \right) \tag{6.15} $$

where the z-axis component of rotation has been removed completely under the assumption of stiction, and the Υ operator is defined as follows

$$ \Upsilon(x) = \begin{cases} 0 & , \text{ if } |x| < \xi, \ \xi > 0 \\ x & , \text{ otherwise} \end{cases} \tag{6.16} $$

(c) Plane Contacts: The only representative of this class of contacts is the face/face contact (see Figure 5.1). Here, the restriction frame is defined as follows

$$ \mathcal{F}_R = \left\{ {}^B(\mathbf{v} \times \mathbf{n}) , {}^B\mathbf{v} , {}^B\mathbf{n} \right\} \tag{6.17} $$

where $\mathbf{v}$ is any unit vector not parallel to $\mathbf{n}$. As before, the rotational matrix ${}^B\mathbf{R}_R$ can be constructed directly from the definition of the restriction frame. Figure 6.3-c illustrates the situation.

Again, a two-stage restriction procedure is employed. The given rotational motion of the contact point is first mapped from $\mathcal{F}_B$ into $\mathcal{F}_R$ (via the rotational matrix ${}^B\mathbf{R}_R$). The second stage then eliminates the rotational component about the contact normal (stiction)

and removes from ${}^R\mathbf{r}'$ small destabilizing rotations about the x and y-axes of the restriction frame, i.e.,

$$ {}^R\mathbf{r}'' = \left(\Upsilon({}^Rr'_x) \, , \, \Upsilon({}^Rr'_y) \, , \, 0 \right) \tag{6.18} $$

where the Υ restriction operator is defined in Eq. (6.16) above.

Computing equivalent restricted wrist motion: Having computed the restricted 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 simulated slave manipulator. This is accomplished by mapping the restricted contact point motion $\Delta^R\mathbf{d}'' = (\mathbf{0}, {}^R\mathbf{r}'')$ into $\Delta^B\mathbf{d}''$ (see Appendix A.3) and computing the corresponding displacement of the slave wrist in base coordinates, $\mathcal{F}_B$ (see Appendix A.4).

Multiple contact pivoting

In this section we extend the results of the previous section to accommodate a restricted, but useful subset of multiple-contact pivoting motions. The restrictions are imposed to aid the operator in performing simple and intuitive multi-contact rotations, keep the geometrical and numerical complexity of the motion analysis low, as well as to limit the complexity of the corresponding execution process at the remote site.

A typical situation which this motion mode is intended to address is one where the operator has brought the movable object into a multi-contact configuration and wishes to align MO with respect to the environment so as to obtain a higher order, and thus more stable, contact type. Figure 6.4-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 with respect to the environment (i.e., edge/face wall contact, Figure 6.4-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 contacts. 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 constraining features, 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-contact situations will arise by sliding the movable object into a second

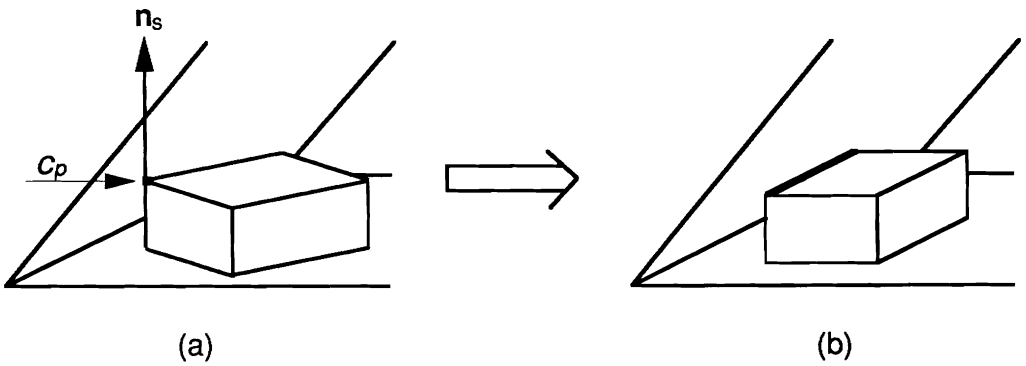


Figure 6.4: Multiple contact pivoting.

contact, 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 contact, 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 contact*, c_p. By convention, we will refer to the mandatory face/face contact as the *secondary* contact, c_s. The motion of MO will then be computed as a pure rotation about the contact point associated with the primary contact, and restricted such that it will not violate any of the constraints, resulting from the secondary contact. 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 contact is a line contact (see Figure 5.1), then motion will only be possible if the corresponding edge direction is parallel to the secondary contact normal $\mathbf{n}_s$ (see Figure 6.4).

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 contact c_p and compute the rotational motion ${}^B\mathbf{r}'$ about its associated contact point as in the case of single pivoting contact above. This rotation must then be restricted so as to retain only the rotation about the axis parallel to the normal of the secondary constraint. We therefore define a restriction frame $\mathcal{F}_R$, such that one of its axes (e.g., z) coincides with this normal direction, i.e.,

$$\mathcal{F}_R = \left\{ {}^B((\mathbf{n}_p \times \mathbf{n}_s) \times \mathbf{n}_s)^* \, , \, {}^B(\mathbf{n}_p \times \mathbf{n}_s)^* \, , \, {}^B\mathbf{n}_s \right\} \tag{6.19}$$

and map the rotation ${}^B\mathbf{r}'$ into this frame to obtain ${}^R\mathbf{r}'$ (see Appendix A.3). The restricted

rotation is then obtained trivially as

$$^{R}\mathbf{r}'' = \left(0, 0, {}^{R}r_z'\right) \tag{6.20}$$

The remaining task is to compute the corresponding motion $\Delta^{B}\mathbf{d}''$ of the slave wrist in the reference frame coordinates ($\mathcal{F}_B$). This is accomplished in a straightforward fashion as for the case of single contact pivoting.

6.4.4 Pushing

Single contact pushing

As mentioned in Section 6.3, accurate prediction of the outcome of a pushing motion is extremely difficult 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 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 3.6) 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.

A third requirement aimed at avoiding slippage along the pushing contact is that the pushing contact plane have a "reasonable" orientation with respect to the sliding surface. We quantify this condition by introducing a *pushing frame*

$$\mathcal{F}_P = \left\{ {}^{B}((\mathbf{n}_p \times \mathbf{n}_s) \times \mathbf{n}_s)^* \, , \, {}^{B}(\mathbf{n}_p \times \mathbf{n}_s)^* \, , \, {}^{B}\mathbf{n}_s \right\} \tag{6.21}$$

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 6.5)[4]:

$$\alpha = \text{atan2}\left(\|\mathbf{n}_p \times \mathbf{n}_s\| \, , \, \mathbf{n}_p \cdot \mathbf{n}_s\right) > \alpha_{min} \tag{6.22}$$

[4]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 $\mathcal{F}_P$.

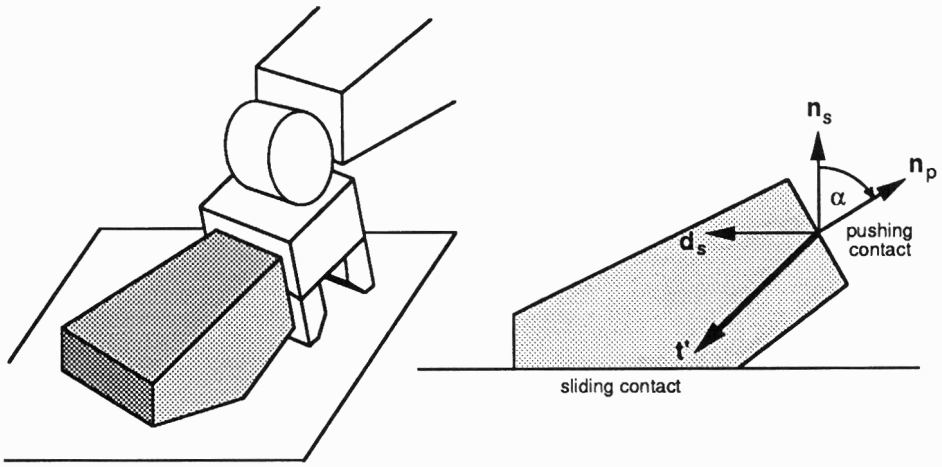


Figure 6.5: Single contact pushing.

Likewise, no sliding motion should be generated unless the commanded displacement vector ${}^B\mathbf{t}$ lies below and has a positive component along the *sliding direction* ${}^B\mathbf{d}_s$ (see Figure 6.5). Rotating the commanded motion $\Delta^B\mathbf{d}$ into the pushing frame, we can express the above conditions in terms of restrictions on $\Delta^P\mathbf{d}$ as follows

$$\Delta^P\mathbf{d}' = \left({}^P\mathbf{t}', \mathbf{0}\right) \tag{6.23}$$

where

$${}^P\mathbf{t}' = \begin{cases} \left({}^Pt_x, 0, 0\right) & , \text{ if } ({}^Pt_x > 0) \text{ and } ({}^Pt_z < 0) \\ \mathbf{0} & , \text{ otherwise} \end{cases} \tag{6.24}$$

Computing the corresponding displacement $\Delta^B\mathbf{d}'$ then gives the resulting pushing motion in the reference ($\mathcal{F}_B$) coordinates.

In order for pushing motion to take place, the operator must first establish a planar contact with some environment object. The operator can signal her intent to push the object by exerting a significant (and therefore easily identifiable) force against it or via the audio interface. If the task model identifies this object 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 restricts subsequent commanded slave motions according to Eq. (6.24) so as to move MO in a straight line along the sliding surface. Similarly, a decisive pull away from the pushing contact or a corresponding voice command will cause the system to exit the pushing mode.

Whereas a number of precautions have 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, the operator has no sense of the frictional forces and 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 rely on her best guess of the relevant dynamic parameters in choosing a reasonable pushing contact and a proper motion velocity.

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. This class of motions is used to align a free (i.e., not grasped) object with respect to a pair of environmental features or to slide an object along an environmental feature by pushing it. The analysis of such aligning and pushing motions is therefore analogous to the analysis of two-contact pivoting and sliding motions, respectively.

6.5 Kinesthetic Feedback

It is well established that force reflection dramatically improves the sense of telepresence in teleoperation [Goertz,1963], [Ferrell,1966], [Hannaford,1988]. 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,1989].

One of the major features of the *teleprogramming* concept is the provision of real-time, bilateral kinesthetic interaction between the operator and the virtual environment of the graphical simulation, despite delayed communication with the remote site. The operator supplies input to the system by exerting forces at the master device and kinesthetically specifies the desired motion of the virtual slave robot. Conversely, a *teleprogramming* system provides the operator with a sense of real-time kinesthetic feedback of the slave's interactions with the virtual environment. Because the actual information arriving from the remote site is delayed, this real-time kinesthetic feedback must be derived from the simulated interactions between the virtual slave and its environment.

In view of Section 6.4, a *teleprogramming* system provides this facility by using the same constraint set $\mathcal{S}$, which was used above to restrict the contact motion of the virtual slave

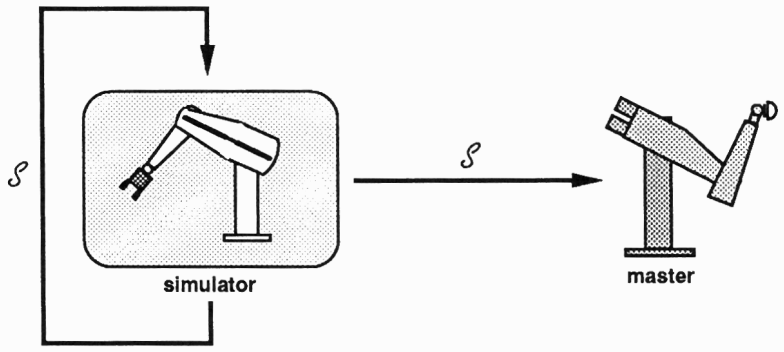


Figure 6.6: Virtual slave and master arm motion restriction.

robot, to also restrict the operator-supplied commanded motion of the master arm, i.e.,

$$\mathbf{D}_m \longrightarrow \boxed{\mathcal{S}} \longrightarrow \mathbf{D}'_m \tag{6.25}$$

Figure 6.6 illustrates the flow of constraint information between the virtual environment of the graphical simulation and the master arm. The constraint set is updated by the simulation process at each step and used to restrict the commanded motion of the simulated slave. It is then transformed into the view independent coordinates (Section 4.3) and sent to the master's controller. Here, the constraint set is scaled into the master's workspace and applied to the input motion as derived from the operator's exerted effort (Section 4.3). The operator-supplied motion command to the master arm $\mathbf{D}_m$ is therefore restricted by the same set of restriction operators, introduced in Section 6.4, to produce the modified set of motion parameters $\mathbf{D'}_m$, which in turn are used to drive the master device. The master is thus actively servoed to resist attempted motion in the constrained directions. This allows the operator holding the master arm to *kinesthetically* feel the impact of contacting a surface, reaching a corner, pivoting about an edge, etc. Despite its simplicity, this geometrically derived kinesthetic feedback provides a good approximation to actual force reflection for rigid-body environmental interactions.

The kinesthetic feedback facility not only provides for real-time pseudo force reflection, but also ensures that the geometric correspondence between the master and the virtual slave is maintained. This is crucial in order to maintain intuitive consistency between the expected and perceived spatial relationships on the part of the operator. The combination of consistent and mutually complementary real-time visual and kinesthetic feedback provides the operator with a strong sense of teleperception, which is essential for natural and efficient remote control of a robotic system.

Chapter 7

Symbolic Command Stream Generation

7.1 General

So far the operator can perform a task in the virtual environment by visually and kinesthetically interacting with the simulation of the remote site. The next important feature of the *teleprogramming* system is that the operator's station software is capable of monitoring the operator's activity in this simulated environment and extract from it a stream of symbolic robot instructions that capture all essential features of the task in progress. Figure 7.1 illustrates the "black-box" view of the command generation process.

Two sources of information are available to this module. The first consists of the low-level position and force trajectories, imparted by the operator, together with the current contact state and motion mode information. This information is provided directly by the graphical simulator and is updated at each simulation step (Chapter 5). The other source of information, which is available to the symbolic command generation module, is the *a priori* information about the task in progress, as defined in Section 3.6. This task model allows the *teleprogramming* system to anticipate, recognize, and correctly interpret special-purpose operations which are being performed by the operator.

The output of the command generation module are symbolic instructions, which can be again classified into two groups, as indicated in Figure 7.1. The first group is composed of *low-level* commands, essentially encompassing *guarded* and *compliant* motions. These commands are generated to execute simple tasks such as free-space navigation, motion into contact with the environment, contour following, etc., and are generated solely on the basis of positional, force and contact state information.

The *special-purpose* class of motions, on the other hand, encompasses special-purpose

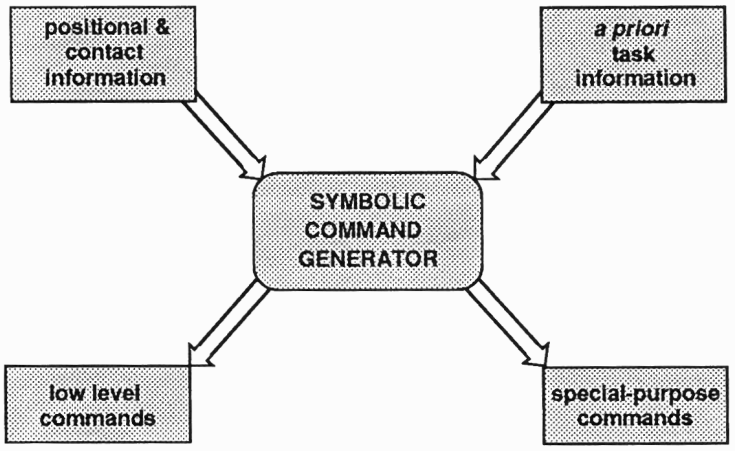


Figure 7.1: The command generation process.

or fine-precision operations, which are best executed autonomously by the slave under local sensory supervision. Examples of such actions include fine precision object alignment, grasping and handling of fragile or deformable objects, high-dexterity dynamically reactive manipulation tasks, etc. In this case, the task model provides global guidance to the process of interpreting low-level motions in the simulated environment, such that these special-purpose operations can be recognized in the input stream and proper symbolic instructions generated. The *teleprogramming* system should also provide a facility, whereby the operator could perform a small portion of a repetitive task (such as sawing, valve tightening, or polishing) and specify to the system to continue executing this task fragment until some terminating condition is met (such as excessive torque for valve tightening, or disappearance of the normal and tangential forces in sawing). These "procedures" should be simple, unparameterized, and defined on-line for one-time use. Again, the task model must contain sufficient information about the structure of the task to allow the system to recognize the operator's intent to initiate such a subtask. Likewise, the correct terminating conditions, corresponding to an initiated iterative procedure should be specified in (or inferable from) the task model.

We will in this chapter focus on the low-level command stream generation as it is the more basic and fundamental of the two command types. Section 7.2.1 describes the overall approach to low-level command generation and motivates the use of a hybrid position/force control model as the framework for the resulting instruction stream. In Section 7.2.2 we introduce the notion of execution environments, while Section 7.2.3 presents the global com-

mand generation algorithm. In Sections 7.2.4 through 7.2.7 we then give detailed description of the command generation process for various motion modes and contact configurations.

7.2 Low-level Command Generation

7.2.1 Approach

As the operator's station based model of the remote environment is only approximate (within known tolerance bounds), the nature of the generated low-level commands must reflect and accommodate the discrepancies between the modeled 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, the system generates sequences of guarded and compliant motion primitives, with the modeling uncertainties built into the motion parameters. Moreover, as the operator's station based simulation of the slave environment is kinematic in nature, the dynamic parameters of the requested motions (e.g., guard and compliance forces, frictional parameters) can not be given precisely. Instead, symbolic (or normalized numerical) values for these parameters are supplied to the slave, which in turn must substitute its estimates of the actual values prior to execution. These estimates are based on the slave's previous interactions with the environment, i.e., task history, and are derived from the local sensory readings at the remote site.

In order to cope with modeling uncertainties, as well as to increase the execution reliability and robustness at the remote site despite sensing and control errors, we adopted the *hybrid force/position* model ([Mason,1981], [Raibert&Craig,1981]) for the command stream generation process, as well as remote site execution. In this control methodology, the Cartesian space of manipulator's end-effector motions is partitioned into *free* and *constrained* directions. A free direction is one along (or about) which the manipulator can move freely, but can not exert any forces (or moments) on the environment. These directions of motion are therefore controlled in position mode. Dually, a constrained direction is one along (or about) which the manipulator can not move, but can exert arbitrary forces (or moments) on the environment. These axes are controlled in force mode. Thus, during free-space motion all six Cartesian motion directions are designated as free and thus position controlled. When in contact, on the other hand, the separation of the Cartesian motion parameters into free and constrained directions is determined by the nature and alignment of contact features. This normally results in position being controlled along some of the

Cartesian axes, while force is controlled along the others. The symbolic language, which the system uses to specify low-level actions to the remote slave, was designed to match this hybrid force/position control paradigm. Appendix B gives a description of the syntax and semantics of the low-level symbolic command language.

7.2.2 Execution Environments

The command generation process proceeds in terms of *execution environments*. An execution environment is a sequence of elementary instructions, which completely specifies a motion primitive and consists of *pre-motion*, *motion*, and *post-motion* phases.

The primary role of the pre-motion phase is to identify the coordinate frame (*task frame* TF) in which the subsequent motion parameters are to be interpreted. One of two predefined coordinate frames, the end-effector frame (EE) or the kinematic-base frame (KB), can be selected, or an entirely new task frame can be constructed from any three component vectors (origin plus any two axes). Moreover, the system can specify whether the task frame is to move along with the manipulator (*dynamic task frame*) or remain fixed with respect to world coordinates throughout the upcoming motion (*static task frame*) (see Appendix B for syntax and details). By convention, free space motions are commanded with respect to EE (dynamic frame). During contact manipulations, the task frame is centered at the primary contact point (Section 6.4.2) and aligned with the contacting features in such a way as to facilitate a clean separation of force and position controlled Cartesian directions for the remote slave manipulator [Mason,1981]. Besides the task frame, the pre-motion phase of an execution environment must specify the force guards in case of a guarded move and ensure that the existing force preloads (if any), as well as the control mode information, are correctly expressed in the new task frame.

The motion phase, in view of Section 6.1, specifies either a free-space movement, a sliding motion, or a pivoting motion. Finally, following the motion, we may need to reset the force guards to their default values (if the motion was guarded) and update the mode information and force preloads to reflect the new contact set (if the motion resulted in addition or deletion of contacts). These instructions are referred to as post-motion instructions.

7.2.3 The Global Algorithm

The *teleprogramming* system generates low-level symbolic commands by monitoring the elapsed time, contact state information, and motion trajectories of the slave manipulator and the movable object(s) in the simulated environment. A new sequence of instructions is

at the end of each simulation step {

1. $t_i \leftarrow$ current time
2. $e_i \leftarrow t_i - t_{i-1}$
3. $\mathcal{C}_i \leftarrow$ current contact set
4. $\mathcal{C}_{i-1} \leftarrow$ old contact set
5. $\Delta^{KB}\mathbf{d} \leftarrow$ end-effector displacement
6. case motion_mode of {

   free_space : Section 7.2.4 (Algorithm 7.2)
   sliding : Section 7.2.5 (Algorithm 7.4)
   pivoting : Section 7.2.6 (Algorithm 7.5)
   pushing : Section 7.2.7

   }
7. if command_generated {

   $t_{i-1} \leftarrow t_i$
   $\mathcal{C}_{i-1} \leftarrow \mathcal{C}_i$
   $T6_{i-1} \leftarrow T6_i$

   }

}

Algorithm 7.1: Low-level command generation algorithm.

issued after each addition or deletion of a contact or after the same contact state has persisted for t_{max} seconds. The time interval t_{max} is a function of the rate at which significant changes occur in the environment. Because this rate is limited by the human neuro-muscular bandwidth, t_{max} can be taken to be on the order of 1 second.

Algorithm 7.1 gives the global outline of the command generation process. Steps 1 and 2 of the algorithm compute the elapsed time since the time when the last execution environment was generated. Steps 3 and 4 make available to the system the current and old contact state information, as maintained by the graphical simulator (Chapter 5). In step 5 the incremental Cartesian end-effector displacement $\Delta^{KB}\mathbf{d} = (\mathbf{t}, \mathbf{r})$ is computed as follows

$$\mathbf{t} = \text{Trans}\left(\Delta^{KB}\mathbf{T}\right) \quad ; \quad \mathbf{r} = \text{RPY}\left(\text{Rot}\left(\Delta^{KB}\mathbf{T}\right)\right) \tag{7.1}$$

where

$$\Delta^{KB}\mathbf{T} = \mathbf{T6}_i * (\mathbf{T6}_{i-1})^{-1} \tag{7.2}$$

and the RPY operator denotes that the corresponding incremental rotational motion is expressed as a roll/pitch/yaw vector. The homogeneous transform $\mathbf{T6}_k$ in Eq. (7.2) above denotes the location of the manipulator's wrist with respect to its kinematic base (KB) at k-th simulation step. The heart of the procedure is step 6, where the changes in the simulated environment since the generation of the last execution environment are examined and the corresponding symbolic instructions generated, if appropriate. Different command generation algorithms apply to different motion modes. We will develop each of these algorithms in the following sections. Finally, if a new execution environment was generated in this step, the relevant current information is stored to serve as "old" information for subsequent iterations of the algorithm.

The following sections address command generation for free-space motion (Section 7.2.4), as well as for each of the three contact motion modes (Sections 7.2.5 through 7.2.7). Each section presents the analysis of representative cases and gives the corresponding execution environments (command sequences), as well as the final algorithm, summarizing the results. In the interest of brevity and proper emphasis on the methodology and semantics, rather than syntax, we will in the following sections occasionally abbreviate the syntax of the language (Appendix B). In particular, where no confusion can arise, we may write **v** in place of the syntactic construct $< v_x, v_y, v_z >$, indicate default parameter values simply as <default>, etc. Other abbreviations and notational conventions will be introduced in the text as needed.

7.2.4 Free-space Motion

During free-space motion, command generation proceeds in a straightforward fashion according to Algorithm 7.2. The vectors **t** and **r** are the incremental translational and rotational wrist-based displacements of Eq. (7.1), appropriately rotated into the current task frame (EE), and t denotes the duration of the requested motion (seconds).

```
if (e_i > t_max) {
    case motion_mode of {
        free_motion    : Move (t; < t_x, t_y, t_z >; < r_x, r_y, r_z >)
        translation    : Move (t; < t_x, t_y, t_z >; < 0, 0, 0 >)
        rotation       : Move (t; < 0, 0, 0 >; < r_x, r_y, r_z >)
    }
}
```

Algorithm 7.2: Command generation algorithm for free-space motion.

7.2.5 Sliding

Case Analysis

(i) motion within contact: $\|\mathcal{C}_i\| > 0$

1. **Slide**$(t; < t_x, t_y, t_z >)$ (7.3)

This case corresponds to sliding in contact (single or multiple) without changing the contact type(s) or multiplicity. As in the case of free-space motion, a new execution environment is generated if the sliding contact state persists for t_{max} seconds. Note that only the motion trajectory information need be specified to the slave. The task frame as well as the associated control modes and force preloads remain unchanged.

(ii) motion into contact: $\|\mathcal{C}_{i-1}\| = 0$, $\|\mathcal{C}_i\| = 1$

$\cdots$

1. **UseFrame**(TF)
2. **GuardForce**$(< 0, 0, F_{contact} >; < 0, 0, 0 >)$
3. **Move**$(\mathrm{t};\ \mathbf{t} + \mathrm{sgn}(\mathbf{t})\epsilon_p\ ;\ \mathbf{r} + \mathrm{sgn}(\mathbf{r})\epsilon_r)$
4. **AssignMode**$(\ \cdots\ )$ (7.4)
5. **Force**$(< 0, 0, -F_{comply} >; < 0, 0, 0 >)$
6. **GuardForce**(<default>)

This case corresponds to the situation where the movable object is transitioning from free-space into a single-contact configuration. Figure 7.2 illustrates representative examples of

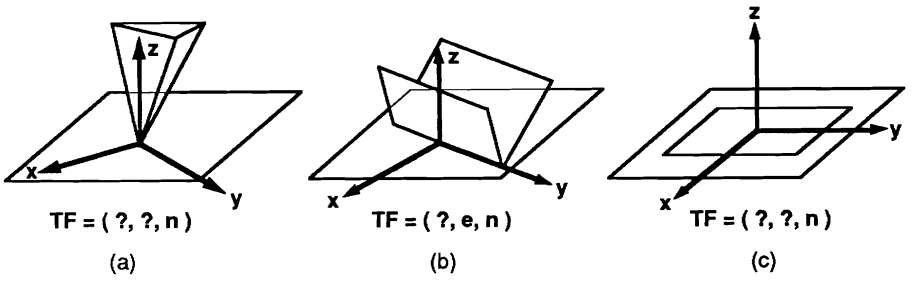


Figure 7.2: Representative examples of the three contact classes.

the three possible contact configurations on impact (see also Figure 5.1). In each case, as indicated by the corresponding figures, the task frame is centered at the impending contact point and aligned with the impending contact features.[1] The task frame TF in each of the above cases is specified with a sequence of instructions of the form[2]

1. **DefineVector**(CP;< cp_x, cp_y, cp_z >:F1)
2. **DefineVector**(X;< ax_x, ax_y, ax_z >:F2)
3. **DefineVector**(Z;< az_x, az_y, az_z >:F3)
4. **DefineTaskFrame**(TF:F4;CP;X;?;Z)

where CP denotes the contact point (task frame origin) and X and Z label the corresponding TF axes. The numeric values of the vectors are obtained from the graphical simulation. The coordinate frames F1 through F4 are not necessarily distinct and must either have been defined with a previous **DefineTaskFrame** command or are equal to one of the predefined coordinate frames (EE or KB). In the interest of brevity we will omit these task frame specification instruction sequences and indicate their presence with an ellipse ($\cdots$), where appropriate, as in the execution environment 7.4 above.

The pre-motion phase (instructions 1–2) of the execution environment (7.4) identify the task frame to be used throughout the environment, and indicate to the slave controller that a contact force in the approach direction (TF z-axis) is expected during the upcoming motion. The contact force $F_{contact}$ is specified as a normalized numerical value, rather than an actual force value. The incremental motion parameters $\mathbf{t}$ and $\mathbf{r}$, as computed from the graphical simulation (Section 7.2.3), are adjusted by the estimated upper bounds on

[1] Note the similarity of this task frame assignment to the assignment of the restriction frame in Chapter 6.

[2] See Appendix B for the explanation of the syntax and semantics of vector and frame specification.

the positional (ϵ_p) and orientational (ϵ_r) uncertainty in the world model (Section 3.2).[3] Once the guard force is encountered and the motion stops, the post-motion phase begins by updating the control mode information. In view of Figure 7.2, the six-vector of modes, specified in instruction 4, is given as follows

(P,P,F,P,P,P)	for a point contact
(P,P,F,F,P,P)	for a line contact
(P,P,F,F,F,P)	for a plane contact

The post-motion phase terminates by specifying a compliance force against the new constraint (along negative TF z-axis), and resetting the force guards to their default values. Again, F_{comply} will be given as a normalized numerical value, and its actual magnitude will be determined at the remote site, based on the frictional properties of the contact. For sliding motions, F_{comply} will normally be a relatively small force to avoid introducing excessive additional contact friction. In particular, we presumably have $F_{contact} > F_{comply}$, and we may therefore want to distinguish between compliance and contact forces by using different normalized values for the two.

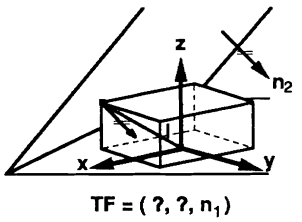
(iii) motion out of contact: $\|\mathcal{C}_{i-1}\| > 0$, $\|\mathcal{C}_i\| = 0$

1. **UseFrame**(EE)
2. **AssignMode**(P,P,P,P,P,P)
3. **Force**($< 0,0,0 >;< 0,0,0 >$) (7.5)
3. **Move**(t;$< t_x, t_y, t_z >;< r_x, r_y, r_z >$)

This case corresponds to the situation where MO breaks contact(s) with the environment and transitions into free-space. The pre-motion phase here specifies that the end-effector frame (EE) is to be used as the task frame (see Section 7.2.2), zeroes any force preloads and compliance forces, and places all task frame axes into position mode. The post-motion phase in this case is null.

[3]The notation sgn(**u**)**v** is used to denote $< \text{sgn}(u_x)v_x, \text{sgn}(u_y)v_y, \text{sgn}(u_z)v_z >$.

(iv) motion into contact: $\|\mathcal{C}_{i-1}\| > 0$, $\|\mathcal{C}_i\| > \|\mathcal{C}_{i-1}\|$



1. **GuardForce**($< f_x, f_y, f_z >; < \tau_x, \tau_y, \tau_z >$)
2. **Slide**($t; \mathbf{t} + \text{sgn}(\mathbf{t})\epsilon_p$)
3. **AssignMode**($\cdots$) (7.6)
4. **Force**($\cdots$)
5. **GuardForce**(<default>)

This case corresponds to the situation where MO, already in contact, reaches a higher contact multiplicity configuration (i.e., another contact). The old task frame TF_{i-1} is used to specify the sliding motion into the new contact state. The pre-motion phase therefore consists only of specifying the guard forces, which are to be expected during the upcoming motion. These guard forces must be expressed in the task frame (TF) and are computed as follows

$$\begin{aligned} {}^{TF}\mathbf{f} &= {}^{TF}\mathbf{R}_{KB} * {}^{KB}\mathbf{f} = {}^{TF}\mathbf{R}_{KB} * {}^{KB}(\alpha\mathbf{n}) \\ {}^{TF}\boldsymbol{\tau} &= {}^{TF}\mathbf{R}_{KB} * \left({}^{KB}\mathbf{l} \times {}^{KB}\mathbf{f}\right) = {}^{TF}\mathbf{R}_{KB} * {}^{KB}(\mathbf{l} \times (\alpha\mathbf{n})) \end{aligned} \tag{7.7}$$

where $\mathbf{n}$ is the contact normal of the upcoming contact c. As before, within the framework of symbolic instruction generation, $\alpha\mathbf{n}$ is taken as the normalized value of the contact force vector ($\alpha \in \mathbb{R}$). The vector $\mathbf{l}$ denotes the KB-frame moment arm connecting the origin of TF with the impending contact point (see the figure).

Following the motion, the new contact $c \in \mathcal{C}_i$ must be reflected in the control mode and force preload information. Therefore, the post-motion phase in this case needs to project the newly added force controlled directions into TF and properly update the mode and force information (instructions 3 and 4). Depending on the type of the new contact, the added force/torque controlled directions are given in c's local frame of reference as shown in Table 7.1.[4] These additional force controlled directions are mapped into TF and their contributions reflected in the TF-based force preload and mode information according to the procedure given in Algorithm 7.3. In view of Table 7.1, steps 2 and 3 of Algorithm 7.3 compute the new force and torque controlled directions in TF, respectively, and return the updated information in the six-vector **force**. Likewise, the updated six-vector of TF control modes is returned in **mode**. The parameter $\lambda \leq 1$ in Algorithm 7.3 denotes the normalized force projection threshold, i.e., if any of the new force/torque controlled directions project onto more that $100\lambda\%$ of a position controlled TF axis, then this axis is designated as

[4]The vector $\mathbf{n}'$ in Table 7.1 is chosen such that $\mathbf{n} \times \mathbf{n}' \neq \mathbf{0}$.

force	torque	contact class
$\mathbf{n}$	$\mathbf{0}$	point contact
$\mathbf{n}$	$\mathbf{n} \times \mathbf{e}$	line contact
$\mathbf{n}$	$\mathbf{n} \times \mathbf{n}'$, $(\mathbf{n} \times \mathbf{n}') \times \mathbf{n}$	plane contact

Table 7.1: Local force/torque constraints for the three contact classes.

---

1. ${}^{TF}\mathbf{n} = {}^{TF}\mathbf{R}_{KB} * {}^{KB}\mathbf{n}$
2. for $j = 1$ to 3 do {

   if ($\|{}^{TF}\mathbf{n}[j]\| > \lambda$) {

   mode[j] = F

   force[j] = - sgn(${}^{TF}\mathbf{n}[j]$)

   }

   }
3. case ContactClass(c) {

   point_contact :

   line_contact : ${}^{TF}\mathbf{u} = {}^{TF}\mathbf{R}_{KB} * {}^{KB}(\mathbf{n} \times \mathbf{e})$

   for $j = 1$ to 3 do {

   if ($\|{}^{TF}\mathbf{u}[j]\| > \lambda$) mode[j+3] = F

   }

   plane_contact : ${}^{TF}\mathbf{u} = {}^{TF}\mathbf{R}_{KB} * {}^{KB}(\mathbf{n} \times \mathbf{n}')$

   ${}^{TF}\mathbf{v} = {}^{TF}\mathbf{R}_{KB} * {}^{KB}((\mathbf{n} \times \mathbf{n}') \times \mathbf{n})$

   for $j = 1$ to 3 do {

   if ($\|{}^{TF}\mathbf{u}[j]\| > \lambda$) mode[j+3] = F

   if ($\|{}^{TF}\mathbf{v}[j]\| > \lambda$) mode[j+3] = F

   }

   }

Algorithm 7.3: Computing force and control mode information.

force controlled and a force preload is specified along it.[5] Note that the correct *direction* of the force preload needs to be identified for each axis in step 2 as compliance forces are specified *against* constrained directions. Moreover, torque preloads are identically zero and so **force[4–6]** need not be computed in step 3.

The mode information as computed in **mode** is then used to instantiate instruction 3 in the execution environment (7.6) above. Likewise, the force information returned in **force** gives the normalized force preloads required by instruction 4. Finally, in instruction 5 the post-motion phase of the execution environment terminates by resetting the force guards that were in effect during the motion into contact to their default values.

(v) motion out of contact: $\|\mathcal{C}_i\| > 0$

$$
\begin{array}{ll}
 & \cdots \\
1. & \textbf{UseFrame}(\text{TF}) \\
2. & \textbf{AssignMode}(\ \cdots\ ) \\
3. & \textbf{Force}(\ \cdots\ ) \\
4. & \textbf{Slide}(\text{t};< t_x, t_y, t_z >)
\end{array}
\tag{7.8}
$$

This case corresponds to sliding from a higher-multiplicity to a lower-multiplicity contact. The pre-motion phase defines the new task frame, based on the new primary contact point and the new contact set $\mathcal{C}_i$. In order to properly reflect the new (reduced) contact set in the control mode and force preload information in the new task frame, we again make use of Algorithm 7.3, where the algorithm is called once for each $c \in \mathcal{C}_i$. The resulting mode and force preload information is then used to instantiate instructions 2 and 3 above.

The Algorithm

The complete command generation procedure for sliding motions is given in Algorithm 7.4.

For simplicity, the above discussion of contact sliding motions omitted the specification of velocity guards (**GuardVelocity** statement, Appendix B). Velocity guards can be specified along force controlled task frame axes in situations where there is a danger of the movable object accidentally being slid off the supporting surface (e.g., the supporting environment feature is small or MO is being slid close to a supporting surface edge). Should MO be accidentally slid off the supporting surface and start falling, a sudden acceleration along the corresponding force controlled task frame axis would trigger the velocity guard condition

[5] A reasonable value for λ may be 0.15 (15%).

```
if (nc_i > nc_{i-1}) {
    case nc_i of {
        1    : case (ii) of Section 7.2.5
        2, 3 : case (iv) of Section 7.2.5
    }
}
else if (nc_i < nc_{i-1}) {
    case nc_i of {
        0    : case (iii) of Section 7.2.5
        1, 2 : case (v) of Section 7.2.5
    }
}
else if (e_i > t_max) {
    case (i) of Section 7.2.5
}
```

Algorithm 7.4: Command generation algorithm for sliding contact motion.

and stop the motion. In fact, we may choose to explicitly specify such velocity guards along every force controlled direction or, indeed, specify them implicitly by making their presence the default.

7.2.6 Pivoting

For the case of pivoting, the slave workcell is asked to execute a motion, which results in a pure translation about the current contact point. We will assume in the case analysis below that the slave manipulator (e.g., MO) has been brought into contact with the environment and that the pivoting motion mode has been selected. In order to aid the remote workcell in executing pivoting motions, we will request a substantial compliance force F_{pivot} against the supporting surface (i.e., $F_{pivot} > F_{comply}$) and assume the frictional contact model [Mason&Salisbury,1985]. The hybrid control modes for the case of pivoting motions are therefore assigned as follows

(F,F,F,P,P,P)	for a point contact
(F,F,F,F,P,F)	for a line contact
(F,F,F,F,F,F)	for a plane contact

Case Analysis

(i) pivoting within contact type: $\|\mathcal{C}_i\| = 1$

This case pertains to the single-contact pivoting situations where the incremental motion does not change the contact type. In view of Figure 7.2, the following single command execution environments are generated for the case of point, line, and plane contact, respectively

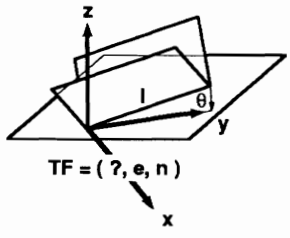
$$\begin{aligned} &\text{(a) } \textbf{Pivot}(\text{t}; < r_x, r_y, r_z >) \\ &\text{(b) } \textbf{Pivot}(\text{t}; < 0, r_y, r_z >) \\ &\text{(c) } \textbf{Pivot}(\text{t}; < 0, 0, r_z >) \end{aligned} \tag{7.9}$$

Note that the rotational contact motion is specified in the old task frame. Also, the existing mode information as well as force preloads (compliance forces) remain in effect. The above commands are generated only after the contact state has persisted for t_{max} seconds.

(ii) pivoting between contact types: $\|\mathcal{C}_i\| = 1$

This case pertains to situations where the motion to be commanded to the slave changes the (single) contact type between MO and the environment. The following four cases detail command generation for a representative subset of all possible situations.

(a) point contact → line contact



...

1. **UseFrame**(TF)
2. **GuardForce**($\mathbf{0}; < -\text{sgn}(\theta_x)(l \cdot F_{contact}), 0, 0 >$)
3. **Pivot**($t; < \theta_x + \text{sgn}(\theta_x)\epsilon_r, \theta_y, \theta_z >$)

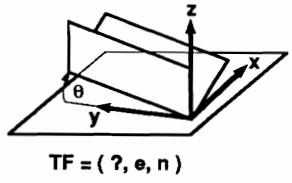
...

4. **UseFrame**(TF')
5. **AssignMode**(F,F,F,F,P,F)
6. **GuardForce**(<default>)

(7.10)

Here, the pre-motion phase aligns TF with the impending edge contact and specifies the proper torque guard on TF x-axis. Following the motion, the task frame is repositioned (no change in orientation) to the center of the new edge contact (instruction 4), and the mode information is updated to reflect the new contact state.

(b) line contact → point contact



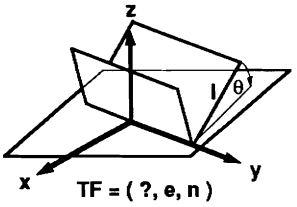
...

1. **UseFrame**(TF)
2. **Pivot**($t; < \theta_x, \theta_y, \theta_z >$)
3. **AssignMode**(F,F,F,P,P,P)

(7.11)

In this case, the pre-motion phase translates the task frame to the upcoming vertex contact. Control modes are updated following the motion.

(c) line contact → plane contact

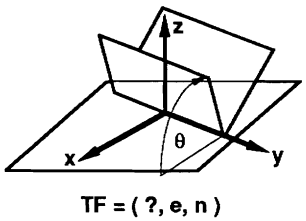


1. **GuardForce**$(\mathbf{0}; < 0, -\text{sgn}(\theta_y)(l \cdot F_{contact}), 0 >)$
2. **Pivot**$(t; < 0, \theta_y + \text{sgn}(\theta_y)\epsilon_r, 0 >)$
   ...
3. **UseFrame**(TF')
4. **AssignMode**(F,F,F,F,F,F)
5. **GuardForce**(<default>)

(7.12)

The pre-motion phase need not change the task frame. However, following the guarded motion into the planar contact, the task frame is translated to the centroid of the contacting planar surface (instruction 3), and the control modes are updated with respect to the new contact state.

(d) plane contact → line contact

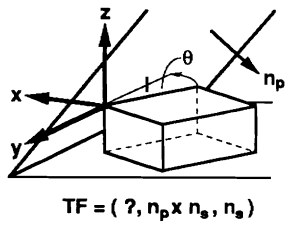


...
1. **UseFrame**(TF)
2. **Pivot**$(t; < 0, \theta_y, 0 >)$
3. **AssignMode**(F,F,F,F,P,F)

(7.13)

In this case, the pre-motion stage positions the task frame at the center of the edge, corresponding to the upcoming line contact feature, and properly aligns TF with the edge direction. Mode information is updated in the post-motion phase.

(iii) pivoting in multiple contact: $\|\mathcal{C}_i\| = 2$



...
1. **UseFrame**(TF)
2. **AssignMode**(···)
3. **Force**(···)
4. **GuardForce**$(\mathbf{0}; < 0, 0, -\text{sgn}(\theta_z)(l \cdot F_{contact}) >)$
5. **Pivot**$(t; < 0, 0, \theta_y + \text{sgn}(\theta_y)\epsilon_r >)$
   ...
6. **UseFrame**(TF')
7. **AssignMode**(F,F,F,F,F,F)
8. **GuardForce**(< default >)

(7.14)

This case encompasses the reorienting pivoting motions with two active contacts as discussed

in Section 6.4.3. Using the conventions and terminology introduced in Chapter 6, the task frame TF is centered at the primary contact point and aligned as shown in the figure above (instruction 1). The pre-motion force preload and mode information is then properly rotated into this new task frame by invoking Algorithm 7.3 for each contact $c \in \mathcal{C}_{i-1}$ (instructions 2 and 3). The pre-motion phase of the execution environment terminates by specifying the necessary guard forces relative to TF (instruction 4). Following the rotational motion about TF z-axis, the task frame is repositioned to the primary contact feature centroid and the mode information is updated to reflect the changed nature of the contact set (instructions 6 and 7). Note that the preload information need not be updated, as zero torque preloads are the default. Finally, as in every guarded move, the guard forces are reset to their default values following the motion.

The Algorithm

The complete command generation procedure for pivoting motions is given in Algorithm 7.5, where c_{i-1} and c_i denote the primary contact before and after the motion, respectively.

7.2.7 Pushing

The command generation for the three basic types of pushing motions — straight-line single contact pushing, rotational realignment pushing, and pushing along a pair of surfaces (Section 6.4.4) proceeds analogously to the command generation for the corresponding sliding and pivoting cases. The only exception is the addition of force/torque preloads along the pushing directions. These are specified in order to aid the slave manipulator in overcoming the frictional forces, working against the desired motion. In the interest of brevity we will omit the detailed description of the corresponding execution environments.

As in the case of sliding, we may also choose to specify velocity guards along the pushing direction to detect conditions such as tipping or twisting of the pushed object. Both conditions would result in sudden acceleration of the slave arm along the pushing direction, which in turn would trigger the velocity guards and safely stop the motion.

```
case nc_i of {
 1 : case ContactClass(c_i) of {
        point_contact : case ContactClass(c_{i-1}) of {
                            point_contact   : case (i.a) of Section 7.2.6
                            line_contact    : case (ii.a) of Section 7.2.6
                            plane_contact   : ?
                        }
        line_contact :  case ContactClass(c_{i-1}) of {
                            point_contact   : case (ii.b) of Section 7.2.6
                            line_contact    : case (i.b) of Section 7.2.6
                            plane_contact   : case (ii.c) of Section 7.2.6
                        }
        plane_contact : case ContactClass(c_{i-1}) of {
                            point_contact   : ?
                            line_contact    : case (ii.d) of Section 7.2.6
                            plane_contact   : case (i.c) of Section 7.2.6
                        }
     }
 2 : case (iii) of Section 7.2.6
}
```

Algorithm 7.5: Command generation algorithm for pivoting contact motion.

Chapter 8

The Remote Slave

The preceding chapters (4 through 7) have dealt with the operator's station portion of the *teleprogramming* system (see Figure 3.2). The operator's station visually and kinesthetically couples a human operator to a graphical simulation of the remote environment and allows her to interactively, via a 6 d.o.f. master device, specify the task to be performed remotely. The final output of the operator's station, as described in Chapter 7, is a stream of execution environments, each containing a description of an elementary motion or action to be performed by the slave workcell.

In this chapter we turn our attention to the remote slave workcell and its interaction with the environment as well as with the operator's station. In the upcoming sections we will first address the instruction parsing and translation (Section 8.1). We will next present a strategy for parsing, scheduling, and executing the received instructions, which guarantees that the time lag between the master and the slave will not increase during the task (Section 8.2). In Section 8.3 we suggest a simple Cartesian level hybrid force/position control algorithm for the slave manipulator, and Section 8.4 closes our brief treatment of the remote workcell with some general comments on error handling and recovery.

8.1 Command Parsing and Translation

As the operator performs a task by interacting with a ground-station based graphical simulation of the remote environment, the operator's station software generates (on line) a stream of symbolic instructions, describing the operator's activity in the simulated environment (Chapter 7). These instructions, grouped into execution environments, arrive to the remote workcell a transmission delay τ after they were generated and sent from the

operator's station. Execution at the remote site then proceeds by

1. parsing and translating the contents of successive execution environments into the local control language,

2. substituting numerical values for the symbolic (or normalized numeric) dynamic parameters (e.g., friction coefficients, compliance force levels),

3. passing the resulting code to the local controller for execution, and

4. monitoring the execution process — detecting error conditions, stopping the slave workcell safely on error, and reporting resulting error state to the operator's station

The symbolic command language, which is used by the *teleprogramming* system to encode elementary slave motion and action information (Appendix B), was designed to be closely compatible with the hybrid position/force control paradigm, as proposed by [Raibert&Craig,1981]. If the slave control software supports the same control strategy, then the process of parsing the incoming symbolic instructions and producing the corresponding instructions, which are directly executable by the local slave controller, is straightforward.

The symbolic command language described in Appendix B is a context-free language and consists of simple declarative statements with no looping or branching constructs. The corresponding BNF grammar can therefore be readily produced and fed to an automatic parser generator (such as **yacc**) to produce a parser (an LALR parser in case of **yacc**) [Aho *et al.*,1986]. Having produced a parser, the code generation process then proceeds as follows:

1. Some of the instructions, such as **UseFrame**, **AssignMode**, **Move**, **Pivot**, and **Slide**, set the corresponding control parameters (i.e., current task frame, control modes, motion time and trajectory) in the slave's controller directly, and no additional processing is necessary.

2. Other instructions, such as **Force**, **GuardForce**, and **GuardVelocity**, however, do require some additional processing. In particular, in view of the kinematic nature of the operator's station based simulation (Chapter 5), the parameters, supplied by these instructions, do not reflect proper dynamics of the slave manipulator and the environmental objects being manipulated. As we saw in Chapter 7, the instructions instead contain symbolic or normalized numeric values to denote dynamic parameters, such as frictional properties of a sliding contact, compliance forces during sliding,

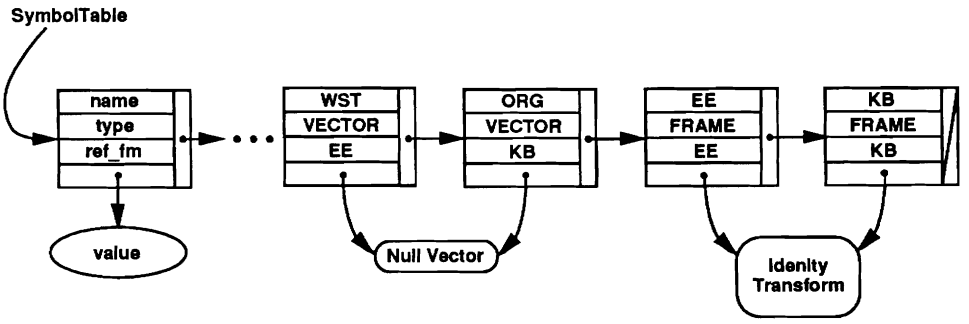


Figure 8.1: Slave controller symbol table.

guard forces while approaching a new contact, etc. The translation process must therefore substitute actual (estimated) values for the symbolic placeholders. These estimates of the dynamic parameters of the slave's interaction with the environment are refined as the task progresses and sensory measurements can be used to get a better sense of the frictional characteristics of the immediate environment, masses and inertial properties of the manipulated objects, etc. The responsibility of obtaining this information lies with the slave controller which must keep track of the relevant dynamic parameters and record the necessary sensory data during motion, in order to maintain updated estimates of their actual values.

3. The third group of instructions, represented by **DefineVector** and **DefineTaskFrame**, serves to update the *symbol table* of currently defined vectors and coordinate frames, known to the slave controller. In view of Section 7.2.2, the symbol table contains four predefined entries and has the general form as illustrated in Figure 8.1.[1] The two frames KB and EE correspond to the kinematic base and end-effector (wrist) frames, respectively. Likewise, ORG and WST denote their respective origins in local coordinates. This suffices to bootstrap the task frame definition process, whereby new vectors can be defined with respect to any defined coordinate frame, and new task frames, in turn, can be composed from any combination of these vectors. In view of Section 7.2.2, task frames are specified to be either static (defined with respect to KB) or dynamic (defined with respect to EE). The symbol table of Figure 8.1 can therefore be thought of as encoding the conceptual data structure of Figure 8.2. In order to correctly translate a statement of the form

[1] For clarity, the symbol table in Figure 8.1 is shown as a linked list. More time and space efficient data structures, such as hash tables, should be used in actual implementations.

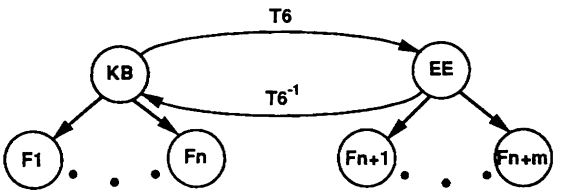


Figure 8.2: Conceptual relationship between task frames.

---

1. dst_fm = Fj
2. src_fm = (Lookup(**v**))→ref_fm
3. dst_root = Lookup(dst_fm)→ref_fm
4. src_root = Lookup(src_fm)→ref_fm
5. $\mathbf{T} = \mathbf{I}$
6. if (dst_fm $\neq$ dst_root) $\mathbf{T} = \mathbf{T} *$ (Lookup(dst_fm)→value)$^{-1}$
7. if (src_root $\neq$ dst_root) $\mathbf{T} = \mathbf{T} *$ ((dst_root == KB) ? **T6** : $\mathbf{T6}^{-1}$)
8. if (src_fm $\neq$ src_root) $\mathbf{T} = \mathbf{T} *$ Lookup(src_fm)→value

---

Algorithm 8.1: Traversing the symbol table to compute ${}^{\text{dst\_fm}}\mathbf{T}_{\text{src\_fm}}$.

DefineTaskFrame(TF:F3;V1;V2;?;V3)

the vectors V1, V2, and V3 must be rotated from their respective reference frames (defined as part of their symbol table entries) into F3 and the resulting vectors used to assemble the new coordinate frame data structure $\mathcal{F}_{TF}$. The corresponding symbol table entry with name=TF, type=FRAME, ref_fm=F3, and value=$\mathcal{F}_{TF}$ is then added to the symbol table. The key operation in this context is fast and efficient mapping of a vector **v**, given with respect to Fi coordinates, into Fj coordinates. In order to affect this transformation, the parser/translator needs to compute the transformation (rotation matrix) ${}^{Fj}\mathbf{T}_{Fi}$. In view of Figure 8.2, the corresponding procedure is given by Algorithm 8.1. Note that further optimizations of the algorithms are possible. They have been omitted here for brevity and simplicity.

8.2 Execution Management and Lag Control

During execution care must be taken to avoid increasing the *lag time* η between the master and the slave manipulators as the task proceeds. A straightforward dequeue-parse-execute

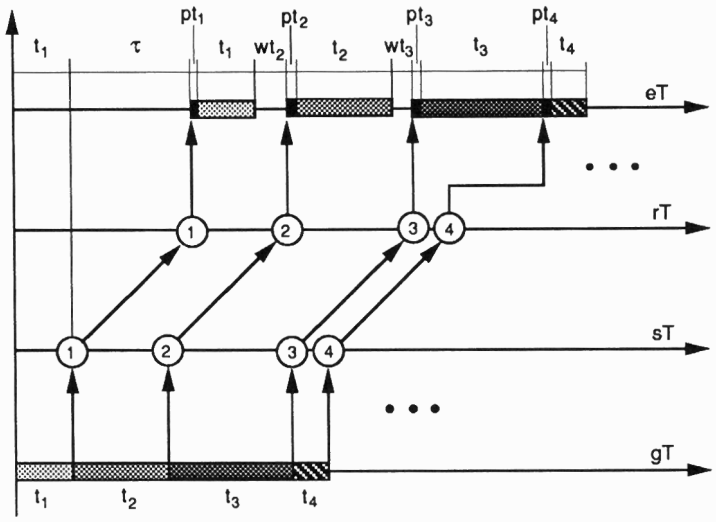


Figure 8.3: Sequential dequeue-parse-execute execution management.

loop leads to the behavior of Figure 8.3, where the lag time increases throughout the duration of the task. The following notation is used in Figure 8.3 and throughout this section

- ⓘ — the i^{th} execution environment
- gT_i — time when ⓘ started being generated
- sT_i — time when ⓘ was sent from the operator's station
- rT_i — time when ⓘ was received by the remote controller
- eT_i — time when ⓘ began executing at the remote site
- t_i — execution length of ⓘ
- pt_i — parsing time for ⓘ
- wt_i — time spent waiting for ⓘ

The waiting time wt_i is defined as follows

$$wt_i = \begin{cases} rT_i - (eT_{i-1} + t_{i-1}) & , \text{ if } rT_i > (eT_{i-1} + t_{i-1}) \\ 0 & , \text{ otherwise} \end{cases} \tag{8.1}$$

Note that the waiting time can not be negative. Then, using sequential dequeue-parse-execute approach, the lag time η_i at time eT_i (just before executing ⓘ) is given by

$$\eta_i = eT_i - gT_i = t_1 + \tau + \sum_{j=1}^{i}(pt_j + wt_j) \tag{8.2}$$

Eq. (8.2) implies that even if the sum of waiting times is bounded, i.e., if

$$\lim_{i\to\infty} \sum_{j=1}^{i} wt_j < W \tag{8.3}$$

for some arbitrarily large constant W, we have

$$\lim_{i\to\infty} \eta_i = \infty \tag{8.4}$$

indicating that the lag time not only increases as the task progresses, but is in fact unbounded.

In order to solve this problem, we employ a *double-buffering* execution scheme, as illustrated in Figure 8.4. In the double-buffering execution paradigm each dequeued execution environment is translated and placed into a *command buffer*. The remote controller maintains two such buffers (A and B in Figure 8.4) — while one is being executed, the other is being constructed by parsing and translating the next execution environment. We will show below that the combination of this parallelism and an artificially introduced *holding time* ht_1, which delays the execution of ① by ht_1, can be used to control the lag. The holding time ht_1 initially increases the lag time, but keeps it *constant* and *bounded* from then on, as will be shown below.

In terms of the above nomenclature, the necessary and sufficient condition to ensure non-increasing lag time η, is

$$\forall i : rT_i + pt_i \leq eT_{i-1} + t_{i-1} \tag{8.5}$$

A stricter version of the above condition can be stated as the following pair of requirements

$$\forall i \;:\; rT_i \leq eT_{i-1} \quad \text{, and} \tag{8.6}$$

$$\forall i \;:\; pt_i \leq t_{i-1} \tag{8.7}$$

Clearly, satisfaction of conditions (8.6) and (8.7) implies satisfaction of condition (8.5). The above requires ⓘ to have arrived at the remote site before the $(i-1)^{th}$ execution environment begins executing, and that ⓘ be ready for execution (parsed and translated into the back-up command buffer) before the $(i-1)^{th}$ execution environment finishes executing. Practical considerations allow us to assume that the requirement of Eq. (8.7) will be satisfied in all situations. What remains to be shown is that we can guarantee the condition of Eq. (8.6). The following proposition establishes this result. See Figure 8.5 for illustration.

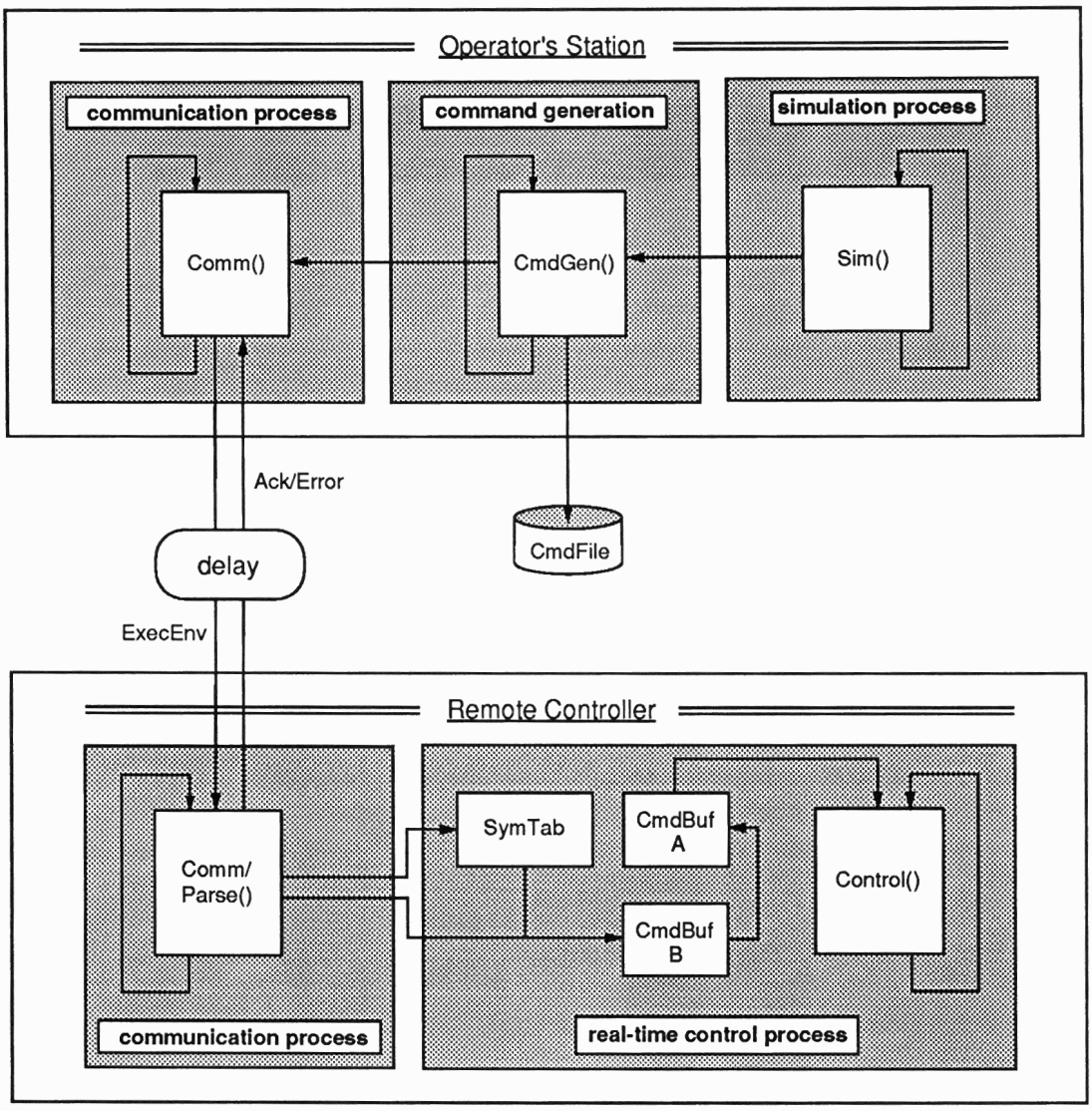


Figure 8.4: Double-buffering remote site execution scheme.

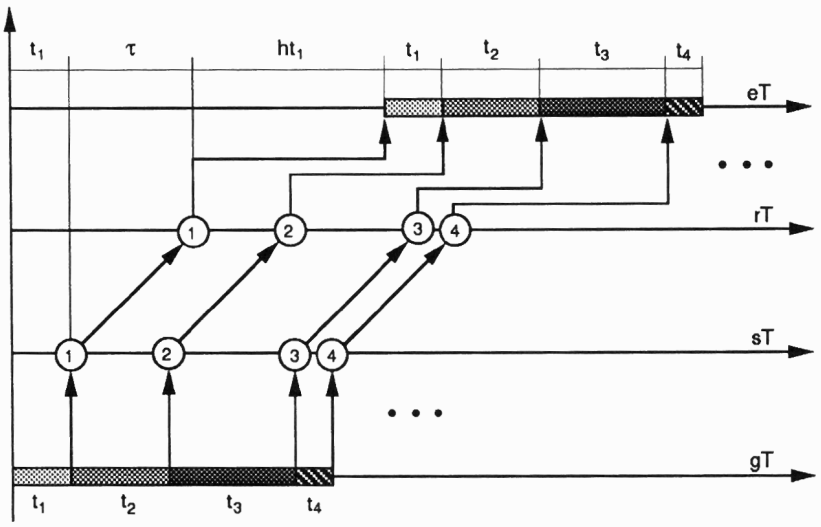


Figure 8.5: Double-buffering execution management.

Proposition : Let $ht_1 = (2\,t_{max} - t_1)$ and assume that $\forall i : pt_i \leq t_{i-1}$. Then

$$\forall i : rT_i \leq eT_{i-1} \quad \text{and} \quad \eta \leq (2\,t_{max} + \tau)$$

Proof : In double-buffering execution paradigm, the times gT_i, sT_i, rT_i, and eT_i are formally defined as follows

$$\begin{aligned} gT_i &= \sum_{j=1}^{i-1} t_j \;;\; t_0 = 0 \\ sT_i &= \sum_{j=1}^{i} t_j = gT_i + t_i \\ rT_i &= \sum_{j=1}^{i} t_j + \tau = sT_i + \tau \\ eT_i &= t_1 + \tau + ht_1 + \sum_{j=1}^{i-1} t_j \end{aligned}$$

Recalling that $t_i \leq t_{max}$ (Chapter 7), we have

$$\begin{aligned} rT_i &= \sum_{j=1}^{i} t_j + \tau \\ &= \sum_{j=1}^{i-2} t_j + t_{i-1} + t_i + \tau \end{aligned}$$

$$
\begin{aligned}
&\leq \sum_{j=1}^{i-2} t_j + 2\,t_{max} + \tau \\
&= \sum_{j=1}^{i-2} t_j + t_1 + (2\,t_{max} - t_1) + \tau \\
&= \sum_{j=1}^{i-2} t_j + t_1 + ht_1 + \tau \\
&= eT_{i-1}
\end{aligned}
$$

Moreover,

$$
\begin{aligned}
\eta &= \lim_{i\to\infty} \eta_i \\
&= \lim_{i\to\infty} (eT_i - gT_i) \\
&= \lim_{i\to\infty} (t_1 + \tau + ht_1) \\
&= 2\,t_{max} + \tau
\end{aligned}
$$

□

This completes the proof that the double-buffering execution scheme keeps the lag time between the master and the slave arms not only bounded, but constant throughout the execution of a *teleprogramming* task.

Notice, however, that the above execution scheme maintains a lag time of $\eta = 2\,t_{max}$ even in the presence of no communication delay ($\tau = 0$). This is a direct consequence of conditions (8.6) and (8.7).

8.3 Control of the Slave Manipulator

The symbolic instructions arriving at the slave site are based on an imperfect model of the actual environment. Despite the fact that the critical motion parameters (e.g., distances to surfaces or edges) have been computed so as to account for the estimated uncertainties in the modeling, other information, such as constraint normals and therefore task frame axes, may be out of alignment with the actual environment. This, coupled with sensing and control errors during execution, may cause execution failures. Consequently, a robust controller and integrated real-time sensing capability is needed to handle contact interactions with imperfectly known environment. The slave execution process must proceed as a high-bandwidth local feedback loop with sensory input participating in the real-time control decisions. Among the sensors that can be used at the remote site are CCD cameras, laser

range finders, sonar scanners, force sensors, etc. At minimum, the slave manipulator needs to be equipped with a sensor of external forces acting on the manipulator's end-effector. This is a central requirement of the hybrid position/force control strategy which relies on the manipulator's ability to realize arbitrary force trajectories in a Cartesian contact-based task frame. Additionally, a small amount of end-effector passive compliance may dramatically reduce the problem of control instabilities on contact with the environment [Xu&Paul,1989].

We propose to use the Cartesian hybrid control algorithm, illustrated in Figure 8.6 [Fisher,1991]. The inputs to the controller are the desired position ($\mathbf{x}_d$) and force ($\mathbf{f}_d$)

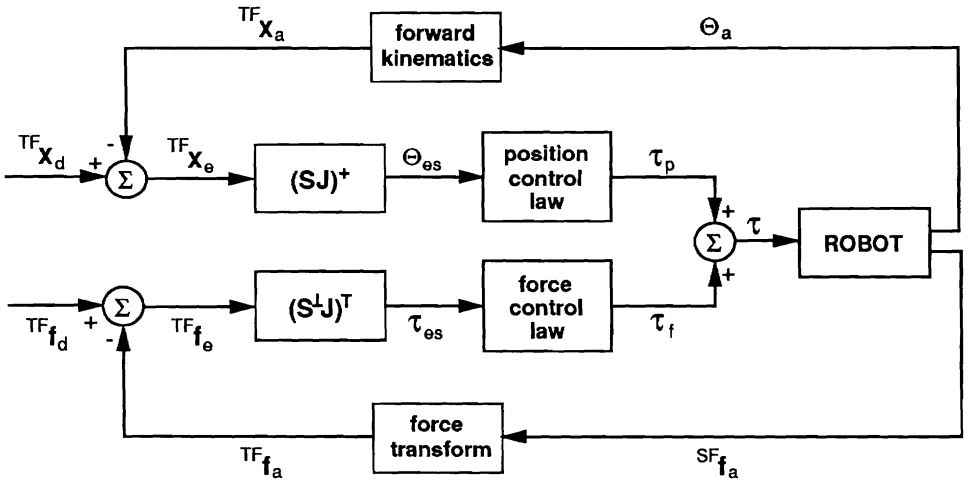


Figure 8.6: Slave hybrid force/position controller.

trajectories of the slave manipulator, along with the current actual position ($\mathbf{x}_a$) and sensed external forces ($\mathbf{f}_a$). All Cartesian input quantities relate to and are expressed in the current task frame (TF). The vectors ${}^{TF}\mathbf{x}_e$ and ${}^{TF}\mathbf{f}_e$ denote the six-vectors of positional and force errors in TF. S denotes the *selection matrix* (functionally analogous to the *mode vector* of Chapter 7), and $\mathbf{S}^{\perp}$ denotes its orthogonal complement. The pseudo-inverse of the selected Jacobian matrix $(\mathbf{SJ})^{+}$ is then used to map the subspace of the selected Cartesian errors into the corresponding joint displacement errors. Similarly, the transpose of the selected Jacobian matrix $(\mathbf{S}^{\perp}\mathbf{J})^{T}$ maps the Cartesian force errors into the corresponding joint torque errors.[2] The position and force control laws then produce control torques, whose sum is fed to the robot. Note also that the feedback quantities must be appropriately transformed into the task frame (the actual joint displacements $\boldsymbol{\theta}_a$ are mapped through direct kinematics into the corresponding Cartesian task frame displacements, and the external forces, measured

[2]See [Fisher,1991] for a detailed description and derivation of these mappings.

in the sensor frame ${}^{SF}\mathbf{f}_a$, are mapped into the equivalent force vector in the task frame).

8.4 Error Handling and Recovery

As we saw in Chapter 7, the low-level contact motions consist of guarded and compliant moves with built-in estimated modeling errors. This, along with the control algorithm of Section 8.3 should provide for stable and reliable execution of the commanded motions at the remote site. However, things still may (and will!) go wrong. Some of the common errors encountered during execution are not reaching an expected motion terminating condition (force, distance), hitting an obstacle in the workspace, stopping prematurely by mistaking friction forces for guard conditions, jamming, etc. The slave should be able to detect most of these error conditions by monitoring its position, velocity, force at the end-effector, and motor torques. Information from the external sensors, such as vision cameras, can be used (if available) to confirm an error condition and aid the system in gathering relevant information about the error state.

Upon detecting an error, the slave must respond in a manner that minimizes the possibility of damage to itself, as well as to environmental objects. If relatively small and static unexpected forces are encountered, the slave controller may choose to stop and maintain the current position until the operator can resolve the situation. Alternatively, the slave may need to comply with large time-varying forces to avoid damage to the arm. Low-level default error handlers should be in place to stop the manipulator when significant forces are encountered along a position controlled direction, and designate the corresponding task frame axis as force controlled until the condition is relayed to the operator and resolved. Likewise, a sudden acceleration (or velocity) along a force controlled direction should stop the motion and place the corresponding axis in position mode, as this situation probably corresponds to loss of supporting surface (i.e., falling).

Upon detecting an error condition, it is critical that the slave be able to gather as much relevant information about the error state as possible, and relay this information to the operator's station. Because of the transmission and other delays between the operator's station and the remote site, the operator learns about an error condition at the slave site $\eta+\tau$ seconds later. During this time, the operator had continued with the task and possibly modified the simulated environment. The error packet arriving from the slave must therefore contain sufficient information to restore the simulated environment (and the display) to the error state and present to the operator the critical remote site sensory data. Moreover,

as discussed in Section 3.8, the error information can also provide local corrections to the operator's station based world model, by giving more accurate information about the location of various environmental features. The error reporting and resolving mechanism can therefore also be used to facilitate on-line refinement of the world model.

If the initial error state information, as received by the operator's station, should not suffice for the operator to understand the nature of the problem at the remote site, she should be able to initiate exploratory actions at the slave workcell (e.g., request additional camera views of the contact area) and gather additional information. Once the operator has determined the cause of the error, she can specify corrective actions to recover from the error and continue with the task.

The critical feature of this approach to error recovery is that the operator is asked to resolve the error condition. This capitalizes on the fact that people are much better at quickly grasping the nature of an arbitrary error state and planning appropriate corrective actions than any automatic state-of-the-art reasoning system. Most importantly, this approach to remote manipulation and error recovery eliminates the need for off-line pre-programming of error handlers for all possible error situations, which is a hopeless undertaking in any realistic application (Chapter 1).

Chapter 9

Experimental Results

9.1 The Experiment

In order to test the *teleprogramming* control methodology, we designed and implemented an experimental *teleprogramming* system, called MERIONETTE. The hardware and software structure of MERIONETTE, as well as the origin of its name, are described in Appendix C.

We have chosen a relatively simple task on which to asses the feasibility of the *teleprogramming* control methodology as well as the performance of our experimental system. The task was to explore the inside of an open box using a near-cubic end-effector (a Kleenex box), which was attached at the end of the slave's compliant wrist assembly (see Appendix C). Figure 9.1 illustrates the task environment. The box exploration task was chosen because of its interactive nature — while performing the task, the operator spends much of the time in contact with the environment (the box), sliding along surfaces, reaching corners, etc. This allows extensive testing of many of the critical features of the *teleprogramming* control paradigm: the kinesthetic interaction between the operator and the simulated environment, on-line low-level symbolic command generation, remote site command translation and hybrid control execution, as well as the preliminary version of error detection and recovery.

The box and end-effector probe dimensions were 41 x 36 x 11 cm and 12 x 12 x 13 cm, respectively. A transmission delay of $\tau = 3$ seconds was artificially introduced into the system, and t_{max} was set at 1 second for a total lag time of $\eta = \tau + 2\,t_{max} = 5$ seconds (see Section 8.2).

A typical experimental run consisted of the operator starting from free space, bringing the end-effector probe into contact with the bottom surface of the box, sliding towards a side of the box, following the edge into a corner, sliding back out of the corner, etc. The

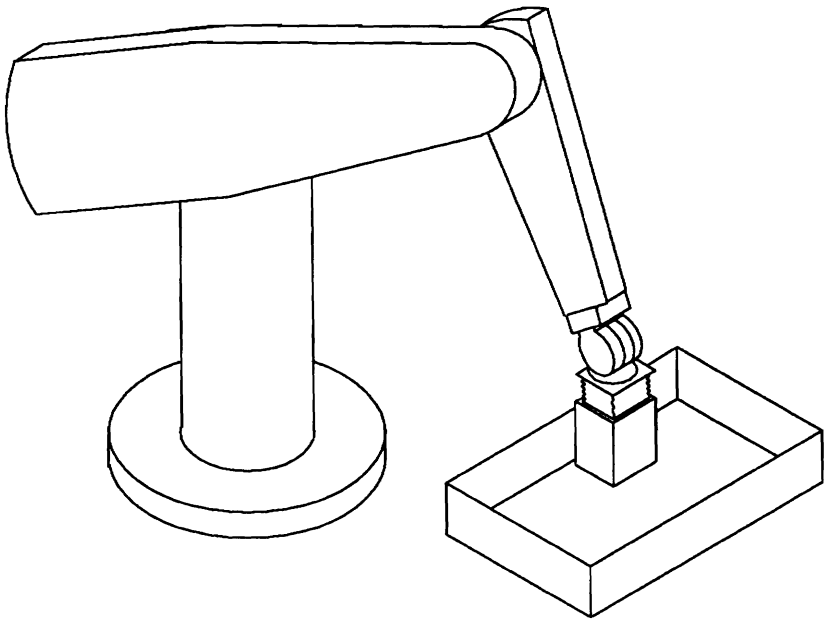


Figure 9.1: The experimental task environment.

critical parameters being observed were:

1. the convenience and naturalness of the operator's interaction with the graphical simulator of the remote environment,

2. the correctness and adequacy of the automatically extracted symbolic instructions describing the operator's actions,

3. the stability and reliability of remote site execution of contact motions,

4. feasibility and effectiveness of on-line error recovery, and

5. overall efficiency of performing the task, despite the substantial transmission delay

The following section presents the main results of a series of preliminary qualitative evaluations of the *teleprogramming* methodology using our experimental system.

9.2 Results

We will in the following paragraphs sequentially address the performance criteria listed in Section 9.1.

9.2.1 The Operator's Station

The operator's station, and in particular the graphical simulation of the remote environment, motion restriction, and generation of the corresponding kinesthetic feedback to the operator, are the most exhaustively tested components of our experimental system. Tests using the operator's station alone, as well as tests involving the entire system, have shown that the combination of three-dimensional computer graphics and real-time kinesthetic feedback allows the operator to interact with the virtual remote environment in a very natural manner. The operator is able to close her eyes and determine the size and orientation of the box with confidence, using only kinesthetic interaction with the simulated environment. Together with real-time visual feedback, the system offers the operator a strong sense of teleperception.

The tests have also confirmed the importance of an audio information channel between the system and the human operator. Even a simple one-way audio channel, used in MERIONETTE (i.e., playback of prerecorded digitized messages), proved to be extremely useful. This is due to the fact that the operator's visual capacity is already committed to the visual interaction with the graphical simulation, and so presenting status and other additional run-time information visually, can easily overload the operator's visual capacity and increase operator fatigue. Audio communication can be used to take advantage of a largely unexploited human input channel – hearing. Likewise, it is easy to imagine the added convenience and power of utilizing human operator's audio output (i.e., speech) as input to the *teleprogramming* system. Future refinements to the system should consider this option.

Other lessons that we learned in experimenting with our particular implementation of the *teleprogramming* system include the importance of a sufficiently high video update rate of the graphical display, as well as the importance of a high-fidelity, singularity-free master input device. Since these and similar conclusions and results pertain more to our particular experimental set-up than to the evaluation of the general concept of *teleprogramming*, we will defer their treatment to Appendix C.

9.2.2 The Low-level Symbolic Language

The box exploration task exhaustively exercises the free-space and sliding contact motion instruction generation. A new execution environments is generated whenever the state of the environment changes "significantly" (e.g., the contact multiplicity or the nature of an existing contact between MO and the environment has changed) and at least once every t_{max} seconds. The experimental system has demonstrated that the process of extracting

symbolic descriptions of low-level slave–environment interactions in the virtual environment can in fact be automated. Moreover, the corresponding instructions can be generated on line, in real time, and have shown to posess sufficient expressive power to describe the low-level activity in the simulated environment in sufficient detail to allow accurate reproduction of the operator's actions at the remote site. Also, owing to the simplicity of the symbolic language, the remote site parser/translator module is straightforward and requires minimal computational resources.

9.2.3 Remote Site Execution

The close match between the nature of the symbolic instructions as generated by the operator's station and the slave workcell's hybrid control execution paradigm allows for reliable execution of the commanded guarded and compliant motions at the remote site. Particularly crucial in this context is the presence of information as to the direction and approximate (relative) magnitude of the dynamic parameters, necessary for safe and reliable execution of the commanded contact motions. These parameters allow the slave workcell to take full advantage of the hybrid control execution paradigm. The Cartesian task frame axes are appropriately partitioned into position and force controlled directions and elementary motion execution proceeds as a high-bandwidth local feedback process, guided by real-time on-board sensory information (e.g., contact force/torque information).

The double-buffering execution management scheme of Section 8.2 successfully ensures that the lag time between the master and the slave manipulators remains constant throughout the execution of a task. Contributing to the importance of this issue is the negative psychological effect of the increasing lag time on the operator, who naturally expects the lag to be constant. This can manifest itself in the frustration on the part of the operator if the setback in the progression of the task, when an error is detected at the remote site, is significantly larger, or even unpredictably different, than expected.

9.2.4 Error Detection and Recovery

Preliminary experiments have shown that reasonably reliable execution behavior can be achieved using a simple Cartesian hybrid controller at the slave workcell. As expected, execution errors relate primarily to the unmodelled static and dynamic effects of real world contact interactions. In particular, friction forces are occasionally mistaken for terminating conditions in guarded moves, or the estimated compliance forces are insufficient to maintain contact during a compliant motion. The slave controller has shown good ability to detect

various error conditions by monitoring its position, velocity, external forces, and its own actuator torques. Experiments have shown that even if an error is not detected immediately, the likelihood of the resulting discrepancy going unnoticed through the remainder of the task is effectively null. Upon detecting an error, a preliminary error recovery mechanism, employed by MERIONETTE, stops the slave motion, alerts the operator to the condition, and updates the operator's graphical model to reflect the error configuration, based on the data supplied by the remote workcell. The operator is then informed (via the audio interface) to resume the task from the error configuration.

Given the simplicity of our example task environment, the current error detection and recovery mechanism is relatively straightforward. Future research will need to address the problem of error recovery in a more general framework, where the slave workcell would be able to gather more detailed information about its error configuration through local exploratory procedures. Likewise, issues related to on-line model refinement should also be investigated in order to make the overall system more flexible and dynamically adaptive.

9.2.5 Overall Performance of the System

In summary, whereas much more work remains to be done in refining our prototype *teleprogramming* system, experimental results have decisively confirmed the validity and effectiveness of the *teleprogramming* control paradigm for remote manipulation in the presence of substantial feedback delays. Remote site autonomy at the level of $nt = 100$ seconds was successfully achieved and exceeded with the box exploration task, verifying the feasibility of near-optimal *teleprogramming* remote control as postulated in Section 1.2. In more challenging applications, the degree of attainable remote site autonomy will dictate the overall efficiency of task execution under *teleprogramming* control.

Chapter 10

Conclusion and Future Work

10.1 Conclusion

The *teleprogramming* concept described in this document provides the necessary bridge between conventional teleoperation (which cannot function in the presence of communication delays) and fully autonomous manipulative capability (which is not yet feasible). *Teleprogramming* a remote manipulator essentially corresponds to *visually* and *kinesthetically* interacting with a virtual world (a graphical simulation of the remote environment), and on-line, automatically generating a sequence of symbolic instructions to the remote robotic system, based on the operator's interactions with the virtual environment. Coupled with a small degree of autonomy at the remote site, this system is able to provide for continuous and efficient remote control of a robotic system, with the flow of control being interrupted only when errors occur at the remote site. The applicability and effectiveness of the *teleprogramming* control paradigm (in terms of the amount of allowable feedback delay) is limited only by the level of autonomy that a robotic workcell can provide and by the amount of time and work that the operator can tolerate losing when an error is reported and she is placed back to the point in the task execution where the error occurred. The latter poses a constraint on the maximum allowable lag time between the master and the slave, and thus the maximum length of the feedback delays that the system can gracefully tolerate.

Experimental results with our prototype *teleprogramming* platform have confirmed the feasibility and effectiveness of the *teleprogramming* concept and provided strong encouragement for future refinements and extensions to the existing methodology. Although the preliminary experiments have demonstrated only very basic functionality, it is important

to note that much of the manipulative capability required in space and underwater applications (e.g., satellite module replacement, locating an eye-bolt and inserting a hook, etc.) relates to the ability to kinesthetically explore the surroundings and locate various features, using sequences of primitive contact motions of the type demonstrated in our system. Future work will be aimed at increasing the dexterity and versatility of the current system to allow for broader applicability in various situations and environments.

As we have mentioned before, we see primary applications of this technology in underwater and shallow space environments, where communication delays preclude direct remote control of robotic systems. However, a *teleprogramming* system can be employed in non-delayed situations as well. In particular, the operator's station portion of a *teleprogramming* system could be used as a stand-alone facility for automatic generation of robot programs, based on operator's task specification in a model-based virtual environment. Industrial robots could thus be reprogrammed quickly by untrained operators, who could simply show the system the desired new task in the simulated environment. Good models of the working environment are generally available in industrial applications.

Application of the *teleprogramming* technology to undersea manipulation would free us from the need to maintain a wide bandwidth communications link (tether) between the operator and the vehicle. While it is within the state-of-the-art to eliminate the tether based on energy considerations [Niksa,1987], it is still impossible to eliminate the tether based on manipulation control considerations. This is due to the fact that acoustic communication links provide insufficient bandwidth to support existing remote manipulator control strategies. Using the *teleprogramming* approach, it would be possible to deploy a submersible from a plane together with an acoustic relay buoy and to then control operations at the ocean bottom remotely over a radio link from a shore-based station. 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 an important consideration.

In shallow space, we see applications of *teleprogramming* in performing a variety of routine exploratory, maintenance, or even construction tasks. Cost justifications in this domain relate to the possibility of eliminating the need for astronauts in performing "extra-vehicular activities" (EVA), or even the prospect of eliminating human crews altogether. In the latter scenario, the entire mission, together with on-board experiments and routine

vehicle maintenance, would be controlled remotely from a ground-based control center, vastly reducing both the cost and risk involved in manned missions.

10.2 Contribution

We see the major contributions of this work in the following areas:

- The overall design of a delay-tolerant control methodology for remote manipulation, which requires a relatively modest amount of remote site autonomy and offers the possibility of near-optimal task performance in the presence of substantial communication delays between the master and slave sites.
- The design and successful experimental verification of an effective approach for providing the operator with real-time kinesthetic feedback despite the communication delays. This information is derived by analyzing the operator's interaction with the simulated environment and has proved to provide a good approximation to actual force reflection.
- The design of a symbolic low-level command language, based on the hybrid position/force control model, which can adequately describe the operator's interaction with the virtual environment to allow accurate reproduction of the operator's activity at the remote site.
- The design and successful demonstration of on-line analysis of operator's activity in the simulated environment and real-time extraction of the corresponding sequences of symbolic robot instructions.
- The design of a remote site execution strategy, relating to parsing, scheduling, and execution management of the incoming instructions, which guarantee a constant time lag between the master and the slave systems.
- Successful experimental verification that the necessary level of remote site autonomy can be achieved using a relatively simple remote site controller.

10.3 Future Work

The *teleprogramming* concept, as described in this document, can be considered as the basis, upon which more general control methodologies can be developed. We will in the

following paragraphs outline some of the immediate as well as some of the more far-reaching extensions to the basic *teleprogramming* control paradigm.

1. **world modeling:** The issue of constructing the initial model of the remote environment has not been addressed in this work. However, as indicated in Section 3.2, the necessary technology, needed to facilitate interactive, off-line construction of the initial environment model, exists. What is needed is a relatively low-complexity, practical approach to integrating existing results and algorithms in image processing and data fusion, along with a convenient mechanism to allow operator's participation in the high-level segmentation process. The operator would participate in this process in a supervisory role, resolving ambiguities when necessary, and ensuring that the extracted model is consistent with the operator's mental model of the remote environment.

2. **Special-purpose commands and procedures:** In Section 7.1 we motivated the need for special-purpose commands in the context of actions, which are more conveniently executed as local high-bandwidth feedback processes at the remote site. As discussed in Section 7.1, examples of such actions are fine precision object alignment, grasping and handling of fragile or deformable objects, and high-dexterity dynamically reactive manipulation tasks. In order to support this level of task specification, a more general framework for interpreting operator's activity in the virtual environment must be designed, which in turn will require a more sophisticated *a priori* knowledge about the task in progress. The corresponding capability to execute such actions under local sensory supervision must exist at the remote site.

   A related enhancement to the system would be the provision of an on-line procedural facility, where the operator would be able to specify a general pattern (cycle) of an iterative subtask, such as sawing, polishing, hammering, etc., and have the system perform the action repeatedly until some terminating condition is reached. Again, the task model would need to contain the necessary information to appropriately terminate the subtask execution.

3. **error recovery:** Error recovery has been addressed only briefly in this work. MERIONETTE uses a simple mechanism, where, upon detecting an error, the slave controller reports the kinematic configuration of the slave manipulator and the manipulated object to the operator's station. There the state of the graphical simulation is updated to reflect the error state. However, because of the discrepancies between the

model and the actual environment, purely kinematic error status information does not suffice to unambiguously reconstruct the state of the remote workcell. At minimum, the remote slave should be able to identify its current contact state (i.e., number and types of currently active contacts). This could be done by employing special-purpose exploratory procedures, which would perturb the slave system in a controlled manner and use the measured response data to construct a model of the error state, which in turn would be relayed to the operator's station for analysis.

A more sophisticated error recovery mechanism may attempt to monitor the operator's corrective actions and resume autonomous execution as soon as it recognizes a state, which belongs to the original task specification. The operator could then be brought back to the point in the task specification, where she was interrupted to attend to the error condition. This would significantly improve the overall system efficiency, as well as operator satisfaction.

4. **on-line model refinement:** Taking the above ideas a step further, we may want to take advantage of the interruption during error recovery and try to refine the operator's station resident model of the remote environment, based on the information supplied by the remote workcell as part of the error state information packet. As the slave encounter various environmental features during task execution, it can record their actual position and orientation and relay this data to the operator's station along with error information. The operator's station software can then use this (possibly sparse) local corrective information to refine the geometric relationships in the model. A key issue in this process is ensuring consistent propagation of local corrections throughout the model.

   Clearly, the on-line refinement mechanism can proceed independently of error recovery as well.

5. **hybrid control:** An important issue that has not been addressed in this work is the stability of the hybrid controller of Section 8.3 in situations where the task frame is significantly removed from the manipulator's wrist, where rotations and translations are normally kinematically decoupled. This separation gives rise to a remote center of compliance, which in turn causes the control of the positional and orientational linkages of the manipulator's kinematic structure to become strongly coupled. Theoretical investigations [Zhang,1986] have shown that, in general, stable control of a manipulator system can be guaranteed only if the center of compliance is coincident

with the point, where translations and rotations are kinematically decoupled (e.g., intersection of the three axes of rotation in a spherical wrist). An interesting direction of research may be to investigate whether alternative strategies for control of contact manipulation can be formulated, such that the compliance center remains fixed at the wrist center, regardless of the contact location and geometry.

6. **virtual editor:** A broader extension of the *teleprogramming* control paradigm may generalize the idea into a model-based virtual editor, in which an operator could control multiple robotic and other devices simultaneously through direct kinesthetic and visual coupling. In such a system, the operator would no longer be constrained by the execution rates of the actual workcells or vehicles (agents) in the remote environment. Instead, she could interleave task specifications for different agents in the virtual editor, with the execution at the remote site proceeding at a slower, agent-dependent rate. In case of an execution error at a specific agent, the operator could (according to a priority scheme) attend to this agent, resolve the problem according to the error recovery scheme in item 3, and resume task specification for other agents. Similarly, having specified the desired actions for a set of agents, the operator may stand by idle while execution at the remote site unfolds. This type of control relieves the operator of continuous interaction with the operator's station and provides the basis for more general forms of supervisory control of robotic devices.

Appendix A

Notation and Coordinate Transformations

A.1 Notation

Both three and six-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.,

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

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

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

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

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

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

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

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.,

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

Likewise,

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

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

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

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 ${}^A\mathbf{r}$

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

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

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

- extracting the new triple of RPY parameters

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

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

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

where $\mathbf{R} = \text{RPYtoM}(\mathbf{r})$.

Appendix B

The Symbolic Command Language

This appendix summarizes the main constructs of the low-level command language interface between the operator's station and the remote workcell. Brief explanations of the corresponding semantics have been included, where deemed necessary.

B.1 Task Frame Management

DefineVector (*name*; $< v_x, v_y, v_z >$: *ref-frame*)

Give a character string label (*name*) to the numeric vector **v**. Only names are used in subsequent vector references, e.g., for task frame construction. Consequently, all vectors must be labeled before they are used. Note that the string label of the reference frame (*ref-frame*), with respect to which the vector's numeric value is given, must also be specified. This frame must have been defined previously. Two predefined vector are recognized — the origin of the kinematic base frame (ORG) and the origin of the end-effector frame (WST).

DefineTaskFrame (*name* : *ref-frame* ; *origin* ; *x-axis* ; *y-axis* ; *z-axis*)

This instruction assembles a right-handed coordinate frame (task frame) from the supplied axes and gives it a symbolic name *name*. Of the four component vectors, only the origin and two of the three coordinate axes need be specified. The axis vector, which is not explicitly specified (if any), is denoted by ?. The task frame reference frame (*ref-frame*) is either the kinematic base frame KB (in which case the task frame remains fixed with respect to the world and is said

to be *static*) or the slave end-effector frame EE (in which case the task frame moves along with the slave robot and is said to be *dynamic*). The coordinate frames KB and EE are predefined.

UseFrame (*frame*)

Use coordinate frame *frame* from now on until overriden. All subsequent coordinate-frame dependent parameters are interpreted with respect to *frame*.

B.2 Force Control Commands

AssignMode (X, X, X, X, X, X) , X ∈ { P, F }

Specify force (X=F) and position (X=P) controlled directions (along task frame axes). A force-controlled direction is assumed to require 0 force compliance (default), unless otherwise specified by a subsequent **Force**() statement. In effect until overriden.

Force ($< f_x, f_y, f_z >$; $< \tau_x, \tau_y, \tau_z >$)

Specifies a force preload $(\mathbf{f}, \boldsymbol{\tau})$ along task frame axes. If a preload is specified on a force-controlled axis, it is interpreted as a *compliance force*. If a preload is specified on a position-controlled direction, it is interpreted as a *preload force* for operations like pushing, screw or valve tightening, etc. In effect until overriden.

GuardForce ($< f_x, f_y, f_z >$; $< \tau_x, \tau_y, \tau_z >$)
GuardVelocity ($< v_x, v_y, v_z >$; $< \omega_x, \omega_y, \omega_z >$)

These instructions specify terminating conditions for the subsequent motion. The argument six-vectors to the above instructions are interpreted in accordance with the current **AssignMode** parameters (hybrid control modes). Therefore, force guards only make sense along position controlled directions and position (and velocity) guards are only relevant along the force-controlled directions. Note that these guards are task dependent and are different from low-level safety force guards (i.e., actuator torque limits), which must be active at all times (task independent). In effect until overriden.

B.3 Motion Commands

All motion commands are subject to velocity constraints imposed on the motion by the time parameter t. If $t = 0$, then the system is expected to determine the best timing/velocity parameters for the motion.

Move (t ; $< p_x, p_y, p_z >$; $< \phi_x, \phi_y, \phi_z >$)

Free-space motions. The argument t is the allowed motion time (in seconds). **p** and $\boldsymbol{\phi}$ give the *incremental* translation and rotation vectors in the current task frame.

Slide (t ; $< \phi_x, \phi_y, \phi_z >$)

Contact sliding — we must be in contact, at least one axis should be force controlled, and a compliance force should be given along that axis. t specifies the allowed motion time (unless $t = 0$), whereas **p** gives the numeric (translational) motion parameters along the sliding surface.

Pivot (t ; $< \phi_x, \phi_y, \phi_z >$)

Perform a pivoting (rotational) motion about the contact point (i.e., task frame origin). The duration of the motion is given by t and the corresponding rotational parameters are given as a roll/pitch/yaw vector $\boldsymbol{\phi}$.

Appendix C

The Experimental System

This appendix contains the description of the experimental *teleprogramming* system, which was designed and built at the GRASP laboratory to test the *teleprogramming* concept. The system was christened MERIONETTE, for Model-based EditoR for Interactive ONlinE Teleprogramming in Time-delayed Environments. Figure C.1 shows an actual view of the operator's station (top) and the remote slave (bottom) in MERIONETTE. At the operator's station, the display on the right shows the real-time graphical simulation of the remote environment, whereas the display on the left provides the delayed actual video information from the remote site. The master arm is shown to the right of the graphical display in the top picture. A close-up view of the slave robot, its compliant wrist sensor, and the example environment (Chapter 9) is shown in the bottom picture.

C.1 Hardware

A schematic diagram of the experimental system's hardware is illustrated in Figure C.2. The master device in our set-up is a Unimation PUMA 250 manipulator. This "joystick" provides 6 d.o.f. of motion within a sufficiently large workspace envelope to give the operator a good sense of spatial maneuvering. Hardware control for the master is provided by the PC-bus based 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 Unimation controller, whose only remaining function is to provide power and the front panel interface. The MMCS chassis is connected to the VME bus via a custom-designed PC/VME adaptor. Mounted at the wrist of the master is a

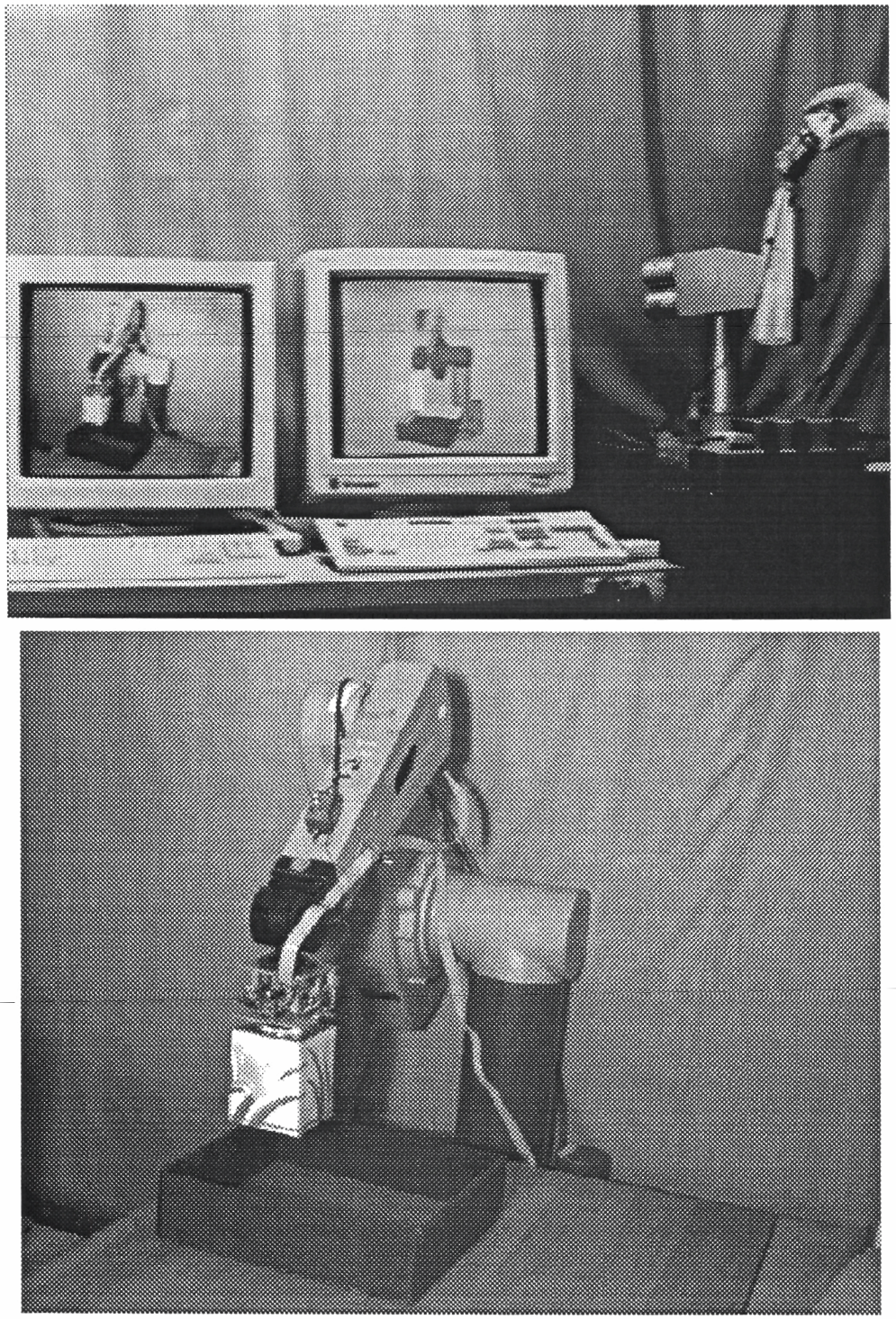


Figure C.1: MERIONETTE: operator's station (top) and remote site (bottom).

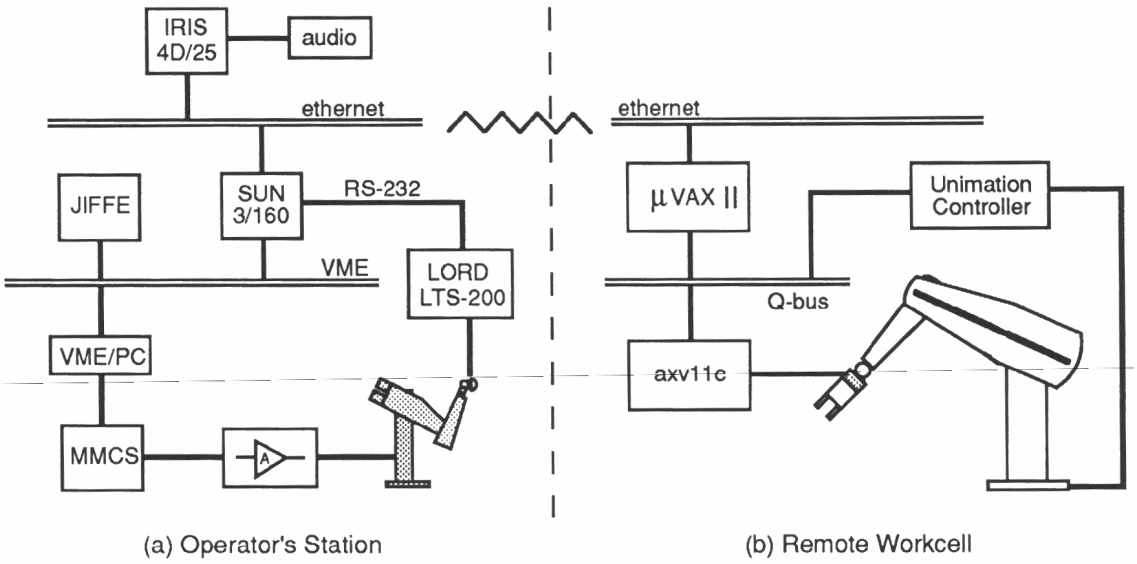


Figure C.2: The experimental *teleprogramming* testbed.

6-axis force/torque sensor (LORD Corp., LTS-200) enclosed within a "whiffle-ball" handle assembly for convenient grasping by the operator. The sensor is read over a serial line and provides information at a rate of 30 Hz.

The computational engine of the operator's station subsystem is JIFFE — a fast, 20 Mflop, VME-based scalar floating point co-processor [Andersson,1989]. The processor has a standard VME interface and physically resides inside the Sun cage (see Figure C.2). 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 (500 Hz), as well as the Cartesian level servo code (30 Hz). It communicates with the host (Sun 3/160) via a JIFFE-resident shared memory segment.

MERIONETTE's graphical workstation is a 16 MIPS Personal Iris 4D-25, equipped with a hardware turbo graphics option to increase its rendering speed. Interfaced to the Iris is an audio speaker, which allows us to use the Irix `/dev/audio` facility for playback of prerecorded, digitized audio messages. Communication between JIFFE and the Iris is accomplished indirectly via the shared memory interface between JIFFE and the Sun and a bidirectional UNIX TCP socket connection between the Sun and the Iris. The round-trip communication latency is on the order of a few milliseconds.

The remote manipulator in our experimental system is a PUMA 560. The low-level joint servo control is accomplished via the standard Unimation controller ($\approx$ 1000 Hz),

while a Microvax II provides for Cartesian control of the manipulator ($\approx$ 35 Hz). The manipulator is programmed using a Cartesian hybrid control algorithm built on top of the RCI programming environment [Hayward,1983], [Lloyd,1985]. The communication between the operator's station and the slave manipulator is again facilitated by a bidirectional UNIX TCP socket link.

A 6 d.o.f. instrumented compliant wrist, mounted at the slave's tip, is used as the remote force sensing device [Xu&Paul,1989], [Lindsay&Paul,1991]. The passive compliance of the wrist and the active compliance of the hybrid control algorithm allow the manipulator to move stably in reliably in contact with the environment.

C.2 Software

The software architecture of MERIONETTE is illustrated in Figure C.3. As is evident from the figure, the software is distributed over four computers and consists of eight main interdependent and intercommunicating processes. We will in the following paragraphs briefly outline the organization and functionality of the major software modules.

At the lowest level in the software hierarchy, JIFFE executes a single real-time process, which controls the master arm at both the joint servo and Cartesian levels. The joint servo loop is a position PD loop with gravity feed-forward and executes at 500 Hz. The Cartesian control bandwidth is limited by the bandwidth of the force/torque sensor, which provides incremental Cartesian directional input to the master arm. The Cartesian servo loop therefore proceeds at 30 Hz. It performs motion restriction of the commanded master displacements $\mathbf{D}_m$ with respect to the current contact set $\mathcal{C}$ (see Chapter 6) and issues the resulting restricted Cartesian displacements $\mathbf{D}'_m$ to the master arm. This enforcement of the current motion constraints thus provides the operator with a sense of real-time kinesthetic feedback.

The desired incremental master displacements are computed by the Sun and passed to JIFFE through a shared memory segment. A dedicated process running on the Sun reads the force/torque sensor as fast as the sensor can supply information (30 Hz) and computes the resulting Cartesian master displacement $\mathbf{D}_m$ according to the algorithm discussed in Chapter 4. A second process on the Sun in the current implementation of the system serves essentially as the communication bridge between the graphical workstation (the Iris) and JIFFE. Its main role is to perform the necessary scaling between the master and the slave workspaces and manage the TCP socket connection with the Iris. It also takes care of the

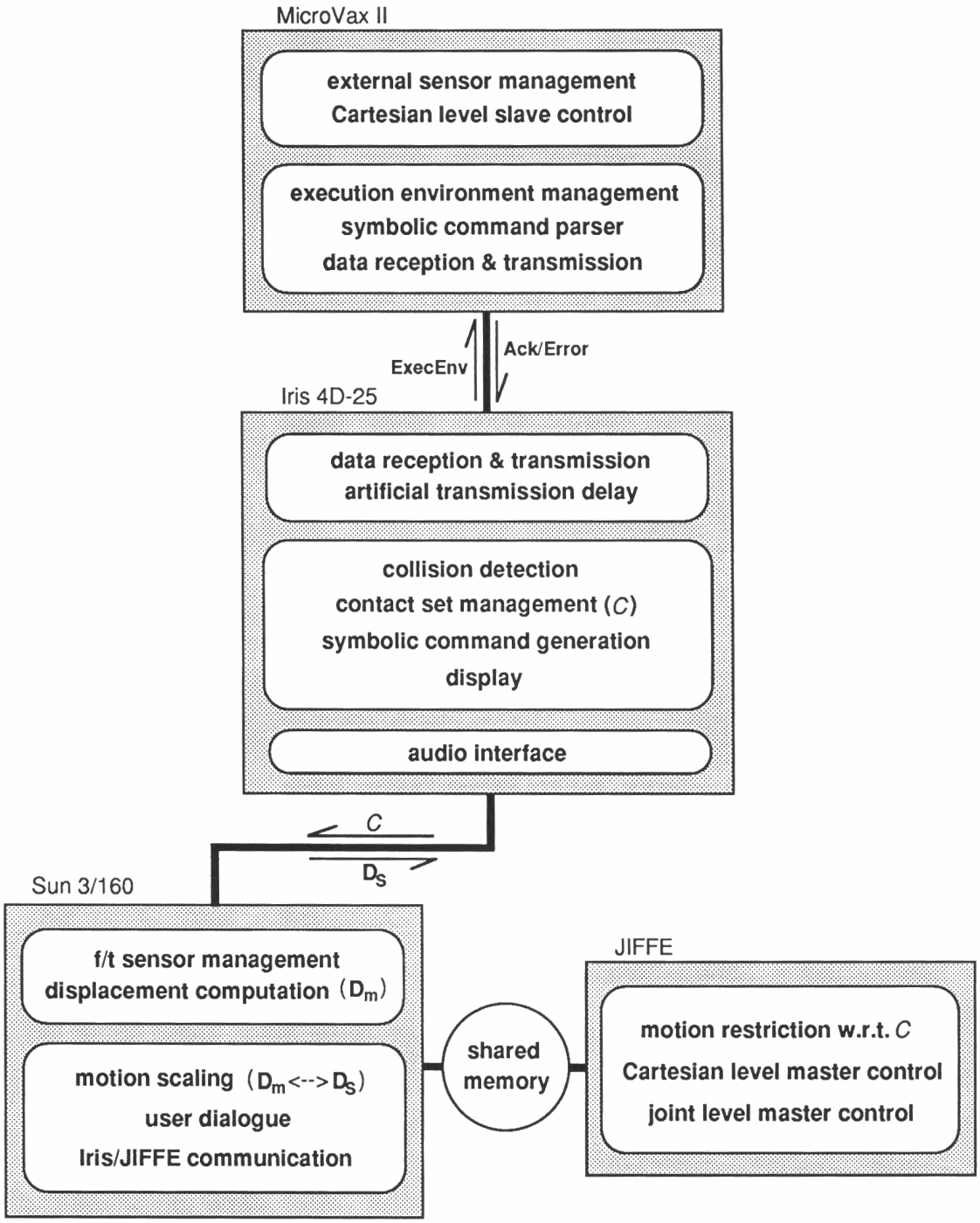


Figure C.3: The software organization of MERIONETTE.

data transfer between the socket buffer and the JIFFE-resident shared memory buffer (i.e., it passes the latest contact set $\mathcal{C}$ to JIFFE and sends the new incremental motion request $\mathbf{D}_s$ to the slave). In later stages of the system design and implementation, the Sun's role will be expanded to include the management of an on-line task-level dialogue with the operator, as discussed in Section 3.6.

The Iris workstation maintains three concurrent, cooperating processes. The simulation process represents the heart of activity at the Iris workstation. This process analyzes the commanded incremental Cartesian displacement $\mathbf{D}_s$ (supplied by the Sun) and checks the resulting motion for possible collisions with the environment. In case of collision(s), it performs the proper motion restriction and contact management as described in Chapter 5. It also ensures that the resulting motion is within the reachable workspace of the slave and away from any kinematic singularities. The output of this stage of processing is the contact set $\mathcal{C}$, which in turn is relayed to the Sun and indirectly to JIFFE, where it is used for constraint enforcement on the master arm. 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]. This basic platform was extended by incorporating a near-linear polyhedral collision detection module [Gilbert&Johnson,1987] and all application-specific software, including the support for accepting animation input from an external source.

The tail end of each simulation step issues a call to the command generation module, which analyzes the current state of the simulated environment according to the algorithms presented in Chapter 7 and generates an instruction sequence (execution environment), if appropriate. The resulting execution environment is passed to the communication module, which artificially delays the data by the communication delay τ and sends it to the remote workcell after the expiration of the delay period. This mechanism is implemented using a shared memory circular queue and a global software timer (resolution ± 10 ms). The simulation process enqueues newly generated execution environments, along with a time stamp indicating the time when they should be sent out, and the communication process ensures that they are in fact sent on their way at the appropriate times.

Finally, the third process executing on the Iris is a background audio process, which accepts requests for audio messages from other processes (through a combination of shared memory and the UNIX signal facility) and issues appropriate calls to the Irix `/dev/audio` interface, which in turn reproduces the requested digitized audio messages through the attached speaker.

The computational platform of the remote workcell is a MicroVax II. The MicroVax maintains two processes, a non-real-time communication and command translation process and a real-time Cartesian control process. The communication process accepts the delayed execution environments and parses them using a simple **lex/yacc**-based parser. The output of the parsing/translation stage is the updated symbol table of defined vectors and coordinate frames in the task environment, and the corresponding command buffer, containing directly executable hybrid control instructions for the slave real-time controller (Chapter 8). The real-time control process implements a modified version of the standard hybrid control algorithm [Raibert&Craig,1981], which is designed on top of the RCI programming environment. The controller integrates the external sensory force data (supplied by the compliant wrist sensor) with the requested motion commands and issues joint displacement requests to the slave manipulator at the rate of 35 Hz.

C.3 Caveats

The experimental results obtained with this experimental testbed are described in Chapter 9. The following paragraphs will outline some of the more MERIONETTE-specific lessons that we learned, while experimenting with the system. Whereas the system performed well and certainly sufficed as a demonstration of the *teleprogramming* principle, certain (primarily hardware) limitations became apparent during testing. In particular, the main bottlenecks in the existing system are the LORD force/torque sensor bandwidth, the kinematics of the master arm, the rendering speed of the Iris workstation, and the computational power of the MicroVax.

At the operator's station, the bandwidth of the force/torque sensor limits the bandwidth of the Cartesian control of the master device. Because the force signal is fairly noisy and therefore has to be filtered with a relatively small low-pass filter gain, the sampling frequency needs to be high in order to avoid introducing a time lag into the signal (Section 4.3). In light of this we have found that the sampling frequency of 30 Hz is insufficient. Moreover, in the current implementation of MERIONETTE the sensor is read by the Sun through a standard UNIX serial port. Due to the non-real-time nature of UNIX, additional variations are introduced into the sampling frequency, which further degrade the sensor information.

As pointed out in Chapter 4, the kinematics of the master device should be completely transparent to the operator. In particular, the master should be free of kinematic singularities in its workspace. In view of this, a standard industrial manipulator, such as PUMA 250,

is not an optimal choice, due to numerous (primarily orientational) singularities throughout its workspace. The resulting frequent reindexing of the master is distracting to the operator and adversely affects the overall efficiency of task performance. Specially designed hand controllers, offering a convenient interface, a singularity-free workspace, and precisely controllable impedance should be considered instead [Bejczy&Salisbury,1983], [Hatamura *et al.*,1990].

Another valuable lesson, which we learned while experimenting with the system, is the importance of sufficient video quality and bandwidth. MERIONETTE's Personal Iris currently allows us to obtain video refresh rates of about 7 Hz for medium complexity environments (i.e., polygon count on the order of 500) and using only partial shading to speed up the drawing process. This has proved to be distracting both in terms of the insufficient update rate, as well as in terms of the poor sense of realism, due to the absence of full shading and the inability to use proper lighting models. However, state-of-the-art graphics hardware, which can resolve both of the above deficiencies to a high degree of satisfaction, is available [Bejczy *et al.*,1990].

On the slave side, the MicroVax has proved to be too slow to adequately support the computational load imposed on it by the *teleprogramming* control paradigm. Due to the low-level, interrupt driven kernel process, which implements the hybrid control algorithm and runs at high priority, insufficient computational power remains for the parsing/translating process. Consequently, this process has trouble supplying command buffers to the control process sufficiently fast to guarantee nonincreasing lag time during execution, as described in Section 8.2.

Finally, the experimental system should be moved away from custom hardware, such as JIFFE and MMCS, to a standard, commercially supported hardware platform.

Appendix D

Example Symbolic Program

The following is a portion of the program as generated by the experimental *teleprogramming* system of Appendix C for the box exploration task, described in Chapter 9.

```
# ........................................
# >> Execution Environment #0.
# >> moving within contact (0 contacts).
UseFrame(EE)
AssignMode(P,P,P,P,P,P)
Move(0.960;<0.001,0.000,0.355>;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #1.
# >> moving within contact (0 contacts).
Move(0.900;<0.002,0.000,0.497>;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #2.
# >> moving within contact (0 contacts).
Move(0.970;<0.791,1.304,6.590>;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #3.
# >> moving into contact (0 --> 1 contacts).
DefineVector(CP;<0.122,0.000,29.232>:EE)
DefineVector(Z;<0.000,0.000,1.000>:KB)
DefineVector(Y;<1.000,0.000,0.000>:KB)
DefineTaskFrame(TF:EE;CP;?;Y;Z)
```

```
UseFrame(TF)
GuardForce(<0.000,0.000,1.000>;<0.000,0.000,0.000>)
Move(0.840;<0.762,-0.659,-7.353>;<0.004,0.000,0.000>)
AssignMode(P,P,F,F,F,P)
Force(<0.000,0.000,-1.000>;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #4.
# >> moving within contact (1 contacts).
Slide(0.900;<0.049,0.093,0.000>)
# ........................................
# >> Execution Environment #5.
# >> moving within contact (1 contacts).
Slide(0.970;<4.931,-2.201,0.000>)
# ........................................
# >> Execution Environment #6.
# >> moving within contact (1 contacts).
Slide(0.980;<7.554,-2.124,0.000>)
# ........................................
# >> Execution Environment #7.
# >> moving into contact (1 --> 2 contacts).
DefineVector(CP;<0.000,0.000,29.233>:EE)
DefineVector(X;<1.000,0.000,0.000>:KB)
DefineVector(Z;<0.000,0.000,1.000>:KB)
DefineTaskFrame(TF:EE;CP;X;?;Z)
UseFrame(TF)
AssignMode(P,P,F,F,F,F)
Force(<0.000,0.000,-1.000>;<0.000,0.000,0.000>)
GuardForce(<0.000,1.000,0.000>;<-0.057,0.000,0.006>)
Slide(0.490;<-0.820,-5.963,0.000>)
AssignMode(P,F,F,F,F,F)
Force(<0.000,-1.000,-1.000>;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #8.
# >> moving within contact (2 contacts).
```

```
Slide(1.000;<-0.144,0.000,0.000>)
# ........................................
# >> Execution Environment #9.
# >> moving within contact (2 contacts).
Slide(0.980;<-4.825,0.000,0.000>)
# ........................................
# >> Execution Environment #10.
# >> moving into contact (2 --> 3 contacts).
DefineVector(CP;<0.000,0.000,29.232>:EE)
DefineVector(X;<0.000,-1.000,0.000>:KB)
DefineVector(Z;<0.000,0.000,1.000>:KB)
DefineTaskFrame(TF:EE;CP;X;?;Z)
UseFrame(TF)
AssignMode(F,P,F,F,F,F)
Force(<1.000,0.000,-1.000>;<0.000,0.000,0.000>)
GuardForce(<0.000,1.000,0.000>;<-0.068,0.000,0.000>)
Slide(0.670;<0.000,-7.063,0.000>)
AssignMode(F,F,F,F,F,F)
Force(<1.000,-1.000,-1.000>;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #11.
# >> moving within contact (3 contacts).
Slide(0.970;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #12.
# >> moving away from contact (3 --> 2 contacts).
DefineVector(CP;<0.000,0.000,29.233>:EE)
DefineVector(Z;<0.000,0.000,1.000>:KB)
DefineVector(Y;<1.000,0.000,0.000>:KB)
DefineTaskFrame(TF:EE;CP;?;Y;Z)
UseFrame(TF)
AssignMode(P,F,F,F,F,F)
Force(<0.000,-1.000,-1.000>;<0.000,0.000,0.000>)
Slide(0.710;<-0.411,0.000,0.000>)
```

```
# ........................................
# >> Execution Environment #13.
# >> moving within contact (2 contacts).
Slide(0.970;<-8.320,0.000,0.000>)
# ........................................
# >> Execution Environment #14.
# >> moving within contact (2 contacts).
Slide(0.970;<-6.413,0.000,0.000>)
# ........................................
# >> Execution Environment #15.
# >> moving within contact (2 contacts).
Slide(0.960;<-6.228,0.000,0.000>)
# ........................................
# >> Execution Environment #16.
# >> moving into contact (2 --> 3 contacts).
DefineVector(CP;<0.000,0.000,29.233>:EE)
DefineVector(X;<-1.000,0.000,0.000>:KB)
DefineVector(Z;<0.000,0.000,1.000>:KB)
DefineTaskFrame(TF:EE;CP;X;?;Z)
UseFrame(TF)
AssignMode(F,P,F,F,F,F)
Force(<1.000,0.000,-1.000>;<0.000,0.000,0.000>)
GuardForce(<0.000,1.000,0.000>;<-0.057,0.000,-0.029>)
Slide(0.860;<0.000,-8.928,0.000>)
AssignMode(F,F,F,F,F,F)
Force(<1.000,-1.000,-1.000>;<0.000,0.000,0.000>)
# ........................................
# >> Execution Environment #17.
# >> moving within contact (3 contacts).
Slide(0.980;<0.000,0.000,0.000>)
# ........................................
.
.
.
```

Bibliography

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison Wesley, 1986.

[2] C. H. An and J. M. Hollerbach. The role of dynamics in Cartesian force control of manipulators. *The International Journal of Robotics Research*, 8(4):51–72, August 1989.

[3] R. J. Anderson and M. W. Spong. Bilateral control of teleoperators with time delay. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 131–137, 1988.

[4] 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.

[5] H. Asada and H. Izumi. Direct teaching and automatic program generation for the hybrid control of robot manipulators. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1401–1406, 1987.

[6] H. Asada and B. Yang. Skill acquisition from human experts through pattern processing of teaching data. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1302–1307, 1989.

[7] N. Badler and R. Bajcsy. Three-dimensional representations for computer graphics and computer vision. *Computer Graphics*, 12:153–160, 1978.

[8] R. D. Ballard. A last long look at Titanic. *National Geographic*, 170/6, December 1986.

[9] D. J. Barber. *Mantran: A symbolic language for supervisory control of an intelligent remote manipulator.* Technical Report 70283-3, MIT Engineering Projects Lab., 1967.

[10] A. K. Bejczy and M. Handlykken. Experimental results with a six-degree-of-freedom force reflecting hand controller. In *Proceedings of the 17th Annual Conference on Manual Control*, June 1988. Los Angeles.

[11] A. K. Bejczy and W. S. Kim. Predictive displays and shared compliance control for time-delayed telemanipulation. In *IEEE International Workshop on Intelligent Robots and Systems*, July 1990. Ibaraki, Japan.

[12] A. K. Bejczy, W. S. Kim, and S. C. Venema. The Phantom robot: Predictive displays for teleoperation with time delays. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 546–551, 1990.

[13] A. K. Bejczy and J. K. Salisbury. Controling remote manipulators through kinesthetic coupling. *Computers in mechanical engineering*, 1(1):48–60, July 1983.

[14] Antal K. Bejczy. 1990. Personal communication.

[15] Paul J. Besl and Ramesh C. Jain. Three-dimensional object recognition. *ACM Computing Surveys*, 17(1), March 1985.

[16] J. H. Black. *Factorial study of remote manipulation with transmission time delay.* Master's thesis, MIT, Department of Mechanical Enginnering, 1971.

[17] Ruud M. Bolle and Baba C. Vemuri. On three-dimensional surface reconstruction methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(1), Jan 1991.

[18] Thurston L. Brooks. Telerobotic response requirements. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 113–120, 1990.

[19] Forrest T. Buzan. *Control of telemanipulators with time delay: a predictive operator aid with force fedback.* PhD thesis, Massachusetts Institute of Technology, 1989.

[20] Peter I. Corke. *A new approach to laboratory motor control: The modular motor control system.* Technical Report, University of Pennsylvania, Philadelphia, PA, 1989.

[21] R. Desai and R. A. Volz. Identification and verification of termination conditions in fine motion in presence of sensor errors and geometric uncertainties. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 800–807, 1989.

[22] Bruce R. Donald. Robot motion planning with uncertainty in the geometric models of the robot and environment: a formal framework for error detection and recovery. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1588–1593, 1986.

[23] A. Fancello, J. P. Porter, and E. R. Reinhart. Force reflection effects on operator performance of remote maintenance and inspection systems. In *Utility/Manufacturer Robot User Group*, 1988.

[24] B. Faverjon and P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. In *IEEE International conference on Robotics and Automation*, 1987.

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

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

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

[28] R. Finkel, R. Taylor, R. Bolles, R. Paul, and J. Feldman. *AL, a programming system for automation.* Technical Report STAN-CS-74-456, Stanford University, 1974.

[29] P. Fischer, R. Daniel, and K. V. Siva. Specification and design of input devices for teleoperation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 540–545, 1990.

[30] W. D. Fisher and M. S. Mujtaba. Hybrid position/force control: a correct formulation. to be published.

[31] C. P. Fong, R. S. Dotson, and A. K. Bejczy. Distributed microcomputer control system for advanced teleoperation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 987–995, 1986.

[32] Q. J. Ge and J. M. McCarthy. Functional constraints as algebraic manifolds in a clifford algebra. To appear in *The International Journal of Robotics Research.*

[33] 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.

[34] R. C. Goertz and R. C. Thompson. Electronically controlled manipulator. *Nucleonics*, 1954.

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

[36] D. D. Grossman and R. H. Taylor. Interactive generation of object models with a manipulator. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-8(9):667–679, September 1978.

[37] B. Hannaford and R. Anderson. Experimental and simulation studies of hard contact in force reflecting teleoperation. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 584–589, April 1988. Vol. 1.

[38] B. Hannaford and P. Lee. Hidden Markov model analysis of force/torque information in telemanipulation. In *Proceedings of 1st International Symposium on Experimental Robotics*, 1989.

[39] Blake Hannaford. A design framework for teleoperators with kinesthetic feedback. *IEEE Transactions on Robotics and Automation*, 5(4), August 1989.

[40] T. Hashimoto, T. B. Sheridan, and M. V. Noyes. Effects of predictive information in teleoperation with time delay. *Japanese journal of ergonomics*, 22(2), 1986.

[41] Y. Hatamura and H. Morishita. Direct coupling system between nanometer world and human world. In *IEEE Conference on Microelectronic and Mechanical Systems*, February 1990. Napa Valley, CA.

[42] Vincent Hayward. *RCCL User's Manual.* Technical Report TR-EE-83-46, Purdue University, 1983.

[43] G. Hirzinger, J. Heindl, and K. Landzettel. Predictive and knowledge-based telerobotic control concepts. In *IEEE International Conference on Robotics and Automation*, pages 1768–1777, 1989.

[44] Richard Hoffman. Automated assembly in a CSG domain. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1989.

[45] Neville Hogan. Mechanical impedance control in assistive devices and manipulators. In *Proceedings of the Joint Automatic Control Conference*, pages TA10–B, San Francisco, 1980.

[46] J. Jennings, B. Donald, and D. Campbell. Towards experimental verification of an automated compliant motion planner based on a geometric theory of error detection and recovery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 623–637, 1989.

[47] O. Khatib. The operational space formulation in robot manipulator control. In *15th ISIR*, Tokyo, Japan, September 1985.

[48] J. P. Kilpatrick. *The Use of Kinesthetic Supplement in an Interactive System.* PhD thesis, University of North Carolina at Chapel Hill, 1976. Computer Science Department.

[49] W. S. Kim, B. Hannaford, and A. K. Bejczy. Shared compliance control for time-delayed telemanipulation. In *First Internnational Symposium on Measurement and Control in Robotics*, June 1990.

[50] L. I. Lieberman and M. A. Wesley. Autopass: An automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research and Development*, 21(4), 1977.

[51] Thomas Lindsay. *Design of a Tool-Surrounding Compliant Instrumented Wrist.* Technical Report MS-CIS-91-30, University of Pennsylvania, 1991.

[52] Y. Liu and R. J. Popplestone. Planning for assembly from solid models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 222–227, 1989.

[53] John Lloyd. *Implementation of a robot control development environment.* Master's thesis, McGill University, 1985.

[54] T. Lozano-Perez, J. Jones, E. Mazer, P. O'Donnell, W. Grimson, P. Tournassoud, and A. Lanusse. Handey: A robot system that recognizes, plans and manipulates. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 843–849, 1987.

[55] T. Lozano-Perez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine motion strategies for robots. In *Robotics Research: The First International Symposium*, August 1983.

[56] J. Y. S. Luh, M. W. Walker, and R. P. Paul. Resolved-acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Control*, AC-25:468–474, 1980.

[57] M. T. Mason and J. K. Salisbury. *Robot hands and the mechanics of manipulation.* MIT Press, Cambridge, Massachusetts, 1985.

[58] Matthew T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(6):418–432, June 1981.

[59] Matthew T. Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 1986.

[60] Matthew T. Mason. On the scope of quasi-static pushing. In *Robotics Research: The Third International Symposium*, October 1985.

[61] NASA. *Flight Telerobotic Servicer, Tinman Concept, In-house Phase B Study – Final Report.* Technical Report, Goddard Space Flight Center, Greenbelt, MD, September 1988. Volumes I and II.

[62] Evar D. Nering. *Linear algebra and matrix theory.* John Wiley & Sons, Inc., New York, 2 edition, 1970.

[63] 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.

[64] Michael A. Noll. Man-machine tactile communication. *SID journal*, July/August 1972.

[65] M. Noyes and T. B. Sheridan. A novel predictor for telemanipulation through a time delay. In *Proceedings of the 20th Annual Conference on Manual Control*, Moffett Field, CA: NASA Ames Research Center, 1984.

[66] M. Ouh-young, D. Beard, and F. Brooks. Force display performs better than visual display in a simple 6-D docking task. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1462–1466, 1989.

[67] M. Ouh-young, M. Pique, J. Hughes, N. Srinivasan, and F. Brooks. Using a manipulator for force display in molecular docking. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1824–1829, 1988.

[68] William J. Palm. *Modeling, Analysis and Control of Dynamic Systems.* John Wiley & Son, Inc., 1983.

[69] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control. MIT Press Series in Artificial Intelligence*, MIT Press, Cambridge, Massachusetts, 1981.

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

[71] A.P. Pentland. Perceptual organization and the representation of natural form. *Artificial Intelligence*, 28(2):293–331, Feb 1986.

[72] R. L. Pepper, D. C. Smith, and R. E. Cole. Stereo tv improves operator performance under degraded visibility conditions. *Optical engineering*, 20:579–585, 1981.

[73] M. A. Peshkin and A. C. Sanderson. Planning robotic manipulation strategies for sliding objects. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 696–701, 1987.

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

[75] M. H. Raibert and J. J. Craig. Hybrid position/force control of manipulators. *ASME Journal of Dynamic Systems, Measurement and Control*, 126–133, June 1981.

[76] G. J. Raju. *Operator Adjustable Impedance in Bilateral Remote Manipulation.* PhD thesis, M.I.T., Department of Mechanical Engineering, September 1988.

[77] Aristides A. G. Requicha. Representations for rigid solids: theory, methods, and systems. *ACM Computing Surveys*, 12(4), Dec 1980.

[78] J. K. Salisbury. Active stiffness control of a manipulator in Cartesian coordinates. In *19th IEEE Conference on Decision and Control*, December 1980.

[79] A. C. Sanderson, M. A. Peshkin, and H. De Mello. Task planning for robotic manipulation in space. In *IEEE Transactions on Aerospace and Electronic Systems*, pages 619–628, 1988.

[80] C. Sawada, H. Ishikawa, K. Kawase, and M. Takata. Specification and generation of a motion path for compliant motion. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 808–815, 1989.

[81] 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.

[82] J. De Schutter and H. Van Brussel. Compliant robot motion I: a formalism for specifying compliant motion tasks. *The International Journal of Robotics Research*, 7(4), August 1988.

[83] J. De Schutter and H. Van Brussel. Compliant robot motion II: a control approach based on external control loops. *The International Journal of Robotics Research*, 7(4), August 1988.

[84] J. De Schutter and J. Leysen. Tracking in compliant robot motion: automatic generation of the task frame trajectory based on observation of the natural constraints. In *Proceedings of the International Symposium on Robotics Research*, pages 127–135, 1987.

[85] Thomas B. Sheridan. Telerobotics and human supervisory control. to be published as a book.

[86] Bruce Shimano. VAL: a versatile robot programming and control system. In *Proceedings of COMPSAC*, Chicago, November 1979.

[87] S. N. Srihari. Representation of three-dimensional digital images. *ACM Computing Surveys*, 13(4), Dec 1981.

[88] L. W. Stark, W. S. Kim, and F. Tendick. Cooperative control in telerobotics. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 593–595, 1988.

[89] Lawrence Stark. Teleobotics: problems and research needs. In *IEEE Transactions on Aerospace and Electronic Systems*, pages 542–551, 1988.

[90] Lawrence Stark. Telerobotics: display, control, and communication problems. *IEEE Journal of Robotics and Automation*, RA-3(1), February 1987.

[91] R. H. Taylor, P. D. Summers, and J. M. Meyer. AML: A Manufacturing Language. *The International Journal of Robotics Research*, 1(3):19–41, 1982.

[92] Daniel E. Whitney. Historical perspective and state of the art in robot force control. *The International Journal of Robotics Research*, 6(1), 1987.

[93] Daniel E. Whitney. Quasi-static assembly of compliantly supported rigid parts. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:65–77, March 1982.

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

[95] Hong Zhang. *Design and Implementation of a Robot Force and Motion Server.* PhD thesis, Purdue University, 1986.

A.2 Contact Operations Using an Instrumented Compliant Wrist

Contact Operations Using an Instrumented Compliant Wrist†

Thomas Lindsay, Janez Funda, and Richard Paul

Abstract

Teleprogramming was developed as a solution to problems of teleoperation systems with significant time delays [5]. The human operator interacts in real time with a graphical model of the remote site, which provides for real time visual and force feedback that is an important tool for teleoperation. The master system automatically generates symbolic commands based on the motions of the master arm and the manipulator/model interactions, given predefined criteria of what types of motions are to be expected. These commands are then sent via the communication link, which may delay the signals, to the remote site. Based upon a remote world model, defined beforehand and possibly refined as more information is obtained, and the commands sent from the master, the slave carries out operations in the remote world and decides whether each command has been executed correctly.

Contact operations involve the robot interactions with the environment, including planned and unplanned collisions, and motion within contact with the environment. A hybrid position/force control scheme using a compliant instrumented wrist has been demonstrated to be very effective for these types of operations. In particular, switching between position and force modes (when contacting a surface, for example) does not present problems for the system. A brief introduction of teleprogramming and contact operations is presented, including a model of sliding motions and early experimental results. Problems with these early experiments are presented, and solutions to these problems are discussed. The criteria for an object to slide rather than tip over are presented, relating to the geometry of the object and the applied forces. Finally, methods are presented to match the experimental results to a simple model, to help the remote manipulator to quickly and robustly sense collisions.

1 Introduction

Teleoperation systems are important for executing tasks in hazardous and unstructured environments. Hazardous environments range from those extremely dangerous to humans, such as contaminated nuclear power plants and hazardous waste sites, to those such as space and deep sea that can be made fairly safe to humans, for short periods at great expense. Completely autonomous activity and manipulation is impractical in unstructured environments with state of the art artificial intelligence.

†This material is based upon work supported by the National Science Foundation under Grant No. BCS-89-01352, "Model-Based Teleoperation in the Presence of Delay." Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

When delays in excess of one second occur, direct force reflecting teleoperation becomes difficult to impossible [6, 12]. Delays can occur on the order of 2-8 seconds for communication with a remote site orbiting the earth (shallow space),and up to 20 seconds for subsea operations (communicating via acoustic link). In order to solve problems associated with communication delays, we have developed a novel teleoperation structure, *teleprogramming* [9]. At the master site, a human operator works with a 6-DOF master arm to guide a simulated slave robot in a geometric model of the remote site. The model provides for monitoring of contacts, and feeds back information to the master arm to give the operator kinesthetic feedback, lacking in most of the current work involving time delays [2, 8, 11, 12].

The master system generates commands based upon the motions and manipulator/model interactions. This information is sent to the remote site, which interprets and executes these low level command steps. Each step is executed autonomously, and the resulting motion of the slave manipulator is analyzed as to whether it succeeded or failed. If it succeeds, an acknowledgment is sent to the master and the slave continues with the next command. Commands from the master are sent continuously, so there is no delay between commands at the remote site if they are executed without an error occurring. If a command fails, information about the error state is sent to the master, and then the slave waits for the human operator to send a new set of commands that will correct the error.

This system closes the force feedback loop at each site, and only uses the delayed communication link between sites to send commands, acknowledgments, and error messages, and not for control feedback. The system can operate with significant time delays, and will operate at the same rate as a direct teleoperation system without time delays unless errors occur in execution.

At the remote site, the slave interprets small execution model steps that make up individual motions. Each execution model step contains information about how long and how far to move, information about contacts and contact forces, and information about what conditions the slave should expect to terminate the motion. For example, a typical move could command the slave manipulator to slide along a surface, pushing against it with a given force, and stop when a wall is encountered. Several errors can occur while trying to execute such a move. The slave system has the task of determining whether any of these errors has occurred. Errors in this example could include falling off the surface, failing to find the specified wall, and encountering an unexpected obstacle.

Identifying such errors in an ideal world would be simple. If the normal force disappears, contact with the surface has been lost. If a contact force in the normal direction to the wall is not found within a specified distance, the wall is not where we expect it. Unexpected forces in other directions, or forces in the wall direction before a specified distance has been traveled indicate that an obstacle has been encountered. In practice, however, there are problems with such a simple approach.

We are using an instrumented compliant wrist for sensory feedback [16]. The compliance is extremely beneficial for the interactions (expected and unexpected) between the manipulator and the environment [10]. However, the compliance makes sliding motions more complex. Depending on the surface friction and the applied forces, the object on the surface may tend to tip over instead of sliding. Control and other problems lead to non-constant steady state forces in the normal and tangential directions. Peaks in these forces, which are used to determine expected and unexpected collisions, can cause a false

identification of an error state. The level of system 'noise' is partially a function of manipulator configuration and direction of movement, so that constant limits that would work successfully in one direction will not work in another. A more robust method of detecting collisions while performing contact operations is necessary.

The rest of the paper is organized as follows. First, the experimental *teleprogramming* testbed is presented. Next, contact operations in the task environment are examined, from both a model based and experimental based perspective. Criteria for an object to slide rather than tip over are presented. Finally, methods to relate the model and experimental results are examined, with an emphasis on a more general and robust method for interpreting sensor readings.

2 Experimental Setup

The GRASP Lab [1] teleprogramming testbed is shown in schematic form in figure 1. The operator's station and the remote workcell are physically separated. The system can be divided into the master site, the remote site, the communication link between these sites, and the task environment.

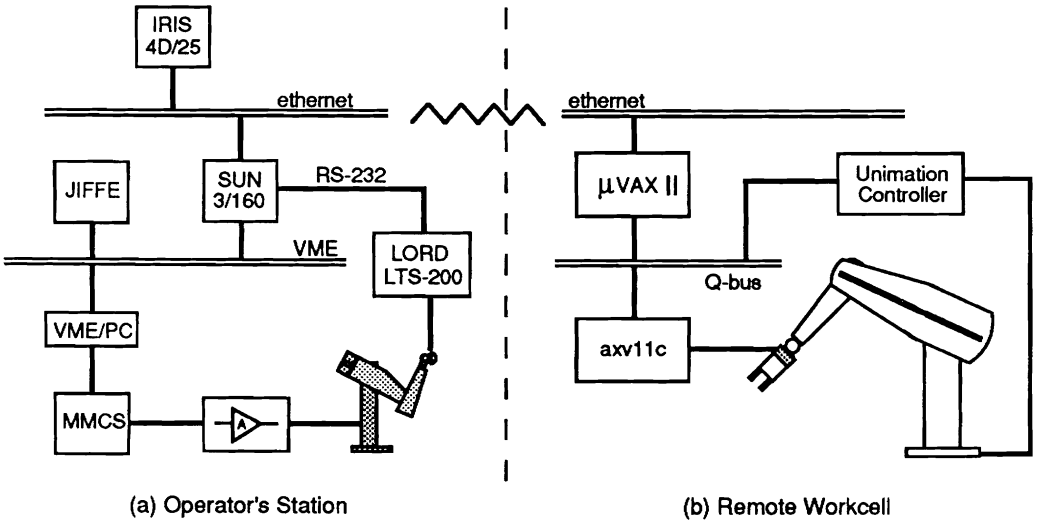


Figure 1: The experimental teleprogramming testbed

2.1 Master Site

The master site is composed of a Unimation Puma 250 robot, acting as a 6-DOF backdrivable input device, and several computers. The Puma hardware is controlled by PC-bus based Modular Motor Control System (MMCS) [4]. There is a 6-d.o.f. force/torque sensor (LORD Corp., LTS-200) mounted at the tip of the 250, which measures the directional input from the human operator. Joint and cartesian level control for the master

[1] General Robotics and Active Sensory Perception Laboratory, University of Pennsylvania Dept. of Computer and Information Science, Philadelphia, PA. Ruzena Bajcsy, Director.

is performed by JIFFE - a 20 Mflop VME-based floating point co-processor [1]. JIFFE communicates with its host (Sun 3/160) via shared memory and with the graphical workstation (Iris 4D/25) via the Sun and ethernet socket connection. The Iris runs a modeling environment for 3-D manipulation of articulated figures, provided by the Computer Graphics Laboratory at the University of Pennsylvania [3]. This software provides the operator with a graphical model of the remote manipulator and its environment. Manipulator/environment interaction is monitored, and is fed back to the master manipulator. This provides the operator with kinesthetic feedback, which is an important part of the teleprogramming system. The master system contains no information about the dynamics or friction at the remote site.

2.2 Remote Site

The remote manipulator is a Puma 560 robot, linked to a MicroVax II. The robot uses a cartesian-based hybrid position/force controller, built upon the low-level RCI robot interface [7]. The hybrid controller has been shown experimentally to be stable in the operating region we are using, as long as the task frame origin is located relatively close (within 20 cm) to the robot wrist point. We use a 6-DOF instrumented compliant wrist for force/torque measurements.

The compliance of the wrist simplifies interactions between the robot and the environment. This is especially beneficial in dealing with the impact forces generated when the robot makes the transition from free space motion to motion in contact with the environment. Both natural and active damping help absorb the energy of impact [14]. Also, the compliance of the sensor helps to make the force control more responsive [10]. Because the wrist is instrumented, the characteristics of the compliance can be changed with the control laws. This means that the effective compliance of the wrist can be changed to suit the task, so that the effective wrist stiffness can be changed for force control and for position control. Also, the position of the environment with respect to the robot is known via the sensors.

2.3 Communication

Communication from master to slave is composed of execution models (EMs), which are automatically generated at the master site. These made up of small packets of commands that together make up the entire program. Each EM step can contain information about the working task frame, the hybrid modes, contact forces, and movement information. Information not supplied in a given EM step is assumed to carry over from the previous step, thus communication time is reduced by elimination of repetition of known information. The EM step does not contain information about the dynamics or friction of the environment.

The communication between the robots in the lab, using an ethernet connection, is virtually instantaneous. Therefore, we have a programmed delay to emulate communication delay. This delay can be varied. Currently, we are using a delay of 3 seconds for our experiments.

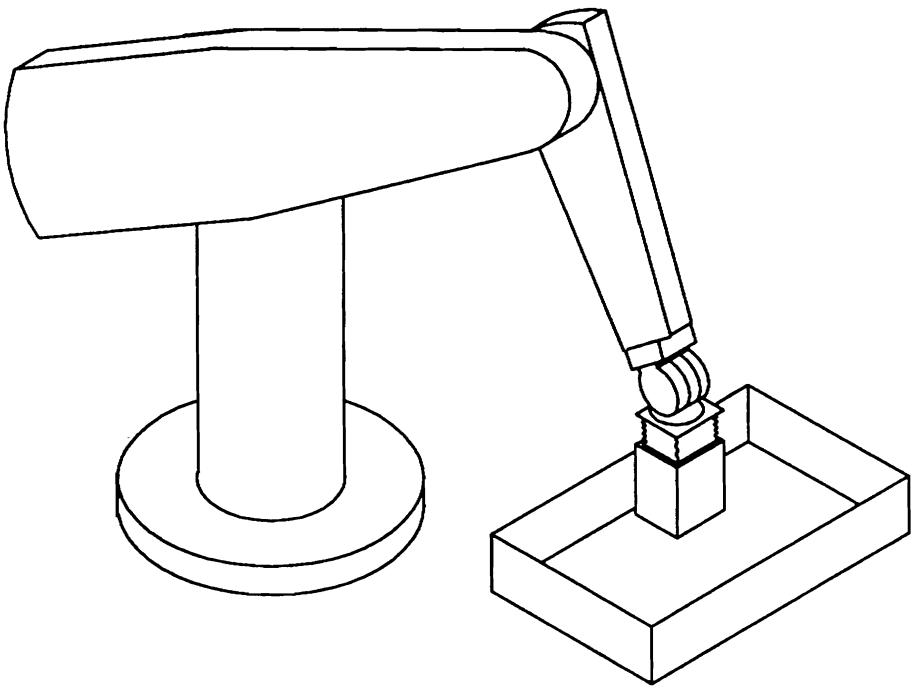


Figure 2: Remote Site and Task Environment

2.4 Task Environment

We are currently experimenting with very simple contact operations. A small box attached to the manipulator is maneuvered into and around a larger box, as shown in figure 2. Elements of tasks include free-space motion, transitions between free-space (position mode) and constrained space (force mode), and constrained motion. In this environment we can test error detection and error recovery. Within this task environment, our command language and teleprogramming concept have been shown to be effective. Problems between theory and experimental work have also been examined, and in many cases we have modified how commands are interpreted at the remote site. Complex procedures can be built using the commands we can now generate. Current work includes creating a new task environment which requires more complex motions.

3 Contact Operations - Experimental Results

Although many tasks include free-space motion, most tasks require interaction with the environment. Free-space motion is a relatively simple operation; there is no need for feedback at the operator's station, and therefore a telerobotic scheme that has only visual feedback (in real time) would work (as with JPL's predictive display). Most of our work concentrates on contact operations, where the manipulator interacts with the environment. The actions presented in this paper include contacting surfaces, and sliding along surfaces.

For two reasons, contact operations are executed semi-autonomously. First, the communication delays make force feedback to the operator impossible. Therefore, the remote site must close the feedback loop locally. Second, there may be inaccuracy in the graphical model at the master site. If the geometry of the environment is known only to a tolerance

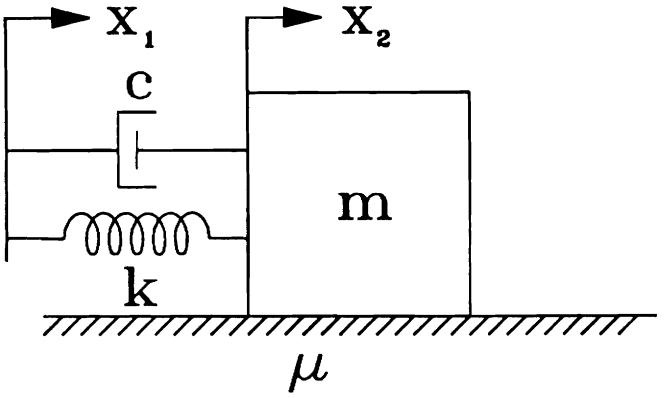


Figure 3: Second Order Model

ϵ, the remote site must locally deal with this inaccuracy. The remote site can deal with these factors; hence the term semi-autonomous. However, the system still runs into problems. A fully autonomous system would have to understand all possible problems and deal with them appropriately; this is beyond the scope of modern artificial intelligence. When the remote system runs into a problem that it cannot correct, it simply sends back information to the human operator, who can reason through the problem and create a suitable correction.

Due to the slow and often unreliable (esp. with acoustic links) communications, the commands sent to and from the remote site need to be minimal. The remote site receives only information about the kinematics of the system. Dynamics and friction must be dealt with locally at the remote site. Further, the remote site must keep pace with the master site, albeit delayed by the communications. The slave therefore has no opportunity to explore the environment beyond the scope of the commanded actions. Thus the system can only gain information about the remote environment, such as friction, while it is also trying to discern expected and unexpected changes from the sensor data. Within these constraints, the remote system must react with the environment and robustly sense forces, contacts, and collisions.

3.1 Contact Model

Motion of the robot/sensor/environment interaction can be simulated for one degree of freedom using a second order model. The second order model is a mass-spring-damper system with a velocity input and coulomb friction. The equation for this model is shown below:

$$m\ddot{x_2} + c\dot{x_2} + k(x_2 - x_1) = f \tag{1}$$

where f represents the coulomb friction.

$$f = \begin{cases} \mu_{sl} N(\frac{\dot{x_2}}{|\dot{x_2}|}) & \text{if } (\frac{\dot{x_2}}{|\dot{x_2}|})\dot{x_2} > v_s \text{ and } F > \mu_{sl} N(\frac{\dot{x_2}}{|\dot{x_2}|}) \\ \mu_{st} N(\frac{\dot{x_2}}{|\dot{x_2}|}) & \text{if } (\frac{\dot{x_2}}{|\dot{x_2}|})\dot{x_2} < v_s \text{ and } F > \mu_{st} N(\frac{\dot{x_2}}{|\dot{x_2}|}) \\ F & \text{otherwise} \end{cases} \tag{2}$$

where

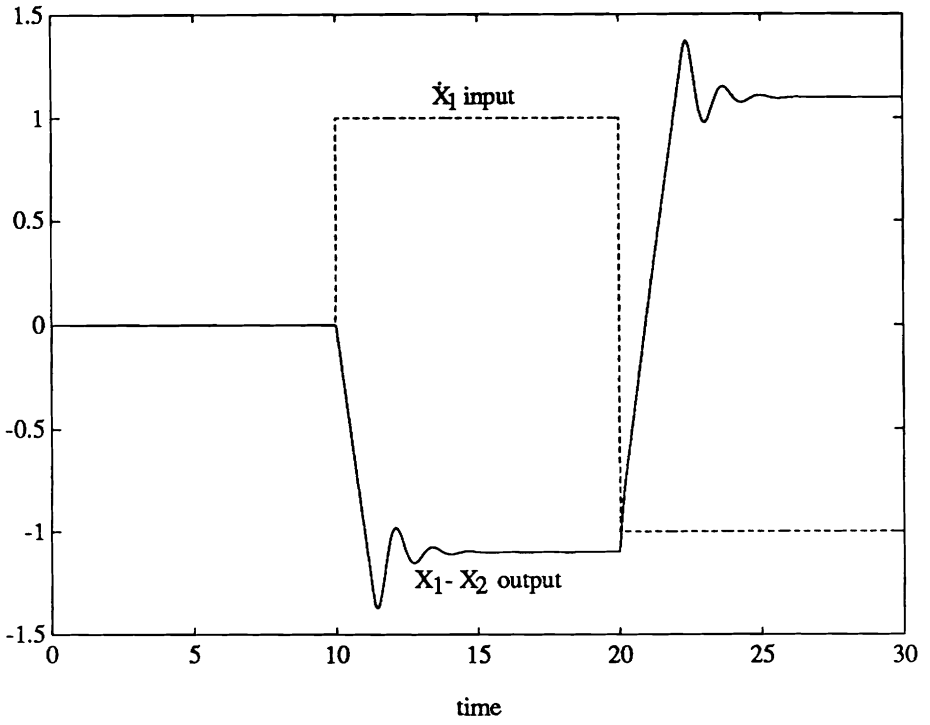


Figure 4: Second Order Model Response

$$F = c\dot{x}_2 + k(x_2 - x_1) \tag{3}$$

and N is the surface normal force. The value of v_s is the cutoff velocity that defines where, for simulation purposes, μ_{st} (static friction) no longer applies, and the value μ_{sl} (sliding friction) is used. Values for m, c, and k are selected to model the wrist behavior, but do not represent the exact physical parameters of the wrist.

The spring-damper subsystem models the wrist sensor. The output from the sensor will be the change in spring length, Δx, and can be interpreted explicitly as a position deflection, or implicitly, using Hooke's law $F = k\Delta x$, as a force. Figure 3 illustrates the second order model. Figure 4 displays data from a simulation of the second order model, with the velocity input shown. For a given mass and input velocity, the rise time and the output level are a function of the spring constant and the coulomb friction. Overshoot is a function of the coulomb friction value and the viscous damping term.

3.2 Experimental Data

Data from the system can be collected and compared to the simulation model. In this section, some of the data will be presented, along with an introduction to some of the problems that were encountered while using the system. One problem was that the box being moved had a tendency to tip over while being pushed. Also, there were many problems associated with sliding along a surface until a wall was encountered. False interpretation of sensor readings, due to uneven frictional force and noise caused by sensor electronics and by arm control, cause the system to stop before hitting a wall or to press on

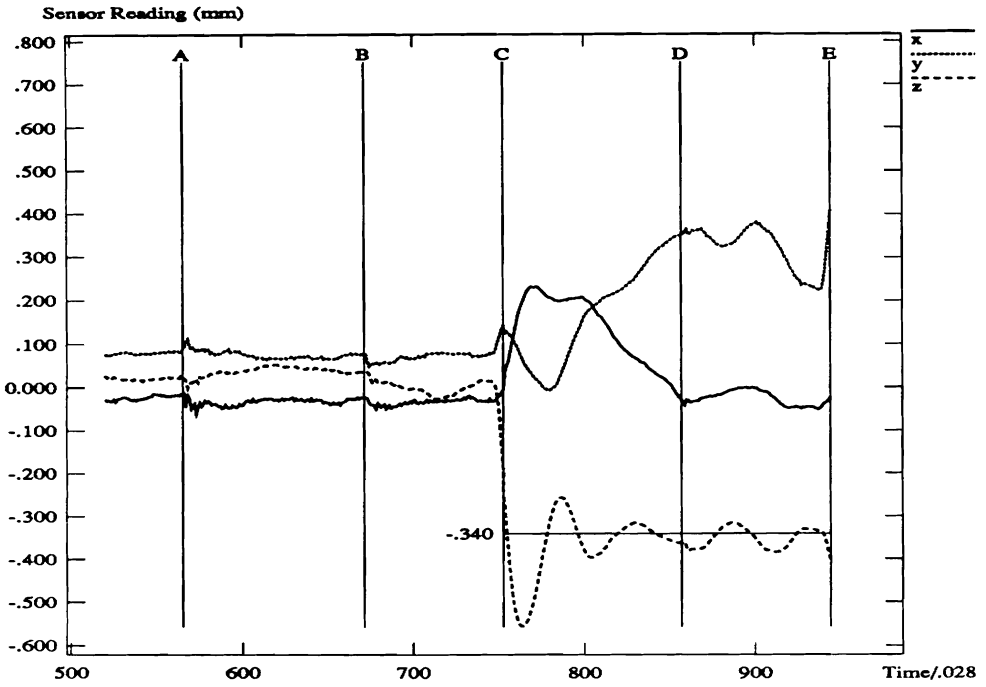


Figure 5: Sensor Readings From a Typical Move

the wall with a large force before deciding to stop. Methods to overcome these problems are presented later in this paper.

Figure 5 shows the sensor readings, for translational directions, of a typical move. The robot moves at approximately 2 cm/sec. Section A-B is a free-space motion. There is a small amount of noise at the beginning of move A-B, which is caused by the transition from the previous free-space motion. Section B-C is a guarded move. At the end of move B-C, the robot comes into contact with the environment. Here, there is a large change in the z-direction sensor reading. The contact is smooth and stable, and the robot never breaks contact with the environment. Move C-D is a standard sliding motion, with the robot in contact with the environment and moving in the negative y-direction. The robot tries to maintain a normal force (z-direction) of approximately 1 lb. (.34 mm) while sliding. There are large, unexpected changes in the x and y sensor readings during this section of the motion. Theoretically, there should be no forces in the x-direction, and the y-direction should have a constant frictional force of μN. The sensors, however, show that the tangential force (y-direction) has a minimum below zero, and a maximum of approximately 0.6 lb. Section D-E is another guarded move, and the robot comes into contact with a wall of the box. The slope of the y-direction sensor reading is high, but the actual value of the reading when the robot touches the wall is not significantly higher than other readings in the D-E section.

Figures 6 and 7 illustrate one of the inaccuracies of the sensor readings. Data "test1" and "test2" are from similar moves. Section B-C shows the robot coming into contact with the environment. In section C-D, the robot moves slightly away from the wall, and in section D-E the robot moves into contact with the wall. Motion is in the x-direction for the "test1" data, while test2 motion is in the y-direction. Notice that while figure 6 shows very similar normal forces for the two tests, section D-E in figure 7 shows very different tangential forces. If we assume that the tangential forces in section C-D are accurate, and

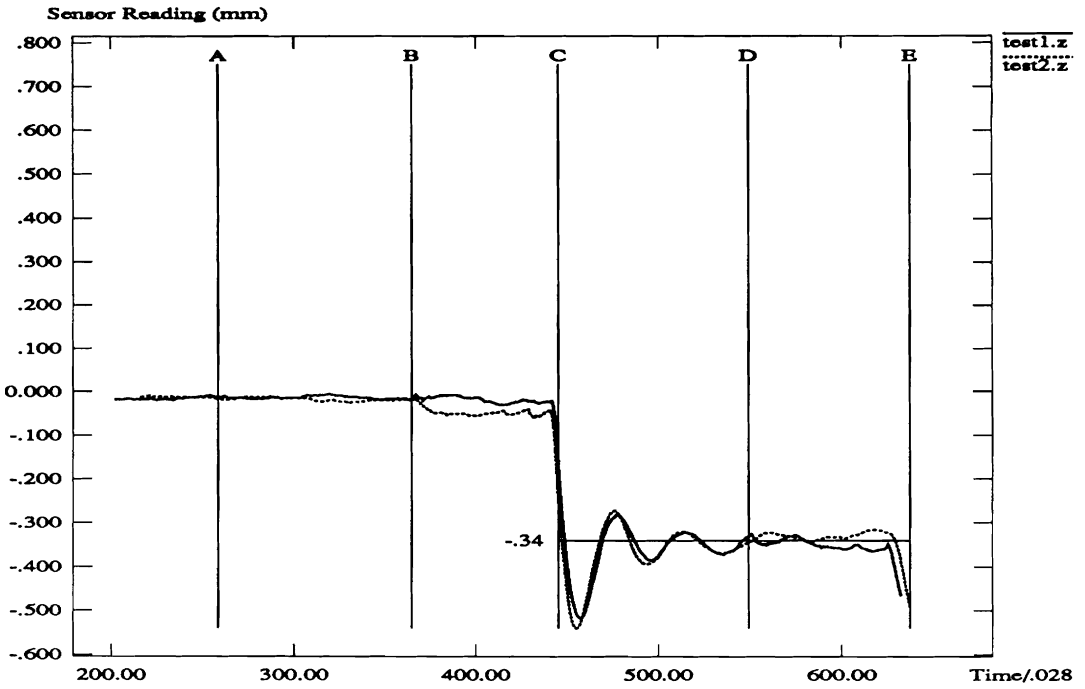


Figure 6: Direction Dependent Sensor Readings: Normal Force

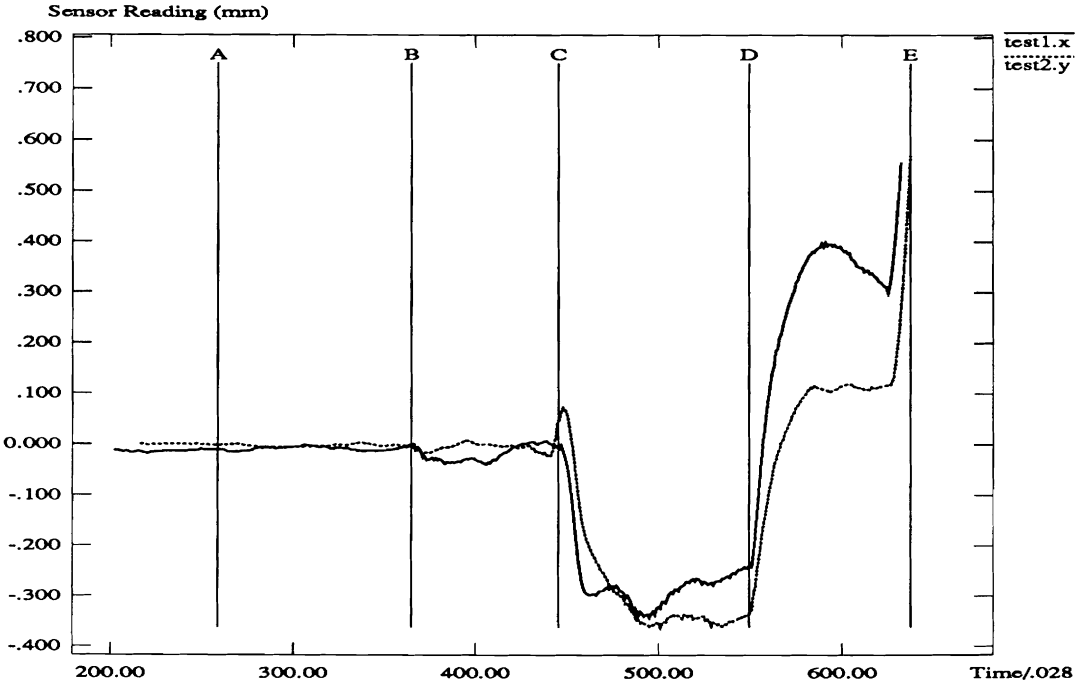


Figure 7: Direction Dependent Sensor Readings: Tangential Force

the frictional force is approximately .5 lb., the force in section D-E for test1 is too high at the mid point of the move, and the force for test2 is very low. The cause of this direction dependent phenomena is unknown, but a method to overcome the problem must be found in order to correctly monitor collisions and contacts.

The data presented above was collected after the tipping problem, presented below, was overcome.

4 Sliding vs. Tipping

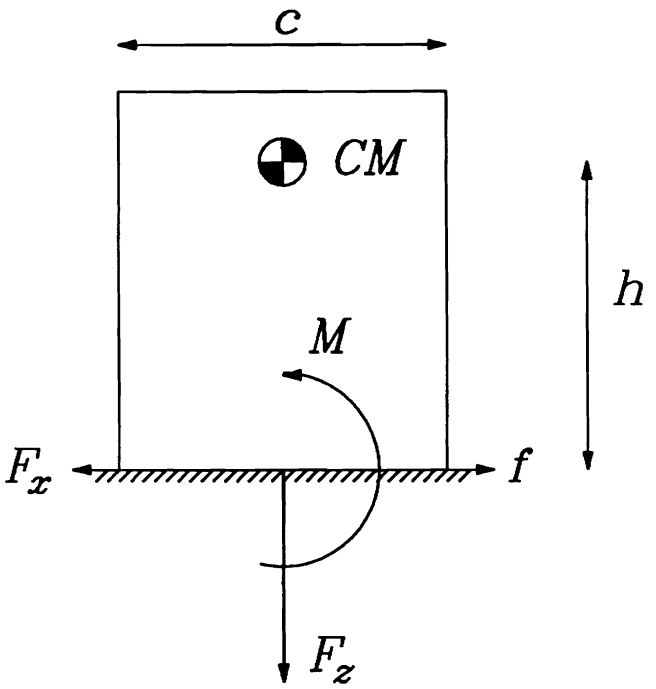


Figure 8: Forces on box in tool tip coordinates

When sliding in contact with the environment, the robot sometimes has the tendency to let the box tip over. There are many factors that contribute to this tipping phenomenon. Three factors discussed here are the height to width ratio of the box, the normal to tangential force ratio, and the effect of rotational compliance upon sliding stability.

Expressing the applied forces in tool tip coordinates [15], which for this case will be the bottom of the box, the conditions for the box to tip over in the positive and negative Y-directions are found by summing the moments about the center of mass. The normal force N will act at the left side of the box if it is tipping about the negative Y-direction (into the page in figure 8), and the criteria for a 2-dimensional box not to tip is:

$$(F_x - f)h + M + N(\frac{c}{2}) \geq 0 \tag{4}$$

The normal force will act at the right side if it is tipping in the positive Y-direction. The criteria for the box not to tip in this direction is:

$$(F_x - f)h + M - N(\frac{c}{2}) \leq 0 \tag{5}$$

where

$$N = F_z + mg \tag{6}$$

$$f = \mu N \tag{7}$$

(See figure 8). Reorganizing,

$$hF_x - (\mu h - \frac{c}{2})F_z - mg(\mu h - \frac{c}{2}) \geq -M \tag{8}$$

$$hF_x - (\mu h + \frac{c}{2})F_z - mg(\mu h + \frac{c}{2}) \leq -M \tag{9}$$

If we assume that mg is negligible compared with applied forces and moments,

$$hF_x - (\mu h - \frac{c}{2})F_z \geq -M \tag{10}$$

$$hF_x - (\mu h + \frac{c}{2})F_z \leq -M \tag{11}$$

In terms of h/c, these equations become

$$\frac{h}{c}(F_x - \mu F_z) - \frac{c}{2}F_z \geq -\frac{M}{c} \tag{12}$$

$$\frac{h}{c}(F_x - \mu F_z) + \frac{c}{2}F_z \leq -\frac{M}{c} \tag{13}$$

These equations are plotted in figure 9, with parameters: $F_x = 1.0, \mu = .1$, and F_z as shown. Note that for a given F_z, if h is large compared to c, then a moment must be applied for the box to remain stable. Also note that as F_z increases, the box will not tip for a greater range of applied moments. It is therefore more stable.

In terms of F_x/F_z, equations 12 and 13 become

$$h\frac{F_x}{F_z} - (\mu h - \frac{c}{2}) \geq -\frac{M}{F_z} \tag{14}$$

$$h\frac{F_x}{F_z} - (\mu h + \frac{c}{2}) \leq -\frac{M}{F_z} \tag{15}$$

$$\tag{16}$$

These equations are plotted in figure 10 with parameters: $h = 5.0, \mu = .1$, and c as shown. For a given value of c, if F_x is large compared with F_z, a moment must be applied for the box to remain stable. As c increases, the range for the applied moment becomes greater, and the box becomes more stable.

The conditions above are intuitive and easy to compensate for. However, in our experimentation the box still tends to tip. The reason has to do with the rotational compliance, and with transforming the applied forces and moment into the tool tip frame.

To transform the forces and moment, the compliance values (inverse of spring constants) of the wrist are needed. There are two parts to the compliance that are important here: the physical compliance and the control compliance. The physical compliance is

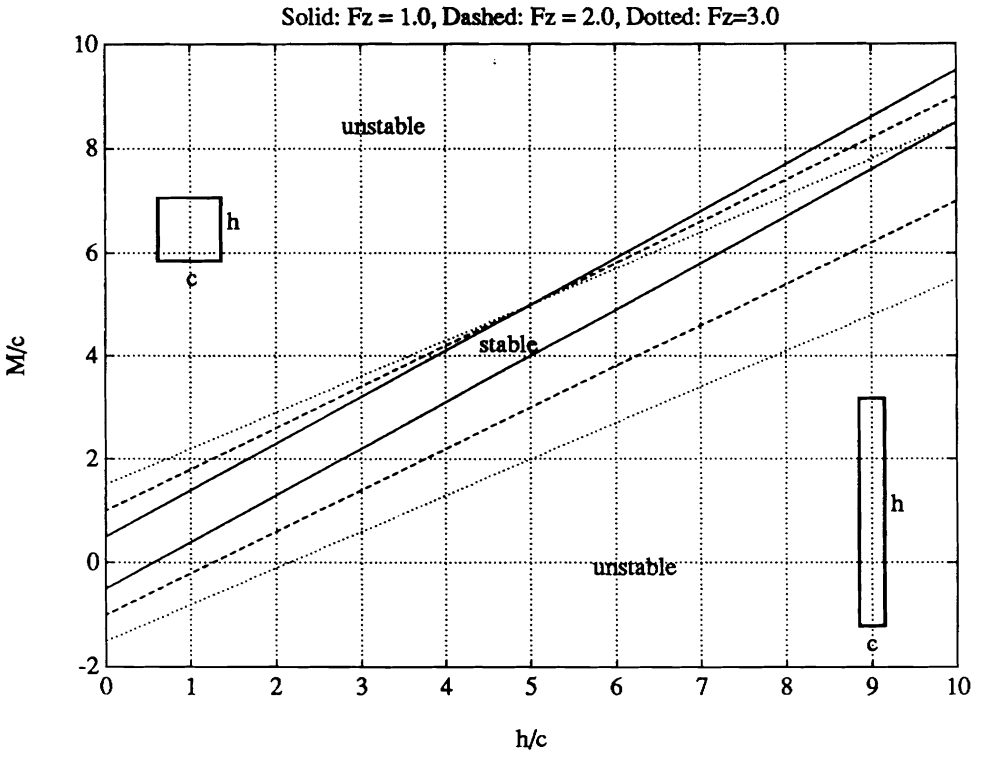


Figure 9: -M/c vs. h/c

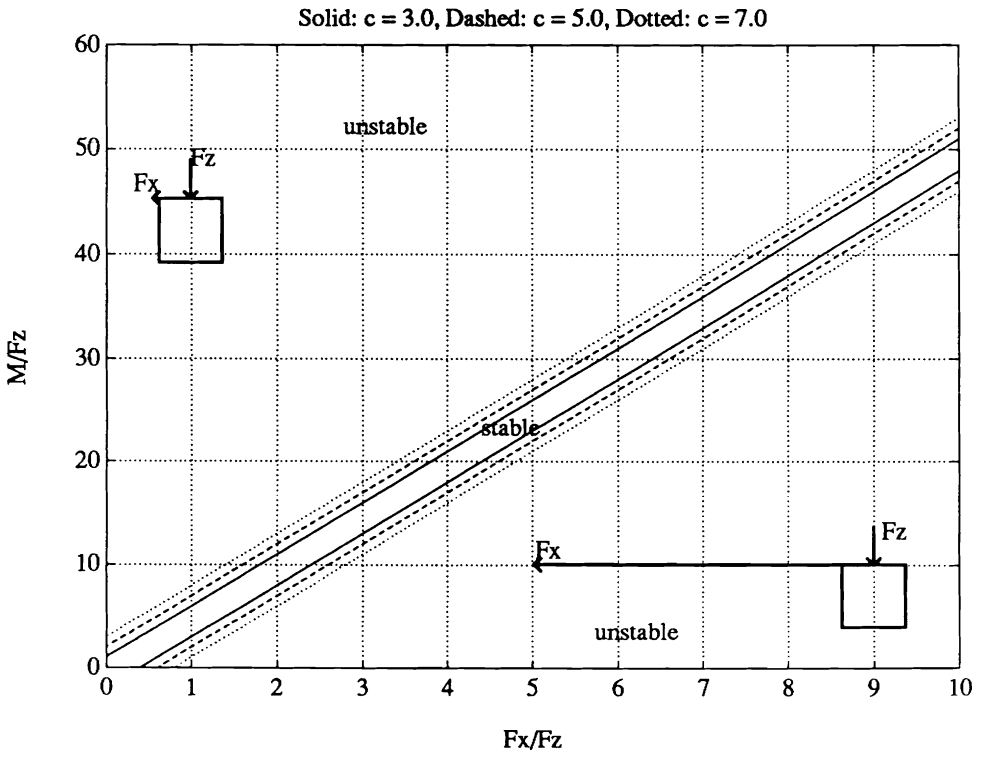


Figure 10: -M/Fz vs. Fx/Fz

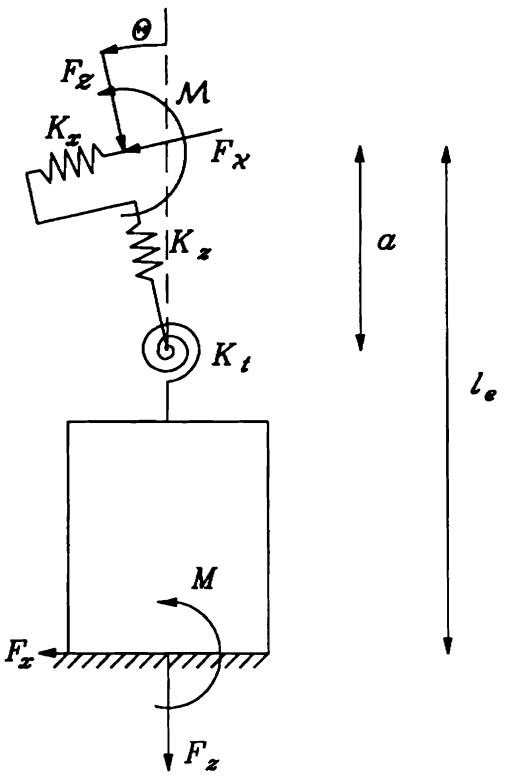


Figure 11: Transformation of forces from application to tool tip coordinates

inherent in the structure of the wrist and its compliant elements. The control compliance is a result of the gains used in the control of the system. A stiff wrist can be made more compliant with higher gains, if it remains stable. The important thing to note is that we can change the control compliance to suit our needs.

To transform the applied forces and moment for the two-dimensional wrist, the following equations are needed (see figure 11)

$$F_x = F_\mathcal{X}\cos(\Delta\theta) - F_\mathcal{Z}\sin(\Delta\theta) \quad (17)$$

$$F_z = F_\mathcal{X}\sin(\Delta\theta) + F_\mathcal{Z}\cos(\Delta\theta) \quad (18)$$

$$M = F_x(l_e - \Delta z) + F_z(\frac{c}{2} - \Delta x) + \mathcal{M} \quad (19)$$

$$\Delta x = F_\mathcal{X}/K_x \quad (20)$$

$$\Delta z = F_\mathcal{Z}/K_z \quad (21)$$

$$\Delta x = (\mathcal{M} + F_\mathcal{X}a)/K_t \quad (22)$$

Substitution yields:

$$F_x = F_\mathcal{X}\cos(\frac{\mathcal{M} + F_\mathcal{X}a}{K_t}) - F_\mathcal{Z}\sin(\frac{\mathcal{M} + F_\mathcal{X}a}{K_t}) \quad (23)$$

$$F_z = F_\mathcal{X}\sin(\frac{\mathcal{M} + F_\mathcal{X}a}{K_t}) + F_\mathcal{Z}\cos(\frac{\mathcal{M} + F_\mathcal{X}a}{K_t}) \quad (24)$$

$$M = F_x(l_e - \frac{F_\mathcal{X}}{K_x}) + F_z(\frac{c}{2} - \frac{F_\mathcal{X}}{K_x}) + \mathcal{M} \quad (25)$$

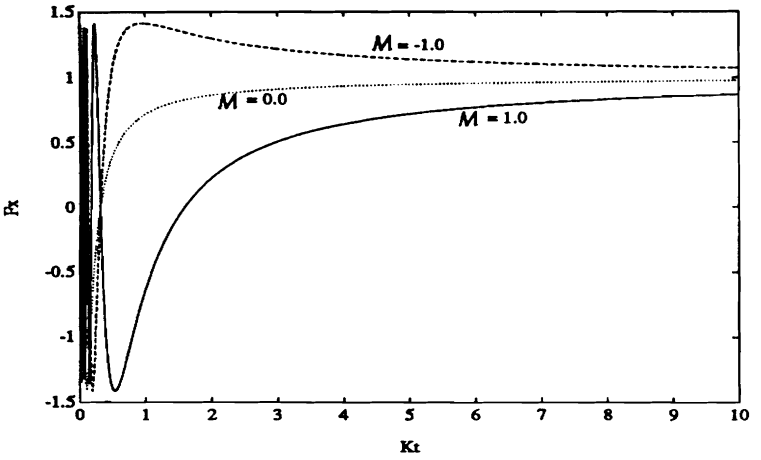


Figure 12: Fx vs. Kt for different values of M

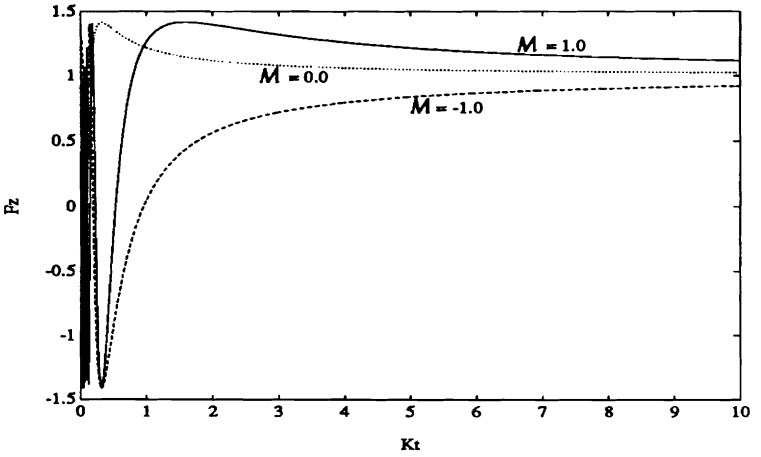


Figure 13: Fz vs. Kt for different values of M

(26)

These equations are plotted in figures 12, 13, and 14. The constants in these equations are chosen to approximate the behavior of the wrist: $K_x = 7.29$ N/mm, $K_z = 12.36$ N/mm, $l_e = 25$ cm, $c = 10$ cm, $F_{\mathcal{X}} = 1$ N, $F_{\mathcal{Z}} = 1$ N, and $\mathcal{M}$, in N-m, as shown. Variable a was chosen to be 25 cm, which would correspond to the case where the center of compliance is at the tool tip (bottom of box, here), although in the wrist it is less than this. The physical value for K_t is 6.93 N-m.

The plots show how the transformed forces and torque vary from those applied to the wrist. It is obvious that small values of K_t do not yield satisfactory performance. After decreasing the control compliance (increased K_t), the box became much more stable. Note here that figures 12, 13, and 14 also show that the tool tip forces are never the same as the applied forces. It is important to the stability of the box that the tool tip forces are controlled accurately, and the compliance of the wrist must be compensated for in the control. By examining equations 23 and 24, it is seen that the smaller the distance from the applied forces to the center of compliance (a), the less effect that the force F_x has upon changing the values of the peg tip forces. Better results would be obtained for the

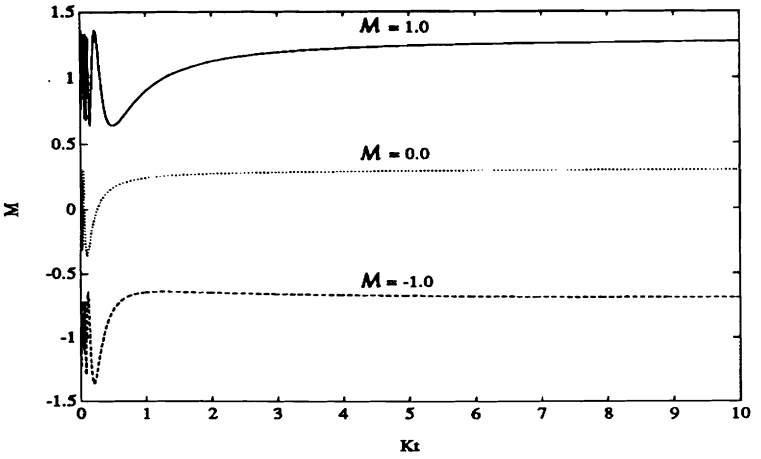


Figure 14: M vs. Kt for different values of M

operation of sliding if the center of compliance coincided with the applied forces. This is much different than the conclusions for peg insertion operations with RCC devices by Whitney [15], for which the center of compliance should be located at the tool tip.

5 Robust Stopping Conditions

The data presented in section 3.2 deviates from the second order model of the system. The deviations have many causes, and as a whole will be termed "noise".

Noise from the sensors is inherent in any system. Initially, the sensor data is conditioned using a low-pass filter in the software, in order to reduce electrically-induced sensor noise. However, the experiments suggest noise that may be dependent on more complex phenomena that may be difficult or impossible to model. Such phenomena include non-homogeneous friction, static friction, sensor coupling (coupling of compliant directions in the sensor), orientation instabilities (tipping, as presented above), and sensor-based hysteresis. These phenomena are all responsible for sensor "noise".

As the manipulator slides around the environment, it attempts to maintain a constant normal force. With a constant normal force, the sliding friction should also be constant, assuming homogeneous surface friction. Contact with a side wall of the box thus could be determined by even a small increase in the tangential force. However, experiments have shown that a small threshold value causes the system to stop on noisy data. Using a constant threshold based stopping condition, a high threshold is needed to keep the noisy data from interfering with normal stopping criteria. Too high of a threshold causes the system to interpret an actual contact with the wall as mere noise. Also, a high threshold causes the box to impact the environment with much more force than is wanted.

The following sections present attempts at developing more robust methods for determining stopping conditions, including ways to reduce the effects of the sensor noise, and to determine stopping criteria under noisy conditions.

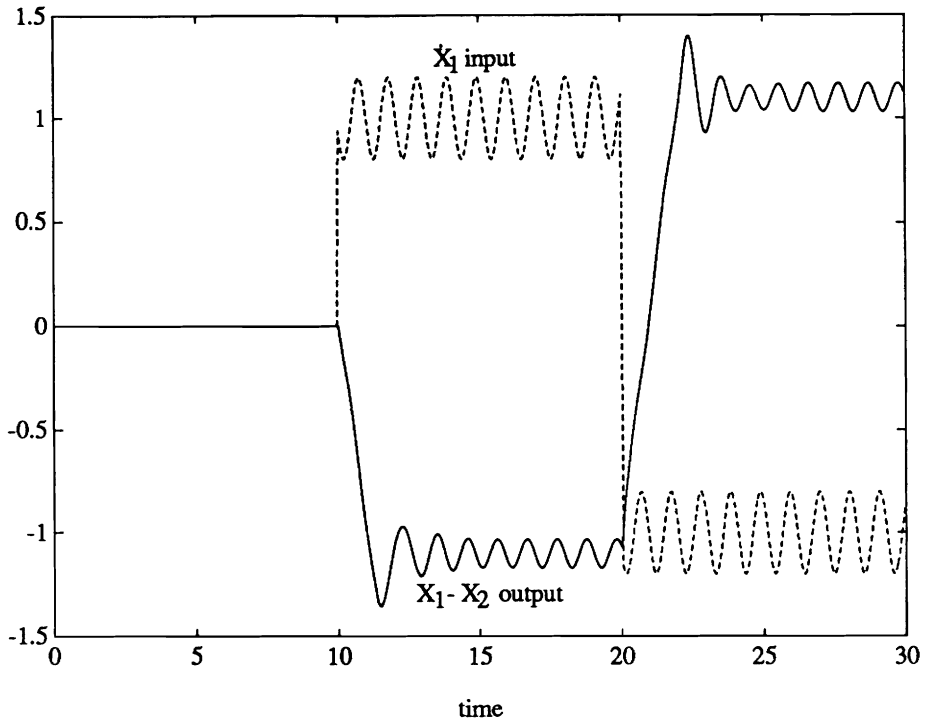


Figure 15: 2nd Order Model With Motion Perturbation

5.1 Torque Preloads

Some of the control noise could be a result of the box being on the verge of tipping over. In order to reduce this noise, a torque preload could be used to make the box more stable. The preload is computed as

$$constant * (F \times v) \tag{27}$$

Using this preload unfortunately does not reduce the control noise, and does not significantly improve the performance of the system. However, it will make the box stable under more adverse conditions, at little computational cost.

5.2 Motion Perturbation

By perturbing the motion of the manipulator with small amplitude sine waves, some of the effects of noise phenomena can be actively reduced. Specifically, static friction problems can be overcome.

Figure 15 shows the simulated output of the second order model with a velocity input as in figure 4, with a superimposed sine wave with an amplitude of 1/5 the constant velocity input. The output is quite similar to that of figure 4, superimposed with a very small amplitude sine wave. The sine wave perturbation causes no adverse effects to the output as long as the frequency is not near the natural frequency of the system.

Experimental results from motion perturbation are shown in Figures 16 and 17. Figure 16 can be compared with figure 5 to show the improvement of the sensor output with motion perturbation. The z-direction output is similar for both moves, but in the

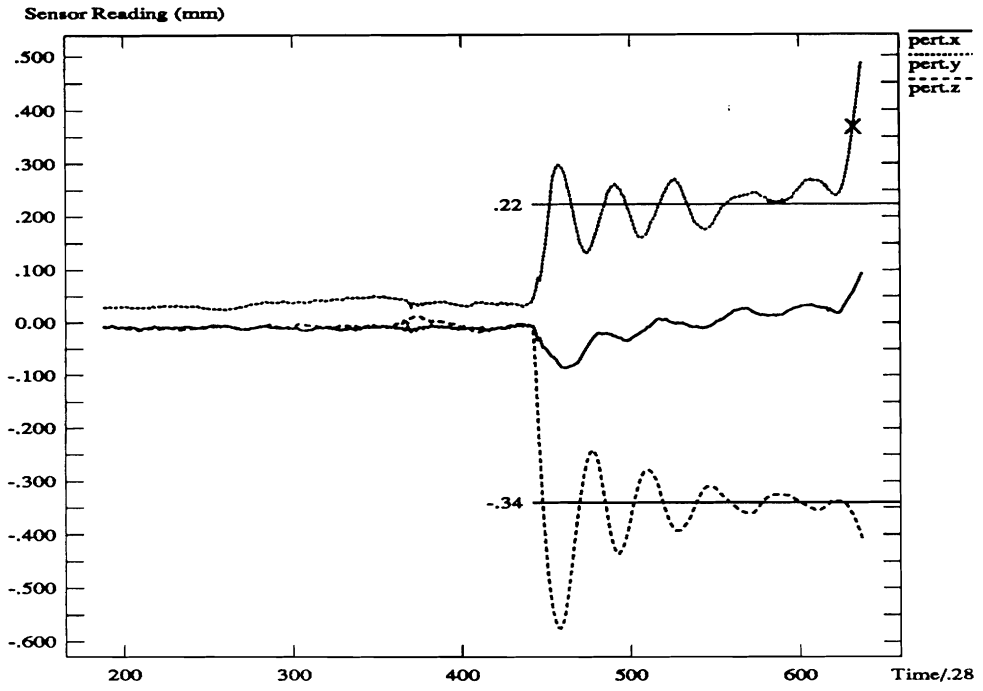


Figure 16: Typical Move With Motion Perturbation

perturbed motion move, the x-direction (normal to motion) output remains close to zero until the wall is encountered. Further, the y-direction (motion direction) output has the characteristics of a 2nd order system. As the motion in the y-direction begins, the sensor output rises to a peak value, and then oscillates about a (in this case never reached) steady state value. Contact with the wall is indicated with a distinct rise in the sensor output. Figure 17 compares multiple moves. The data spread for the steady state value of the output is much smaller than similar moves without perturbation. The point at which the box comes into contact with the wall (at the end of the data), can therefore be determined more accurately using a constant threshold.

With data as shown in figures 16 and 17, the use of a constant threshold value for determination of contact can be revised. If a move that is known to terminate in contact is long enough to create a model, contact with the wall can be determined by a data point that falls outside n standard deviations computed from data collected after the rise time of the move (see figure 17). A simple collision detection algorithm is shown in figure 18. The "X" in figure 16 indicates where a collision would have been detected using this algorithm, with $n = 3.5$. Figure 19 shows a closeup of the data from figure 17 where the wall is encountered. The "X" marks indicate where the wall would have been detected.

Two refinements to this algorithm can be made. The first is to retain an absolute maximum constant threshold, so that if the data readings are very noisy, or if an obstacle is encountered before an adequate model can be built, the robot can still stop on a given force. This would eliminate the possibility of damage to the robot and the environment. Second, the algorithm as it stands uses all of the data points after the rise time to compute the mean and standard deviation. This is computationally expensive. Computing the mean and standard deviation from only the previous N data points would be faster, and may lead to even better results.

Motion perturbation, while improving the performance of the system by reducing

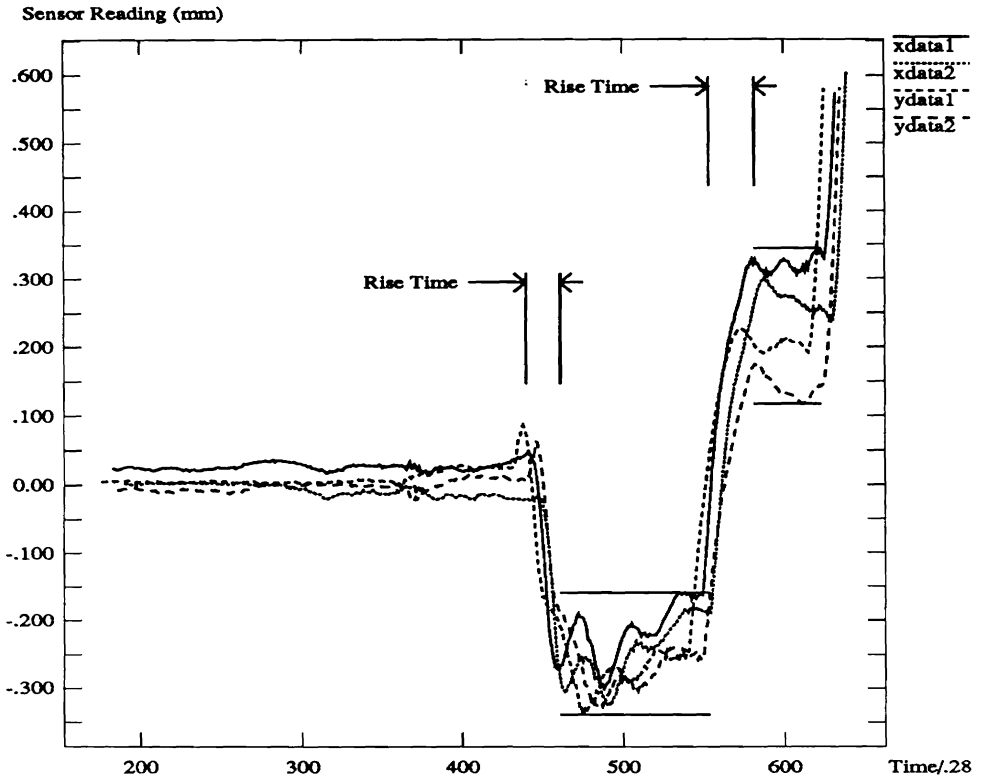


Figure 17: Experimental Data With Motion Perturbation

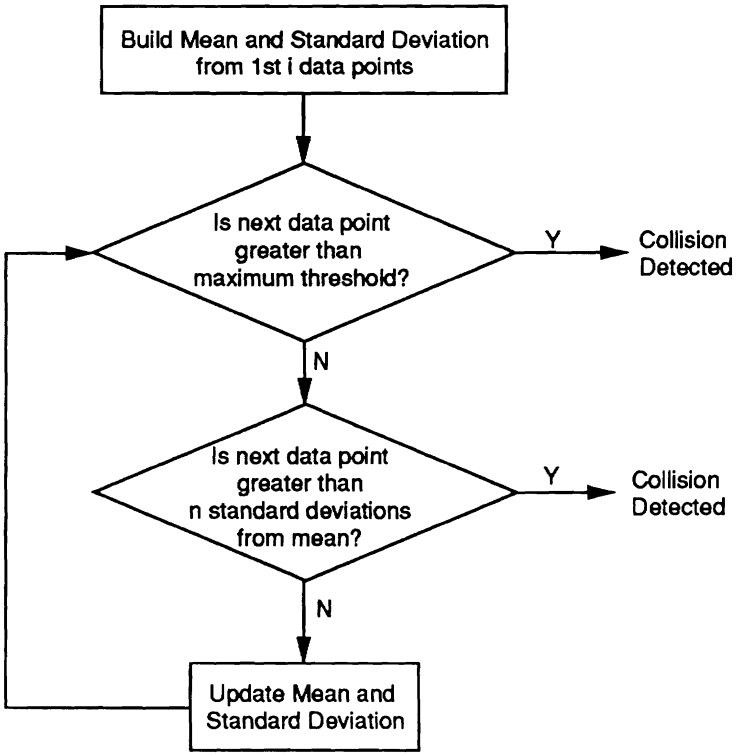


Figure 18: Collision Detection Algorithm

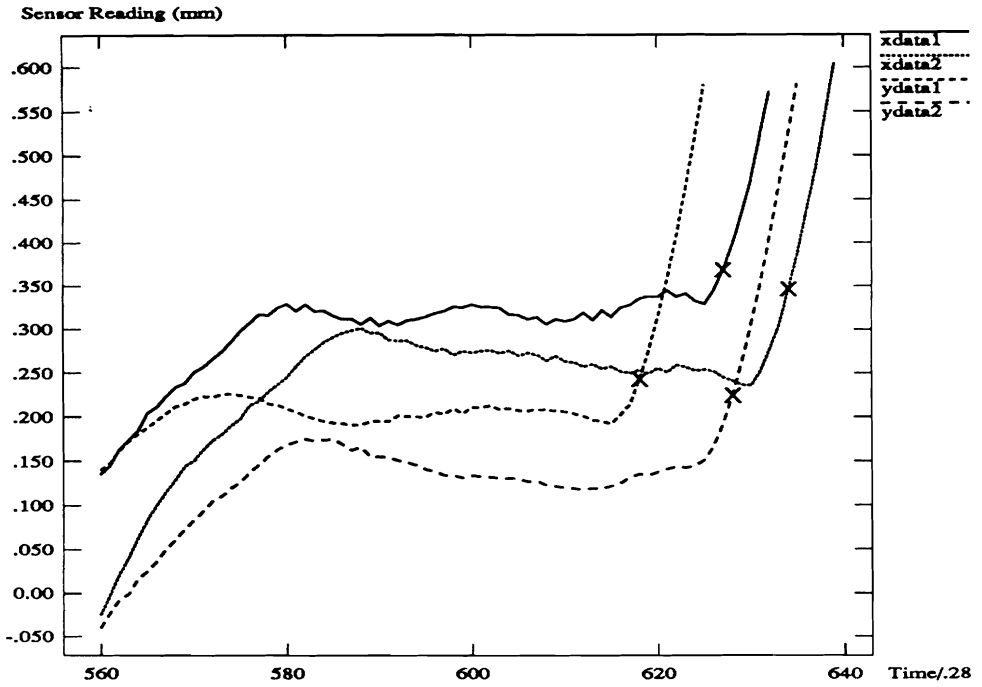


Figure 19: Magnified View of Collision Data

the effects of static friction, still does not overcome the "noise" associated with non-homogeneous friction. It does, however, produce sensor output that conforms well to a system model. The system can detect collisions sooner using the collision detection algorithm. This algorithm is also beneficial when the environment contains other surfaces with different coefficients of friction. A constant threshold based stopping criteria would not be able to adapt to these different conditions.

5.3 Exploratory Procedures

In some instances, surface conditions may impede command execution to the point that contact operations are impossible under the current model of the environment. An example of this would be trying to detect collision with a foam rubber wall while sliding across a very rough surface. In cases like this, it may become necessary for the remote site to autonomously explore surface conditions while the human operator waits.

A more refined model of the environment obviously leads to more accurate analysis of sensor data. The operator works in a model world dealing with kinematics only. While the slave manipulator operates in the real world, data about the environment can be gathered, analyzed, and used to refine new incoming data. Many surface attributes can be recovered through normal operation of the manipulator, including penetrability, hardness, compliance, compressibility, deformability, and surface roughness [13]. These criteria may be enough to refine the environment model to the point where contact operations can again be accomplished using the same types of commands from the master site as before. However, there may be surfaces where the current paradigm of contact operations cannot be used. At this point, the human operator must adapt the motion strategies to reflect the surface attributes. Instead of sliding along a surface to find a wall, for example, the operator may have to move above the surface, poking the surface occasionally to make

sure that "contact" has not been lost, until the wall is encountered.

6 Conclusions

Although the criterion for contact operations, including collision and error detection, appear to be simple, it is shown that using real world sensors and control, a much more robust set of rules must be used. By utilizing robust criterion for error detection, limited execution model commands can be successfully executed, and actual error states can be discerned from spurious data.

References

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

[2] Forrest T. Buzan. *Control of Telemanipulators with Time Delay: A Predictive Operator Aid with Force Feedback.* PhD thesis, Massachusetts Institute of Technology, 1989.

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

[4] Peter I. Corke. *A New Approach to Laboratory Motor Control: The Modular Motor Control System.* Technical Report, University of Pennsylvania, Philadelphia, PA, 1989.

[5] Janez Funda. *Teleprogramming: Towards Delay-Invariant Remote Manipulation.* PhD thesis, University of Pennsylvania, 1991.

[6] Blake Hannaford and Won S. Kim. Force reflection, shared control, and time delay in telemanipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 133–137, 1989.

[7] Vincent Hayward. *RCCL User's Manual.* Technical Report TR-EE-83-46, Purdue University, 1983.

[8] B. Hirzinger, J. Heindl, and K. Landzettel. Predictive and knowledge-based telerobotic control concepts. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1768–1777, 1989.

[9] 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.

[10] Randall K. Roberts, R. P. Paul, and Benjamin M. Hillberry. The effect of wrist force sensor stiffness on the control of robot manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 269–274, April 1985.

[11] P. S. Schenker and A. K. Bejczy. Workspace visualization and time delay in telerobotic operations. In *13th Annual AAS Guidance and Control Conference, Aerospace Human Factors Session*, February 1990. Keystone, CO.

[12] Thomas B. Sheridan. Telerobotics and human supervisory control. To be published as a book.

[13] Pramath R. Sinha, Yangsheng Xu, Ruzena Bajcsy, and Richard P. Paul. *Robotic Exploration of Surfaces With a Compliant Wrist Sensor.* Technical Report MS-CIS-90-92, GRASP LAB 244, University of Pennsylvania, Philadelphia, PA, 1990.

[14] Richard Volpe and Pradeep Khosla. Experimental verification of a strategy for impact control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1854–1860, 1991.

[15] Daniel E. Whitney. Quasi-static assembly for compliantly supported rigid parts. In M. Brady, J.M.Hollerback, T.L.Johnson, T.Lozano-Perez, and M.T.Mason, editors, *Robot Motion: Planning and Control*, pages 429–462, MIT Press, 1982.

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