INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Virtuality and reality of a near-optimal time-delayed teleoperator control system based on teleprogramming paradigm.

Damian Daniel Haule

B.Sc.Eng. (University of Dar Es Salaam, TANZANIA), 1986
M.Eng. (McGill University, Montreal, CANADA), 1990

McGill Research Center for Intelligent Machines
Electrical Engineering Department
McGill University
Montréal

December, 1996

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy (Ph.D.)

"In the name of God"

"Ad Majorem Dei Gloriam"

Abstract

The *teleprogramming control paradigm* is suggested as a means to efficiently perform teleoperation tasks in situations where the remote and local manipulator systems are connected via a low bandwidth delayed communications link. The effects of communication delays in the order of seconds can be reduced by building a *virtual reality* simulated model of the remote site with which the operator can interact to receive immediate quality feedback using a *Human-Machine Interface (HMI)* for *telepresence* applications. This concept overcomes the delay by transmitting not Cartesian or joint level informations in the form of *signals*, but rather *symbolic*, error tolerant, command program segments to the remote site. Symbolic instructions are send to the remote station every time the contact state changes or every second if no change of contact state has occurred.

Remote robotic systems are often very complex and difficult to operate, especially as multiple robots are integrated to accomplish difficult tasks in an unstructured or hazardous environments. In addition, training the operators is time-consuming and costly. A simulated virtual reality based system will provide a means by which operators can be trained to operate in an intuitive, and cost-effective way. Operator interaction with the remote system is at a high, task-oriented, level. Real-time state monitoring can prevent illegal robot actions and provides interactive feedback. A teleprogramming based simulator is essential for cost-effective *Teleoperator Interface & Training (TIT)* using *supervisory control* approach. An intelligent virtual interface is required which provides a rich means of presenting diagnostic and visual state information to the operator with reduced fatigue in real-time.

The *Mobile Servicing System (MSS) Operations and Training Simulator (MOTS)* will be used as a leading edge implementation of the teleprogramming concepts. MOTS provides high-fidelity, functional kinematic and dynamic software simulation of the MSS Space Segment in on-orbit configuration. MOTS is a real-time simulation environment of varying degrees of fidelity, along with an aggregate of software tools intended for the support of MSS space operations and training of crew and ground personnel. Primary interface to MOTS simulation models is through a *Common Data Base (CDB)* where telecommands are stored in a common shared memory. Hence, all telecommand data elements that are used to control the simulation modules are exported through the CDB by the *Human Computer Interface (HCI)* pages, hand controllers (H/C) and Display & Control (D&C) panel. Communication between simulation modules is achieved through the CDB in real-time.

Résumé

Un *paradigme de commande téléprogrammée* est proposé comme moyen d'effectuer efficacement des tâches de téléopération dans des situations où les systèmes de manipulation à distance et locaux sont reliés via une liaison de communications différées à basse largeur de bande. Les effets des délais de communication (de l'ordre de plusieurs secondes) peuvent être réduits par l'élaboration d'un modèle de simulation en réalité virtuelle de la station distante, avec laquelle l'opérateur peut intervenir pour recevoir une réaction immédiate efficace en utilisant une *Interface homme-machine* (IHM) pour des applications de *téléprésence*. Les délais sont éliminés par ce concept parce l'on transmet non pas des signaux contenant des données cartésiennes ou de position des articulations, mais plutôt des segments de programme de commande *symbolique* à tolérance d'erreur. Des instructions symboliques sont envoyées à la station chaque fois que l'état des contacts change, ou à chaque seconde s'il n'y a pas de changement dans ceux-ci.

Les systèmes robotiques télémétriques sont souvent très complexes et difficiles à opérer, surtout si de multiples robots sont intégrés pour accomplir des tâches difficiles dans des environnements non-structurés ou hasardeux. En outre, la formation des opérateurs est longue et coûteuse. Un système basé sur la réalité virtuelle est un moyen pour former les opérateurs de manière autodidacte avec un rendement plus efficace et moins coûteux. Les intéractions entre l'opérateur et le système à distance se font à un haut niveau. La commande en temps réel prévient des actions indésirables du robot et favorise les réactions interactives. Un simulateur téléprogrammé est essentiel pour une *Interface de formation téléopérateur* (IFT) efficace et moins coûteuse utilisant l'approche *contrôle de surveillance*. Une interface virtuelle intelligente procure un excellent moyen de présenter un diagnostique et des informations visuelles à l'opérateur avec une fatigue réduite en temps réel.

Le *simulateur d'opérations et de formation (SOF)* pour les *Systèmes mobiles de service (SMS)* sera utilisé comme système de fine pointe de téléprogrammation. Le SOF est un logiciel fonctionnel de simulation cinématique et dynamique fiable du SMS dans une configuration en orbite. Le SOF est une simulation en temps réel d'un environnement de degrés variables d'exactitude, assorti de logiciels prévus pour le support des opérations du SMS et pour la formation de l'équipage et du personnel au sol. L'interface primaire du modèle SOF se trouve dans une *base commune de données (BCD)* où les télécommandes sont stockées dans une mémoire commune. Donc, toutes les données de télécommande qui sont utilisées pour commander les modules de simulation sont exportées à travers la BCD à l'intérieur de l'interface conçue pour l'utilisateur, les commandes manuelles et le tableau d'affichage de commande. La communication entre les modules de simulation se fait à traver la BCD en temps réel.

Acknowledgements

Many people, in one way or another, have assisted me in preparing this dissertation. At the outset I would like to thank my mentor Prof. Alfred S. Malowany, who initiated the ideas developed in this work and is largely responsible for the intellectual content of this dissertation. His inexhaustible supply of ideas made sure that no hurdle proved fatal and his good humour and encouragement made me trust that there indeed was an end to the tunnel. Lastly, but not least, he was instrumental in influencing the organization of this dissertation.

I also wish to express my sincere gratitude and appreciation to Prof. Vincent Hayward and Prof. Hannah Michalska. They were very instrumental during research work and for taking their time to review my progress and offer their comments and suggestions. Several colleague students have helped me in writing programs, preparing data, laboratory experiments, picture output, carrying out relevant library research, etc. In this regard , I would like to thank especially Marco Petroni. It is my pleasure to give particular credit to all those who, in one way or another, have given their encouragement and dedicated effort toward this technical documentation. Moreover, the rest of the fellow at McGill research *Centre for Intelligent Machines (CIM)* and staff, who provided a fun and stimulating working environment while at McGill without forgetting to praise them for their help, suggestions and friendship. My sincere thanks also goes out to the CIM's systems staff, namely Jan Binder, Steve Robbins, and Mike Parker, who have kept the computers up and running through thick and thin, and who have been responsible for maintaining an excellent computing environment.

I would like also to thank the Department of Electrical Engineering, University of Dar es Salaam, Tanzania for granting me an educational leave of absence for further studies. The financial support of the *Institute for Robotics and Intelligent Systems (IRIS)* and McGill University is also gratefully acknowledged. In this regard, special thanks go to Greville Smith, the donor of the McGill Major Fellowship, from which I was a recipient for three years (i.e. 1992/95).

Finally, my sincere appreciations goes to CAE Electronics for allowing me to be a team member on MOTS project and sharing their favourable hardware and software environments for this research.

Damian Daniel Haule

Table of Contents

List of Figures

List of Tables

Chapter 1 Problem Statement in Telerobotics

1.1 Introduction on Telerobotics

Robotics is the computerized control of a mechanical system or machines which work in contact with its environment (e.g. a manipulator arm which performs *pick & place* motions). Its control is programmable and behaves in an *intelligent* way (i.e. the robot senses and reacts to changes in its environment) [Sheridan, 1989]. In the 1990's robots are leaving the factory floor to work in new areas such as the resource industries, the sub-sea and in space. Unlike industrial automation where the robot works in *its own* world, these environments are not structured, are sometimes dynamic, and cannot be completely modelled. It therefore becomes imperative to have a human operator present who can guide or supervise, rather than control the robot, using a *computer-assisted-interface (CAI)*. Work in this area of cooperative robot-operator interaction is called *telerobotics*, and the application to *intervention* environments is the focus of current researchers [Browse and Little, 1991, Grinstein and others, 1992, Haule and others, 1991, Giralt and others, 1991]. The emerging field of intervention robotics poses new technical and scientific challenges.

In the resource industries such as forestry and mining, in undersea operations and in space, the word environment is only partially known, has little structure, and may not be static. Hence, robot actions must be *sensor-based*, i.e. the execution of an action must be self-monitored, so that if the circumstances change, the action can be suspended or aborted or otherwise changed. Most robot manipulators have at least four dof's in order to provide independent control in position and orientation in 3D. In the context of telerobotics, the operator-robot interface may provide extra operational sophistication such as the possibility of storing the operator input and resulting robot motions in order to improve the performance or re-directing the operator input to a simulator for training purposes. Robotics had good success in industrial environments, while robots have failed to become the logi-

cal conclusion of the industrial revolution. This is due to the fact that factory automation is characterised by a structured, well-controlled, 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; 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.

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 labour. A parallel application of robotics has been in environments where people can not perform work themselves. These environments are hazardous and unstructured[1] or semi-structured[2]. Examples of such applications are performing work in areas of biological, chemical, or nuclear contamination, which is hazardous or detrimental to humans [Alami and others, 1990, Thayer and others, 1992]. 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 [Sayers *et al.*, 1992, Goforth and Dominy, 1988]. The latter class of applications is characterised 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 adequately model the complexities of real world and the possibility for on-line learning and performance improvement through *experience*.

The classical approach to tackle these problems has been to introduce problem solvers and expert systems as part of the *remote robotic workcell*[3] control system. However, such

[1] *Structured* environments are designed and engineered to somehow *cooperate* with the machine, i.e. known to the human operator.

[2] A *semi-structured* environment is one about which much - but not everything - is known *a priori*.

[3] A *Robotic Workcell* is a collection of robots, sensors, and other industrial equipment grouped in a cooperative environment to perform various complex tasks.

systems tend to be limited in scope and application domain in order to remain intellectually implementable. 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 environments and becomes hopeless in situations where the environment is only partially known and significant modelling, 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.

1.2 Master-Slave Mode Teleoperation

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 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. The early prototypes were essentially mechanical pantographic linkages of kinematically similar master & slave arms. Despite their simplicity, they provided for good kinesthetic remote control. 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 & forces) needed to be scaled upward or downward into the task domain so that the relative-motion control would be more natural to the operator.

The introduction of electrically actuated teleoperators under computer control removed the limitations of the mechanical linkages, but were unable to provide kinesthetic feedback to the operator. 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 [Schenker *et al.*, 1991, Hirzinger

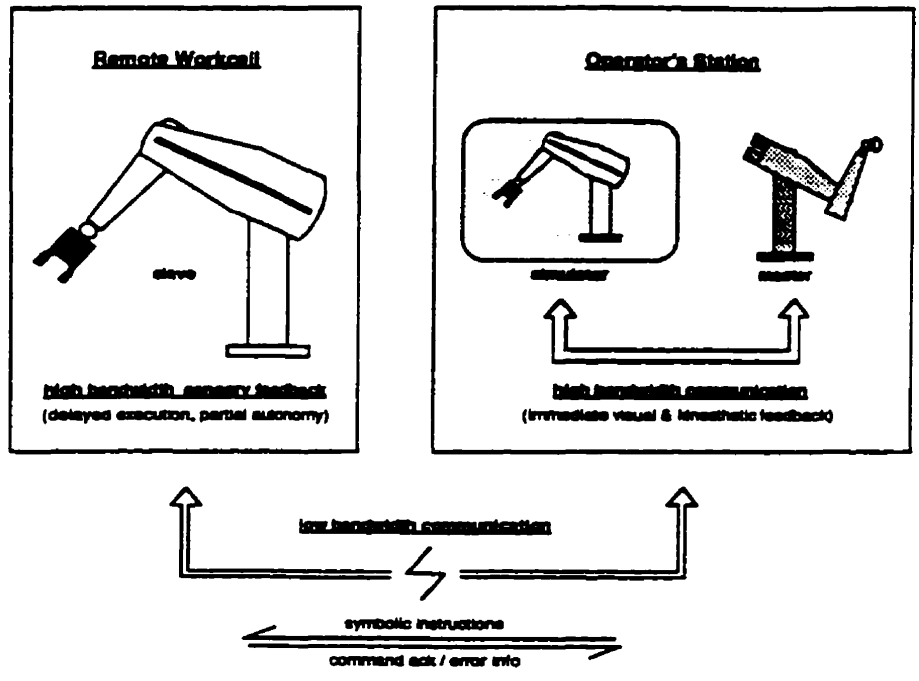


Figure 1.1: Overview of a classical teleoperator control system

and others, 1989, Grinstein and others, 1992]. 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 he is actively present in the remote environment [Adnan and Cheatham, 1992, Cole and others, 1991, Ince and others, 1991, Yamakita and others, 1992]. Fig. 1.1 shows an overview of a classical master-slave teleoperator control system.

1.3 Human Supervisory Control

Supervisory control refers to a human operator who is intermittently programming and con-

tinually receiving information from a computer that itself closes an autonomous control loop through artificial effectors and sensors to the controlled process or task environment [Sheridan, 1992]. This leads to a *virtual presence, virtual environments, artificial reality* or *virtual reality*, i.e. a feeling of being present in an environment other than the one the person is actually in. With sufficient good technology a person would not be able to discriminate among actual presence, telepresence and virtual presence. The current motivations to develop *supervisory control* systems are:

- to achieve the accuracy and reliability of the machine without sacrificing the cognitive capability and adaptability of the human;
- to make control faster and unconstrained by the limited pace of the continuous human sensorimotor capability;
- to make control easier by letting the operator give instructions in terms of objects to be moved and goals to be met;
- to eliminate the demand for continuous human attention and reduce the operator's workload;
- to make control possible even where there are time delays in communication between human and teleoperator;
- to provide a *fail-soft* capability when failure in the operator's direct control would prove catastrophic; and
- to save lives and reduce cost by eliminating the need for the operator to be present in hazardous environments and for life support required to send the operator there.

This list of motivations for supervisory control reflects the research *objectives*. The basic supervisory control paradigm is that the human operator provides largely *symbolic* commands (i.e. concatenations of typed symbols or specialized key presses) to the computer. However, some fraction of operator commands may be *analogic* (e.g. hand-control

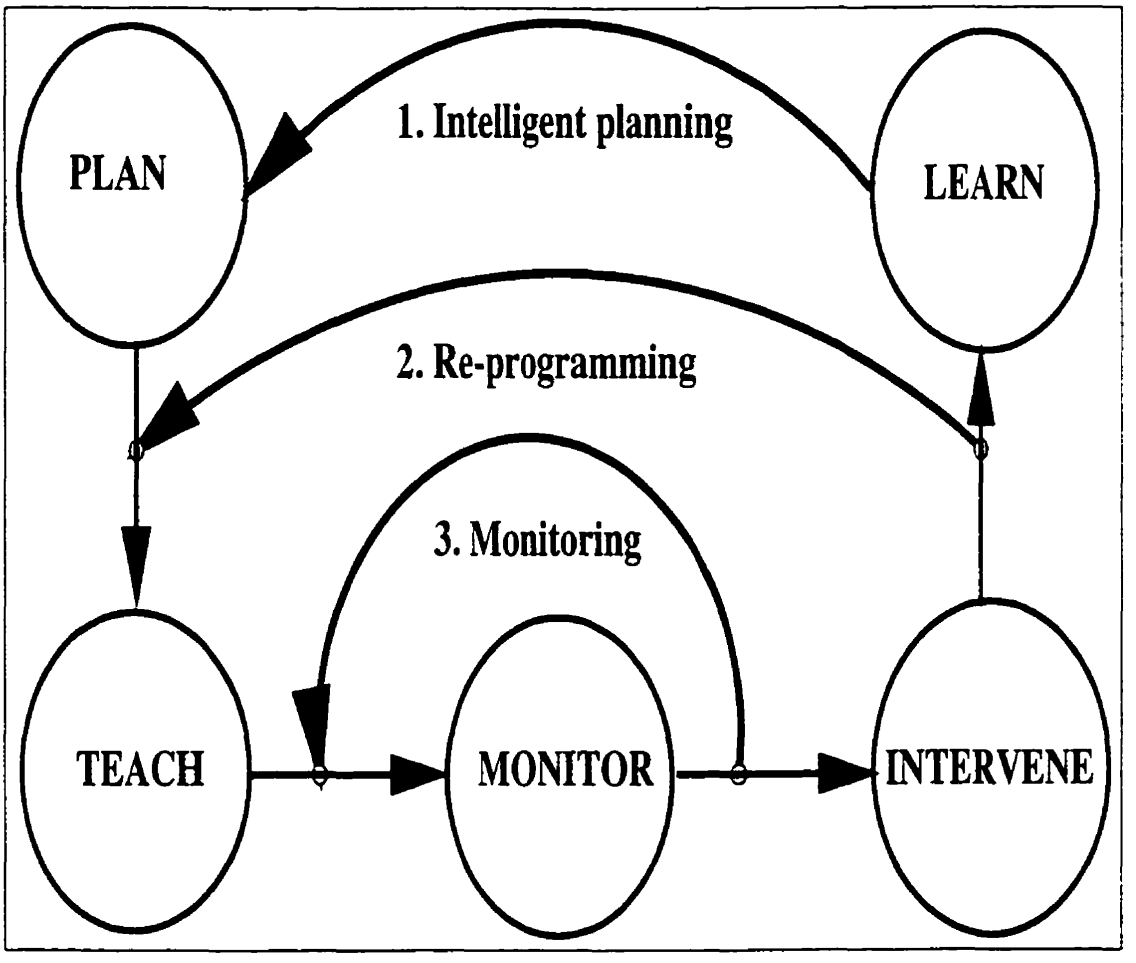


Figure 1.2: Five supervisor functions as nested loops

movements) in order to point to objects or otherwise demonstrate to the computer relationships that are difficult for the operator to put into symbols. The *local* or *human-interactive computer* thus should be human-friendly. Meanwhile, the subordinate *remote* or *task-interactive computer* that accompanies the controlled process must receive commands, translate them into executable strings of code, and perform the execution, closing each control loop through the appropriate actuators and sensors. [Sheridan, 1992] defined *five* generic supervisory functions of the human operator as follows:

1. **Planning:** planning what task to do and how to do it;
2. **Teaching:** teaching (or programming) the computer what was planned;
3. **Monitoring:** monitoring the automatic action to make sure all is going as planned and to detect failures;

4. **Intervening:** the supervisor supplements ongoing automatic control activities, takes over control entirely after the desired goal state has reached satisfactorily, or interrupts the automatic control in emergencies to specify a new goal state and re-program a new procedure; and

5. **Learning:** learning from experience so as to do better in the future.

These functions are usually time-sequential steps as nested loops. Fig. 1.2 shows these nested loops and there are *three* of them. The first loop (i.e. outermost one) is from *learning* back to *planning* where after the learning process, *intelligent* planning can be achieved. The second loop (i.e. middle one) is from *intervene* to *teach*. Here, *re-programming* can be done during intervention process. The third loop (i.e. innermost) is basically constant *monitoring* of the automatic control process.

1.4 Problem Statement in Teleoperation

Limiting factor in direct teleoperation is the communications link between the operator and slave manipulator. This is due to delayed communications and limited bandwidth channels. Communication delays have devastating effects on *task performance & telepresence* [Frank and others, 1988, Held and Durlach, 1991]. These tend to disorient teleoperators and dramatically decrease the operator's performance. Classical teleoperation assumes direct 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-Mars [Campbell, 1988, Wojcik, 1992, Smith and others, 1987, Haule and others, 1991]) or are forced to communicate over a limited bandwidth communication link (e.g. acoustic link to an underwater manipulator [Kotoku, 1992]).

Under such circumstances, both the instructions to the slave manipulator (i.e. desired velocities and forces) as well as the feedback from the slave back to the operator (i.e. 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 him until a communication delay later when the feedback arrives. A typical operator's response

under such circumstances is to adopt a *move-and-wait* strategy [Ferrell, 1965], 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. This is a typical manual control system which has devastating effects due to delays [Hess, 1984]. Moreover, these delays tend to increase overall mission costs during remote manipulations.

Communication delays affect task performance while doing teleoperation. Consider a situation with a one-way time delay τ which is due to the combination of transmission and other delays in the system. From now on, this lumped delay will be referred to as the *communication delay* or the *feedback delay*. Let T_{task} be a time to execute a given task without delay. By executing elementary commands, each of which takes on average time t to execute, then the total time to execute the same task in the delayed environment by using the *move-and-wait* approach is T_{total} [Ferrell, 1965]; where:

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

Fig. 1.3 illustrates the effect of communication delays on the total task completion time using the *move-and-wait* strategy for $t = 1$ sec. and five different values of the communication delay τ (i.e. $\tau = [0, 2, 5, 10, 15]$. For $\tau = 0$ corresponds to the case where there is no delay in the control loop at all, i.e. $T_{total} = T_{task}$. Hence, in view of Equation 1.1, consider a twenty minute task ($T_{task} = 20\ min.$), with an elementary command time of 1 sec. ($t = 1 sec.$) and with a feedback delay time of 10 sec. ($\tau = 10 sec.$), then the total time to execute the task would be 7 hours! Clearly this is NOT satisfactory.

Feedback delays can severely reduce the efficiency of task performance by forcing the operator to wait and can severely degrade (even destroy) the sense of *remote presence* during remote manipulation [Sheridan and Ferell, 1963, Adnan and Cheatham, 1992, Adams, 1962]. 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. Delay in receiving both visual and kinesthetic information causes a problem, however delays in receiving force information has been shown to be perceptually more significant.

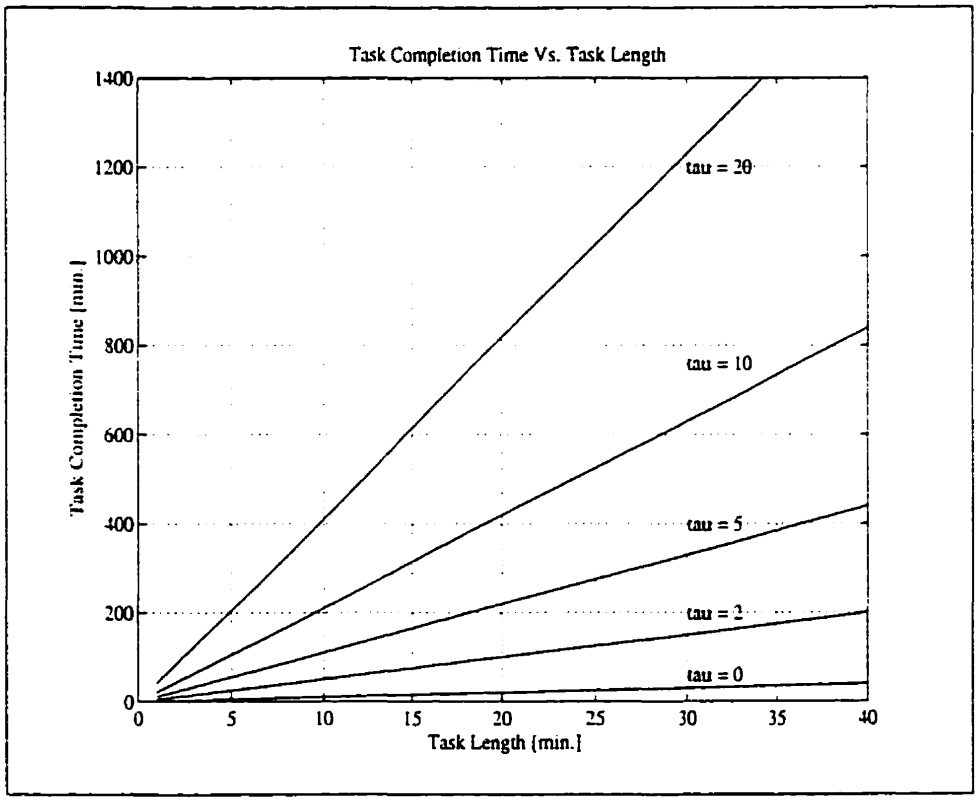


Figure 1.3: Effects of communication delays on task performance

Physiological studies have shown that the neurological control of human musculoskeletal movements operates at the rate of 5 Hz., and that a time delay of approximately 300ms ($\approx$ 1/3 sec.) is clearly perceptible and distracting to humans [Beil and Warrick, 1949, Smith, 1963]. Consequently, delays approaching 1s severely destabilize the performance of a human operator relying on real-time feedback information [Boff and others, 1986].

Unfortunately in space and undersea, communication delays often exceed the one second 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 [Bailey and others, 1987]. If teleoperated work is to be performed in shallow space (e.g. moon, mars), 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 [McMillan *et al.*, 1994, Sayers *et al.*, 1992]. 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, 1989].

1.5 Research Summary

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, to develop a delay-tolerant control strategy, which will allow for continuous and efficient control in *real-time* for most, if not all, earth-based, ocean-based, as well as shallow space telerobotic applications. This is accomplished by providing ways and means in order to have satisfactory quality sensory feedback to the operator.

According to basic control theory, sustained, stable closed-loop control in the presence of significant time delay is not possible [Sheridan, 1992]. However, various control strategies and ways of sharing the necessary control functions between the remote site and the local station in a remotely controlled robotic system are possible, which can dramatically improve the ability to perform useful and effective work over large distances. This research work, presents and demonstrates a solution to the problem, based on the concept of *teleprogramming paradigm* for the remote robotic workcell. By so doing, virtual environments are used to demonstrate the feasibility of the method. The virtual reality technology allows the operator to reach an ultimate in *immersion, interactivity & involvement* for potential ubiquity using many logical input devices for multi-media interface through a common shared memory where telecommands can be stored in the form of *Common Data Base (CDB)*. The CDB can be accessed in real-time by all simulation modules whenever needed. The operator can view and modify the CDB via a *Human Computer Interface (HCI)* pages.

Chapter 2: Gives a detailed study as a background framework towards solving the stated problem in different fields pertaining manual control during teleoperation. A lot of previ-

ous work on overcoming communication delays while enhancing operator performance and remote presence is reported. Moreover, advantages and disadvantages of these methods are pointed out so that new methods can take advantage of them in implementing new teleoperator control systems. It also gives an overview of remote motion analysis vital for the simulation purposes in terms of motion collision avoidance, classifications and restrictions.

Chapter 3: Presents the teleprogramming paradigm: a new delay-tolerant control strategy to be used in most telerobotic applications. Its concept is fully outlined which focuses on elementary symbolic command generation, transmission and interpretations from the local site to the remote site. Initial analytical evaluation of the method is also given.

Chapter 4: In order to understand the rationale of *Virtual Reality (VR)*, this chapter summarises the origin and evolution of VR by providing its potential usage and benefits in telerobotics applications. It also gives a survey of VR drawbacks and possible solutions for ultimate *ubiquity* at present and in the future. Moreover, it outlines the design concepts of general purpose VR systems to be used for *Teleoperator Interface & Training (TIT)*.

Chapter 5: Outlines the concept of symbolic elementary telecommand generation process. Different motion phases are identified in terms of *execution environments* and an algorithm to be used is provided. The semantics of a symbolic telecommand language is summarised. The transmission and interpretation mechanisms of these telecommands in terms of *parsing, translation & execution* is also outlined. This allows *meta-interaction* between the local (master) and the remote (slave) workcell. A management scheme for lag control using a *double-buffering* execution scheme is designed and proven satisfactory.

Chapter 6: A control scheme for robotic visual tracking is designed. The scheme is divided into two parts: a *predictor* and an observer-based *double-loop feedback* scheme. Its performance is later evaluated using computer simulations in frequency domain by minimising a squared-error cost function of the motion parameters.

Chapter 7: The flexibility of using *Common Data Base (CDB)* as a shared memory section to which all simulation modules have access at run-time is presented. Primary methods for interfacing with the simulation via CDB telecommand labels are introduced which include *hand controller (H/C), display & control (D&C) panel*, and *human-computer inter-*

face (HCI) pages. Process scheduling, management and simulation performance analysis is outlined using the available CDB-based software utilities such as *Distributed Interactive Simulation (DIS)* module.

Chapter 8: Summarizes the design of MOTS software in two categories: firstly simulation models and secondly tools to support execution and configuration management using CDB telecommand data elements or labels.

Chapter 9: Covers experimental analysis and results obtained. This includes validating the teleprogramming concept, visual tracking simulation results, and MOTS technical performance parameters based on CDB telecommand data elements.

Chapter 10: Summarizes main contributions of this research and suggests possible work for future improvement with concluding remarks. The dual usage of space-related technologies is also outlined from which civil-related programs can benefit.

Appendices: Cover all additional materials which are vital in understanding this dissertation. Most important, Appendix A gives a summary of the Manipulator Development and Simulation Facility (MDSF) which is used as the truth model or benchmark for real-time space robotic simulations.

Chapter 2 Background on Related Work

This chapter offers a brief review of related work in three main areas:

- Manual approaches used to overcome time delays during remote manipulation;
- Teleoperator human-machine real-time interfaces; and
- Automatic generation of robot control programs for hostile environments [Milgram and others, 1995].

2.1 Manual Control with Delays

Delayed visual feedback has long been known to affect the control input human beings make when they are an integral part of a system [Crane, 1984, Frank and others, 1988, Garvey and others, 1958, Hess, 1984]. The latency between system input and output forces human controllers to change how they maintain a desired system state. These changes have been documented, as have various schemes to compensate for a delay [Crane, 1983, Kim *et al.*, 1993, Leslie and others, 1966]. These developments span work on human tracking and the control of movement, vehicle simulation (e.g. flight training), robotics for manufacturing, space exploration and current work on virtual reality displays [Pausch *et al.*, 1992, Liang *et al.*, 1991]. Human input to systems where visual feedback about the results of the input is delayed has long been a topic of interest. There are many ways time can be lost in the real world. They may cover the time needed for communications to travel vast distances or they may come from the time expended on the calculations needed to define the response of a system and to display it graphically to a human operator. For experimentalists, delay of visual feedback has served as a reliable variable for human control performance [Archer and Namikas, 1958, Beil and Warrick, 1949].

Manual control and changes to it caused by the presence of a feedback delay can be thought of as determined partly by the dynamics of the controlled system, partly by the

size of the delay (or the human controller's ability to compensate for it) and partly by the requirements of the task being performed. A sizable literature about the control of systems with delays has accumulated and much of it is relevant to developments where human beings will operate either in artificial, computer-generated worlds or remotely through sensors [Deyo and others, 1988, Bryson and Fisher, 1990]. This work started in the late 1950s as studies of human tracking performance where individuals controlled single-axis systems with uncomplicated dynamic responses. The purpose was to investigate the acceptability of the first- and second-order lags of mechanical systems [Beil and Warrick, 1949, Garvey and others, 1958, Adams, 1961, Conklin, 1957]. At about that time also, the relatively long transmission times associated with remote space and planetary exploration stimulated interest in how well human controllers could accommodate delays of up to several seconds [Adams, 1962].

The 1970s brought computer image generation to flight simulation and while the delays involved were only on the order of 100 to 200 msec., they were long enough to produce demonstrable changes to the control behaviour of pilots. The wide field-of-view displays of flight simulators made apparent the behaviour of elements in the visual scene and many of the shortcuts taken in the simulation's mathematical model. Worry about the fidelity of visual or motion cues stimulated the development of software compensators for display delays. Indeed, much of the work measuring the effects of delayed visual feedback has been performed in the context of flight simulation because of the need for an update rate of at least 30 Hz. for the visual scene [Smith and Sarafian, 1986, Pausch *et al.*, 1992, Crane, 1984]. Current effort is to form networks of distributed simulators and the bandwidth of the communication between them forces delay to be a problem still. While interest in simulation fidelity still continues, work on delay compensation changed focus to vehicle control when significant computational delays were introduced by the development of digital *fly-by-wire* aircraft [Berry and others, 1982]. In addition, plans for a national space station included a remote manipulation capability so that astronauts could use a robotic teleoperation system for repair [Albus and others, 1986]. A teleoperator may be using an artificial visual display which may contain significant computation or network delays.

Current interest in graphical *environments* and the technology to allow people to interact with them (using head-mounted displays, position tracking devices, directional sound, gesture gloves and suits) faces many of the same problems encountered during the widespread development of simulation for flight training. Additional time or computational power is required to increase the resolution or complexity or number of images a display system can provide. The equivalence of time and computational resources is the trade-off that has maintained interest in the problems caused by delays.

2.2 Overcoming Communication Delays

The problem of communication delays has been recognised as one of the central areas of research in telerobotics for some time [Sheridan and Ferell, 1963, Stark and others, 1987, Sheridan, 1992]. Many researchers have proposed approaches to solve this problem. Ferrell proposed slowing down the motion so as to minimise the effect of the delay [Ferrell, 1965]. He also proposed strengthening the slave arm and the objects which it manipulates in order to avoid damage (e.g. underwater remotely operated vehicles (ROV's)). Finally, he proposed 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]. Another strong proposal was that of *"supervisory control"*: limited autonomy at the remote site, i.e. 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 and Sheridan, 1967]. [Hirzinger and others, 1989] suggested formally modelling up-link and down-link delays by augmenting the dynamic state-space model of the system (environment & slave) - delays are modelled as delay lines on the output and introduce additional number of states.

Other researchers proposed a control theoretic approach such as modelling a teleoperated system as a two-ported network, and devising control laws which attempt to cancel the effects of feedback delays [Anderson and Spong, 1988, Hannaford, 1989]. Another approach is that of using *predictive displays*[1] to allow the operator to *preview* the command effects on

[1] *Predictive display* is a graphical simulation of the remote workcell interacting with its environment.

the remote environment [Bejczy and others, 1990, Christensen and others, 1991]. Full autonomy at the remote site is another approach whereby automatic on-line sensing, sensory data interpretation, strategy generation, task and motion planning, execution monitoring, error detection, re-planning and recovery, etc. are being incorporated [Stein and Paul, 1992, Lindsay, 1992, Wakita and others, 1992]. Most of these approaches have proven to be not entirely satisfactory. For delays in excess of one second, simple *move-and-wait* strategies become impractical for most applications. However, 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 on *predictive displays* offers the best compromise between the desirable and the feasible.

Supervisory control [Lee and Lee, 1992, Sheridan, 1992] provides a broad conceptual framework for the design of effective telerobotic systems inspite of communication delays. The central idea is to distribute decision making and control between the operator's station and the slave workcell in favour 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 during 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 is the need to anticipate, detect and provide pre-programmed 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 for 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 workcell and its environment. These models are then graphically displayed to the operator, and the effects of operator's commands are computed in the simulated visual feedback. [Bejczy and others, 1990] used the so called *Phantom Robot* for predictive displays while doing teleoperation with time delays. State of the art predictive displays can synchronise and overlay real-time computer graphics with the incoming delayed video camera signal on the same physical display. [Kotoku, 1992] applied a force feedback to realize predictive displays of a remote manipulation system with transmission delays. [Merritt and Cole, 1991] applied stereo-scopic 3D cues in a rapid sequential positioning task for evaluating motion parallax for teleoperator displays.

Overcoming the communication delay problem has been one of the central issues in telerobotic technology. Refer to [Sheridan, 1993] for its intense review and prognosis of the problem. So far, a *teleprogramming* concept is thought to be a promising way to come over it [Funda and others, 1992, Kaczor *et al.*, 1993, Paul *et al.*, 1992]. In teleprogramming systems [Paul *et al.*, 1993], it is assumed that sufficient data has been received from the slave site so as to enable the construction of a model of the remote environment. The operator can interact both visually and kinesthetically with the simulated environment. The operator's actions are then transformed into a sequence of robot program commands which, when executed by the slave manipulator, seek to mimic the operator's actions after the delayed time. In this respect, in order for the the teleprogramming systems to realistically work well, the following two problems should basically be solved:

- how the master site automatically generates the sequence of robotic commands;
- what format of the commands is adequate for their execution of the remote slave.

[Funda and others, 1992] proposed a class of symbolic language based on the hybrid force/position model as a solution to the basic problems. [Simon and others, 1994] presented the approach, algorithms and processes to perform cross-country autonomous nav-

igation using the teleprogramming context for mobile robots. [Jimenez and others, 1993] used the teleprogramming concept to predict the dynamic behaviour of a planetary vehicle while designing it and its associated control mechanisms. [Giralt and others, 1993] considered using the teleprogramming concept for the class of intervention robots (e.g. Mars Rover) that have to perform tasks in remote environments which may impose painful, harsh or utterly dangerous conditions to humans. [Stein, 1993] discussed a case study for portability issues in any teleprogramming system with a behaviour-based controller [Stein and Paul, 1994]. [Lindsay and Paul, 1993] used the teleprogramming concept to simplify the operator's interaction with the manipulator/tool system using the adaptive sensing algorithm of the remote system. [Lee and Lee, 1994] presented a new method for designing an optimal time-delayed teleoperator control system based on *Smith's principle* for conventional teleprogramming. [Cho *et al.*, 1995] proposed a *discrete-event-based* teleprogramming system for enhancing the tolerance to the error condition due to the geometric uncertainty of the remote slave environment. A new planning and control methodology, which enables the system to overcome the geometric uncertainty problem as well as the communication delay problem was designed and verified. [Mitsuishi and others, 1995] described the construction of a tele-handling and tele-machining system at the macro and micro scales that uses the Internet for communication. A method to compensate for transmission time delays using the physical model of the object is also described. Moreover, [Stein and others, 1995] presented an experimental effort to validate the teleprogramming system using the Internet as the sole medium of communication. The experiment employed a supervisory control approach to time-delayed remote manipulation where an operator directs the actions of a semi-autonomous remote manipulator. Other researchers use intelligent monitoring systems [Wakita *et al.*, 1995] for limited communication path during overseas connections through the Internet.

2.3 Kinesthetic Feedback

Force reflection dramatically improves the sense of *teleperception* [Ferrell, 1966, 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.

It has been shown that kinesthetic feedback can be at least as important as 3-D visual information and that, in some circumstances, force feedback alone can be more valuable than visual feedback alone [Ouh-young *et al.*, 1989, Kotoku, 1992]. Communication delays preclude direct reflection of the reaction forces, experienced by the slave, to the operator's hand controller. Many studies have shown that 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. 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*. 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, 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 and linear laws have been proposed for 1-D force reflection [Fong *et al.*, 1986].

An interesting application for extracting force information from a graphical display was proposed by [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 visualisation alone. *Goodness of fit* is characterised by minimising 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. Current telerobotic systems use sophisticated and costly *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]. [Marapane and others, 1992b] used a real and virtual robot head for active vision research. In this research, they implemented a *Graphical Simulation and Animation (GSA)* environment for a 10 d.o.f. Robotic Research Head. The power and usefulness of the GSA system as a research tool was demonstrated by acquiring and analysing stereo images in the virtual world. Same researchers have applied the GSA environment for the design of flexible robotic structures [Marapane and others, 1992a]. They demonstrated its capabilities by simulating the behaviour of a flexible beam driven by a motor and controlled using a simple PD controller. To overcome the effects of delays, [Schebor and Turney, 1991] developed key components of a prototype forward simulation subsystem, the *Global-Local Environment Telerobotics Simulator (GLETS)* that buffers the operator from the remote task. GLETS totally immerses an operator in a real-time, interactive, simulated, visually updated artificial environment of the remote site. Using GLETS, the operator, will, in effect, enter into a telerobotic virtual reality and can easily form a gestalt of the *virtual local site*[2] that matches remote site interactions.

2.4 Automatic Robot Programming

Robots can be used to perform non-contact tasks such as inspection or surveillance. However, the majority of robotic manipulation tasks require that the robot physically interact with its environment. This 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. 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 and Craig, 1981], etc. The most popular of these control strategies is the *hybrid position/force* control method [Raibert and Craig, 1981]. This approach separates the robot's Cartesian d.o.f. of motion into force and position (velocity) controlled directions. Mason proposed a theoretical framework which can analyse the geometry of contact(s) be-

[2]The concept of *virtual reality* technology is fully covered in Chapter 4.

tween the robot and the environment and define mutually orthogonal *naturally constrained* and *artificially constrained* directions [Mason, 1981]. 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: *WAVE, AL, AUTOPASS, VAL,* etc. The target application for most of these languages were assembly problems in manufacturing and automation, and programs were designed either off-line or interactively through a step-by-step interpretative process. Recently, work has been done on at least partially automating the process of generating robot programs. [Grossman and Taylor, 1978] used the manipulator itself as a 3-D pointing device to interactively generate object models and automatically produce the corresponding object declarations for the *AL* language. [Asada and Izumi, 1987, Asada and 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 and Leysen, 1987, De Schutter and Van Brussel, 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 include: *Representational Formalisms & Formal Frameworks for Planning Strategies* [Sanderson and others, 1988, Hoffman, 1989, Lozano-Perez and Brooks, 1985]. Formal models for synthesizing com-

pliant 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 modelling errors [Donald, 1986, Jennings *et al.*, 1989]. Automatically generating assembly programs from design information by searching through a graph of contact formations [Desai and Volz, 1989].

One can note from the above efforts that automatic generation of robot programs in the presence of significant modelling, sensory and control errors is extremely difficult and in general, quite possibly unachievable. Typically, these methods analyse 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, 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 [Noorhosseini and Malowany, 1994a, Noorhosseini and Malowany, 1994b], 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) which limit the scope and usefulness of such schemes. Adaptive behaviour and on-line learning techniques are needed for successful autonomous planning, error detection and re-planning in the presence of uncertainties. Thus, the potential ultimate in automatic programming is to have *object-oriented* languages.

2.5 Programming by Human Demonstration

Different methods are presented by which robots can learn new tasks by monitoring the performance of a human operator. Programming a robot to behave in a desired manner is one of the most challenging tasks in robotics. Methods such as *teach-by-showing, textual programming, teleoperation & automatic programming* [Kang and Ikeuchi, 1993] have merits, but they can be either inconvenient or impractical in many situations. To remedy these problems, new approaches that combine elements of teleoperation and automatic programming were proposed by many researchers [Takahashi and Ogata, 1992, Pook and Ballard, 1993, Kuniyoshi *et al.*, 1992, Ikeuchi and Suehiro, 1994]. In these approaches, a robot learns ma-

nipulation skills by observing a human instructor performing tasks (e.g. assembling). By extracting relevant task knowledge from the observations, the robot can generate an equivalent action sequence. Although different implementations differ in the way by which learning occurs, they can be categorized by the realism experienced by the human instructor during the teaching process. In the system presented by [Takahashi and Ogata, 1992], the instructor teaches a robot in a virtual world, with no tactile feedback. In [Pook and Ballard, 1993], the human instructs the system by teleoperating an actual robot with no force feedback. These two methods direct real-time sensory feedback to the system, since the robot actively participates in the assembly process, either as a virtual robot or as a teleoperated robot, during the learning process. While various tasks may be programmed using these methods, they provide limited sensory reflection to the human operator. This makes it difficult for the human instructor to exercise hand-eye coordination during the teaching phase. In the systems of [Kuniyoshi *et al.*, 1992] and [Ikeuchi and Suehiro, 1994], the human directly manipulates the objects, so this difficulty due to the lack of human contact with the world does not occur. These systems employ visual sensors to observe the tasks being carried out by a human and deduce from these observations the state transitions of a given task.

In [Tung and Kak, 1995] a method in which a robot can learn new assembly tasks by monitoring the performance of a human operator wearing a *Data-Glove*[3] [Inc., 1993b] is presented. In particular, the system records the motions of the *Data-Glove* as it is used to manipulate actual objects and, by using geometric reasoning, deduces the assembly task that is being performed. Subsequently, the assembly steps are translated automatically by a task planner into a robot manipulation program. In experimental demonstrations of this system, the robot can now learn assembly tasks involving *pickups*, *put-downs* and various mating operations. Instead of using vision sensors to observe the human assembly performance, [Tung and Kak, 1995] employed a *Data-Glove* to obtain the trajectories of the manipulated objects in real-time. This allows for fast and efficient computation of the positions and orientations of the objects and provides a natural mechanism through which assembly operations can be deduced and then transformed into a robot manipulation program. The position

[3]A *Data-Glove* is a device that allows the user to measure the gestures of a human hand: it consists of a cloth glove with ten optical fibers attached to the back and passing over the finger joints.

and orientation of the hand in space is measured by using a *Polhemus 3Space* tracking sensor [Inc., 1993a], which is mounted on the back of the *Data-Glove*. [Onda and others, 1995] constructed a teaching system for assembly tasks using a force/position simulator which extracts a sequence of contact state transitions from the motion performed by the operator. This makes feasible to achieve an error-tolerant automated assembly motion and studies of the automatic assembly task system can progress based upon this information. Automatic programming [Lozano-Perez, 1981] requires no human intervention after the specification of the task, however there is overwhelming complexity of the subtasks. [Paul and Ikeuchi, 1995] and [Ikeuchi and Suehiro, 1994] presented the Assembly Plan from Observation (APO) system which observes a human operator perform an assembly task, analyzes the observations, models the task, and generates the programs for the robot to perform the same task. This method incorporates the simplicity of *teach-by-showing* and teleoperation.

In [Hirai and Asada, 1993], they used geometry to analyze the kinematics of contacts and then model robot assembly tasks using a contact state network. [Ikeuchi and Suehiro, 1994] proposed the concept of partitioning contact state space and using it to build task models for a multi-finger hand. Another area of related research is assembly motion planning. Work such as by [Lozano-Perez, 1981, Lozano-Perez *et al.*, 1984, Mason, 1981] has related the concept of compliant-guarded motions during assembly to paths traversing the faces of Cartesian-space obstacles. [Tso and Liu, 1995] described a method of generating robot program codes automatically from perception of human demonstration. The movement of a marker-based mechatronic input device representing the end-effector of the manipulator is driven by an operator, and recorded by a special visual measurement system. The captured spatial path is divided into different segments for individual processing. This robust system of task-specification capturing and understanding can be applied to the APO system of a robot arm. With the incorporation of a corresponding mechatronic input device, the method can be extended to robot-hand operation as well. Some of the typical high-level robot commands that can be achieved via programming by human demonstration are summarized in Table 2.1. Stanford University's Aerospace Robotics Laboratory has developed the *Task-Level Command (TLC)* architecture [Miles and Cannon, 1995] as a means for a human operator to direct highly autonomous robotic platforms with simple, intuitive com-

Command	Interpretation
OPEN	to open the gripper
CLOSE	to close the gripper
MOVE(x,y,z)	to translate the robot end-effector from the current position to the (x,y,z) position
ROTATE(*roll,pitch,yaw*)	to rotate the robot end-effector from the current orientation to the (*roll,pitch,yaw*) orientation
GRASP(*roll,pitch,yaw,d*)	to move the robot end-effector along the (*roll,pitch,yaw*) and out of the local origin direction for a distance *d*; then to close the gripper at the end
DEGRASP(*roll,pitch,yaw,d*)	to do the opposite of GRASP; i.e. to open the gripper at the beginning and to move the robot end-effector along the (*roll,pitch,yaw*) and towards the local origin direction for a distance *d*
PATH(*position-array*)	to control the robot movement so that the end-effector point will track the trajectory described by the (*position-array*) at regular intervals

Table 2.1: Typical high-level robot commands

mands. The TLC architecture is a hierarchical approach to robot control that enables an operator to graphically specify dynamics and sophisticated tasks through an interactive user interface that is updated in real-time. The low-level details of carrying out those tasks are handled autonomously by the robot, and therefore do not burden the operator as he is left free to concentrate on high-level issues. This novel control approach exploits the complementary capabilities of both robotic control and human decision-making to construct a powerful human/robot team operating in a semi-structured environment [Miles and Cannon, 1995]. In this approach, the human operator assists the robot in perceiving unexpected situations in the environment through simple *point-and-click* type interaction with a live video display from on-board cameras. This approach overcomes the challenges of semi-structured environments without sacrificing the high-degree of autonomy and resilience to time delay of the TLC architecture.

2.6 Telerobotic Simulation

The main problems in teleoperation include communication time delays and limited sensory feedback. These tend to disorient teleoperators and dramatically decrease the operator's performance during remote manipulations. A solution is to design a key component of a prototype forward simulation subsystem that buffers the operator from the remote task. An example of such solution is that of [Kim and Bejczy, 1991], where they developed high-fidelity real-time computer *Graphic Displays* for operator aid in telemanipulation. The roles of graphic displays in teleoperation include task visualisation, preview prior to actual execution and predictive displays under communication time delay. Graphic displays are also useful for task analysis, task planning and operator training. [Schebor and Turney, 1991] developed a *Global-Local Environment Telerobotic Simulator (GLETS)* which totally immerses an operator in a real-time, interactive, simulated, visually updated artificial environment of the remote telerobotic site. Using such simulators, the operator will, in effect, enter into a telerobotic virtual reality and can easily form a gestalt of the virtual *local site* that matches his normal interactions with the remote site. To simplify teleoperation, a designed telerobotic simulator will be able to: (i). simulate a remote manipulator and its local environment; (ii). continually update the simulated environment using remote visual sensors; (iii). manipulate the simulated environment with an easy-to-use, gesturally and voice controlled operator interface, which provides rich visual and audio cues; and (iv). provide a flexible object-oriented software architecture with underlying standard robotics algorithmic support, which results in software that is reusable, extensible, reliable and portable.

The effectiveness of a telerobotic system as a tool depends largely on the way the system is *interfaced* with the operator. Poor interfaces result into extremely *expensive* teleoperator training. The key to solving this problem is to make the interface to the simulator more *human-like* rather than requiring the operator to be more *machine-like*. A human-machine system that requires direct cooperation between the agents that take part in task execution is an example of a teleoperated work environment. Multi-agent cooperation can take place in two general modes (i.e. *interaction & interface*) as shown in model of Fig. 2.1. by [Ntuen and others, 1991]. Using a telerobotic simulator, the operator will, in effect, be able to in-

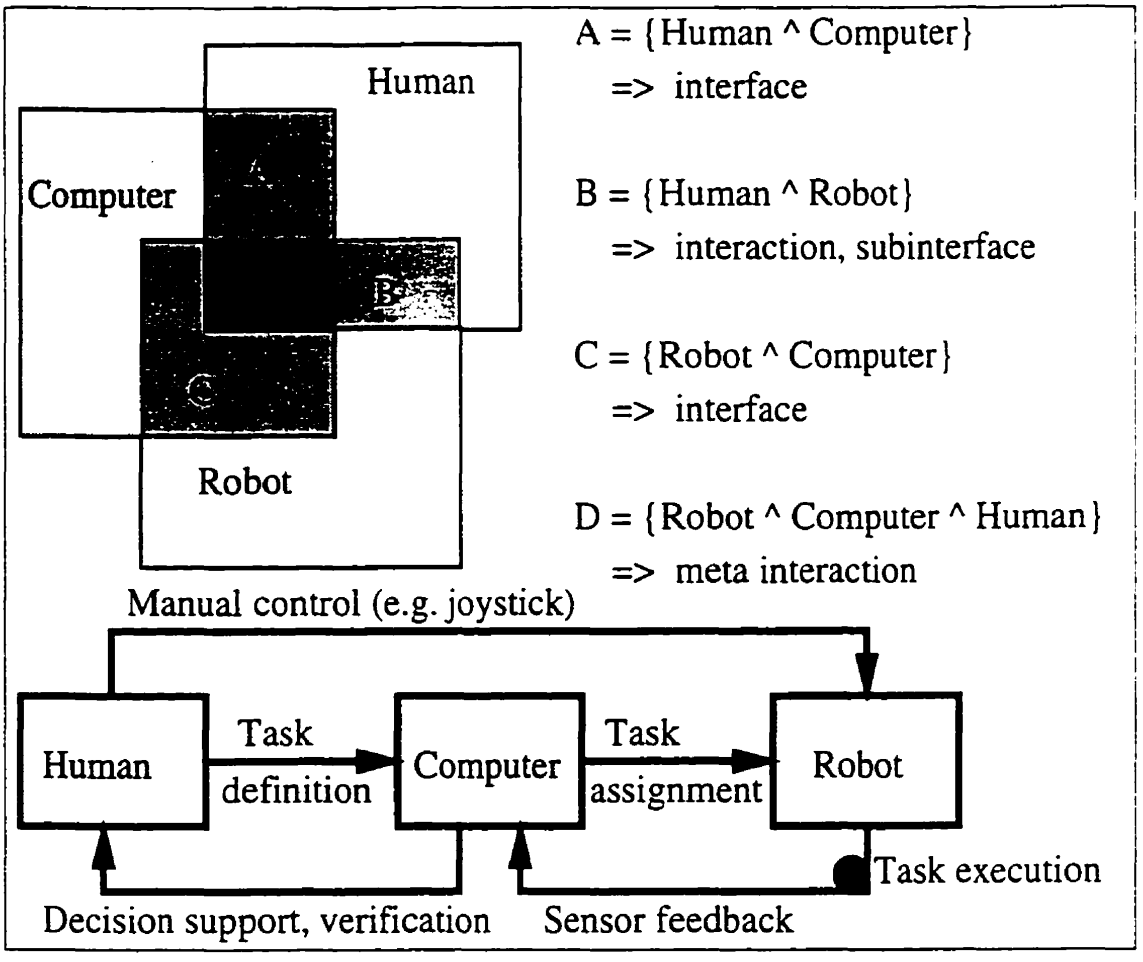


Figure 2.1: Human-machine interaction (interfacing) in a telerobotic system

teract with a telerobotic virtual reality and to form a gestalt of the virtual *local telerobotic site* that matches the operator interactions with the actual remote site. These capabilities can be provided by: (a) displays in the form of 3-D stereoscopic, shaded surface graphics and synthesised speech; (b) computer-linked glove that allows the system to monitor the position, orientation and finger configuration of an operator's hand, which permits the operator to control the end-effector. The glove also permits the hand to be used as a *pick & place* device on *virtual control panels*, or through the use of gestures, to provide complex instructions to the telerobotic simulator, such as *grasp, release, insert, open or expand*; and (c) speech synthesis and voice recognition which provides input of more abstract commands.

Voice input and output have contributed productive gains in graphical workstation environments. [Apostolos and others, 1992] have done experiments on benefits of using auditory cues for cuing operator manual control actions. The experiments were done on the sim-

ulated Solar Maximum Satellite Repair to examine a task of unbolting an electrical connector screw based on the apparent significance of auditory signals. A combination of *audio, vision & force* feedback gives the best performance. Unless a robot simulator can maintain both a correct and current world model of its environment, it cannot function in a telerobotic application. Thus, a software architecture has to be designed with visual update systems that allow the simulator to interact with a remote vision system for update of the world model. The visual update system serves two purposes: first to determine the minor discrepancies between the real and simulated world environments and to update the simulator when these discrepancies become too large to tolerate; second to locate objects that are completely lost by the simulator. The simulator's flexibility and extend-ability are very important, hence the need for *object-oriented programming* environments.

However, *kinematic* simulation of the motion of the remote and the manipulated objects is most simplified or favourable. Hence, the simulation 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. From now on, the movable portions of the remote environment, which are directly under operator's control, will be collectively referred to as the *Movable Object (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 virtual model through the available environment updating mechanisms rather than direct simulation. This applies to the dynamic changes caused by the slave, as well as those produced by external environmental agents (e.g. winds, water currents). While kinematic simulation may seem restrictive, it is the most practical approach since only approximate information about the world is available, no complete information about the *masses, inertias, frictional properties*, etc. of the objects in the environment is available. In addition, the unmodelable and unpredictable external agents may significantly affect the dynamic state of the world, further diminishing the utility of a *dynamic* simulation. [Bejczy and others, 1990] discussed whether partial or full dynamic simulation has to be incorporated in the original conceptual design of the teleoperator control system for an unstructured environment.

Motion Mode	Classification
Free Space Motion	**Free motion** (rotations & translations) **Translation** (fixed orientations) **Rotation** (fixed position)
Contact Motion	**Sliding** (fixed orientation) **Pivoting** (fixed position) **Pushing**

Table 2.2: Motion mode classifications

2.7 Motion Mode Classifications

A teleprogramming system should offer the operator control over a wide range of slave workcell motions both in free space (i.e. while approaching/leaving the work area) and in contact with the surroundings (i.e. while performing the work). Moreover, the operator should be able to select different subsets of the physically realizable motion, which would allow him 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. To simplify general motion for both the operator and the slave robot, a natural way is to separate rotations and translations whenever possible [Funda and others, 1992]. This is crucial in contact motion, as the contact point is normally physically removed from the wrist-based reference location ($\mathcal{F}_w$) where motion is commanded. This separation gives rise to a remote compliance centre 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 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. Table 2.2 summarizes a set of elementary classes of motion mode classification.

Given a set of elementary motion modes, a mechanism to switch between them needs to be designed. In order to minimise 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, 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. During free space motion, the system should offer the operator the maximum possible maneuver-ability. At the same time it should aid the operator preserve positional/orientational parameters that he 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 him 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.

As shown in Table 2.2, there are three contact motion modes. 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 he clearly indicates such intent. This aids the operator in preserving high-order contacts (which are presumed preferred), while still allowing him to transition to an arbitrary adjacent contact. The second mode of contact motion is **pivoting** whereby the operator can adjust the orientation of MO or transition between adjacent contacts by rotating or pivoting about the contact point. In this mode the contact point is not allowed to slide along or depart from the supporting environment contact feature. As the contact type 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 re-orienting and contact changing motions of MO.

Moreover, 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 is **pushing**. It is provided to support a rudimentary pushing capability. Predicting the exact outcome of pushing motions in actual situations is extremely difficult. This is because the motion of a pushed object critically depends on the complex interactions between the microscopic features of the two sliding surfaces [Mason, 1981]. 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 [Funda, 1991] and [Haule and Malowany, 1995a] described how the teleprogramming system detects collisions, examines contacts and maintains the resulting collection of all currently active contacts as the contact set C. Associated with each contact are one or more mutually orthogonal Cartesian constraints on the relative motion of the contacting objects [Mason, 1981]. The number of resulting motion constraints is a function of the geometric contact type and physical properties of the contacting surfaces (e.g. friction). For situations where multiple contacts define the current contact state between two objects, one must 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. In this case, one needs to define a set of orthogonal reference coordinates and project the constraints associated with each contact into the common reference coordinate frame to obtain the *constraint set* S. Any subsequent motion imparted on an object, whose contact set is given by C, is thus constrained by S. In this case, the objects are the MO and the environment. For each contact configuration a *restriction frame* $\mathcal{F}_R$[4] is defined. For single contact configuration, the commanded motion, appropriately mapped into the restriction frame, will be restricted

[4]A *restriction frame* $\mathcal{F}_R$ 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 S.

with respect to the only constraint $\{c_i\} = C$. For MO/environment configurations with contact multiplicity greater than 1, on the other hand, the constraints associated with the contacts $c_i \in C$ will be mapped into this restriction frame and the commanded motion will be restricted based on the resulting orthogonal set of motion constraints. Thus, a motion restriction is needed for each mode during object motions with single- or multi-contacts.

Chapter 3 The Teleprogramming Methodology

3.1 Introduction

There is clearly an established and growing need to perform work in remote environments which are unreachable or unsafe for humans as discussed in Chapters 1 & 2. A purely autonomous manipulative capability would provide the solution to the communication delay and real-time feedback problem. However, its realization remains beyond the state of the art in modern robotics. On the other hand, direct teleoperation using the *move-and-wait* strategy in the presence of feedback delays degrades task performance and sense of telepresence. This research proposes to solve the problem of time-delayed remote manipulation by using a new control methodology, based on incremental *teleprogramming* of the *remote workcell*. 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. [Giralt and others, 1989] proposed a high-level view of teleprogramming system shown in Fig. 3.1.

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 between the operator and the simulated task environment. The immediate visual feedback provides a strong sense of *teleperception* while immersed in virtual environments. As the operator performs the task in the virtual model, the system continuously monitors the oper-

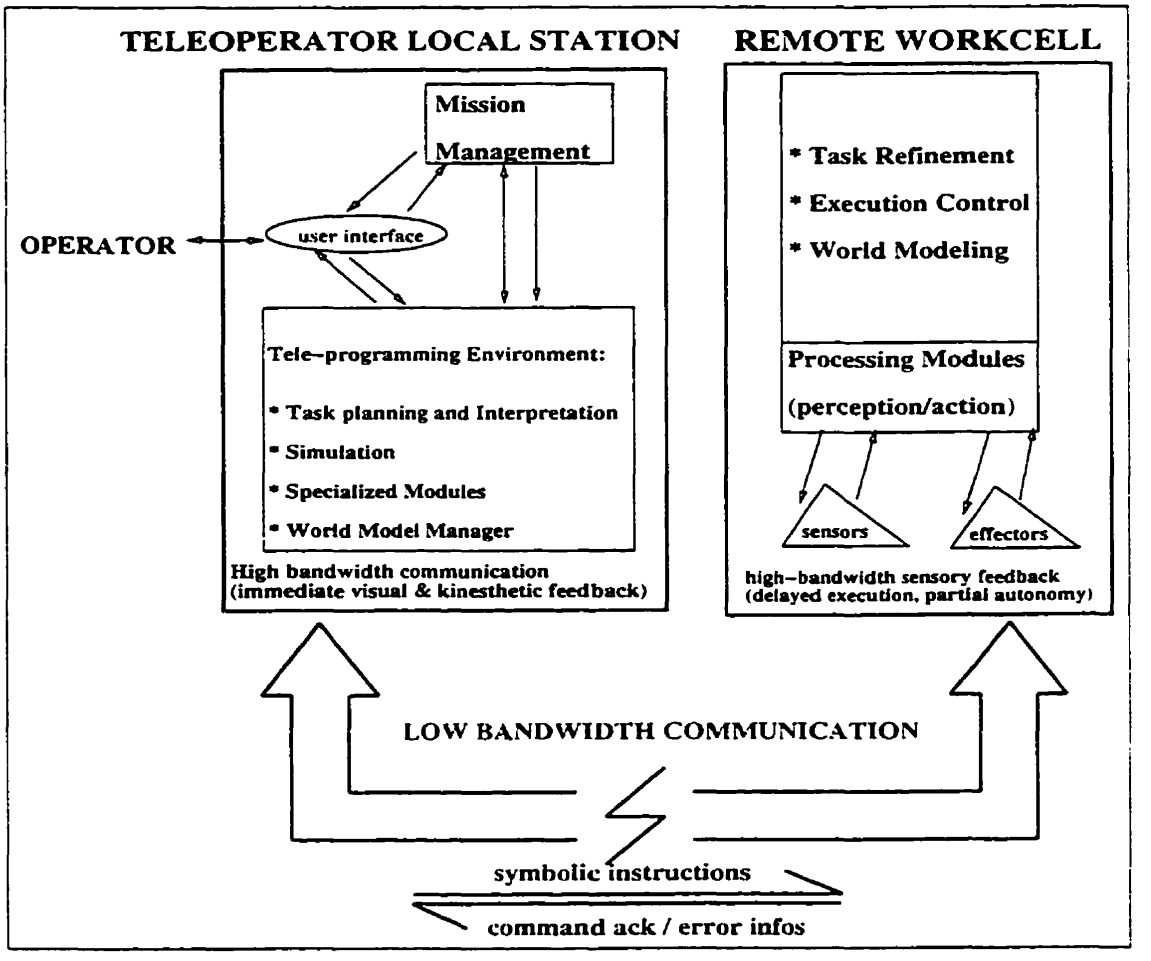


Figure 3.1: High-level view of a teleprogramming system

ator's actions and generates a stream of symbolic robot instructions, capturing all essential features of the task in progress. The action interpretation and telecommand 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 real world and the virtual model. The instructions are generated automatically, on-line, as the task progresses and are sent to the remote site 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 modelling, 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. This information is used to alert the operator about the error

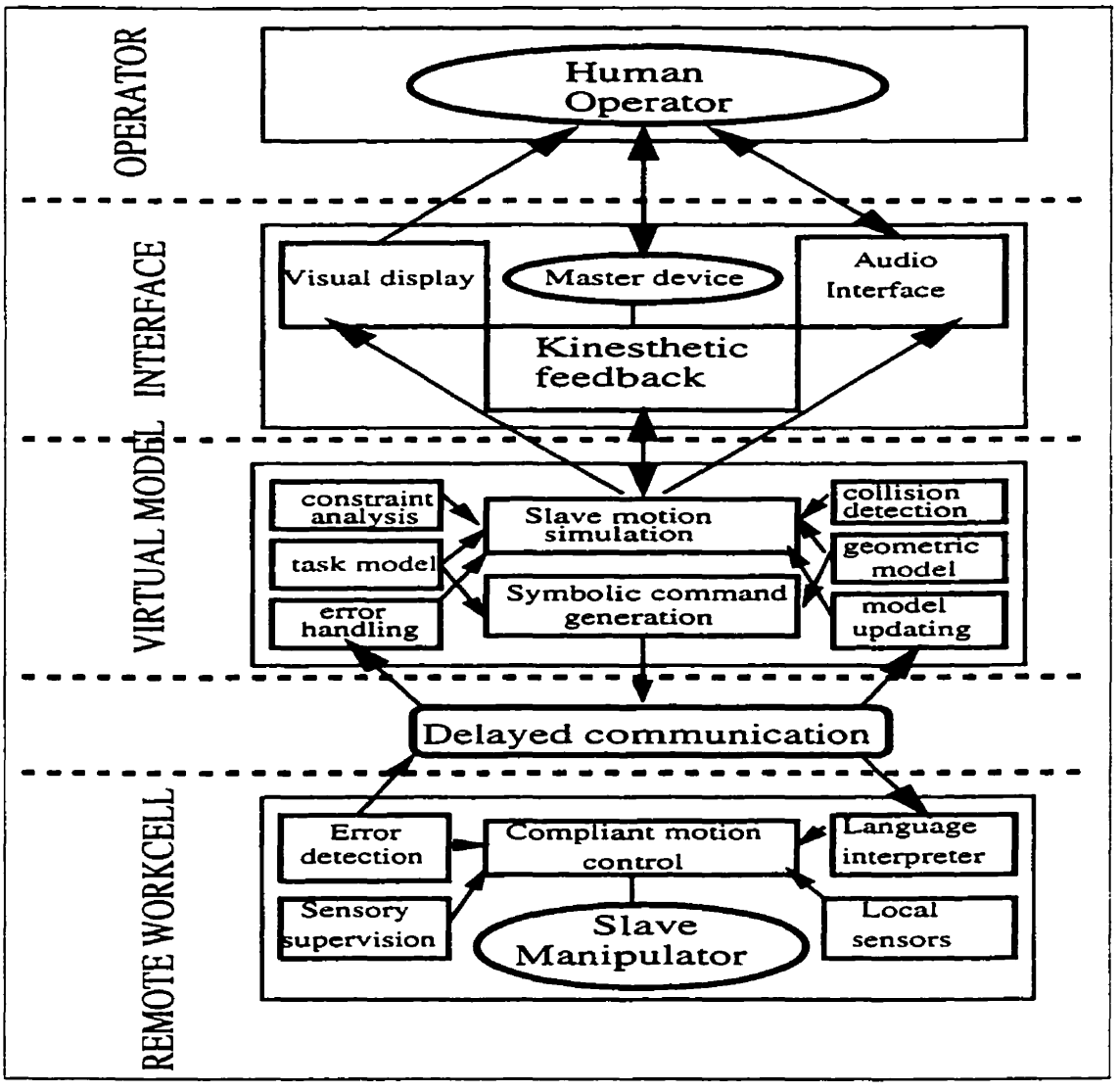


Figure 3.2: Conceptual organisation of a teleprogramming system

condition, properly adjust and update the virtual model, and allow the operator to specify the necessary corrective actions and proceed with the task. An overview of a more detailed *teleprogramming control paradigm* is as shown in Fig. 3.2 which illustrates all major components of the conceptual system architecture and indicates the basic inter-relationships.

3.2 Virtual World Model

The assumption in this work is 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 virtual 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 3-D objects [Bolle and 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 virtual model is constructed only once at the beginning of a *teleprogramming* session, it is proposed 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. 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 models, etc. By augmenting this virtual world with a corresponding model of the manipulator workcell itself, a real representation of the remote environment can be obtained, which can be displayed, animated and manipulated in real-time using standard input devices in virtual environments. 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 virtual world model. This information will be later used by the symbolic telecommand generation module as well as by the on-line model refinement process.

After constructing a 3-D virtual model of the remote workcell and its environment, it becomes possible for the operator to interact with this simulated world and specify tasks to be performed by the actual remote workcell. With this goal in mind, input devices can be

interfaced to the graphical display, allowing the operator to control positional and orientational parameters of the simulated workcell. While operating in an 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 modelling effects such as hydrodynamics and buoyancy in underwater applications, suggests that a non-dynamic, kinematic simulation of the remote environment including the slave robot is to be employed. The role of a virtual simulator in a *teleprogramming* system is to provide a real-time, realistic graphical animation of the slave workcell operating in the simulated environment under the 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 are all other collisions which normally involve some non-effector part of the slave robot and an environment obstacle.

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, as well as by the telecommand generation process. 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. 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 done 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. This constraint set is also used to provide the operator with real-time sense of kinesthetic interaction with the environment.

3.3 Generation of Elementary Symbolic Telecommands

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. 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. This is not critical during free space motion but it is vitally important when attempting to establish or maintain contact with the environment. 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 instruction 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 recognise and correctly interpret the operator's intent to initiate special-purpose subtasks such as 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 unparameterised, unnamed, one-time *procedure* and be executed repeatedly until some terminating condition is reached. The decision-level support, allowing the telecommand generation module to correctly disambiguate and interpret operator's input, is provided by the task model. Some knowledge of the general goal of the task in progress is required in order for the simulator software to correctly interpret the operator's actions 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 & 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 virtual simulator to efficiently manage the collision computation load. Similarly, an indication of the relevant relationships between environmental objects, involved in the the execution of the task (e.g. which objects are rigidly attached to their support, which ones are detachable, etc.), can be used by the simulator and the telecommand generation process.

The task model should encode the knowledge of the special-purpose actions and iterative procedures, as well as their associated terminating conditions. This information can then guide the telecommand 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 so as to provide the operator an unoccluded and intuitive view of the work area throughout the execution of the task. This task information can be gathered either by using a pre-prepared *Common Data Base (CDB)* in a shared memory, 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, these approaches 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. A detailed method on how symbolic telecommands can be generated, transmitted and executed is covered in Chapters 5 through 7.

3.4 The Remote Robotic Workcell

The operator's station sends the symbolic instructions to the *remote robotic workcell (RRW)* 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 telecommand 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. The RRW must be capable of some autonomy in executing the commanded motion primitives. In order to support its expected degree of autonomy, the RRW needs to be equipped with sufficient sensing capability to carry out elementary motion telecommands 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 [Raibert and Craig, 1981], 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 the 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 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. 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 RRW.

During task performance, the slave workcell must monitor its execution status, verifying that elementary motions terminate correctly or identifying that an execution error has oc-

curred. In either case, this information should be propagated to the operator's virtual model to report the status of the RRW. While everything can be 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 synthesised voice warnings, etc. as organised in Fig. 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 maneuver at the remote site to clarify the resulting state of the RRW. 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 virtual 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.

3.5 Error Handling and Recovery

The symbolic instructions arriving at the remote 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 to account for the estimated uncertainties in the modelling, 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 remote 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 scan-

ners, force sensors, etc. At minimum, the remote 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. For real-time meta-interactions with the remote workcell, data can be computed and stored in a shared memory using the CDB utility as covered in Chapter 7.

The low-level contact motions consist of guarded and compliant moves with built-in estimated modelling errors. This, along with the control algorithm should provide for stable and reliable execution of the commanded motions at the remote site. However, things may still go wrong as already mentioned. 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 remote workcell 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 to confirm an error condition and aid the system in gathering relevant information about the error state. Upon detecting an error, the remote workcell must respond in a manner that minimises the possibility of damage to itself, as well as to environmental objects. If relatively small and static unexpected forces are encountered, the remote controller may choose to stop and maintain the current position until the operator can resolve the situation. Alternatively, the remote workcell 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 remote workcell 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 remote site $\eta + \tau$ seconds later (where η is the lag time as discussed in Section 5.5). During this time, the operator had continued with the task and possibly modified the simulated environment. The error packet arriving from the remote 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, the error information can also provide local corrections to the operator's station based virtual 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 virtual model. Once the operator has determined the cause of the error, he can specify corrective actions to recover from the error and continue with the task. 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.

3.6 Pre-evaluation of Teleprogramming Paradigm

The *teleprogramming control methodology*, as outlined in this chapter, distributes decision-making and control between the human operator (who provides for task planning and error recovery) and the RRW control system (which provides for low-level autonomous execution and control, as well as error state identification). Within this paradigm, telecommands 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 telecommands under local sensory supervision a communication delay τ later. Therefore, the operator doesn't wait for explicit feedback from the remote site following each elementary telecommand. When an error occurs, the remote control system stops the robot and alerts the operator. The operator then re-plans from this point, once again starting a stream of telecommands to be executed autonomously by the slave. In view of earlier discussion on the total task completion times

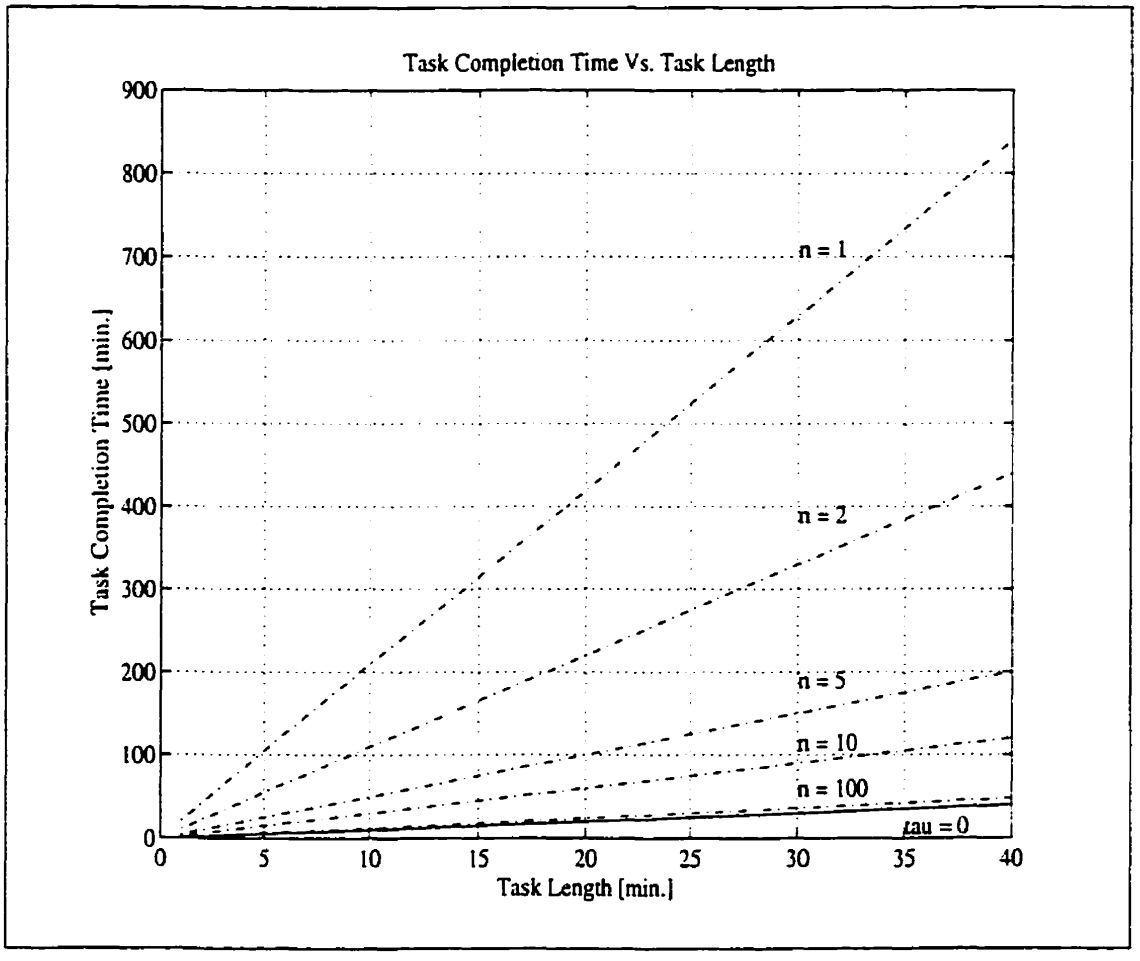


Figure 3.3: Effects of n telecommand segments on task performance

using the *move-and-wait* strategy (Section 1.4), one can now compute or evaluate the corresponding behaviour for the teleprogramming paradigm. If n is the number of elementary symbolic telecommands that are executed by the slave workcell on average, without entering an error state, then the total time to perform a given task is given by T'_{total}[1]; where:-

$$T'_{total} = (1 + 2\frac{\tau}{nt})T_{task} \tag{3.1}$$

Clearly, in the interest of minimising the overall completion time, Equation 3.1 suggests that the desired behaviour or condition of the teleprogramming system must be:-

[1]Other parameters are as defined before in Section 1.4.

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

Fig. 3.3 illustrates total completion times (T'_{total}) versus task length (T_{task}) for τ = 10 sec., t = 1 sec. and five different values of n, i.e. $n = [1, 2, 5, 10, 100]$. Note that for the case of $n = 1$, it corresponds to *move-and-wait* strategy and the solid line on the very bottom corresponds to direct teleoperation with no communication delay (i.e. $T'_{total} = T_{task}$). Fig. 3.3 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 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 $\tau \leq 10$ sec., and so $nt \leq 100$ sec. is the acceptable level of remote site autonomy. This is clearly within the state of the art of modern robot control strategies. Thus, the *teleprogramming control methodology* 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 RRWs. Ideas covered in this chapter will later be augmented with VR technology to come up with a real-time teleoperator control system. Finally, it can be shown that for $nt = 100$, the delay for a given task is only 1% of the delay when $nt = 1$ (i.e. 99% time savings as proven in Section 9.3). Thus, nt will be the performance determining factor of a teleprogramming system in place as it dictates the level of autonomy.

Chapter 4 Virtual Reality and Telerobotics: an overview

4.1 Introduction & General Overview on VR

Virtual Reality (VR) is the realm of the technology that facilitates the operation of complex systems, consuming information and turning it into knowledge (that most valuable of human resources). Although the terms cyberspace and VR have been around for years, VR as an industry is in its infancy. [Sutherland, 1965] demonstrated the first head-mounted stereo display in 1965. VR takes a fresh look at human interaction which evolves from user interface design, visual simulation and telepresence technologies. VR is unique in its emphasis on the *experience* of the human participant. Thus, VR focuses the user's attention on the experience as its quality is crucial. To stimulate creativity and productivity, the virtual experience must be credible. The *reality* must both react to the human participants in physically and perceptually appropriate ways, and conform to their personal cognitive representations of the micro-world in which they are engrossed. The experience does not necessarily have to be realistic - just consistent. The essence of what VR is and will be, is defined within three basic ideas taken together, i.e. *immersion, interactivity & involvement* [Morie, 1994]. The unique aspect of VR is that all three can exist at the same time. VR involves the creation and experience of environments. Its central objective is to place the participant in an environment that is not normally or easily experienced. This objective is satisfied by establishing a relationship between the participant and the created environment. Accordingly, a three-tiered definition of VR, shown in Fig. 4.1, addresses respectively what VR is, how it is accomplished and its effect - both of VR on the participant and of the participant on the environment. The distinguishing *what* of VR is its extension of the *human-computer interface* [Boff and others, 1986].

Virtual environment displays are interactive, head-referenced computer displays that give users the illusion of displacement to another location. In other words, *virtual environ-*

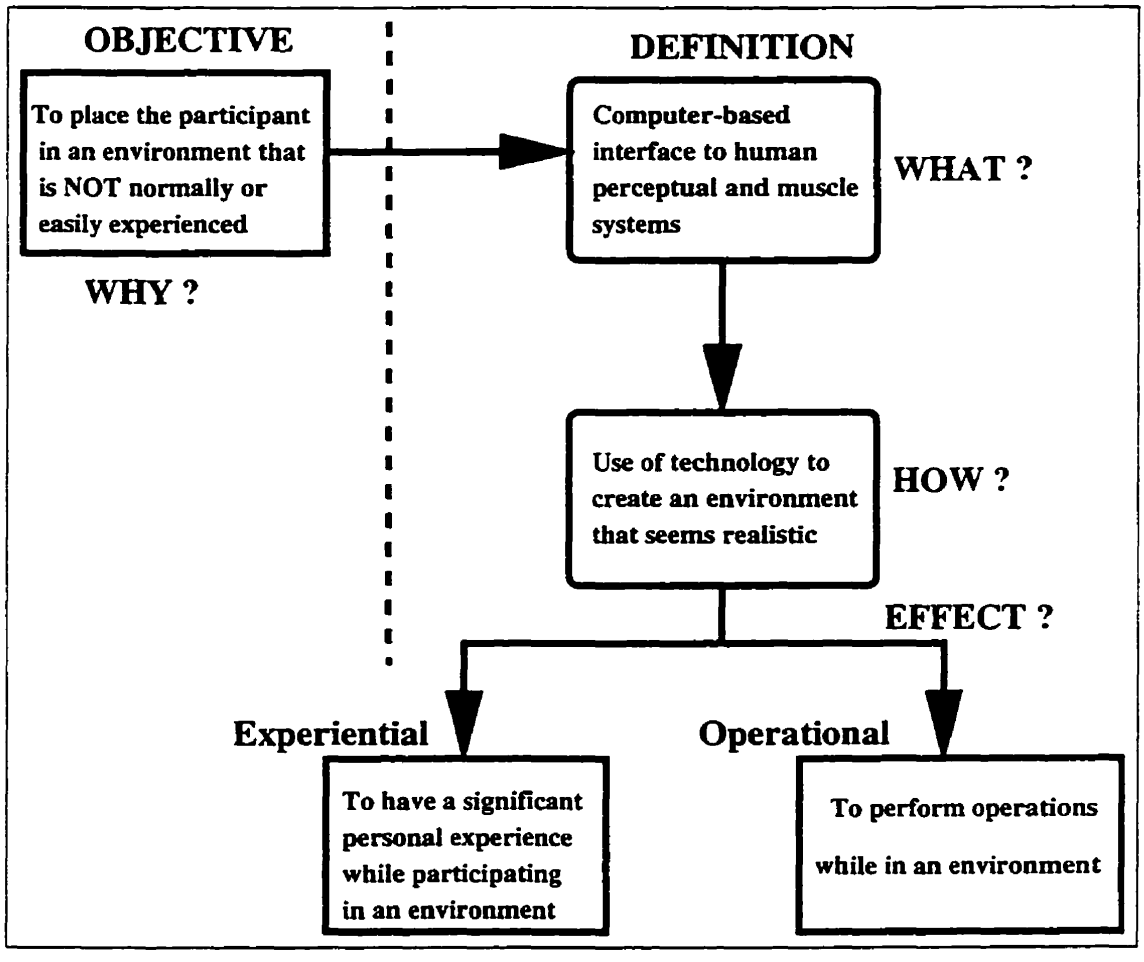


Figure 4.1: VR objective and definition

ments can be defined as *interactive, virtual image displays enhanced by special processing and by non-visual display modalities, such as auditory and haptic, to convince users that they are immersed in a synthetic space* [Ellis, 1991]. [Appino and others, 1992] presented an architecture for virtual worlds used for a computational fluid dynamics simulation. Virtual environment displays potentially provide a new communication medium for human-machine interaction. In some cases, they might prove cheaper, more convenient and more efficient than former interface technologies. In fact, teleoperations like tasks requiring coordinated control of a viewing position and a manipulator, are the tasks most likely to benefit from a virtual environment interface. In teleoperation or planetary surface visualisation, virtual environments offer techniques for solving control problems caused by time delays or awkward camera placements. Additionally, the completely synthetic character of purely virtual environments allows the introduction of visual, auditory and hap-tic interac-

tion modes totally unrealizable in physical environments. Considered as communications media, virtual environment displays have broad applications potential - *in education, procedure training, high-level programming, teleoperation, remote planetary surface exploration, exploratory data analysis, and scientific visualisation, as well as entertainment.*

The research and development community associated with vehicle simulation and teleoperations interface development have the technical training and applications background required to design usable virtual environment displays and to constitute a tradition of expertise in this field. The illusory virtual environment is created through the operation of three types of hardware:

1. *Sensors, such as head position sensors, to detect the operator's body movement;*
2. *Effectors, such as a stereo-scopic display, to stimulate the operator's senses; and*
3. *Special-purpose hardware that links the sensors and effectors to produce sensory experiences resembling those in a physical environment.*

In a virtual environment, a simulation computer establishes this linkage. In closely related technology of head-mounted teleoperation display, the linkage is accomplished by robot manipulators, vehicles, control systems, sensors and cameras at a remote work site. The display technology works by developing a real-time, interactive, personal simulation of the content, geometry and dynamics of the environment [Foley, 1987]. The software for a virtual environment must address three separate functions: (a) the shape and kinematics of the actors and objects; (b) their interactions among themselves and with the environment; and (c) the extent and character of the enveloping environment. A successful environmental simulation must provide adequate communications channels to address these functions. [Latta and Oberg, 1994] developed a conceptual VR model that isolates and describes the human and technical elements that create the participatory environments of VR systems.

Our primary physical connection to the world is through our hands as we perform most everyday tasks with them. However, when we work with computer-controlled applications, we are constrained by clumsy intermediary devices such as keyboards, mice and joysticks.

Little of the dexterity and naturalness that characterise our hands transfers to the task itself. In an effort to change this, people have been designing, building and studying ways of getting computers to *read* user's hands directly free from the limitations of intermediary devices. The development of electronic gloves has been an important step in this direction. A basis for understanding the field by describing key hand-tracking technologies and applications using glove-based input is provided by [Sturman and Zeltzer, 1994]. Samples of developed electronic gloves are summarized in Table 4.1. The teleoperation of multi-finger robot hands requires a dextrous master that is a multi-DOF controller worn on the operator's hand. Use of the hand gestures is a natural form of control that can bring significant improvements in teleoperation efficiency. Commercially available dextrous masters such as Data-Glove, control the position of a robot hand or of a simulated hand (in virtual environment applications) in the open-loop without force or touch feedback to the operator. There is a need for portable systems that have force feedback, but are still sufficiently compact to be desktop. [Burdea and others, 1992] discussed a sample prototype master providing force feedback for the Data-Glove.

VR technologies have not yet crossed the threshold of usability due to display resolutions rendering the user legally blind. Head- and hand-tracking devices are inaccurate and of very limited range. Most setups can generate only the crudest of scenes without update lags that ruin the feeling of immersion. However, VR has so far shown more promise than practical applications. The promise looks bright for fields such as data visualisation and analysis. For such problems, VR offers a natural interface between human and computer that will simplify complicated manipulations of the data. VR also provides an opportunity to rely on the interplay of combined senses rather than on a single or even dominant sense. So far, it is not known whether VR is better than other visualisation and analysis approaches for certain classes of data and if so, by how much. The payoff will come not for those applications or tasks for which VR is merely better, even if significantly, but for those applications or tasks for which it offers some unique advantages. To answer some of these questions, [Ribarsky and others, 1994] embarked on a multi-pronged program involving the *Graphics Visualisation and Usability Centre* and other research groups at Georgia Institute of Technology. Integration is mandatory, since these questions involve basic considerations: how immersive

Glove Type/Name	Developer	Year
Sayre Glove	Sayre *et al*, Illinois	1976
MIT LED Glove	MIT Media Lab	1980
Digital Data Entry Glove	Gary Grimes, AT&T Bell Labs	1983
VPL Data-Glove (fiber-optic)	VPL	1987
Exos Dexterous HandMaster (exoskeleton)	Exos	1989
Power Glove (low cost)	Mattel toy company	1989
CyberGlove (18 sensors)	Virtual Technologies	1990
Space Glove	W Industries - Virtuality systems (England)	1991

Table 4.1: Developed electronic gloves

environments affect user interfaces and human-computer interactions; the ranges and capabilities of sensors; computer graphics and the VR optical system; and applications' needs. [Ribarsky and others, 1994] reported some of their results which include using *Glyph-maker* to create customised visualisations of complex data variables onto graphical elements in the virtual environments. More research issues in scientific visualisation can be found in [Rosenblum, 1994]. These issues include volume visualisation, perception and user interfaces, data modelling for scientific visualisation, foundations of visualisation, vector and tensor field visualisation, etc.

Interactive graphics and especially VR systems, require synchronisation of sight, sound and user motion if they are to be convincing and natural. A method to accurately predict sensor position to more closely synchronized processes in distributed virtual environments is to be proposed. Problems in synchronisation of user motion, rendering and sound arise from three basic causes: noise in the sensor measurements, the length of the processing pipeline

(i.e. delay times) and unexpected interruptions. Most VR systems either use raw sensor positions or they make an ad-hoc attempt to compensate for the fixed delays and noise. A typical method for compensation averages current sensor measurements with previous measurements to obtain a smoothed estimate of position. The smoothed measurements are then differenced for a crude estimate of the user's instantaneous velocity. Finally, the smoothed position and instantaneous velocity estimates are combined to extrapolate the user's position at some fixed interval in the future [Haule and Malowany, 1995b]. [Friedmann and others, 1992] presented a solution to these problems based on the ability to more accurately predict future user positions using an optimal linear estimator and on the use of fixed-log data flow techniques that are well known in hardware and operating system design.

4.2 Telepresence and Virtual Presence

The term *telepresence* is often used in discussions of teleoperation, however it has never been adequately defined. According to [Akin and others, 1983], telepresence occurs when the following conditions are satisfied:

> *"At the worksite, the manipulators have the dexterity to allow the operator to perform normal human functions. At the control station, the operator receives sufficient quantity and quality of sensory feedback to provide a feeling of actual presence at the worksite".*

A major limitation of this definition is that it is not sufficiently operational or quantitative. It does not specify how to measure the degree of telepresence. A high degree of telepresence is desirable in a teleoperator system primarily in situations in which the tasks are wide ranging, complex and uncertain (i.e. when the system must function as a *general-purpose system*). In such situations, a high degree of telepresence is desirable because the best general purpose system known to us (as engineers) is us (as operators). In a passage that is relevant both to this issue and to the definition of telepresence, states:

> *"Teleoperators offer the best means of transmitting man's remarkably adapting problem solving and manipulative skills into inhospitable environments. The*

anthropomorphic motor approach calls for development of teleoperator subsystems which sense highly detailed patterns of visual, auditory and tactile information in the remote environment and display the non-harmful, task-relevant components of this information to an operator in a way that very closely replicates the pattern of simulation available to an on-site observer. Such a system would permit the operator to extend his sensory-motor functions and problem solving skills to remote or hazardous sites as if he were actually there".

In addition to its value in a general purpose teleoperator system, telepresence is likely to be useful in a variety of other applications. More specifically, it should enhance performance in other applications such as in virtual environments, where the operator interacts with synthetic worlds created by computer simulations (i.e. virtual worlds). The most obvious cases in this category are those associated with training people to perform certain motor functions (e.g. flying an airplane) or with entertaining people (i.e. providing imaginary worlds for fun). Telepresence is also important for cases in which the system is used as a research tool to study human sensorimotor performance and cases in which it is used as an interactive display for data presentation [Fisher, 1987]. An important obstacle at present to scientific use of the telepresence concept is the lack of a well-defined means for measuring telepresence. However, the core issue is how one achieves telepresence. In other worlds, what are the factors that contribute to a sense of telepresence? In fact, what are the essential elements of just plain *presence?* Or, how can the ordinary sense of presence be destroyed? Sensory factors that must certainly contribute to telepresence include high resolution and large field of view. In addition, the devices used for displaying the information to the operator's senses in the teleoperator station should, to the extent possible, be free from the production of artifactual stimuli that signal the existence of the display. The most crucial factor in creating high telepresence is, perhaps, high correlation between the movements of the operator sensed directly via the internal kinesthetic senses of the operator and the actions of the slave robot sensed via the sensors on the slave robot and the displays in the teleoperator station. In general, correlation will be reduced by time delays, internally generated noises, or non-invertible distortions that occur between the actions of the operator and the sensed actions

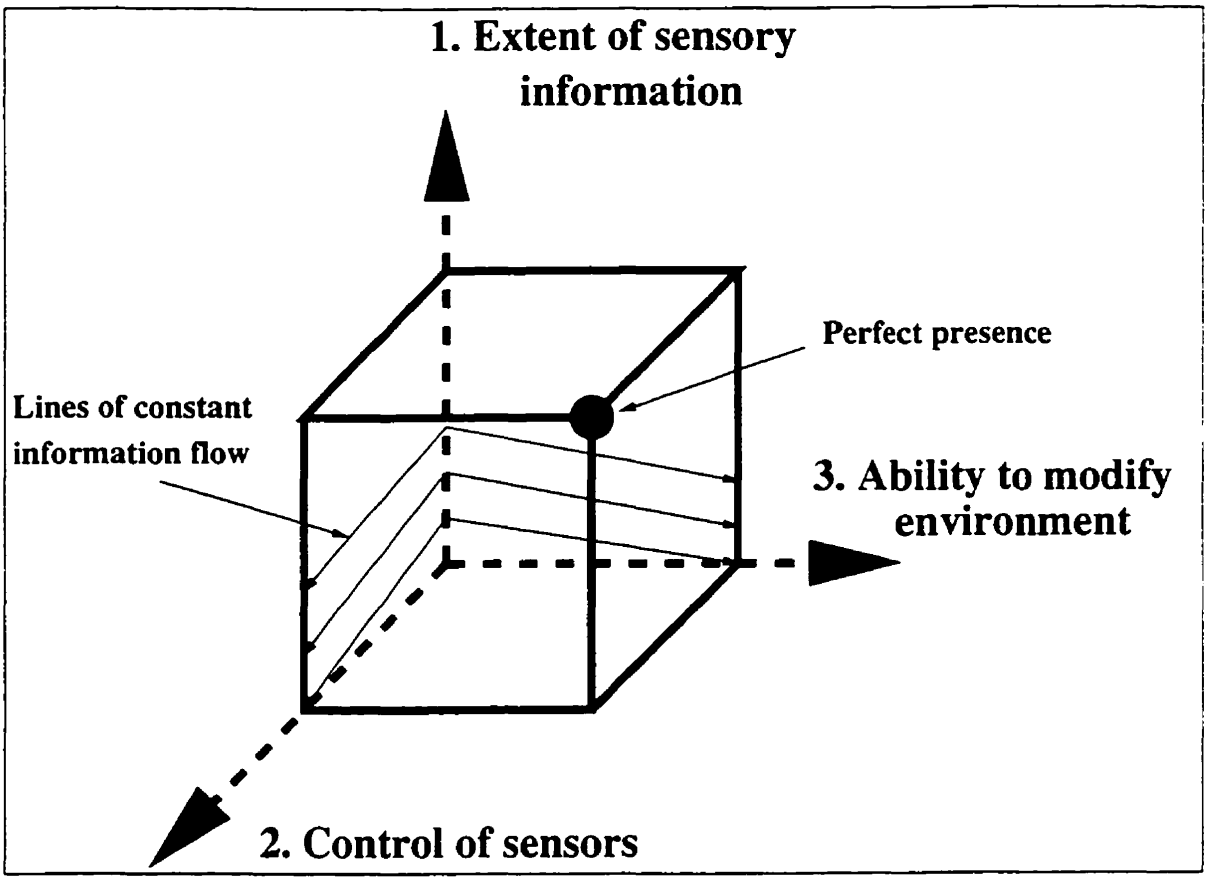


Figure 4.2: Principal determinants for sense of presence

of the slave robot. It also seems plausible that identification of above factors and therefore telepresence, would be increased by a similarity in the visual appearance of the operator and the slave robot.

It is important to consider the extent to which telepresence can increase with operator familiarisation by adaptation, training, learning, etc. [Ogata and Takahashi, 1994, Held and Durlach, 1991]. Although the impression of *telepresence* or *remote presence* is just now becoming a familiar phenomenon in connection with teleoperators and virtual displays, a closely related phenomenon has received attention in the past by both philosophers and perceptionists. The phenomenon, which has been referred to as *externalisation* or *distal attribution*, is this:- that most of our perceptual experience, though originating with stimulation of our sense organs, is referred to external space beyond the limits of the sensory organs [Loomis, 1992]. For a given task, [Sheridan, 1992] proposes three measurable physical variables that determine telepresence and virtual presence as shown in Fig. 4.2. Sheri-

dan discussed several aspects of human performance that might be (differentially) affected by these forms of presence and suggests models by which to characterise both kinematic and dynamic properties of the human-machine interface and how they affect both sense of presence and performance. Recent developments promise to make the virtual environment a medium that can engage researchers attention, which has generated a great deal of public interest. [Zeltzer, 1992] presented a taxonomy of graphic simulation systems, based on three salient components: *autonomy, interaction & presence (AIP)*. The resulting *AIP-cube* (shown in Fig. 4.3) provides a useful qualitative tool for describing, categorising, comparing and contrasting virtual environments, as well as more conventional computer animation and graphic simulation systems. Moreover, such a taxonomy can help researchers to identify application areas as well as avenues of research to pursue. Most existing research on VR concerns issues close to the interface, primarily how to present an underlying simulated world in a convincing fashion. However, for VR to achieve its promises as a rich and popular artistic form, as have the novel, cinema and television, it will be necessary to explore well beyond the interface, to those issues of content and style that have made traditional media so powerful, e.g. VR for art and entertainment. Broad exploration is required if VR is to achieve its promise of letting researchers *go anywhere & do anything* [Bates, 1992].

4.3 Virtual Human Interface: applicational areas

VR has been mainly developed in visual applications. In order to improve *reality*, force information is very important. Several force display systems have been already proposed so far. However, the motion ranges are relatively small. Since most of the force display systems have multi-link structures such as robots, large mechanical part of force display systems or large robots are necessary to attain enough motion space. However, large mechanical parts or large robots are not desirable for safety and cost reasons. To overcome the difficulty, [Kawamura and others, 1995] proposed a new type of force display using a wire drive system for the development of a virtual sports machine (i.e. virtual tennis). To realize virtual tennis, it is necessary to provide player hands with reaction forces from a tennis ball. Moreover, a head mounted display is needed which is capable of showing moving objects at high speed. In order for humans to interact more effectively with comput-

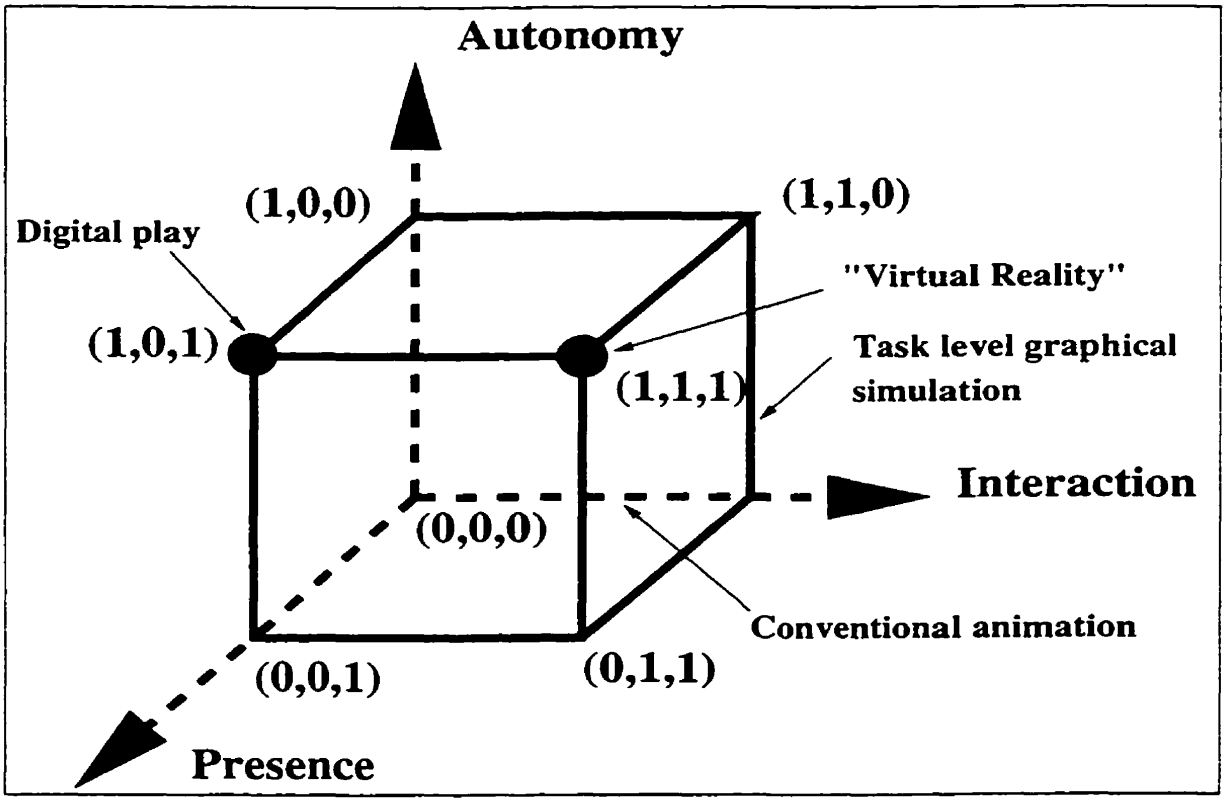


Figure 4.3: The AIP-cube (taxonomy)

ers, the capability of humans to use various types of sensory data must be exploited. One means is by haptic perception of forces and torques on a user's hand while interacting with an artificial environment. Rendered graphics and virtual reality displays can generate realistic appearances of simulated environments, but do not enable the virtual world to be felt or manipulated directly. An ideal haptic interface would give a computer user the ability to feel, grab and manipulate virtual objects [Hui and Gregorio, 1995, Hayward, 1995, Colgate *et al.*, 1995]. The dynamic reactions of objects in the simulated world would be instantly transmitted to the hand of the user as the motions or forces of the user's hand are sampled by the interface. Tasks that have been integrated with some type of haptic interface include flight simulators, force-reflecting teleoperation & telepresence, simulated molecular docking, etc. Additionally, any computer interface task which currently uses an input device such as a mouse, tablet, or joystick can benefit from an effective haptic interface. [Berkelman and others, 1995] developed a high-performance magnetic levitation haptic interface to enable the user to interact dynamically with simulated environments by holding a levitated structure and directly feeling its computed force and motion responses. The re-

sponse of the levitated device has been made successfully to emulate virtual devices such as gimbals and bearings as well as different dynamic interactions such as hard solid contacts, dry & viscous friction, and textured surfaces.

Dextrous manipulation of remote objects can be performed using a multi-fingered robot hand as the slave end-effector placed at the remote end. A well accepted approach to teleoperate such a robot hand is by using a glove-like haptic interface capable of measuring the human finger motions and projecting the measured motions on to the robot hand. In a true teleoperation, the human operator should feel as if he is directly handling the remote objects with bare hands. To achieve such realism, the human operator must be provided with such perceptual information about the remote world as vision, touch, force, sound and vibration [Shimoga, 1993]. [Shimoga *et al.*, 1995] described a touch reflection system suitable for dextrous interaction with remote and virtual environments. The basic framework involves a robot hand, teleoperated by a human master wearing a VPL *Data-Glove* [Inc., 1993b]. The touch reflection system consists of tactile sensors attached to the tips of the robot fingers and micro-actuators attached to the finger tips of the human operator's hand. [McKee and Schenker, 1995] proposed the use of automated viewing during teleoperation by combining deliberative task models, modes of human perception and reactive architectures. For more applicational areas of human virtual interfaces worldwide, one can refer to the following references [Kahaner, 1994, Encarnacao and others, 1994, Voyles and Khosla, 1995, Caldwell *et al.*, 1994, Anderson and Davies, 1994].

4.4 VR for Teleoperator Interface & Training (TIT)

Robots are often required to function in environments which would be extremely dangerous or expensive when using direct human labour, however, computer control and intelligence are not sufficiently developed to permit the robots to perform these advanced technical tasks under their own initiative and there is always a human operative in the loop. Ideally the operator would wish to input body motions (from legs, arm, hand and head) which the robot would duplicate and receive from the remote sensors full *visual, audio & tactile* feedback of a quality and form comparable with that normally produced by the eyes, ears and skin. Thus, this research considers the development of input, control and feedback (visual, audio

& tactile) systems (*human-machine interface*) (HMI) for a machine (e.g. a robot) to be used in the telepresence applications. This multi-purpose human-machine interface provides the user with an enhanced degree of true control of and *feel* for the task in hand. Training the operators of complex robotic systems is time-consuming and costly. Thus, a VR-based robotic simulation system is required. The VR system provides a means by which operators can operate and be trained to operate complex robotic systems in an intuitive, cost-effective way. Operator interaction with the remote workcell is at a high, task-oriented level. Continuous state monitoring prevents illegal robot actions and provides interactive feedback to the operator and real-time training for novice users. With a VR-based interface, the operator is fully immersed in the graphical environment of the remote system. The VR system can be set up to stimulate both normal and exceptional behaviours to enhance the training of operators. The VR system will allow telecommands to be captured, previewed and down-loaded to the remote workcell for execution.

In both teleoperator and virtual environment systems, the human operator is projected into a new interactive environment mediated by artificial electronic and electro-mechanical devices. The operator's performance, experience and sense of presence in these new environments depend strongly on the HMI and the associated environmental interactions. The primary focus of this research is the understanding and design of these interactions and interfaces. Secondary foci include: (a) the human operator's own cognitive and sensorimotor systems, particularly those elements of these systems that are directly related to the HMI or to the sense of presence; (b) the more peripheral components of the two types of systems; namely the telerobotic mechanism and its environment in the case of virtual environments; and (c) the impact of transformed presence, achieved by either teleoperators or virtual environments. VR is a technique for creating simulated experience, or rather, an experience of a simulated external world. In VR, the visuals, sounds and sensations create an actual experience, leaving the freedom to the operator to explore the environment, gather information and effectively solve problems. Advances in computer technology are making possible the development of 3D graphical and VR simulations of ever-increasing realism and afford-ability. Virtual worlds can be used as direct interfaces for teleoperators (i.e direct sensing & manipulation user-interface) using VR tools from 3D position sensors to sensing

gloves, stereo viewing devices and 3D sound generators. The main objective is to put the user in a 3D environment that enables him to explore the virtual environment and interact with it in various ways. A comprehensive and interactive TIT system is to be developed using VR technology. Thus, TIT is a *computer-assisted-interface (CAI)* to the teleoperator. Salient features of TIT system include telepresence, telesupervision and autonomous operation with HMI. A systematic approach for the development of TIT will reduce the lead-times and costs of facilities for recurrent basic tasks. 3D modelling is realized through VR programming using VR toolkits. VR environments are most useful when the communications bandwidth is low or the time delay is large.

TIT will consist of a virtual world creator and its virtual interface. A multi-media system consisting of 3D interactive graphics workstation (SGI), 3D mouse, head tracker, stereoscopic display systems, glove devices with the position and orientation of the hand registered by a tracking device and an audio-speech devices are used to enhance the effectiveness of TIT to be developed. One of the most promising areas to use TIT is in an unstructured or hazardous environments such as nuclear plants, biological or chemical contaminations, space, etc. Robots replace humans because of the hostility of the environment or potential risks involved during operations. These operations range from handling hazardous materials to maintenance operations. However, the problem that immediately arises is the reduction in the level of supervision achievable by a human operator as he looses his two senses, i.e. *touch & vision*. VR provides excellent tools for TIT while compensating this disadvantage. A teleoperator can be interfaced to the RRW, in a natural and very efficient way. The main emphasis will also be teleoperator training in simulated environments. Moreover, stored plans from a TIT virtual environment simulation can be used to control a real remote robot and compensate for lengthy transmission delays. The TIT software interface can do real-time modelling of the digitised images that the remote robot senses. This is a tremendous improvement to the slow, frustrating *move-and-wait* approach (i.e. a back-breaking operation that resulted in fatigue and frequent errors) [Ferrell, 1965].

VR is a computer-based virtual environment in which operators can interact with a system by means of stereoscopic glasses, 3D pointing devices, hand gesture interface devices,

etc. The operator demonstrates a task in a computer-generated VR space (i.e. *training environment*). The robot performs the task autonomously in a physical world called *task environment*. The system recognises the operations in the training environment as elementary symbolic commands and transfers them to a robot. The robot by itself uses sensor data and command interpretation rules to execute the task. The virtual world concept allows the human operator to interact with real world in a more natural way, i.e. provides a natural user interface by putting him in the scene during remote task operation. Two modes of operation are proposed for TIT: the simulation mode and real-world mode. In both modes, the constructed world is presented to the user in a 3D visual form using 3D stereoscopic VR interface devices (i.e. 3D mouse, head tracker, data gloves, Crystal Eyes, etc.). The TIT creates the virtual world interacting with robots, and then refines it using the information from sensing devices mounted on the robots already in the remote scene and the teleoperator's knowledge from TIT training. In simulation mode, the virtual world is built out of imagination and with the help of a knowledge-based system for training purposes. TIT can simulate the individual operations and the interaction of the robots with the world, and finally a teleoperator can practice the whole mission beforehand. TIT also provides an excellent test-bed for trial experiments. It allows examining different algorithms and provides a better understanding of robots behaviour under certain circumstances.

4.5 Virtuality and Control of a Remote Workcell

The elements of a remote workcell in a collaborative job can be observed better in the virtual world than in real world. TIT provides communication tools for interacting with the real world while in a simulated environment. A complete interface has to be bi-lateral, i.e. the *machine-man & man-machine linkage*. TIT allows the teleoperator to feel and sense the constructed world as if he is actually present at the remote scene. It provides an excellent supervisory environment for the teleoperator in command to successfully accomplish a mission. The teleoperator can command the robots and other active elements operating in the remote world. Regular tools such as keyboards or mouse would not be efficient and even impossible for a teleoperator to use as communicating tools with the robots while in remote areas. Instead, teleoperator's voice, hands and head's gestures can be used to control

the RRW. The voice commands, 3D mouse, head tracker & glove devices with the position and orientation registered by a tracking device allows for development of TIT in VR environment. For example, in fine manipulation of an object, the robot's hand can imitate the teleoperator's hand and exert the same forces and in the same way that he exerts on a virtual object, or in some operations a teleoperator can guide two robotic hands to perform the job by using two gloves and performing the same operation himself in his chamber and letting the robots imitate his hands. With this concept, TIT can have two modes of operation, i.e. *navigation & manipulation* modes. Thus, during *navigation* mode, the VR input devices are used for navigation purposes and during *manipulation* mode, same devices are used for manipulation purposes. It is also possible to operate using both modes concurrently if sufficient HCI peripherals are available.

At the heart of *HMI* is a powerful Workstation (currently the Silicon Graphics (SGI)) where the user-interface program resides. TIT will provide enough feedback to the teleoperator to achieve telepresence and sufficient (i.e. quality & quantity) *sensory feedback* to approximate actual presence at the remote site. Use of *stereo vision* enhances teleoperator depth perception. The system will provide a user-interface with which an inexperienced operator can easily teach a task. The operator shows an example of task movements in the virtual environment and a finite automation defined task-dependently is used to interpret the movements as a sequence of high-level representations of operations. When the system is commanded to operate the learning task, it observes the task environment, checks the geometrical feasibility of the task and if necessary re-plans the sequence of operations so that the robot can complete the task. Then the system uses task-dependent interpretations rules to translate the sequence of operations into manipulator-level commands and executes the task by replicating the operator's movement in the virtual environment. Using TIT system, the operator interacts with the simulated environment in real-time, with the actual RRW responding after the time delay. This approach tends to decouple the control of the RRW at the two sites, compensating for the time delay.

Ogata *et al* suggested that any VR TIT system will have three phases of operations [Ogata and Takahashi, 1994]. *Phase 1: Setting-up the Training Environment*:- the operator spec-

ifies task components by selecting the part name and its colour attribute from the menu. Then, he specifies its rough layout in the training environment by dragging a mouse. *Phase 2: Instruction in the Training Phase*:- the operator uses VR interface devices to perform the task in the training environment created during Phase 1. The system, recognising the operator's movements from the interface device-data and the layout information, generates task-level commands such as *pick-up block A* during a *pick-and-place* task environment. *Phase 3: Operation in the Task Environment*:- the system interprets the command into manipulator commands by using the task-dependent interpretation rules defined in advance. Sensors (i.e. a colour image processing system & a force-torque) are used to determine the part's geometrical position and orientation, and the feasibility of the task is examined. If necessary, the system re-plans the sequence of the commands so that a robot can execute the task. Finally, the robot uses the interpreted commands to perform the task. Collaborative VR TIT systems will allow multiple network users to simultaneously access the same virtual environment, interacting with each other and one or more remote robots. Since the underlying communications protocol used is TCP/IP, users located thousands of miles apart can use the Internet to inhabit a common environment and share control.

VR is a type of human-computer interface where the user is immersed in an environmental simulation with which he interacts [Helsel and Roth, 1991, Miner and Stansfield, 1994]. As the computer-generated environmental models become more realistic and the user interaction more intuitive, the virtual world becomes more of a reality. The VR TIT system being developed for remote control, allows participants to interact with realistic simulations of complex systems in a natural way using task-level voice commands and hand or head gestures. Due to this natural interaction and task-level orientation, participants may be trained to use the system easily and commands may be generated, previewed and executed rapidly. The user is immersed in the graphical virtual environment through a stereo viewer which tracks the user's head or hand position and orientation. As the operator's view changes, the graphics are updated so that the feeling of immersion inside the virtual environment is achieved. When users are immersed in the virtual environment, they can get any view of the graphical model of a remote workcell by simply looking at the desired destination and *walking* or *flying* to the location. This type of human-computer in-

teraction is much more intuitive than a flat screen display using a 2D mouse to change the view. Traditional graphical programming systems have several limitations such as lack of accurate depth perception, lack of non-visual feedback (audio, force, etc.) and awkward control interfaces (2D mouse, keyboard, teach pendant and space-ball). VR has the potential to move the human-robot interface to a new, intuitive and user-friendly level [Miner and Stansfield, 1994]. Moreover, VR can be used to train operators by taking the physical robots out of the loop. By using VR TIT system, operators can be trained before the actual robot hardware is available. During training, the operators progress can be monitored and real-time feedback on their performance can be provided. Training scenarios are easily set-up, including emergency situation procedure training. By having the operator experience the emergency situation in the virtual world, they are much better prepared to carry out the correct procedures if the actual emergency should arise. Trainers can also interact with the trainees in the virtual simulation system to further enhance the training exercise. [Takahashi and Ogata, 1992] proposed a VR interface for robotic assembly task teaching. Tasks were executed by an operator wearing a VPL Data-Glove, and hand gestures were recognised and translated into robot commands. [Miner and Stansfield, 1994] described a VR simulation system which provides a different method of complex robotic system interaction by taking more of a task-level, supervisory approach to the problem. Extensive telerobotics research using VR techniques has been done by NASA-Ames Research and JPL for control of remotely deployed robots [Stark and others, 1987, Bejczy, 1980].

4.6 Operator Interactions with VR-based TIT Simulator

The VR-based TIT system can use a combination of off-the-shelf and customised hardware and software. The operator interface consists of an accurate, 3D graphical model of the test bed which is viewed through a stereoscopic viewer. A Silicon Graphics, Inc. Indigo/Extreme Workstation can be used to drive the graphics and simulation and provide audio feedback. Operators issue voice commands to interact with the VR TIT simulation. The voice commands are recognised by the voice recognition systems (e.g.VERBEX 7000) and are communicated to the SGI host via a serial link. Audio feedback provides the user with

guidance throughout operations of the system. Fig. 4.4 shows a block diagram of a general-purpose TIT system configuration. The operator can interact with the system through the use of an immersive stereo viewer and voice input. Audio feedback can be used to continuously guide the operator and to provide command confirmation. The operator enters the virtual environment through VR supported input devices. The stereo viewer uses tracking devices to continuously monitor the location *(x,y,z)* and orientation *(yaw, pitch, roll)* of the participant's head or hand. The SGI Workstation polls the stereo viewer about 10 times per second and updates the graphical display so that the user's movements are synchronised with his view of the virtual environment. In this way, the participant is given the sensation of actually being in the graphical world. The operator moves through the graphical environment by using VR interface devices while in the *navigation mode* for walking or flying. Operators become accustomed to the environment very quickly and require little instructions in how to use the system. The VR environment can be developed using the software platforms (e.g. WorldToolkit by Sense8) which are the modelling and simulation systems that are used to graphically display the robot motions during robot programming, previewing and monitoring. These platforms provide the capability of adding custom device drivers and user developed code using the object-oriented C^{++} language on which they are based. During operation, the operator receives continuous audio feedback from the system. This serves several useful purposes. First, the operator obtains confirmation of voice commands.

Advances in the field of computer graphics favor the emergence of relatively low-cost visual simulation systems which can be used in a number of situations where the safety of people and/or equipment is critical. The growing complexity and inherent risks associated with the operation of power transmission and distribution systems have made this industry one of the prime settings for the development of VR-based job-training simulators. Such systems offer the trainee the opportunity to be exposed to a range of scenarios and to exceptional conditions which either occur rarely or are hazardous to reproduce. Thus, ESOPE-VR[1] is a prototype developed for a distributed client-server oriented VR system aimed at the training of operators working in power utility switching or dis-

[1] *ESOPE-VR* constitutes an extension of the *ESOPE* (Expert-System for Operations Environment) simulator by Hydro-Quebec [Okapuu-von Veh and others, 1996].

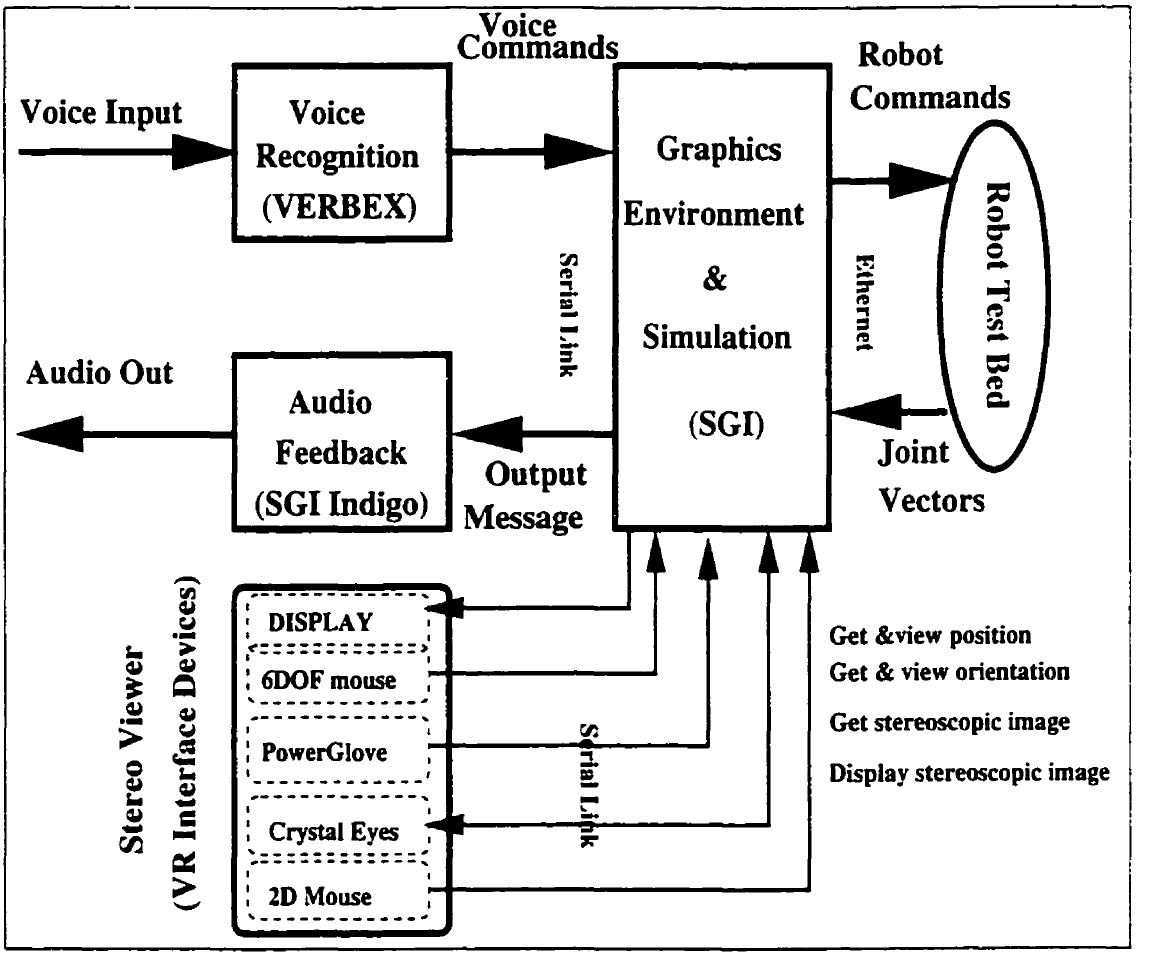


Figure 4.4: General-purpose VR-based TIT system block diagram

tribution stations. ESOPE-VR is developed jointly by Ecole Polytechnique de Montreal and McGill University with the logistic and technical support of Hydro-Quebec. It allows the user to exercise operations that typically consist of changing the topology of electrical networks by energizing or opening transmission lines, isolating equipment such as circuit breakers and transformers in order to perform maintenance or repair work, providing appropriate compensation or redistributing the load. The architecture and interaction metaphors of ESOPE-VR are described in [Garant and others, 1995]. Special emphasis is made on the generation of the virtual environment resulting from a complex conversion process between the 2-D schematic representation of a typical power station and its 3-D equivalent. ESOPE, like other systems, is a job training simulator based on 2-D user interfaces. While easy to use, they lack the realism and feeling of spatial immersion, which can be provided by a VR interface. The efficiency of the learning process can be

greatly improved when trainees not only hear and see, but also practice what they are being taught. Hence, ESOPE-VR aims to provide improved learning conditions by successfully integrating a 3-D visual interface, speech recognition, verbal feedback, multimedia facilities, navigation and manipulation devices along with expert system support. When these human-computer interaction techniques and their support tools are brought together, a training environment closer to reality is achieved thus reinforcing the soundness of the apprenticeship. The overall ESOPE-RV architecture can be found in [Garant and others, 1995, Okapuu-von Veh and others, 1996]. ESOPE-VR is a highly interactive 3D modelling system designed to explore the use of input devices and utilization of new 3D interaction techniques based on the World-Toolkit for SGI platform. Many initial teleprogramming-based experiments were also performed using the ESOPE system.

4.7 VR in the Future & its Social Implications

After the novelty of visiting a virtual world wears off, it all comes down to one question: why are we there? There needs to be a reason for being in a virtual environment, whether it is for escapism (e.g. entertainment), training or education. It is this reason that gives substance to the idea of *involvement*. Involvement is what reaches out and engages our minds; it is the stuff that sparks our imaginations. It is what completes the building of alternative worlds in our brains [Morie, 1994]. What it will take for VR to truly become the medium of the future (i.e. the thing that will let it fulfill its true potential) is *ubiquity*. This brings us to the concept of VR as a widespread and pervasive medium. Present rudimentary efforts involve networking some few computers so multiple players can share the same space and experience. Eventually VRs will be linked in a vast global network. One can see this happening in a non-graphic form on today's internet. Graphic servers such as *xMosaic* gives first glimpses of a more visual type of network. It is conceivable to imagine that same sort of functionality for networked VRs, i.e. common graphic and networking protocols that enable rapid transmission of all the digital data associated with a virtual world to all participants. TV, phones, cable systems and the telephone are beginning to merge into one integrated whole, i.e. the so-called *information super-highway*. When immersive technologies are finally folded into this global communication network, VR will finally achieve a

ubiquitous state. Within the global network, immersive technologies will have widespread artistic, social and entertainment implications. With the information super-highway barely in the planning stage and media forerunners establishing the first interactive television networks, we're poised on the edge of a future that could change the very fabric of our lives. However, we are still operating under old paradigms; the new ones are still to come.

What could VR become if it reaches its potential (i.e. if VR could move closer to the continua previously discussed)? What if it could reach the ultimate in *immersion, interactivity, involvement & ubiquity*? How can it do this? If VRs actually attained ultimate immersion, we could feel and sense them (i.e. indistinguishable from real life). Recreating real life is only one type of experience. With ultimate interactivity, we can actually construct worlds or modify them from within. In the future, the worlds will be ultimately responsive. Ultimate involvement will bring us worlds intriguing enough to engage us on high emotional and intellectual levels. High levels of immersion, interactivity and involvement will result in synthetic environments of a type which can scarcely be dreamed of now. For all this potential to be achieved, these environments cannot be rare structures that are accessible to an elite group. They must be easy to find, easy to use and there must be multitudes of them. Through new technologies, the computer is taking over functions heretofore regarded as non-rote *pattern recognition* and *thinking*: interacting with sensing devices, building databases of *knowledge* or *world models*, associating what is known from the past with what is currently happening, making decisions about what actions to take, and implementing those actions. The human operator is becoming more and more as a *supervisor* characterized in three terms [Sheridan, 1992]:

- **Super:** the human operator is above the computer hierarchically;
- **Tele:** the supervising is from a distance, both literally and figuratively; and
- **Meta:** the human operator may oversee many tasks, and may interact with many other supervisors by communication channels.

Thus, new technologies such as VR for telerobotics, etc. pose a serious challenge for the years ahead. As robotics automation continues to grow, the optimistic scenario is that

telerobots will grow in variety and numbers, becoming available to us to do our beck and call in our homes, schools, government facilities, hospitals, and across the entire spectrum of our workplaces (i.e. factories, farms, mines, etc.). Some will be of human size, some the size of insects and some much larger. Positive social implications of these new technologies include: improved task performance and reliability, improved human safety, reduction of human labour as machine labour costs much less, participative technology advancement and better appreciation of human intelligence in relation to artificial or virtual intelligence. However, new technologies have some negative social implications for the individual (i.e. *separation & alienation*). Developments in broadband communication technology have allowed the human supervisor to become physically separated from the locus of action. In large man-machine complexes (e.g. factories, power plants) the human supervisors are drawn into centralized control rooms to perform their instructing and monitoring activities aided by flexible and sophisticated command languages and means to communicate either through keyboards or voice, by exotic multi-colour computer-graphic displays, etc. Thus, the human supervisor becomes an *alien* to the physical process [Sheridan, 1980]. The components of potential alienation include: threatened or actual unemployment, silent failures (i.e. undetected for long periods), erratic mental workload and work dissatisfaction, desocialization, technological illiteracy, sense of not contributing, abandonment of responsibility, blissful enslavement, daily living by remote control (i.e. reduction of social contact), automation of ecoexploitation, electronic tele-governance, etc. As a result of all these, in today's world there are increasingly many situations where our use and appreciation of technology is closely tied to our *trust* in it. This is particularly true of large-scale technological systems such as air traffic control, electric power generation, computerized banking and fund transfer, and military command and control systems. Some of the attributes of trust include: *reliability, robustness, familiarity, understandability, usefulness, dependence, etc.*

Chapter 5 Symbolic Telecommands: Generation & Meta-Interaction

5.1 Introduction

The operator can perform a task in the virtual environment by visually and kinesthetically interacting with the simulation of the remote site. Another important feature of the *teleprogramming* system is that the operator's station software is capable of monitoring the operator's activity in the simulated environment and extracting from it a stream of symbolic robot instructions that capture all essential features of the task in progress. Fig. 5.1 illustrates the *black-box* view of the telecommand 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 virtual simulator and is updated at each simulation step. The second source of information, which is available to the symbolic telecommand generation module, is the *a priori* information about the task in progress. This task model allows the *teleprogramming* system to anticipate, recognise and correctly interpret special-purpose operations which are being performed by the operator. The output of the telecommand generation module are symbolic instructions, which can be again classified into two groups. The first group is composed of *low-level* commands, essentially encompassing *guarded & compliant* motions. These telecommands 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 motion commands, on the other hand, encompasses special-purpose or fine-precision operations, which are best executed autonomously at the remote site under local sensory supervision (e.g. fine precision object alignment, grasping, etc.). In

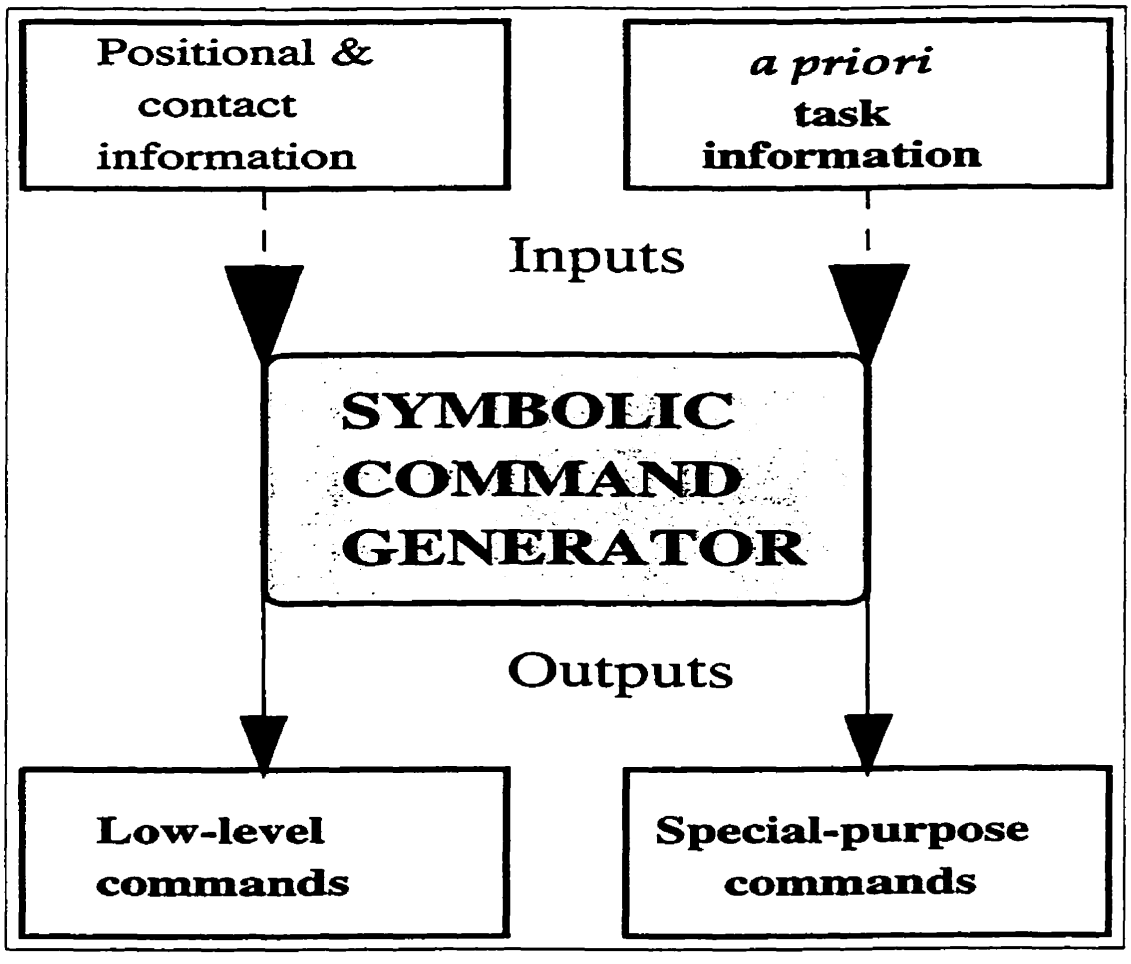


Figure 5.1: Telecommand generation process

this case, the task model provides global guidance to the process of interpreting low-level motions in the virtual environment, such that these special-purpose operations can be recognised 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. These *procedures* should be simple, unparameterised and defined on-line for one-time use. Moreover, the task model must contain sufficient information about the structure of the task to allow the system to recognise the operator's intent to initiate such a subtask. Likewise, the correct terminating conditions corresponding to an initiated iterative procedure should be specified in the task model. The rest of this chapter will focus on the *low-level telecommand generation* as it is the more basic and fundamental of the two telecommand types. The main

constructs of the low-level telecommand language interface between the operator's station and the remote workcell are also provided.

5.2 Low-level Telecommand Generation

The operator's station based model of the remote environment is only an approximation (within known tolerance bounds). Hence, the nature of the generated low-level telecommands must reflect and accommodate the discrepancies between the modelled and the actual world. This is not critical during free-space motion, but it is vitally important when attempting to establish or maintain a contact with the environment. Consequently, for the case of contact motion, the system generates sequences of guarded and compliant motion primitives with the modelling uncertainties built into the motion parameters. Moreover, as the operator's station based virtual model 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 (normalised numerical)* values for these parameters are supplied to the remote site, which in turn must substitute its estimate of the actual values prior to execution. These estimates are based on the workcell'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 modelling uncertainties, as well as to increase the execution reliability and robustness at the remote site despite sensing and control errors, the *hybrid force/position* model can be adopted [Raibert and Craig, 1981] for the symbolic telecommand 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*[1] and *constrained*[2] directions.

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

[1] 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*.

[2] A *constrained direction* is one along (or about) which the manipulator can not move, but can exert forces (or moments) on the environment; these axes are controlled in *force mode*.

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 workcell, is designed to match the hybrid force/position control paradigm. A description of the *syntax & semantics* of the low-level symbolic telecommand language can easily be augmented to *VR toolkit's* languages. The telecommand 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 (i.e. *task frame (TF)*) in which the subsequent motion parameters are to be interpreted. One of the 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*). By convention, free-space motions are commanded with respect to EE (dynamic frame). During contact manipulations, the task frame is centred at the primary contact point 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 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 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 motion 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.

5.3 Symbolic Telecommand Generation Algorithm

The *teleprogramming* system generates low-level symbolic telecommands by monitoring the elapsed time, contact state information and motion trajectories of the remote manipula-

tor and the movable object(s) in the simulated environment [Haule and Malowany, 1995c]. A new sequence of instructions is issued after 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. The global outline of the symbolic telecommand generation process is given in Algorithm 5.1. 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 virtual simulator. In step 5, the incremental Cartesian end-effector displacement $\Delta^{KB}d = (t, r)$ is computed as follows:

$$t = Trans(\Delta^{KB}T; \qquad r = RPY(Rot(\Delta^{KB}T)) \tag{5.1}$$

where

$$\Delta^{KB}T = T6_i \star (T6_{i-1})^{-1} \tag{5.2}$$

Algorithm 5.1 (Global low-level telecommand generation)

---

at the end of each simulation step {

1. $t_i \Leftarrow$ *current time*
2. $e_i \Leftarrow t_i - t_{i-1}$
3. $C_i \Leftarrow$ *current contact set*
4. $C_{i-1} \Leftarrow$ *old contact set*
5. $\Delta^{KB}d \Leftarrow$ *end-effector displacement*
6. *case motion-mode of* {
   - *free-space* *(use sub-algorithm for free-space motion)*
   - *sliding* *(use sub-algorithm for sliding motion)*
   - *pivoting* *(use sub-algorithm for pivoting motion)*
   - *pushing* *(use sub-algorithm for pushing motion)*

   }
7. *if command-generated* {

   $t_{i-1} \Leftarrow t_i$

   $C_{i-1} \Leftarrow C_i$

   $T6_{i-1} \Leftarrow T6_i$

   }

}

The *RPY* operator denotes that the corresponding incremental rotational motion is expressed as *roll/pitch/yaw* vector. The homogeneous transform $T6_k$ denotes the location of the manipulator's wrist with respect to its kinematic base (KB) at the 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 telecommand generation algorithms do apply for different motion modes. 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. Each sub-algorithm presents the analysis of representa-

tive cases and gives the corresponding execution environments (telecommand sequences), as well as the outline of the sub-algorithm (i.e. summarising the results). For example during *free-space motion*, telecommand generation proceeds in a straightforward fashion as shown in Algorithm 5.2. The vectors t and r are the incremental translational and rotational wrist-based displacements of Equation 5.1, appropriately rotated into the current task frame (EE) and t denotes the duration of the required motion (in seconds). The main constructs of low-level telecommand language interface between the operator's station and the remote workcell are as summarized in Table 5.1 representing three main categories of motion telecommands: (a) *task frame management*; (b) *force control*; and (c) *motion control*. All symbolic telecommands must be pre-labelled in CDB data elements and initialized as detailed in Chapter 7. For detailed analysis of these telecommands, one can refer to my published papers [Haule and Malowany, 1995c, Haule and Malowany, 1995b, Haule and Malowany, 1995a].

Algorithm 5.2 (Telecommand generation algorithm: free-space)

---

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*$(t; <0, 0, 0>; <r_x, r_y, r_z>)$
  }
 }

5.4 Telecommand Parsing and Translation

The operator's station visually and kinesthetically couples a human operator to a graphical simulation of the remote environment. The operator interactively, via an input devices supported in the virtual environment, specifies the task to be performed remotely. The final output of the operator's station is a stream of *execution environments*, each containing a description of an elementary motion or action to be performed by the slave workcell. Thus, this section and the next one focus on the *meta-interaction* of the remote workcell with its

Telecommand Class	Syntax & Semantic
Task Frame Management	**DefineVector**(*name*; $< v_x, v_y, v_z >$: ref_frame) **DefineTaskFrame**(*name:ref-frame;origin;x-axis;y-axis;z-axis*) **UseFrame**(*frame*)
Force Control Telecommands	**AssignMode**(*X, X, X, X, X, X*), $X \in \{P, F\}$ **Force**($< f_x, f_y, f_z >; < \tau_x, \tau_y, \tau_z >$) **GuardForce**($< f_x, f_y, f_z >; < \tau_x, \tau_y, \tau_z >$) **GuardVelocity**($< v_x, v_y, v_z >; < w_x, w_y, w_z >$)
Motion Telecommands	**Move**($t; < p_x, p_y, p_z >; < \phi_x, \phi_y, \phi_z >$) **Slide**($t; < \phi_x, \phi_y, \phi_z >$) **Pivot**($t; < \phi_x, \phi_y, \phi_z >$)

Table 5.1: Low-level symbolic telecommand language: syntax & semantics

environment as well as the operator's station virtual model. The analysis will include the instruction parsing and translation, a strategy for parsing, scheduling and executing the received instructions, which guarantees that the time lag between the virtual and the remote workcell will not increase during the task. More results were published in [Haule and Malowany, 1994a]. The operator performs a task by interacting with a local-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. These instructions, grouped into *execution environments*, arrive to the remote workcell a transmission delay τ (i.e. T_d in Figure 5.2) after they were generated and sent from the operator's station. Remote site execution proceeds by:

1. Parsing and translating the contents of successive execution environments into the local control language;

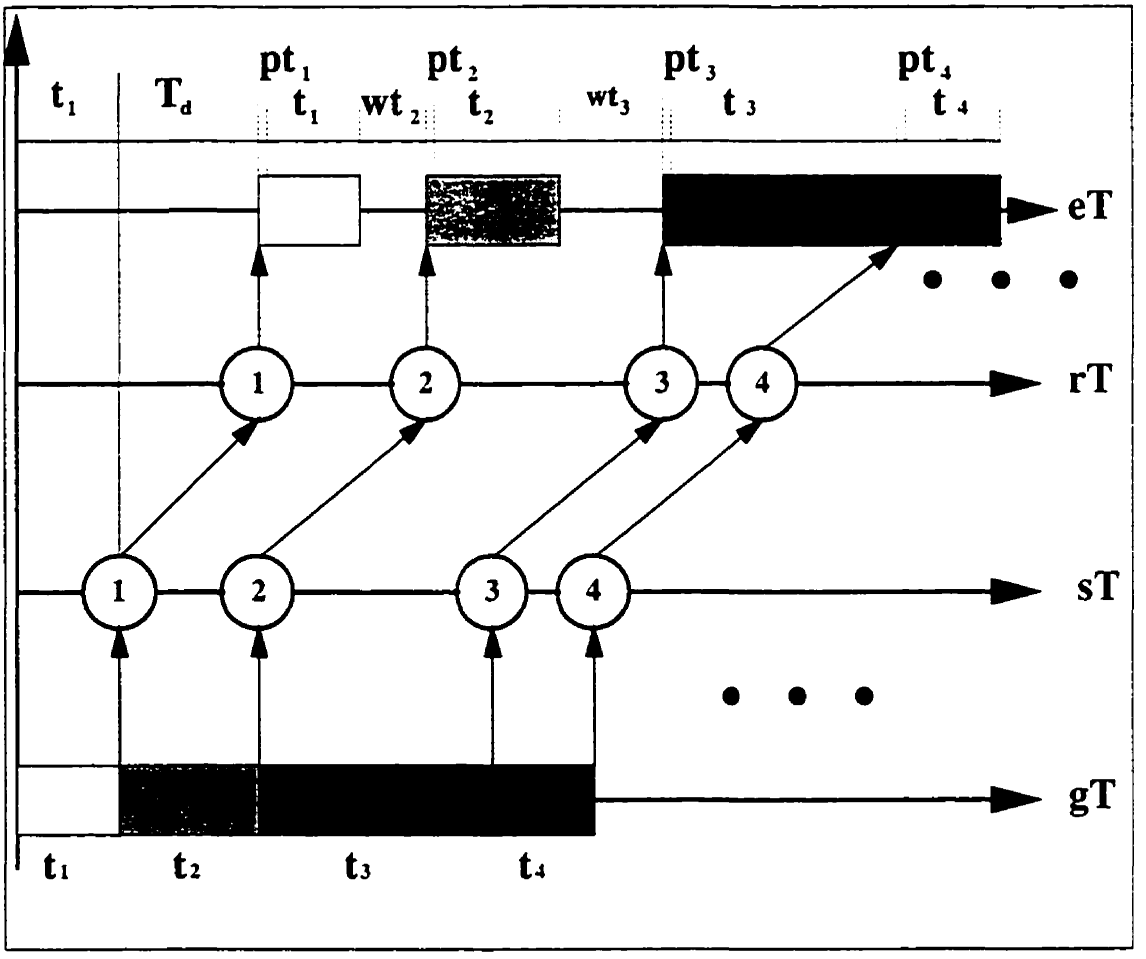


Figure 5.2: Sequential execution management

2. Substituting numerical values for the symbolic (or normalised 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; i.e. detecting error conditions, stopping the remote workcell safely on error and reporting resulting error state to the operator's station.

The *symbolic telecommand language*, which is used by the teleprogramming system to encode elementary motion and action information, is designed to be closely compatible with the *hybrid force/position* control paradigm [Raibert and Craig, 1981]. If the remote 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 a local remote controller, is straightforward. The symbolic telecommand lan-

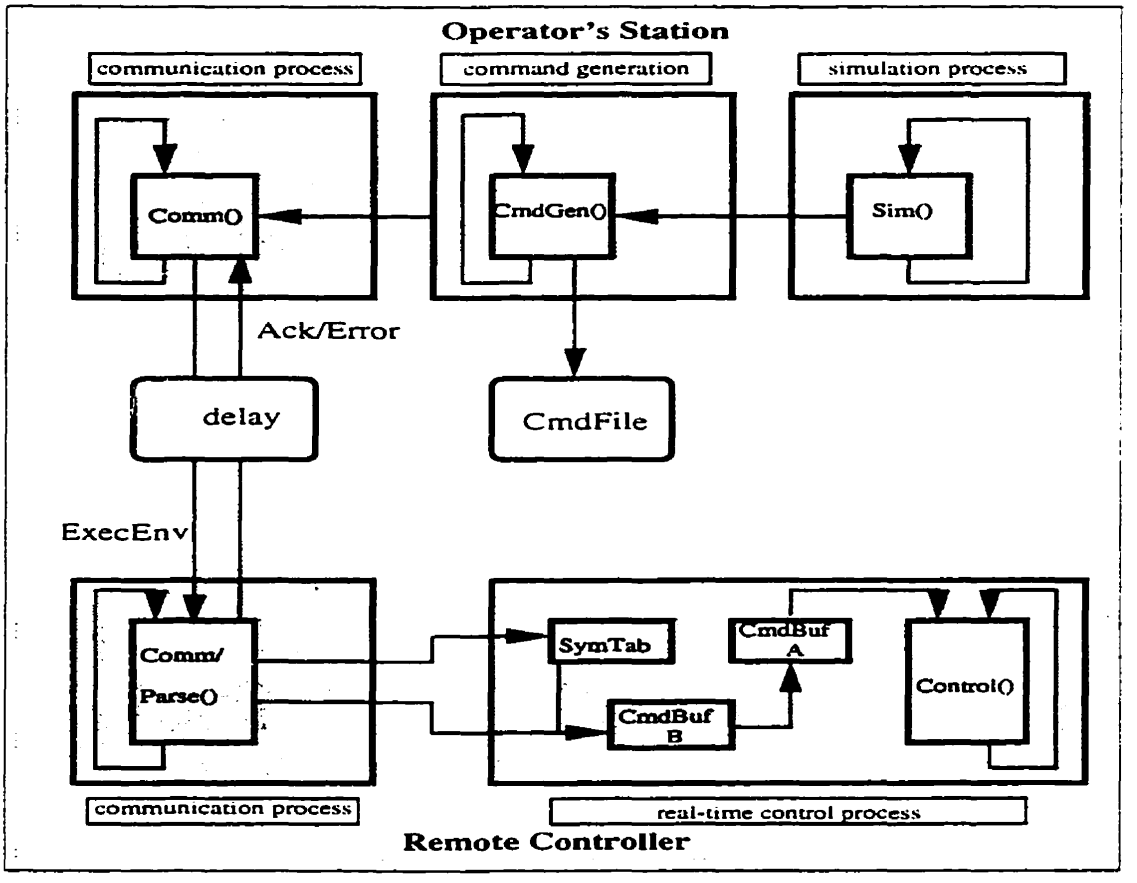


Figure 5.3: Double-buffering execution scheme

guage 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. Having produced a parser, the code generation process then proceeds in three steps. *First step:* Some of the instructions, such as *UseFrame, AssignMode, Move, Pivot, Slide*, etc. set the corresponding control parameters (i.e. current task frame, control modes, motion time and trajectory) in the remote controller directly and no additional processing is necessary.

Second step: Other instructions, such as *Force, GuardForce, GuardVelocity*, etc. do require some additional processing. In particular, in view of the kinematic nature of the operator's station based simulation, the parameters, supplied by these instructions, do not reflect proper dynamics of the remote manipulator and the environmental objects being manipulated. These instructions contain symbolic or normalised numeric values to denote dynamic parameters, such as frictional properties, compliance forces during sliding, guard forces

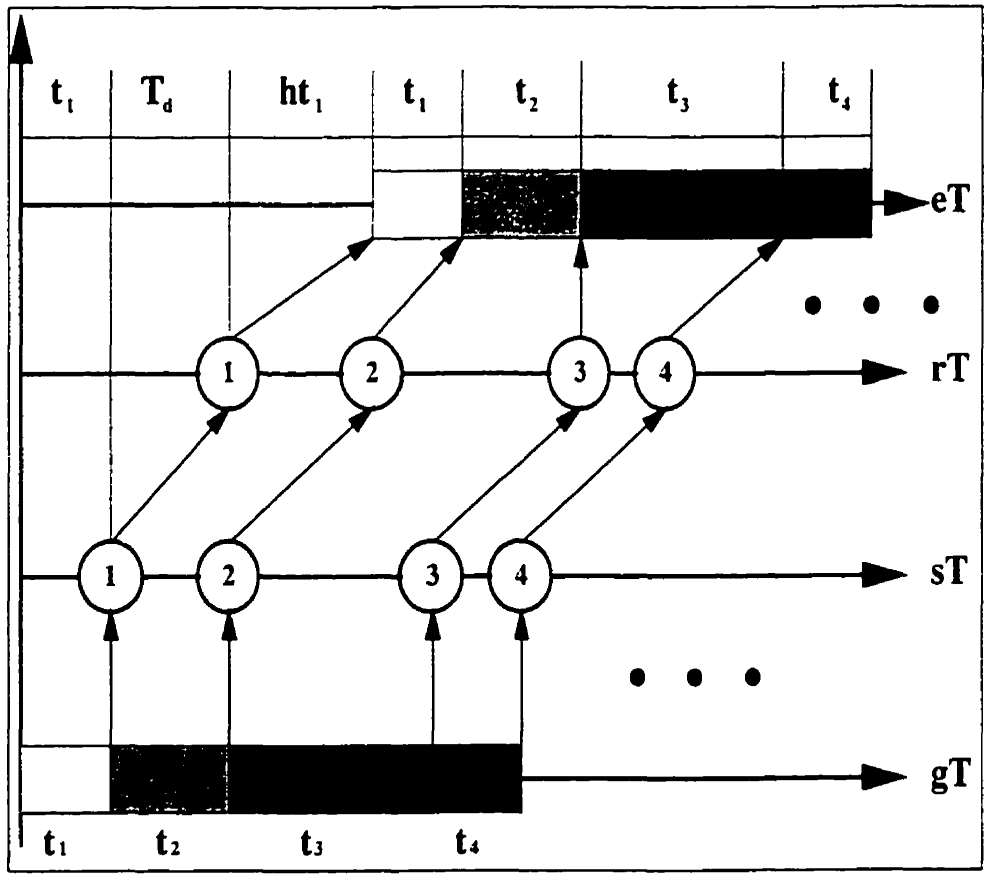


Figure 5.4: Double-buffering execution management

while approaching a new contact, etc. The translation process must therefore substitute actual (estimated) values for the symbolic place-holders. These estimates of the dynamic parameters of the manipulator'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 remote 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. *Third step:* Instructions, represented by *DefineVector* and *DefineTaskFrame*, serve to update the *Symbol Table* of currently defined vectors and coordinate frames, known to the remote controller. For clarity, the symbol table is a linked list. More time and space efficient data structures, such as hash tables, should be used in actual implementations. The two frames KB and EE correspond to the kinematic base and end-effector (wrist) frames respectively. Likewise,

Notation	Explanation
ⓘ	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 ⓘ

Table 5.2: Sequential dequeue-parse-execute notations

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 combination of these vectors. Task frames are specified to be either static (defined with respect to KB) or dynamic (defined with respect to EE).

5.5 Lag Control During Execution

During execution care must be taken to avoid increasing the *lag time* η between the virtual and the remote workcell as the task proceeds. A straightforward dequeue-parse-execute loop leads to the behaviour of Fig. 5.2, where the lag time increases throughout the duration of the task. The notations used in Fig. 5.2 are summarised in Table 5.2 and will be adopted from now on. The waiting time is defined as follows:

$$wt_i = \begin{cases} rT_i - (eT_{i-1} + t_{i-1}); & \text{if } rT_i > (eT_{i-1} + t_{i-1}) \\ 0; & \text{otherwise} \end{cases} \tag{5.3}$$

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 (i.e. just before executing ⓘ) is given by:

$$\eta_i = eT_i - gT_i = t_1 + \tau + \sum_{j=1}^{i}(pt_j + wt_j) \tag{5.4}$$

Equation 5.4 implies that even if the sum of waiting times is bounded, i.e. if

$$\lim_{i\to\infty} \sum_{j=1}^{i} wt_j < W \tag{5.5}$$

for some arbitrarily large constant W, we have

$$\lim_{i\to\infty} \eta_i = \infty \tag{5.6}$$

indicating that the lag time not only increases as the task progresses, but is in fact unbounded. In order to solve this problem, a *double-buffering* execution scheme shown in Fig. 5.3 is employed. The remote controller maintains two such buffers (i.e. CmdBuf A and B). While one is being executed, the other is being constructed by parsing and translating the next execution environment. The combination of this parallelism and an artificially introduced *holding time* ht_1, which delays the execution of the 1^{st} telecommand by ht_1, can be used to control the lag. This execution management scheme is also shown in Fig. 5.4 for clarity and is for its analytical validation refer to Section 9.2.

Chapter 6 Design of Teleprogrammable Control Scheme

6.1 Introduction on Visual Tracking

A new method for designing an optimal time-delayed teleoperator control system based on *prediction* of object position and its velocity using *observers* is presented [Hacksel and Salcudean, 1994, Zhu *et al.*, 1992, Haule and Malowany, 1994b]. An observer-based approach to the determination of moving object positions can be used to correct object velocity estimates generated by observers, leading to accurate, low-noise estimates during remote contact tasks. The control scheme can be used for collision avoidance purposes, visual object tracking, visual servo systems, etc. A design of a teleprogramming controller consisting of a *predictor* with an observer-based *double-loop feedback* for visual robotic tracking of moving objects during remote manipulation is demonstrated. The performance of the controller is evaluated using MATLAB computer simulations in the frequency domain by minimising a squared-error cost function of the motion parameters for a variety of motion trajectories. The designed controller is fast and robust enough as it will be shown by simulation results and control analysis.

A telerobotic system consists of a local and a remote manipulator. Position telecommands are sent forward from the master to the slave. In order to improve the performance of the system, force information is send back from the slave to the master. Often there is a transmission delay incurred when communicating between the two subsystems which causes instability in the force reflecting teleoperator. Teleprogramming methodology solves the instability problem as shown in [Haule and Malowany, 1995a]. Other control-based solutions include minimising the behaviour of a lossless transmission line, a scattering technique developed for a teleoperator system [Anderson and Spong, 1989]. PID controllers and a lag-lead compensators for a robot system were also developed [Chen, 1989]. 2-port network models of bilateral remote manipulation are also employed to solve the problem

[Raju and others, 1989, Anderson and Spong, 1988], etc. Although the resulting control laws were shown to stabilise an actual teleoperator system and are intuitively stable because of their passivity properties, stability for the system has not yet been proven. Thus, the designed tracking controller can be incorporated into the characterisation of the nonlinear, distributed parameter system, human operator and the environment. Tradeoffs between static accuracy, system stability and insensitivity to disturbances with sampling rate remain to be derived for a more realistic discrete time system model. Both theoretical analysis and real tests are required using various control theories. Extensions to include the development for the full n-DOF system and the use of a gain element is to be accommodated. The resulting controller has to guarantee stability for any passive task object at the slave port and any passive human impedance at the master port. Different control schemes exist in both manual and supervised automatic modes of control [Bejczy and Kim, 1995, Tarn and others, 1995].

One technique for teaching a robot to perform a certain task is to teach it the trajectory that the mobile object should track to accomplish the task. Once the robot learns the trajectory, it can achieve the task by tracking the trajectory. [Endo and others, 1995] described a method for controlling the trajectory tracking of a mobile object, enabling the robot to accomplish some task involving a target object. A visual servoing method [Walters, 1994, Yoshimi and Allen, 1994] can be used to position a robot with respect to an object before tracking it and estimating its velocity [Chaumette and Espiau, 1991]. A model registration system capable of tracking an object through distinct aspects in real-time is presented by [Ravela and others., 1995]. Their system integrates tracking, pose determination and aspect graph indexing. The tracking combines steer-able filters with normalized cross-correlation, compensates for rotation in 2D and is adaptive. [Bensalah and Chaumette, 1995] described a real-time visual target tracking using the generalized likelihood ratio (GLR) algorithm by first introducing the visual servoing approach and the application of the task function concept to vision-based tasks. They also presented a complete control scheme which explicitly enables pursuing a moving object. In order to make the tracking errors as low as possible, they used the GLR test, an algorithm able to detect, estimate and compensate abrupt jumps in target motion. Visual servoing [Walters, 1994, Espiau *et al.*, 1992,

Hashimoto, 1993] is now a classical approach to realize various robotics tasks (i.e. positioning, grasping, target tracking, etc.) in closed loop with respect to visual data. As far as target tracking is concerned, [Papanikolopoulos *et al.*, 1993] used classical approaches in control theory to track a moving object. However, they consider the object motion as disturbance, which implies tracking errors in the image. On the other hand, [Allen and others, 1993] developed an object motion estimation algorithm, based on $\alpha - \beta - \gamma$ filters, in order to reduce the observed tracking errors. Other similar techniques are based on the use of Kalman filters [Lee and Lee, 1994, Chaumette and Santos, 1993]. While the first approach has computational advantages, the second one seems much more appealing due to the adaptability of its coefficients for tracking various target motions. Different researchers design different control schemes to achieve the same goal while dealing with visual tracking [Richards and Papanikolopoulos, 1995, Papanikolopoulos *et al.*, 1994, Sharma and Hutchison, 1994, Bishop *et al.*, 1994, Yokokohji *et al.*, 1994, Lee *et al.*, 1994].

6.2 Design Methodology: for controllability & observability

A design was performed for the visual robotic tracking of moving objects during remote manipulation. Using the knowledge of gripper and object positions at uniform time intervals, a control system was designed. The design consists of two phases: first an *estimator* or *predictor* was designed; second a *double-loop feedback controller* which receives signals from the predictor was designed. A predictor algorithm which gives future positions of the moving object two sampling instants later was implemented in software. The motion of the object was estimated by minimising a squared-error cost function of the motion parameters [Isidor and Byrnes, 1990]. The performance of both predictor and controller was evaluated by performing MATLAB computer simulations. The results are satisfactory for a variety of remote object motion trajectories. Detailed results are presented in [Haule and Malowany, 1995b]. The configuration of the system under discussion is shown in Fig. 6.1. A number of fixed cameras act as sensors capturing the position of both the robot's gripper and the moving object every T seconds. In response to this information, the robot system must react by moving its gripper towards the moving object. Once the gripper is within a tolerable distance from the object, it must maintain that relative position.

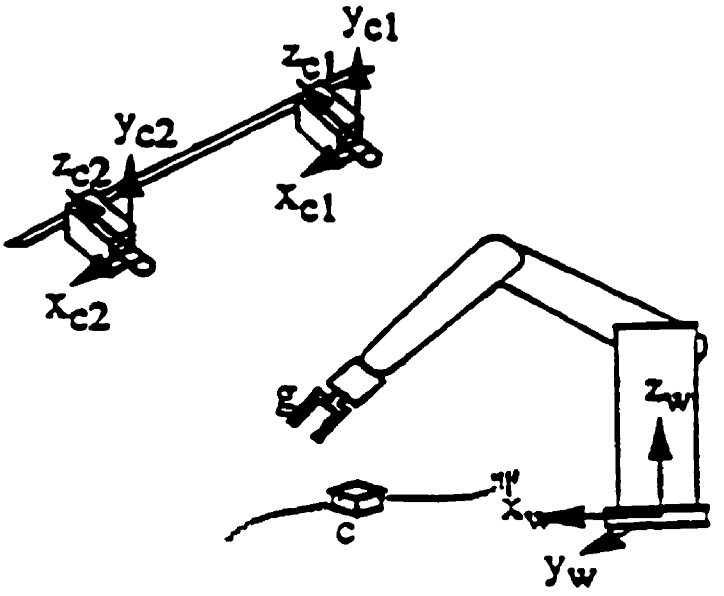


Figure 6.1: A generic remote robotic workcell

The design assumptions include: dynamics of the robot (i.e. its matrices) are known [Lei and Ghosh, 1993], no noise or friction is present, the visual sensors are fixed cameras that give the position of the gripper and the moving object, the information from the camera is already in the world coordinate frame (WCF) whose origin is at the base of the robot frame, the moving object has a smooth and continuous motion in time (i.e. no acceleration) within the vision field of all the cameras, the robot is actuated such that the speed of its arm is greater than that of the moving object so as to ensure that the object can be reached and the object is confined to move in the X-Y plane. The information available from the sensors is limited to the positions of both the gripper and the moving object in WCF. The solution is directly related to the minimisation of the position error between the gripper and the object. The gripper has to be actuated in such a way that its motion is directed towards a predicted future position of the object in order to *catch* it. This implies that the gripper's speed has to be greater than that of the moving object. A tracking scheme is shown in Figure 6.2. The controller responds to the position error between gripper positions $X_g(k)$ and object posi-

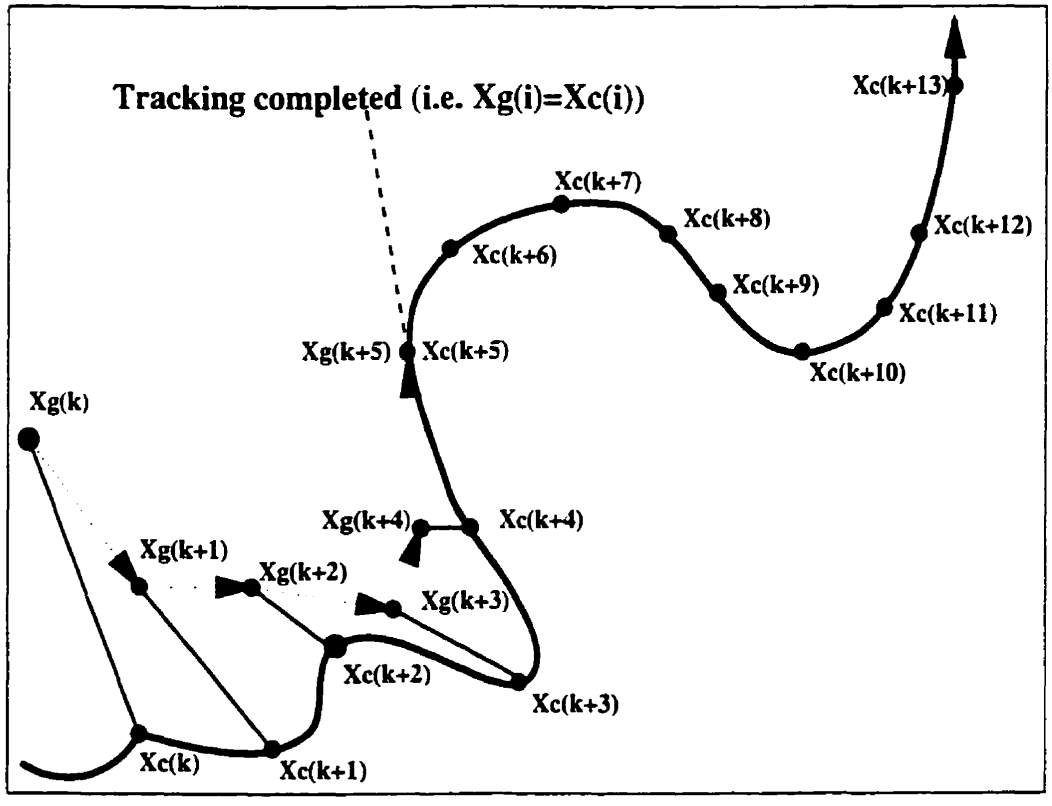


Figure 6.2: Error between gripper's and object's position

tions $X_c(k)$ and drives the robot arm while the predictor estimates the position of the object at time $t + kT$, where k is an integer taking care of the delays present in the control system. Equation 6.1 gives the matrices of the continuous-time robot system [Lei and Ghosh, 1993]. The output of the robot system $y(t)$ represents the position of the gripper in WCF. This corresponds to the three components of the state vector in Equation 6.2[1].

$$A = \begin{bmatrix} O_{3x3} & I_{3x3} \\ O_{3x3} & O_{3x3} \end{bmatrix}, \quad B = \begin{bmatrix} O_{3x3} \\ I_{3x3} \end{bmatrix}, \quad C = (I_{3x3}\ O_{3x3}), \quad D = 0 \tag{6.1}$$

$$\vec{Y}(t) = (X_g(t)\ Y_g(t)\ Z_g(t))^T \tag{6.2}$$

The other three states of the robot system represent velocities in 3-D space. Equation 6.3 shows a complete state vector.

[1] A *state* of a system is the minimum amount of information necessary to fully describe the system.

$$\vec{X}(t) = (X_g(t)\ Y_g(t)\ Z_g(t)\ \dot{X}_g(t)\ \dot{Y}_g(t)\ \dot{Z}_g(t))^T \tag{6.3}$$

Matrices **A** and **C** are unit-less, matrix **B** has units of 1/mass so that the input $u(t)$ to the robot system is a force and the output $y(t)$ is a 3×1 column vector having units of position. For no coupling between the three orthogonal coordinates, the overall system can be simplified and divided into three subsystems, one for each axis as shown in Equations 6.4, 6.5 and 6.6.

$$U_x(t) = \ddot{X}_g(t) \tag{6.4}$$

$$U_y(t) = \ddot{Y}_g(t) \tag{6.5}$$

$$U_z(t) = \ddot{Z}_g(t) \tag{6.6}$$

Each subsystem was assumed to have identical state-space equations given as:

$$\begin{bmatrix} \dot{X}_1 \\ \dot{X}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} U \tag{6.7}$$

$$Y = X_1 \tag{6.8}$$

Since the sensors (i.e. cameras) are discrete-time systems and the plant to be controlled is given in the continuous-time domain, a discretization of the plant is necessary. This is achieved by installing a *zero-order hold (ZOH)* in front of the plant. This discretization is assumed to be done with a periodic sampling T. The ZOH equivalent state-space equations of each subsystem are as given in Equations 6.9 and 6.10.

$$\begin{bmatrix} X_1(k+1) \\ X_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_1(k) \\ X_2(k) \end{bmatrix} + \begin{bmatrix} 0.5T^2 \\ T \end{bmatrix} U(k) \tag{6.9}$$

$$Y(k) = X_1(k) \tag{6.10}$$

The *controllability*[2] of the ZOH equivalent model was assessed before designing the controller. Similarly, the *observability*[3] was counter-checked. The controllability (C') and observability (O') matrices are as given in Equation 6.11. Since C' and O' exhibit full rank[4], each subsystem is controllable and observable [Franklin and others, 1990]. This means that the states can be placed arbitrarily via a controller and that an observer can be designed to estimate the unavailable state $X_2(k)$. Still, due to the presence of two simple real poles on the unit circle in the Z-plane, the subsystem is unstable. One requirement for the controller is thus to stabilise the closed-loop system.

$$C' = \begin{bmatrix} 0.5T^2 & 1.5T^2 \\ T & T \end{bmatrix}; \quad O' = \begin{bmatrix} 1 & 0 \\ 1 & T \end{bmatrix} \tag{6.11}$$

6.3 Prediction Scheme

The discrete-time predictor receives the current position of the object as given by the sensors and sends an estimate of the future position of the moving object in WCF to the feedback control system. Since the motion of the object is unknown, a discrete time model of Equation 6.12 or 6.13 was used to approximate the motion of the centroid of the object.

$$\vec{X}(k+1) = \alpha\vec{X}(k) + \beta \tag{6.12}$$

or equivalently

$$\begin{bmatrix} X_c(k+1) \\ Y_c(k+1) \\ Z_c(k+1) \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} X_c(k) \\ Y_c(k) \\ Z_c(k) \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \tag{6.13}$$

A minimisation of the cost function J was done in order to generate the parameters of the

[2]The system is *controllable* if the states can be moved in any direction in the state space, i.e. poles can be arbitrarily placed.

[3]*Observability* is the ability to estimate the system states from a record of output measurements.

[4]For observability & controllability, matrices C & O must have *full rank n* (i.e. n = dimension of C or O).

motion equation which is subsequently used to predict positions of the object.

$$J = \sum_{i=1}^{N} (\vec{X}(i) - \alpha \vec{X}(i-1) - \beta)^2 \tag{6.14}$$

The cost function J is a second order polynomial in α and β with positive coefficients in front of the parameters of highest degree. Thus, J has only one extremum point which is a minimum whose value is J_{min}, obtained by differentiating J with respect to α and β and equating it to zero. After differentiation was completed (see Section 6.7), the parameters minimising the cost function were found by solving Equation 6.15 where the matrix M is given in Equation 6.16. A computer algorithm whose structure is outlined in Algorithm 6.1 was used to compute the parameters necessary for prediction using MATLAB software on a Sun Workstation. Appendix B gives the MATLAB listing of the controller program.

Algorithm 6.1 (Outline of predictor algorithm)

1. *Initialisation of variables;* $i = 0$;
2. *Get position of object from cameras:* $i = i + 1$;
3. *Compute* α *matrix and* β *vector for all positions collected;*
4. *Check error between predicted and actual positions, if error is less than the maximum allowable (i.e.* ϵ*) then go to step 5, else go to 2;*
5. *Send the predicted position* $(i+2)$ *to the control system based on known* i *positions of the object;*
6. *Get position of the object from cameras:* $j = j - i + 1$;
7. *Use the last* i *positions of the object to compute the predicted* $j = j + 2$ *position;*
8. *Go to step 6 (predicted positions are thus generated up to the completion of the experiment).*

$$\begin{bmatrix} M_{4x4} & O_{4x4} & O_{4x4} \\ O_{4x4} & M_{4x4} & O_{4x4} \\ O_{4x4} & O_{4x4} & M_{4x4} \end{bmatrix} \begin{bmatrix} \alpha_{11} \\ \alpha_{12} \\ \alpha_{13} \\ \beta_1 \\ \alpha_{21} \\ \alpha_{22} \\ \alpha_{23} \\ \beta_2 \\ \alpha_{31} \\ \alpha_{32} \\ \alpha_{33} \\ \beta_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} X_c(i)X_c(i-1) \\ \sum_{i=1}^{N} X_c(i)Y_c(i-1) \\ \sum_{i=1}^{N} X_c(i)Z_c(i-1) \\ \sum_{i=1}^{N} X_c(i) \\ \sum_{i=1}^{N} Y_c(i)X_c(i-1) \\ \sum_{i=1}^{N} Y_c(i)Y_c(i-1) \\ \sum_{i=1}^{N} Y_c(i)Z_c(i-1) \\ \sum_{i=1}^{N} Y_c(i) \\ \sum_{i=1}^{N} Z_c(i)X_c(i-1) \\ \sum_{i=1}^{N} Z_c(i)Y_c(i-1) \\ \sum_{i=1}^{N} Z_c(i)Z_c(i-1) \\ \sum_{i=1}^{N} Z_c(i) \end{bmatrix} \tag{6.15}$$

$$M = \begin{bmatrix} \sum_{i=1}^{N} X_c(i-1)X_c(i-1) & \sum_{i=1}^{N} X_c(i-1)Y_c(i-1) & \sum_{i=1}^{N} X_c(i-1)Z_c(i-1) & \sum_{i=1}^{N} X_c(i-1) \\ \sum_{i=1}^{N} Y_c(i-1)X_c(i-1) & \sum_{i=1}^{N} Y_c(i-1)Y_c(i-1) & \sum_{i=1}^{N} Y_c(i-1)Z_c(i-1) & \sum_{i=1}^{N} Y_c(i-1) \\ \sum_{i=1}^{N} Z_c(i-1)X_c(i-1) & \sum_{i=1}^{N} Z_c(i-1)Y_c(i-1) & \sum_{i=1}^{N} Z_c(i-1)Z_c(i-1) & \sum_{i=1}^{N} Z_c(i-1) \\ \sum_{i=1}^{N} X_c(i-1) & \sum_{i=1}^{N} Y_c(i-1) & \sum_{i=1}^{N} Z_c(i-1) & 1 \end{bmatrix} \tag{6.16}$$

It can be noticed that the predictor algorithm is divided into two parts: (1) At start up of the experiment, the predictor uses as many sensed positions of object as necessary to satisfy a convergence criterion. If the distance vector between the predicted position at time t

and the current position of the object is less than a certain value (say ϵ), then convergence is achieved. Otherwise, more sensed positions of the object are needed to meet the error criterion and more time is required in order for the predictor to generate a reasonably accurate prediction; (2) Once the parameter approximation is done, the predictor starts sending signals to the control system. The predictor receives current positions of the object (at time t) and sends to the control system the predicted values at times $t + T$ and $t + 2T$ where T = sampling period. The predictor also updates the value of the motion parameters at each sampling instant.

6.4 Double-loop Feedback Scheme: observer-based

Position error alone as input to the control system cannot achieve tracking nor stabilise the closed-loop system. The feedback of the gripper velocity was added into the design, i.e. *double-loop feedback* controller. Since the velocity state is assumed not to be available for measurement, an observer had to be designed in order to generate estimates of the velocity. This was achieved by selecting identical initial values for the estimates to be equal to the true state. This is possible since the initial state of the gripper is known to be motionless. Thus, the observer equations are as presented below.

$$\hat{X}_2(0) = \bar{X}_2(0) = X_c(0) \tag{6.17}$$

$$\hat{X}_2(k+1) = \hat{X}_2(k) + TU(k) + K_e\left[Y(k+1) - X_1(k) - \frac{T^2}{2}U(k) - T\hat{X}_2(k)\right] \tag{6.18}$$

Note that the observer is of first order. Since the dynamics of the gripper are decoupled, the motion is divided into three independent components. Thus, we have three first order observers (i.e. one for each Cartesian coordinate). Due to the choice of the initial state, the gain K_e can be arbitrary since the bracketed terms it multiplies have their sum equal to zero at each sampling instant. Still, for the sake of rigour, the gain K_e was calculated to be $1/(2T)$ due to the choice of the poles of the state error equation which was conveniently placed at $z = 0.5$. By so doing, the speed of the observer was made faster than that of the robot system. The overall design block diagram of the visual tracking scheme is shown in

Fig. 6.3 which is based on the *observer* and *predictor* configurations. This control scheme was implemented in software form. The control law used is quite simple and is given by:

$$U(k) = K\,[E_1(k) - E_2(k)] = K\,[E_1(k) - T_d X_2(k)] \tag{6.19}$$

Algorithm 6.2 (Outline of the computer control algorithm)

1. *Initialisation of variables;* $m = 0$;
2. *Receive predicted position of object:* $m = m + 1$;
3. *Compute the states* $x_1(m), y_1(m), z_1(m)$ *and the inputs;*
4. *Compute the estimated states* $x_2(m), y_2(m), z_2(m)$ *from the observer equation;*
5. *Go to step 2 (outputs of the system are thus generated up to the end of experiment).*

The computer was used to compute recursively the value of the states of the gripper and of the observer. The algorithm used for the controller is as summarised in Algorithm 6.2 and its program written for MATLAB software simulations is given in Appendix B. During simulations, the choice of the gains K and T_d was based on the desired properties and characteristics of the overall closed-loop system such as stability, speed of response and settling time. However, it was found that values of $K = 1$ and $T_d = 1.5$ gave a dead-beat response to a step input. Still, the system subjected to other types of inputs will not exhibit dead-beat response. This is of no major consequence as long as the output response of the closed-loop system is stable, has a relatively small overshoot and settles down in a reasonable amount of time. Finally, in order to achieve smooth tracking, K was reduced to 0.6.

6.5 Initial Stability Analysis

For stability reasons, the overall closed-loop control system was initially analysed. The graph of the *root locus* based on a T_d value of 1.5 with respect to a varying gain K is shown in Fig. 6.4. It has to be noted that this plot is obtained by neglecting the observer dynamics since the state equation of the observer is identical to that of the robot system for state $X_2(k)$. From Fig. 6.4, it can be noted that the closed-loop transfer function with a gain of

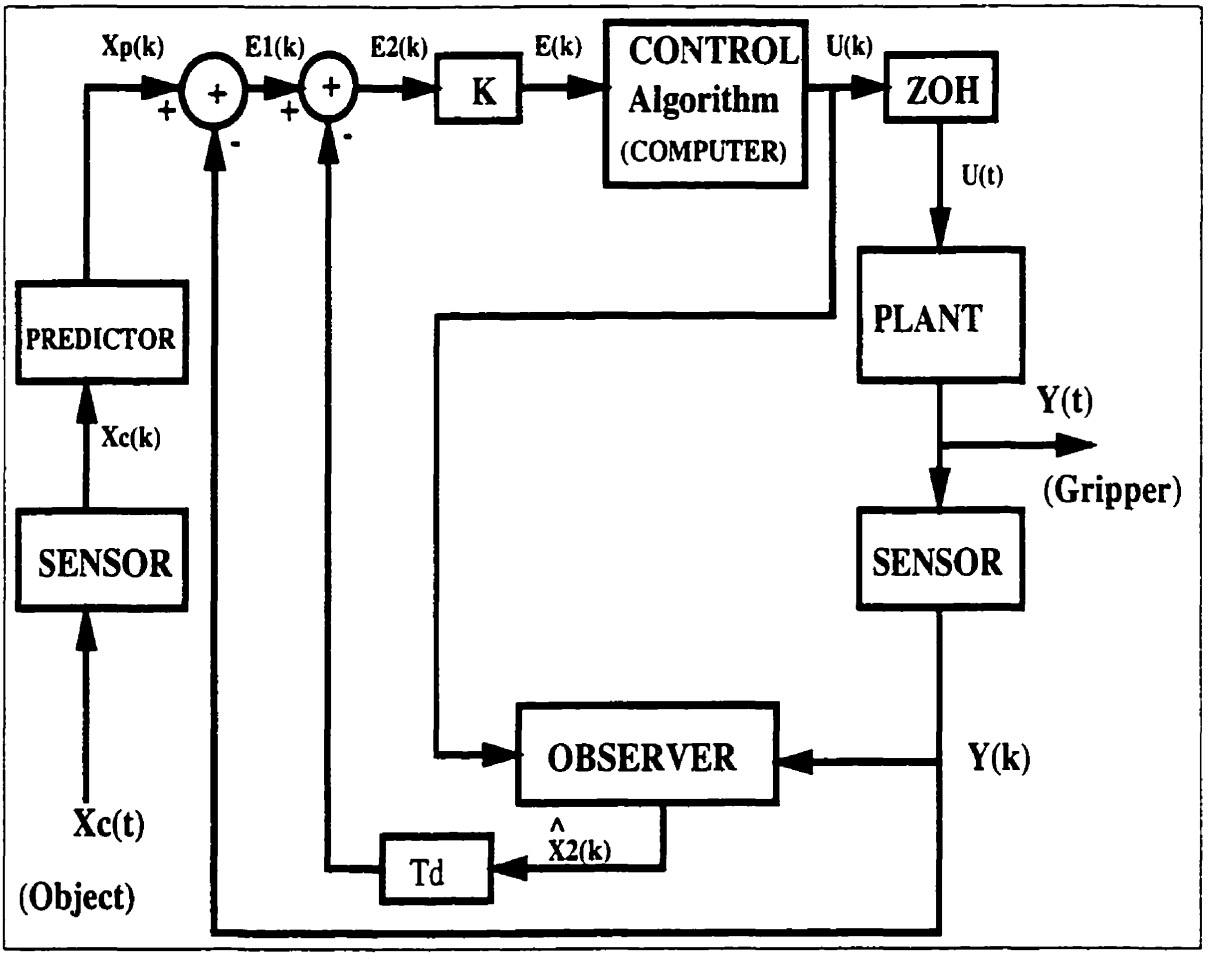


Figure 6.3: Block diagram of tracking control system

$K = 0.6$ corresponds to a point on the root locus lying inside the unit circle. This implies that the stability criterion is met and smooth tracking can be achieved. Thus, this gain value was chosen for further manipulations and was kept constant throughout the rest of the experiment. A complete derivation of the overall closed-loop transfer function is given in Section 6.8. Detailed experimental simulation results is covered in Section 9.4.

The rank of the matrix M depends on the type of motion taken by the moving object along the x-, y- and z-directions. Since the motion of the object was confined to the XY plane, the z-component has no contribution to the computation of the parameters. Thus, the rank of the matrix M was reduced from size four to three. If the motion of the moving object along the x-axis is linearly dependent on that along the y-axis, the rank of the matrix M is further reduced to size two. The choice of *ramp-, circular-, square- and wave-like* motions (see Section 9.4) was meant to prevent another reduction of the rank of matrix M. The algorithm

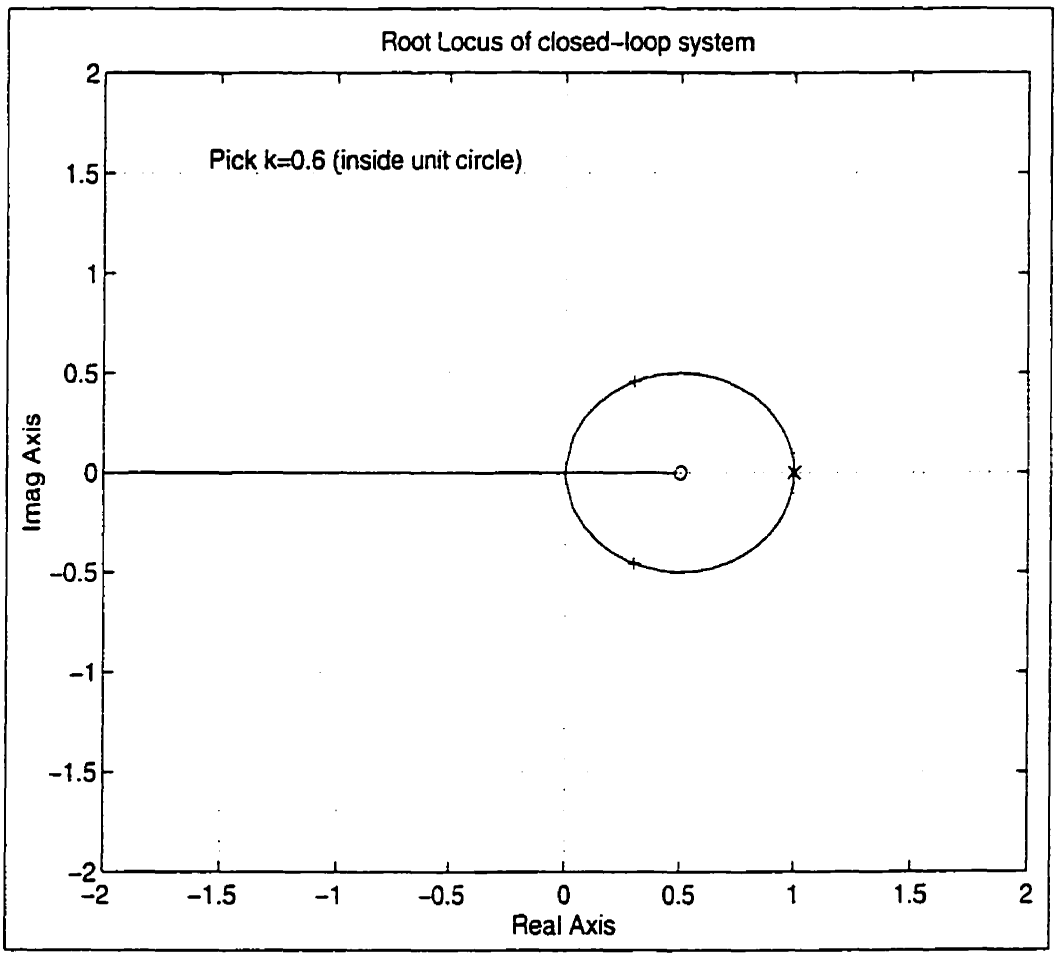


Figure 6.4: Root Locus of the closed-loop system

for the predictor has been set to check the rank of the matrix M before computation of the parameters is done. This step is necessary at the initialization stage. The accuracy of the predictor depends on the tolerance specified and complexity of the trajectory of the moving object. Prediction of future positions of the moving object can be done for one or more sampling instants ahead. At present time, the only information available to the predictor about the motion of the moving object is its present and past positions. In order to predict more than one sampling instant ahead, the predictor has to use the last prediction, present and past positions. This situation is desired in order to take into account the delays present in the overall system. However, it was found that two step prediction gives more accurate tracking of the moving object.

The time it takes for the gripper to go to the moving object and move together with it depends on the trajectory of the object. Since the predictor algorithm requires a certain num-

ber of actual positions of the object in order to estimate the motion parameters, the gripper cannot move before this calculation is completed. This period of time is approximately 3 seconds (refer to Section 9.4). In fact, it takes three steps of batch computation to converge within the desired accuracy of 0.1 units of position. Once the gripper starts moving, it is very fast to catch the object. This is true for the proportional feedback controller which has no parameter penalizing the size of its output. Hence, the error between estimates of object positions and gripper positions is going to be within the pre-specified tolerance rather quickly. However, the error between actual position of the object and gripper does not necessarily go to zero depending on the predictor's accuracy. Thus, a need for improvement using linear quadratic (LQ) controller is required as outlined below.

6.6 Improved LQ Controller

A K value of 0.6 was used for proportional feedback control. Still, the gripper takes more time to stabilize its motion around the object's position as compared to the case where K was larger. So varying K is changing the stability configuration of the system, the smoothness of the motion, as well as the demand on the inputs. This can be related to the performance index of a *Linear Quadratic* (LQ) controller. A compromise between speed of response and control effort can be found and this resulting compromise affects the shape of the motion: smoother control inputs implies smoother motion of the gripper in time. An improvement of the design would be to use LQ control. The controller drives the robot via forces in each Cartesian direction. Since the goal of tracking is to make the position and the relative velocity between the gripper and the object go to zero quickly, it is possible that an excessive demand be placed on the actuators. The purpose of the LQ controller is to reduce the effect imposed on the actuators so as to avoid saturations. The major drawback of such a controller is the slowing down of the tracking process. Still, the performance index J_{LQ} can be defined and offers the possibility of compromising between speed of tracking and demand on the control input.

$$J_{LQ} = \sum_{k=0}^{N} (\vec{X}^T Q_1 \vec{X}(k) + Q_2 U^2(k)) \tag{6.20}$$

where

$$N = \frac{Finite\ time\ of\ experiment}{Sampling\ period\ T} \tag{6.21}$$

The matrix Q_1 and scalar Q_2 are given as follows:

$$Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}; \quad Q_2 = \Phi \tag{6.22}$$

The greater Φ is, the smaller is the control input (force). A compromise between a reasonable speed of tracking and a maximum input force must be selected using Φ. Concerning stability, since the robot system is controllable and that Q_1 is symmetric and positive semi-definite, the LQ controller always gives a stable closed-loop system. The computer control algorithm uses a proportional controller with output feedback regulation of a reference input. In order to apply LQ control to the robot system, the block diagram form of Fig. 6.3 would have to be changed so that no reference input appears and that the state is fed back to the input of the controller. To achieve this new state space representation of the system, states would have to be redefined as follows:

$$\Delta\vec{X}(t) = (\Delta X(t)\Delta\dot{X}(t))^T \tag{6.23}$$

where

$$\Delta X(t) = X_g(t) - X_p(t); \quad \Delta\dot{X}(t) = \dot{X}_g(t) - \dot{X}_p(t) \tag{6.24}$$

It is assumed that the moving object has no acceleration. Thus, the relative acceleration between the object and the gripper would be equal to the acceleration of the gripper alone. The main difference between the LQ controller and the proportional feedback controller is that the gain K is a function of time, characterized by $K(k)$. Therefore, the LQ controller would be found by calculating the gain $K(k)$ at each sampling instant such that the performance

index J_{LQ} is minimized. The control input would be calculated as follows:

$$U(k) = -K(k)\Delta\vec{X}(k) \tag{6.25}$$

It has to be noted that all simulations were performed only in the discrete-time domain. The control input was held constant between sampling instants by the ZOH. Still, the robot motion is in the continuous-time domain. In discrete-time, the behaviour of a system is considered only at the sampling instants. It is known that a system can have a stable discrete output response and be unstable between samples in the continuous-time domain. Therefore, inter-sampling behaviour has to be addressed so as to check the continuous-time output response of the system. If the number of data needed to achieve convergence of the batch computation is large, then some form of weighted recursive calculation has to be performed so as to reduce the computation time. To design a more realistic control system, the recursive computation could be optimized with respect to its complexity and time consumption. Another addition to the control system would be a *disturbance-rejection* feature. Disturbance could be modelled based on friction or drag present in any real physical environment. In presence of no physical obstacles, the gripper tends to keep its velocity constant once it has caught up to the moving object. To reject a disturbance input to the control system, another compensator would have to be added into the design.

6.7 Motion Estimation of Moving Object

A prediction algorithm used to estimate the motion of a moving object was discussed in Section 6.3. The motion of the centroid of the moving object can be expressed as:

$$\vec{X}(k+1) = \alpha\vec{X}(k) + \beta \tag{6.26}$$

or equivalently

$$\begin{bmatrix} X_c(k+1) \\ Y_c(k+1) \\ Z_c(k+1) \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} X_c(k) \\ Y_c(k) \\ Z_c(k) \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \tag{6.27}$$

where $\vec{X}(k)$ is the position vector of the object. Using the present and past positions of the moving object, its future position is estimated by computing the parameters of the matrices α and β by minimizing the cost function J:

$$J = \sum_{i=1}^{N} (\vec{X}(i) - \alpha\vec{X}(i-1) - \beta)^2 \tag{6.28}$$

In x-coordinate, the cost function is given by:

$$J_x = \sum_{i=1}^{N} (x(i) - \alpha_{11}x(i-1) - \alpha_{12}y(i-1) - \alpha_{13}z(i-1) - \beta_1)^2 \tag{6.29}$$

Differentiating with respect to α_{11} yields:

$$\frac{\delta J_x}{\delta \alpha_{11}} = -2\sum_{i=1}^{N} (x(i) - \alpha_{11}x(i-1) - \alpha_{12}y(i-1) - \alpha_{13}z(i-1) - \beta_1) \tag{6.30}$$

Equating to zero and rearranging terms, yields:

$$\sum_{i=1}^{N} x(i)x(i-1) = \alpha_{11}\sum_{i=1}^{N} x(i-1)x(i-1) + \alpha_{12}\sum_{i=1}^{N} x(i-1)y(i-1) + \alpha_{13}\sum_{i=1}^{N} x(i-1)z(i-1) + \beta_1\sum_{i=1}^{N} x(i-1) \tag{6.31}$$

The procedure is repeated for the rest of the parameters (i.e. α and β) in J_x, J_y and J_z to obtain a total system of twelve linear equations as follows:

$$\sum_{i=1}^{N} x(i)y(i-1) = \alpha_{11}\sum_{i=1}^{N} x(i-1)y(i-1)+\alpha_{12}\sum_{i=1}^{N} y(i-1)y(i-1)+\alpha_{13}\sum_{i=1}^{N} y(i-1)z(i-1)+\beta_1\sum_{i=1}^{N} y(i-1) \tag{6.32}$$

$$\sum_{i=1}^{N} x(i)z(i-1) = \alpha_{11}\sum_{i=1}^{N} x(i-1)z(i-1)+\alpha_{12}\sum_{i=1}^{N} z(i-1)y(i-1)+\alpha_{13}\sum_{i=1}^{N} z(i-1)z(i-1)+\beta_1\sum_{i=1}^{N} z(i-1) \tag{6.33}$$

$$\sum_{i=1}^{N} x(i) = \alpha_{11}\sum_{i=1}^{N} x(i-1) + \alpha_{12}\sum_{i=1}^{N} y(i-1) + \alpha_{13}\sum_{i=1}^{N} z(i-1) + \beta_1 \tag{6.34}$$

$$\sum_{i=1}^{N} y(i)x(i-1) = \alpha_{21}\sum_{i=1}^{N} x(i-1)x(i-1)+\alpha_{22}\sum_{i=1}^{N} x(i-1)y(i-1)+\alpha_{23}\sum_{i=1}^{N} x(i-1)z(i-1)+\beta_2\sum_{i=1}^{N} x(i-1) \tag{6.35}$$

$$\sum_{i=1}^{N} y(i)y(i-1) = \alpha_{21}\sum_{i=1}^{N} x(i-1)y(i-1)+\alpha_{22}\sum_{i=1}^{N} y(i-1)y(i-1)+\alpha_{23}\sum_{i=1}^{N} y(i-1)z(i-1)+\beta_2\sum_{i=1}^{N} y(i-1) \tag{6.36}$$

$$\sum_{i=1}^{N} y(i)z(i-1) = \alpha_{21}\sum_{i=1}^{N} x(i-1)z(i-1)+\alpha_{22}\sum_{i=1}^{N} z(i-1)y(i-1)+\alpha_{23}\sum_{i=1}^{N} z(i-1)z(i-1)+\beta_2\sum_{i=1}^{N} z(i-1) \tag{6.37}$$

$$\sum_{i=1}^{N} y(i) = \alpha_{21}\sum_{i=1}^{N} x(i-1) + \alpha_{22}\sum_{i=1}^{N} y(i-1) + \alpha_{23}\sum_{i=1}^{N} z(i-1) + \beta_3 \tag{6.38}$$

$$\sum_{i=1}^{N} z(i)x(i-1) = \alpha_{31}\sum_{i=1}^{N} x(i-1)x(i-1)+\alpha_{32}\sum_{i=1}^{N} x(i-1)y(i-1)+\alpha_{33}\sum_{i=1}^{N} x(i-1)z(i-1)+\beta_3\sum_{i=1}^{N} x(i-1) \tag{6.39}$$

$$\sum_{i=1}^{N} z(i)y(i-1) = \alpha_{31}\sum_{i=1}^{N} x(i-1)y(i-1)+\alpha_{32}\sum_{i=1}^{N} y(i-1)y(i-1)+\alpha_{33}\sum_{i=1}^{N} y(i-1)z(i-1)+\beta_3\sum_{i=1}^{N} y(i-1) \tag{6.40}$$

$$\sum_{i=1}^{N} z(i)z(i-1) = \alpha_{31}\sum_{i=1}^{N} z(i-1)x(i-1)+\alpha_{32}\sum_{i=1}^{N} z(i-1)y(i-1)+\alpha_{33}\sum_{i=1}^{N} z(i-1)z(i-1)+\beta_3\sum_{i=1}^{N} z(i-1) \tag{6.41}$$

$$\sum_{i=1}^{N} z(i) = \alpha_{31}\sum_{i=1}^{N} x(i-1) + \alpha_{32}\sum_{i=1}^{N} y(i-1) + \alpha_{33}\sum_{i=1}^{N} z(i-1) + \beta_3 \tag{6.42}$$

Therefore, these 12 equations (i.e. Equations 6.31 through 6.42) were used to solve for the parameters as shown in Equation 6.15. Since, from the assumption, the motion of the object is confined within XY plane, then all terms containing a z-coordinate vanish. Hence, the dimension of the matrix M is reduced to 3 by 3.

6.8 Derivation of Overall Transfer Function

The overall block diagram of the tracking control system is shown in Figure 6.3. Using this diagram, the closed-loop transfer function can be obtained as follows (neglecting the dynamics of the observer):

$$E_1^*(s) = X_p^*(s) - Y^*(s)$$

$$E^*(s) = K\left[-T_d X_2^*(s) + X_p^*(s) - Y^*(s)\right]$$

$$E^*(s) = U^*(s)$$

However;

$$Y(s) = G_{ZOH}(s).G_{plant}(s).U(s) \tag{6.43}$$

Where;

$G_{ZOH}(s) = \frac{1-e^{sT}}{s}$ where T is the sampling period;

$G_{plant}(s) = \frac{1}{s^2}$ (for each coordinate axis); and

$G_{observer}(s) = s$, i.e. $X_2^*(s) = [sY(s)]^*$

So;

$$Y(s) = \frac{(1 - e^{-sT})}{s}.(\frac{1}{s^2}).U^*(s) \tag{6.44}$$

Starring;

$$Y(s) = \left[1 - e^{-sT}\right]^*.U^*(s).(\frac{1}{s^3})^* \tag{6.45}$$

Now,

$$\begin{aligned} X_2(s) &= sY(s) \\ &= s\left[\tfrac{(1-e^{-sT})}{s}.(\tfrac{1}{s^2}).U^*(s)\right] \\ &= (1 - e^{-sT}).(\tfrac{1}{s^2}).U^*(s) \end{aligned}$$

Starring;

$$X_2^*(s) = \left[1 - e^{-sT}\right]^*.U^*(s).(\frac{1}{s^2})^* \tag{6.46}$$

Using the fact that $U^*(s) = E^*(s)$, we get:

$$K\left[-T_d X_2^*(s) + X_p^*(s) - Y^*(s)\right] = \frac{Y^*(s)}{[1 - e^{-sT}]^*(\frac{1}{s^3})^*} \tag{6.47}$$

Regrouping terms we get:

$$K\left[-T_d X_2^*(s) + X_p^*(s)\right] = Y^*(s)\left[K + \frac{1}{[1 - e^{-sT}]^*(\frac{1}{s^3})^*}\right] \tag{6.48}$$

Since $X_2^*(s)$ is known (from Equation 6.46), substituting and rearranging it, gives:

$$\frac{Y^*(s)}{X_p^*(s)} = \frac{K(\frac{1}{s^3})^*[1 - e^{-sT}]^*}{K[1 - e^{-sT}]^*[\frac{1}{s^3}]^* + 1 + KT_d[\frac{1}{s^2}]^*[1 - e^{-sT}]^*} \tag{6.49}$$

Now in terms of *z-transforms* we get:

$$Z\left\{(1 - e^{-st})^*\right\} = 1 - z^{-1} = \frac{z-1}{z}$$

$$Z\left\{[\frac{1}{s^3}]^*\right\} = \frac{T^2 z(z+1)}{2(z-1)^3}$$

$$Z\left\{[\frac{1}{s^2}]^*\right\} = \frac{Tz}{(z-1)^2}$$

The pulse transfer function of the closed-loop system is thus given as follows:

$$\frac{Y(z)}{X_p(z)} = \frac{K\frac{T^2 z(z+1)}{2(z-1)^3} \cdot \frac{(z-1)}{z}}{K[\frac{z-1}{z}][\frac{T^2 z(z+1)}{2(z-1)^3}] + 1 + KT_d\frac{Tz}{(z-1)^2} \cdot (\frac{z-1}{z})} \tag{6.50}$$

By reducing the expression we get:

$$\frac{Y(z)}{X_p(z)} = \frac{\frac{KT^2}{2}(z+1)}{\frac{KT^2}{2}(z+1) + (z-1)^2 + T_d KT(z-1)} \tag{6.51}$$

By factorization and regrouping terms we get:

$$F(z) = \frac{Y(z)}{X_p(z)} = \frac{\frac{KT^2}{2}(z+1)}{z^2 + z(\frac{KT^2}{2} + T_d KT - 2) + (1 + \frac{KT^2}{2} - T_d KT)} \tag{6.52}$$

$F(z) = \frac{Y(z)}{X_p(z)}$ is the closed-loop transfer function of the proportional feedback control system. All the constants (i.e. K, T & T_d) were selected accordingly (see section 6.4) and substituted into Equation 6.52 before the initial stability analysis was done using *root-locus* method (see section 6.5).

Chapter 7 Managing Telecommands using CDBs

7.1 Overview & Rationale of CDBs

Telecommand labels will be handled in real-time using *Common Data Base (CDB)*[1] [CAELIB., 1995s] which is a section of shared memory to which all simulation modules have access during run-time in real-time. The CDB concept is modelled after the Fortran Common Block. It is defined by CDB declaration file that is processed into a set of binary files which are used during run-time to access the telecommand data elements. The simulation models export their CDB labels by embedding data import/export statements. The Visualization And Display (VAD) item and the simulator communicate with all of their interface data elements through the simulation CDB interface. All data that is used to control the simulator must be exported through the CDB by the Human-Computer Interface (HCI) pages. The list of HCI page variables is amongst the Telecommand Data Elements Table. The CDB interface between the VAD and the simulator hardware allows the simulation, video recording, playback and distribution equipment, as well as reading the hand controllers and switch panels. The CDB VAD interface for MOTS system (see Chapter 8) is as shown in Figure 7.1. The MOTS system will be used as a valid example of a teleprogramming system since the author is part of the team working on the project. The author worked as a system CDB integration specialist in which he was fully responsible in formulating and designing the CDB interface telecommand labels for all simulation models. The MOTS Interface Control Document (ICD) was used as a key guidance to formulate most telecommands and other relevant data elements needed for telemetry, etc. Furthermore, the author was responsible for their organization and scheduling all telecommand data elements in the simulation environment running SIMEX as discussed in Section 7.5. Finally, the author was responsible for the creation and processing of CDB source files known as CDB

[1] Refer to Appendix A for more details on CDB Utility.

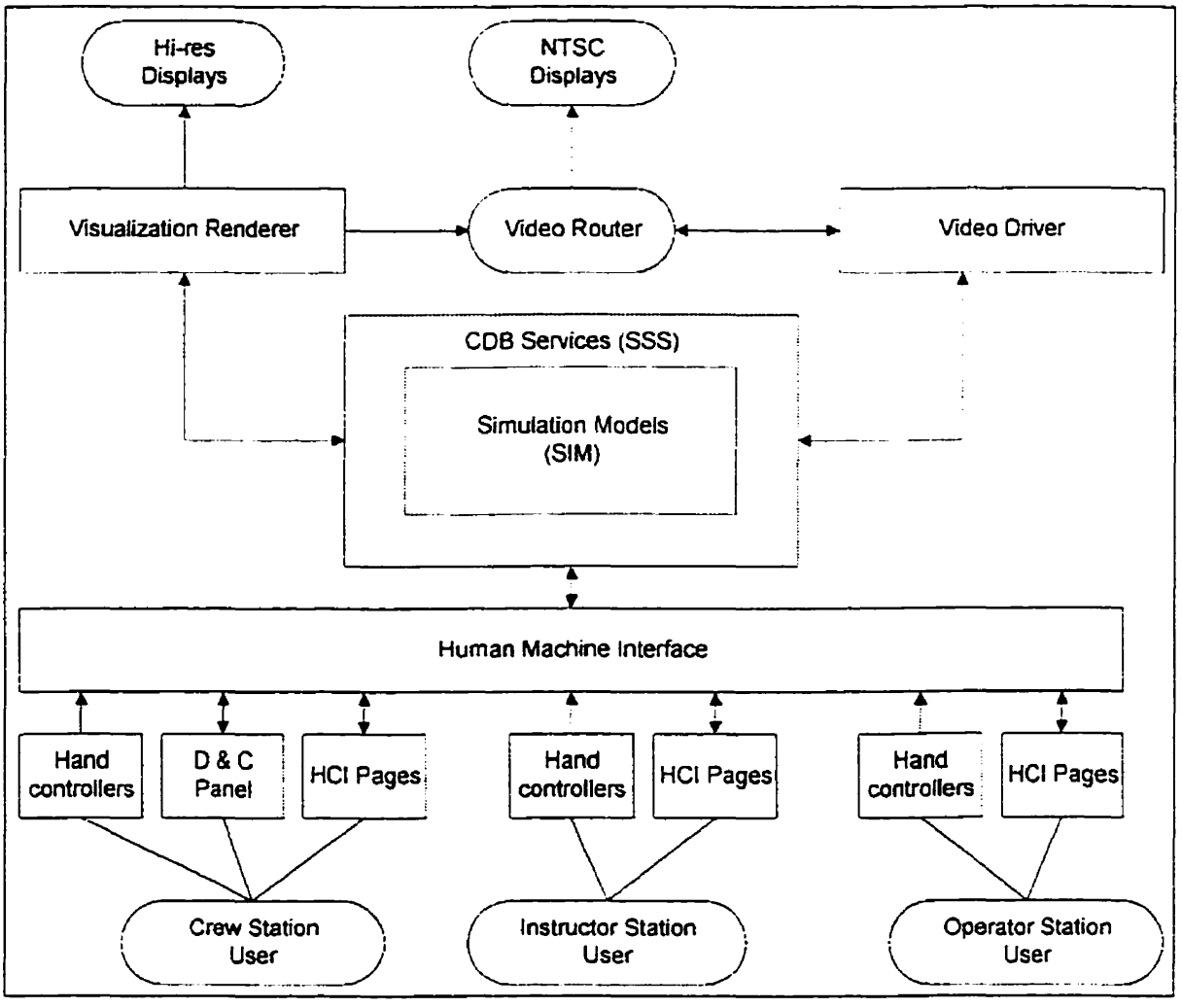


Figure 7.1: MOTS internal interfaces with CDB

bases (in total MOTS has 7 bases).

Source files containing all telecommands and their associated data are maintained, and are referred to as the CDB data elements. When processed the CDB has many support files. The *Common Data Base Processor (CDBP)* uses the CDB source file(s) to produce the block data file(s), the label file(s), and the index file(s). The label file contains all the essential information contained in the Common Database source file, plus the relative position of data items. The label file provides a detailed description for each data item, which can be retrieved using the Common Database access routines. The FORTRAN Pre-Compiler, the C Pre-Compiler and Instructor's Facilities programs make intensive use of these routines. Following any modification other than adding spare labels or telecommands using the *Common*

Database Spare Utility (CDBS), the Common Database source file must be re-processed by the CDBP. If the CDB is segmented and any of these segments has been recompiled by the CDBP utility, one must use the *XMerge* utility to rebuild the integral CDB to reflect the changes. Every FORTRAN/C program using the telecommand labels of the CDB source file must be recompiled and replaced in the configuration. The Common Database is structured into bases of related data. Each CDB segment requires one data block or base.

7.2 CDB-based Software Interface Structure

The source data definition line format for each telecommand data element or label comprises the following (including their respective column locations):

- **Compulsory information:**
  1. Label Name [up to 32 characters] (COL: 1-32),
  2. Data Type (COL: 33-36),
  3. Initialization and Alignment Boundary (COL: 33-36),
  4. Default Initial Value and Dimension (COL: 38-51),
- **Optional information:**
  1. Description (COL: 52-105),
  2. Circuit breaker bus (COL: 92-97),
  3. Interface Assignment (COL: 99-104),
  4. Unit (COL: 106-113),
  5. Display Format (COL: 114-118),
  6. RAP (Record And Playback) (COL: 120),
  7. Schematic or ID Number (COL: 122-128)

Each segment source file must be processed individually by CDBP to produce a set of files that are used by the foreground software or the background utilities. The data files are used to load the CDB into memory at simulator load time. When the user loads the simulator, a CDB-LOAD routine will use the CDB description inside that file to load the data into memory. The other files (except the log file) are used by some other CDB oriented utilities (e.g. *CDBS, CDBA, CDBCOM, Mini, XMerge, FPC and CPC*) to extract information about

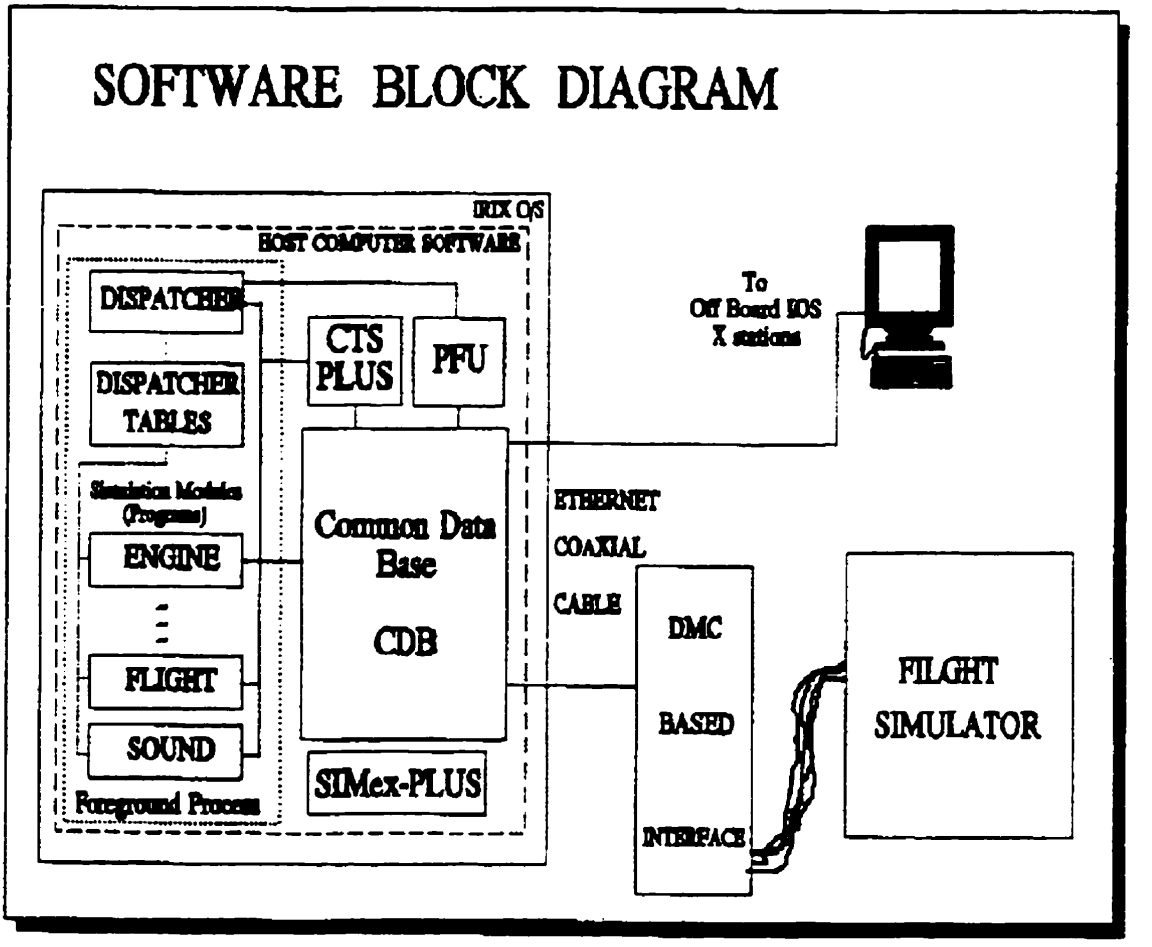


Figure 7.2: CDB-based interface to simulation softwares

the CDB segment. Communication between different simulation utilities which need to access real-time simulation data must be interfaced via CDB which is itself interfaced to the simulator via a Data Management Controller (DMC). A software organization structure for a general purpose flight simulator (such as MOTS) is as shown in Figure 7.2.

The software involved in MOTS simulation (as discussed in Chapter 8) is defined as a set of sub-programs, data, input/output and utilities used to operate and maintain the simulation. This includes both the operating system software and a written software that is specific to MOTS simulation environment. Figure 7.2 shows the interaction of software and utilities in the simulation environment. The software development environment was designed to aid the user during the development life cycle of real-time software. It consists of several house keeping tools, a screen editor, several CDB tools, a Pre-Compiler and Compiler (for both

FORTRAN & C/C++), a run-time executive, a run-time library, a debugging tool, a configuration management tool and performance monitoring tool. These tools are designed primarily to reduce engineering costs and provide the user with a flexible, attractive, state-of-the-art software environment. They are closely integrated through the use of global common database, logical names and file revision handling system. Most tools support the following features: telecommand recall, line editing, online help and input re-direction. The software is re-usable for a wide range of real-time applications and can be available on VMS/VME and most UNIX computer systems.

7.3 CDB Management and Task Scheduling

A configuration is a set of vital resources (software, data, processors) that is currently used to control the simulation. Each configuration is defined in a configuration data base binary file used by the SIMEX-Plus utility. For each configuration a user can load alternate modules, protect files, monitor the configuration history and automatic building of simulation load modules. The mother process (MOM) is the heart of simulation. It is the memory resident high priority real-time process which is started up at boot time. It also has access to operating system call routines. Its functions are to: monitor the simulation status and inform users via system messages; communicate with other background tasks; execute operation commands for SIMEX such as: *LOAD, FREEZE, RUN, etc.* Moreover, there will be auxiliary mother processes (i.e. MOMn) for each simulator CPUn which reports to the Host Computer Mother task (CPU0). The relationship between users and the Mother Process is as shown in Figure 7.3 using a Data Management Controller (DMC) with Digital Force Control (DFC) algorithms. All simulation communication interfaces are through a shared memory in the form of CDB data elements or labels.

The run-time environment consist of the *Application Software*, the *Executive Software* and the *Run-Time* library. In order to meet real-time constraints required for real-time simulation, all simulation modules cannot be run at once. It then becomes necessary to prioritise sub-programs so enough spare time will be left for background activities to take place. Hence, it is mandatory to maintain a **scheduler or dispatcher** tables which prioritise the simulation modules forcing some modules to run at a lower apparent rate than oth-

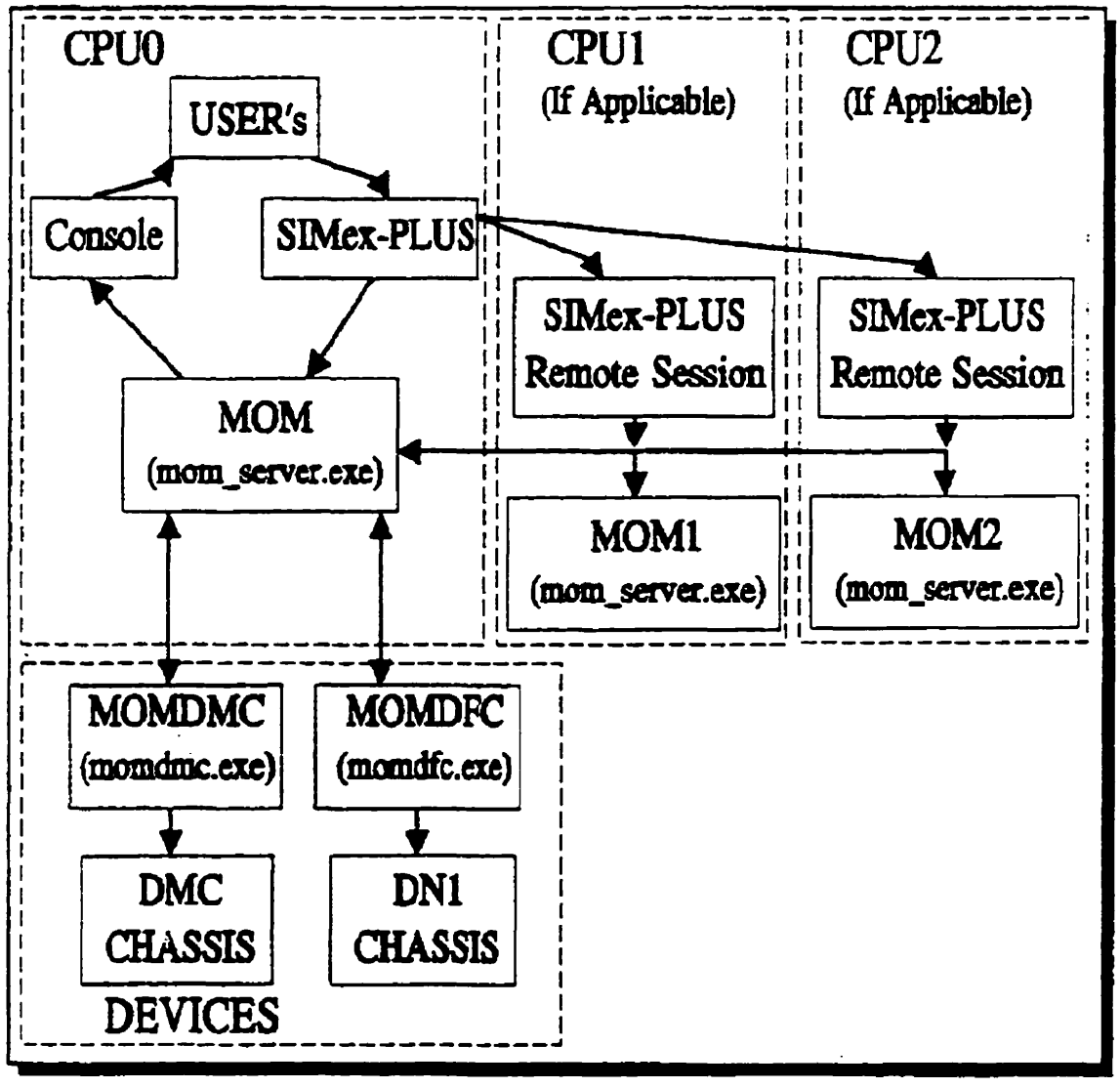


Figure 7.3: MOM communication hierarchy

ers. This technique provides enough spare time so that effective work can be completed on the computer even if the simulator is running. The Executive Software consists of the **"main"** program of each real-time process (i.e. the Dispatcher) and servers that process incoming requests from the *PerFormance monitoring Utility* (i.e. PFU) or from the *Computerized Test System* (i.e. CTS-PLUS) tool. The Dispatcher functions' include to schedule sub-systems and compute run-time statistics such as CPU spare time and over-runs.[2] The

[2]**Over-runs** occur when there is insufficient CPU time to meet hard deadlines, i.e. the execution time of the synchronous process exceeds the basic frame time.

Dispatcher can be configured to meet **"hard deadlines"** (i.e. Synchronous Dispatcher) or **"soft deadlines"** (i.e. Asynchronous Dispatcher). A real-time system will require several sub-programs, some using the synchronous dispatcher and others using the asynchronous dispatcher.

Some of the computer's memory is loaded with executable instructions which are necessary to maintain the simulation. When several simulation modules and other related files are linked into one process, the process is known as a *task*. For purposes of the simulation there are two types of tasks: *synchronous & asynchronous* which are *foreground* tasks. The remaining time on any iteration is referred to as *background* (i.e. spare time). Foreground tasks have priority over background tasks, but within the foreground, synchronous tasks have a higher priority than asynchronous tasks. The CDB is organized in such a way that the variables are separated in two groups:

- **The restorable area.** This contains variables that describe the state of the simulation. Once restored, the simulation can be restarted from the point where the recording was made using the snapshot.
- **The non-restorable area.** This contains variables that are used by non-simulation modules, such as the RT Dispatcher, Snapshot and Data Gathering.

Communication through CDB also helps the interlocking mechanism between the operator input and the simulation models. In either case, the same CDB variables are used as input to the simulation module; the interlocking mechanism is transparent to the simulation modules. The same data is shared through the CDB by all sub-modules, which are executed at different rates within the synchronous and asynchronous dispatcher processes. The RT Dispatcher is responsible for controlling the execution of all simulation modules using three simulation processes; namely:

1. **sp0:- main synchronous process:**
    - synchronously dispatched;
    - solves the equations of motion;

- updates modal coordinates;
- computes generalized coordinates constraint forces, etc.;
- handles inputs; and
- handles transients.

2. **ap0:- main asynchronous process:**

   - asynchronously dispatched;
   - updates equations of motion; and
   - initializes simulation.

3. **ap1:- low priority asynchronous process:**

   - asynchronously dispatched; and
   - handles low priority functions such as I/O and snapshot.

The process **sp0** is scheduled each basic iteration period of the simulation. Its critical functions are performed within a super-band of **sp0** at the integration rate. Other functions are typically scheduled at the basic iteration rate or more slowly within a sub-band. This process is not interruptible by other modules. The process **ap0** is scheduled in order to perform functions at a regular rate of at least 1 Hz. The scheduling of **ap0** is controlled by **sp0** which allows a fixed number of iterations to elapse in between. The priority of **ap0** is lower than that of **sp0** so that critical computations, which can be very time consuming, do not prevent the simulation outputs from being delivered in real-time. The process **ap1** is used to perform low priority tasks such as snapshots. This process has the lowest priority and its scheduling is not controlled by **sp0**. Due to the fact that RT-DS functions are performed on separate processes, a buffering mechanism is required to ensure data integrity. Two sets of buffers are utilized, one for transfers from **sp0** to **ap0** referred to as *output* buffers. The second set used for transfers from **ap0** to **sp0** is referred to as *input* buffers. The proper synchronization of data transfers between processes is necessary to avoid run to run variation. This is achieved in RT-DS by swapping output buffers only before **sp0** schedules **ap0** and swapping input buffers a fixed number of iterations later. If **ap0** has not been completed before input buffers are to be swapped, an overrun is recorded[3].

[3]The overrun counter must always be zero to ensure simulation repeatability.

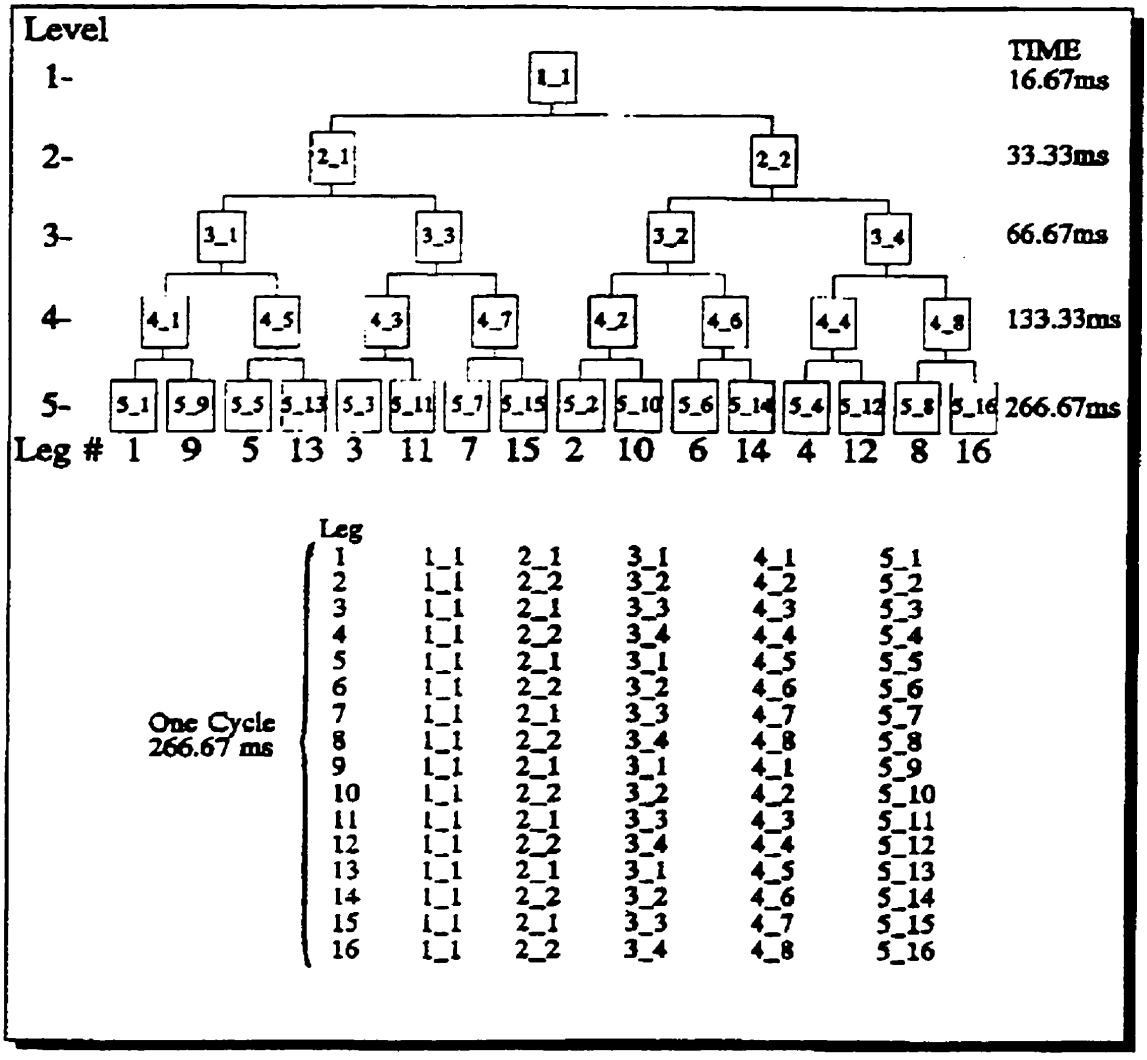


Figure 7.4: Synchronous banding scheme

7.4 Process Banding Scheme

The purpose of processes or tasks banding scheme is to pre-define the order and rate of execution for all available sub-programs in the simulation environment. Each task is subdivided into units known as bands where modules can be placed according to their average execution times. The proper placement of modules into appropriate bands is the key to providing the contractual amount of spare time. There are four levels of bands which start off at a base rate as described by the logicals **"system-frame"** and **"system-asched"** for the synchronous and asynchronous tasks respectively. When the simulator is running and the tasks are called upon, each task will start at its base rate and pick up a band from each level

```
;
; ----- Critical Band
;
crit_table:
          leg 1_1              ; DESCRIPTION OF CRITICAL BAND
end_crittable:
;
; ----- Non-Critical Bands
;
sync_table:
          leg  2_1 3_1 4_1 5_1; LEG COMPOSITION
          leg  2_2 3_2 4_2 5_2
          leg  2_1 3_3 4_3 5_3
          leg  2_2 3_4 4_4 5_4
          leg  2_1 3_1 4_5 5_5
          leg  2_2 3_2 4_6 5_6
          leg  2_1 3_3 4_7 5_7
          leg  2_2 3_4 4_8 5_8
          leg  2_1 3_1 4_1 5_9
          leg  2_2 3_2 4_2 5_10
          leg  2_1 3_3 4_3 5_11
          leg  2_2 3_4 4_4 5_12
          leg  2_1 3_1 4_5 5_13
          leg  2_2 3_2 4_6 5_14
          leg  2_1 3_3 4_7 5_15
          leg  2_2 3_4 4_8 5_16
end_synctable:

Inclusion of simulation modules into a band.

band2_1:
;     "PROG" STATEMENT USED TO INCLUDE SIMULATION MODULES
;       |     ENTRY POINT NAME OF SIMULATION MODULE
;       |      |      "FREEZE/UNFREEZE" FLAG
;       |      |      |
      prog avnla.u     ; Avionics EIS, FMS, CAWS mapping
      prog avnlg.u     ; Avionics GPWS bus mapping
      prog dind.f      ; ECS Instrumentation
      prog vturcof.f   ; Turbulence coefficient
      :    :     :
```

Figure 7.5: Sample synchronous program table

below it whose composition will be unique on each iteration. These unique compositions are called **"legs"**. It is important that the execution time of each leg is roughly the same within a task for balancing reasons.

The number of legs a task has will determine how many iterations are necessary to complete a **"cycle"**.[4] Figure 7.4 shows a sample of a synchronous banding scheme. Each band name is comprised of a level number and a placement number. For example, the band-name **"3-2"** is a third level band in second place. Note that the logical **"system-frame"** is set at 60 Hz (i.e. 16.67ms). A sample format of a synchronous program Table is shown in Figure

[4]A cycle is the minimum amount of iterations required to execute all modules within a task.

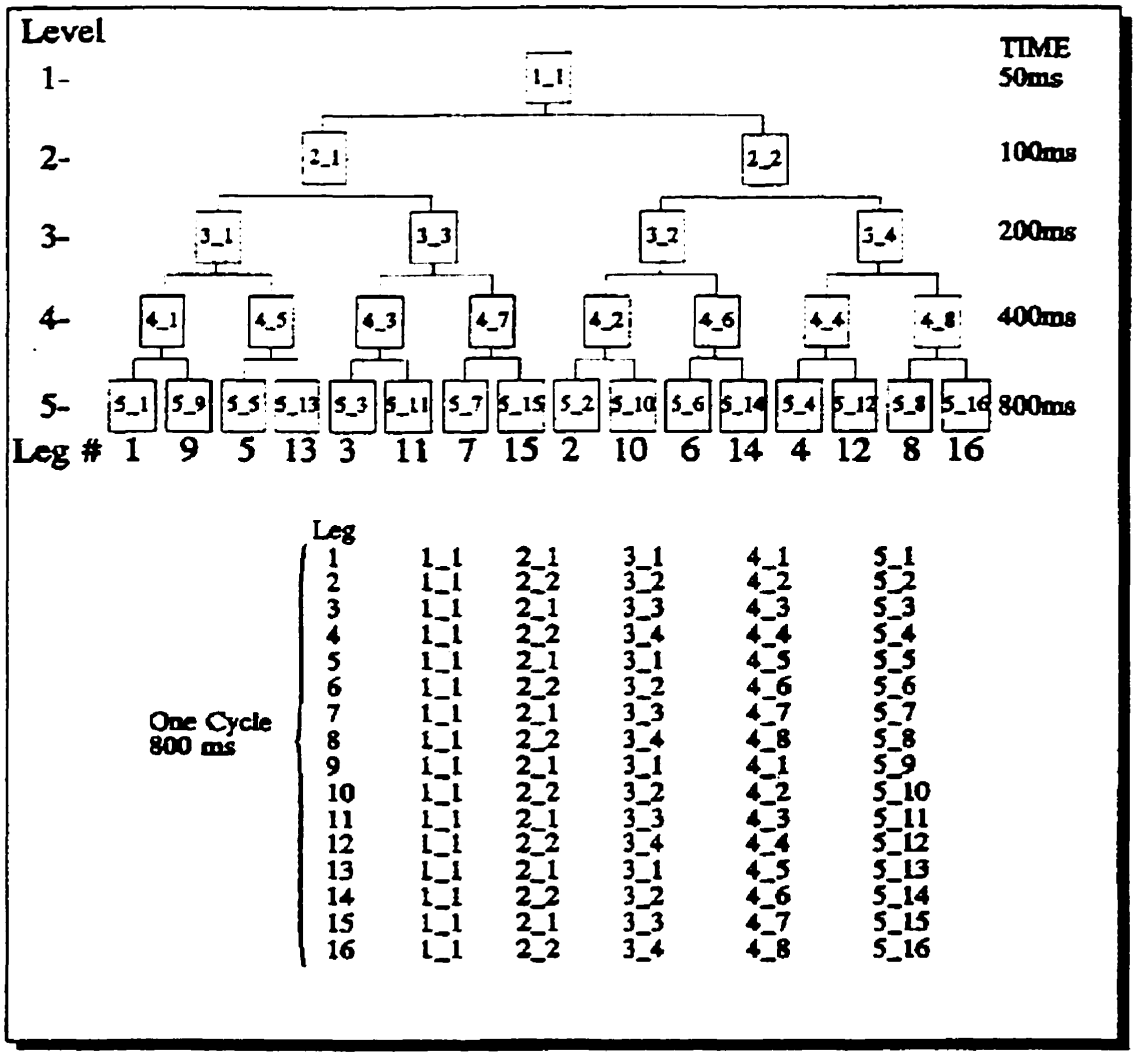


Figure 7.6: Asynchronous banding scheme

7.5. Similarly, Figure 7.6 and 7.7 show a sample of an asynchronous banding scheme and a corresponding program Table. In this case, the logical **"system-asched"** is set at 50 Hz (i.e. 50ms). Also note that at each iteration, the dispatcher traverses one leg of the tree from root to leaf, thus calling the modules in the visited nodes. During one complete cycle, all 16 legs are traversed in the indicated priorities (i.e. leg #1 will be computed first and leg #16 last). The user can modify the scheduler/dispatcher tables if a simulation module requires moving from one band to another to achieve balance and/or possibly more spare time.

```
;
; ----- Critical Band
;
crit_table:
end_crittable:
;
; ----- Non Critical Bands
;
sync_table:
          leg  1_1 2_1 3_1 4_1
          leg  1_1 2_2 3_2 4_2
          leg  1_1 2_1 3_3 4_3
          leg  1_1 2_2 3_4 4_4
          leg  1_1 2_1 3_1 4_5
          leg  1_1 2_2 3_2 4_6
          leg  1_1 2_1 3_3 4_7
          leg  1_1 2_2 3_4 4_8
end_synctable:
;*************************************
;       User update section           *
;*************************************
band1_1:
        aprog call_local_copy.u
        aprog xsca.u
        aprog gdmn.f                  ! W/X: Landmass i/o
        aprog glod.f                  ! W/X: Landmass i/o
        aprog host_ovp.u              ! IOS: OVP
band2_1:
        aprog tcnia.u                 ! IOS : cnia
        aprog x9da.f                  ! IOS: digital voice
        aprog x9play.f                ! IOS: digital voice

...

band5_9:
band5_10:
band5_11:
band5_12:
band5_13:
band5_14:
band5_15:
band5_16:
end:
```

Figure 7.7: Sample asynchronous program table

7.5 CDB Simulation Expert Utility

SIMex-Plus[5] is the primary utility used for the development, maintenance and operation of a CDB-based simulation software such as MOTS (as discussed in Chapter 8). It provides a controlled environment for the software life-cycle of the simulator. SIMex-Plus is responsible not just for storing the simulation software, but for maintaining a history of the evolution of the projects development and use, letting users know when and why steps were taken, and allowing them to return to previous steps. It is also responsible for the security and integration of the simulation, providing an environment in which users can follow a set

[5]SIMEX is an acronym for SIMulation EXpert.

of basic rules that will result in no loss of files of significance and no confusion about which files must operate together to create a configuration. It will protect and group files not only for organizational reasons, but to reduce the chance of accidental or malicious corruption or loss. Thus, the responsibilities of SIMex-Plus can be grouped into three categories:

- Configuration Control (i.e. configuration file management);
- Software Construction (i.e. *BUILDING* elements); and
- Simulation Operation (i.e. *LOAD, UNLOAD, RUN, SUSPEND, FREEZE*).

SIMex-Plus is a general purpose simulation manager and operator. The ability to use it on a wide variety of CDB-based simulators and on different computer systems with minimal alteration is highly desirable. To meet this objective, SIMex-Plus was designed to be transportable between projects and computer systems with no actual changes to the code. Functionality specific to the operation of the computer, such as File I/O, has been extracted into a separate general software package. Project specific information such as which compilers to use for building or the order of loading executable files has been extracted into a set of data files to be read by SIMex-Plus, and is known collectively as the *Specific Support Environment (SSE)*. The remainder is the set of functions standard to all applications of SIMex-Plus, such as all of the user telecommands and the user interface, and is contained in the code of the *General Management Program*. Some of the advanced features of SIMex-Plus include: *file reservation, smart invalidation, parallel build, source file versioning, external build & load, semaphore (purge, operation & configuration), etc.* Finally, with SIMex-Plus, one can easily perform numerous procedures for software maintenance such as add a new simulation module; modify an existing simulation module; delete a simulation module; reschedule a simulation module; change an initial CDB value; add a new CDB label; and CDB updates.

7.6 CDB-based Simulation Performance & Test Utilities

The Program Performance Utility (PFU) provides an accurate set of execution times of real-time programs. It allows the user to do individual time measurements for specific sub-programs or group of sub-programs during a pre-selected number of iterations. A summary containing timing information on all sub-programs can also be requested. Sub-programs

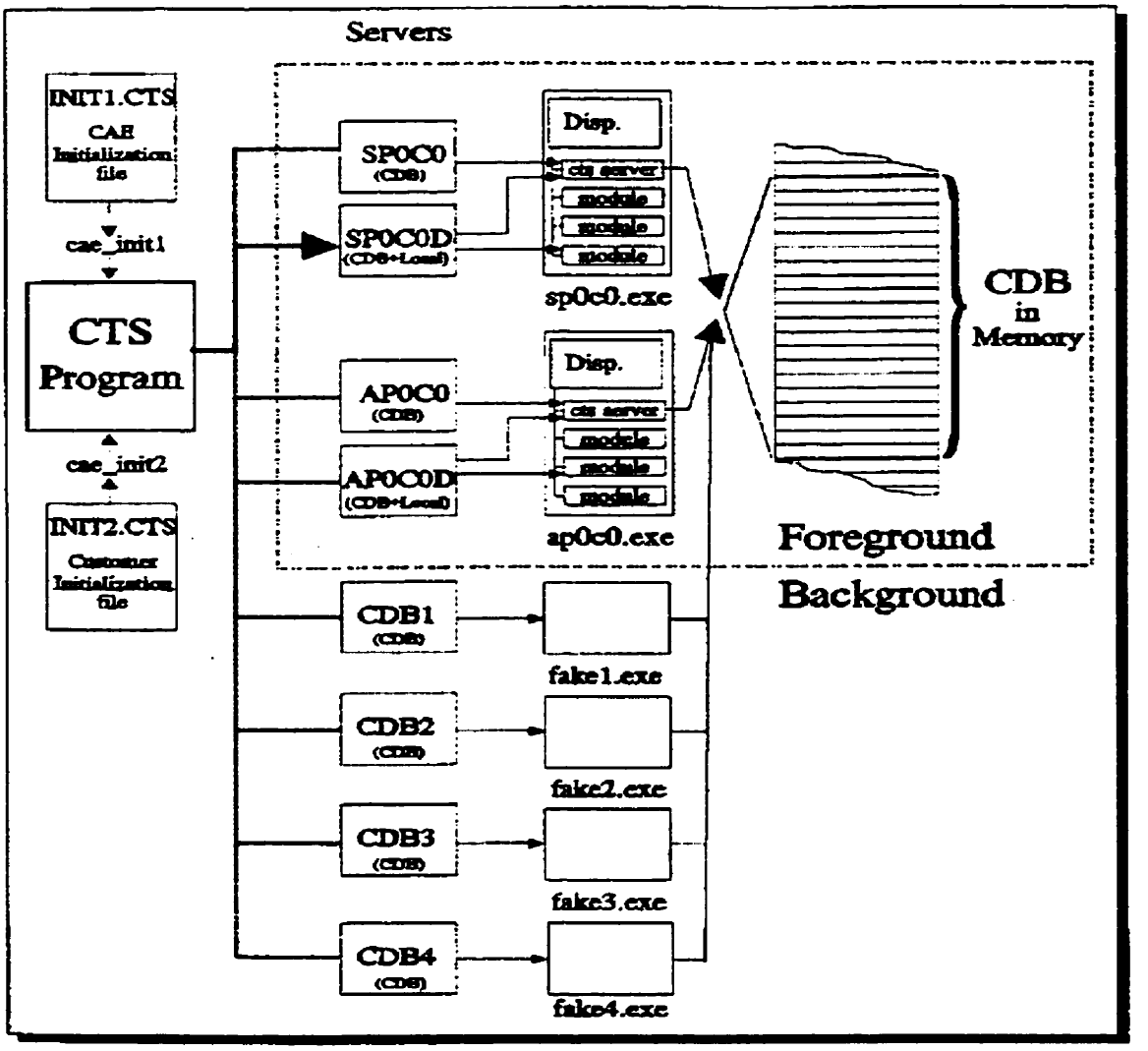


Figure 7.8: CTS linkage to processes and CDBs

can be linked or unlinked from the real-time process and moved from one execution band to another. Timings are available in the following forms: *band timing; iteration timing; sub-program timing; & summary table.* Band, leg and module sampling/monitoring displayed in tabular form with minimum, average and maximum time statistics. The total CPU time used, based on the basic iteration time, is also given in the summary sheet.

The Computerized Test System (CTS) is a powerful tool used to debug and test real-time software. It is designed to assist in development, debugging and validation of the simulator programs. It supports both interactive and telecommand file modes. CTS-Plus provides control over program execution, gives access to variables, and automates the test process.

It also records real-time data and produces tabular or graphic outputs. CTS-Plus can examine or modify, in real-time, local and global CDB variables interactively (using EXAMINE & DEPOSIT commands) or continuously (using MONITOR command). The variable values can be collected in memory (using COLLECT command) or continually updated (using DRIVE command). These values can be plotted on screen or paper (using PLOT & GRAPH commands) and/or saved on disk for future use (using PUT command).

The purpose of the CTS-Plus utility is to link the user's modules with an independent CDB and a CTS server. Independent CDB refers to an area of memory which is similar to the real CDB but which is not permanently mounted, it is part of the executable. The CTS server is also a module, but it has the ability to communicate between the user's CTS-Plus session and the user's modules or the independent CDB. Hence, it allows the user to work on one or several modules in a stand-alone mode, independently from the simulation. This means that while simulation is running, you can run your modules, have them communicate together with your own CDB, and have access to all the CTS-Plus features. The structural organization of the CTS-Plus is as shown in Figure 7.8. Note that a **"fake"** file is maintained that will allow the simulator to run even though modules are not linked into the executables where they have been defined. A source file named *"fake-entry"* is maintained containing the entry point names of all modules in the tasks used by the simulator.

7.7 Distributed Interactive Simulation Module

The Distributed Interactive Simulation (DIS) module is an additional tool or utility for CDB-based real-time simulations. DIS module is meant for interfacing and interacting with other simulation applications across a network. The exchange of information across the network is performed by sending and receiving Protocol Data Units (PDUs). The interface with the network is performed by two external PDU Manager processes, i.e. *PM-WRITE* and *PM-READ*, which are launched by the *dis-main* object. The *PM-WRITE* process initializes the outgoing ethernet connection and writes PDUs directly to the network. It receives these PDUs from objects in the DIS module via a queue in shared memory. The DIS objects write to this queue, from which the data is read by *PM-WRITE* to be sent to the network. The *PM-READ* process initializes the incoming ethernet connection and reads PDUs directly from

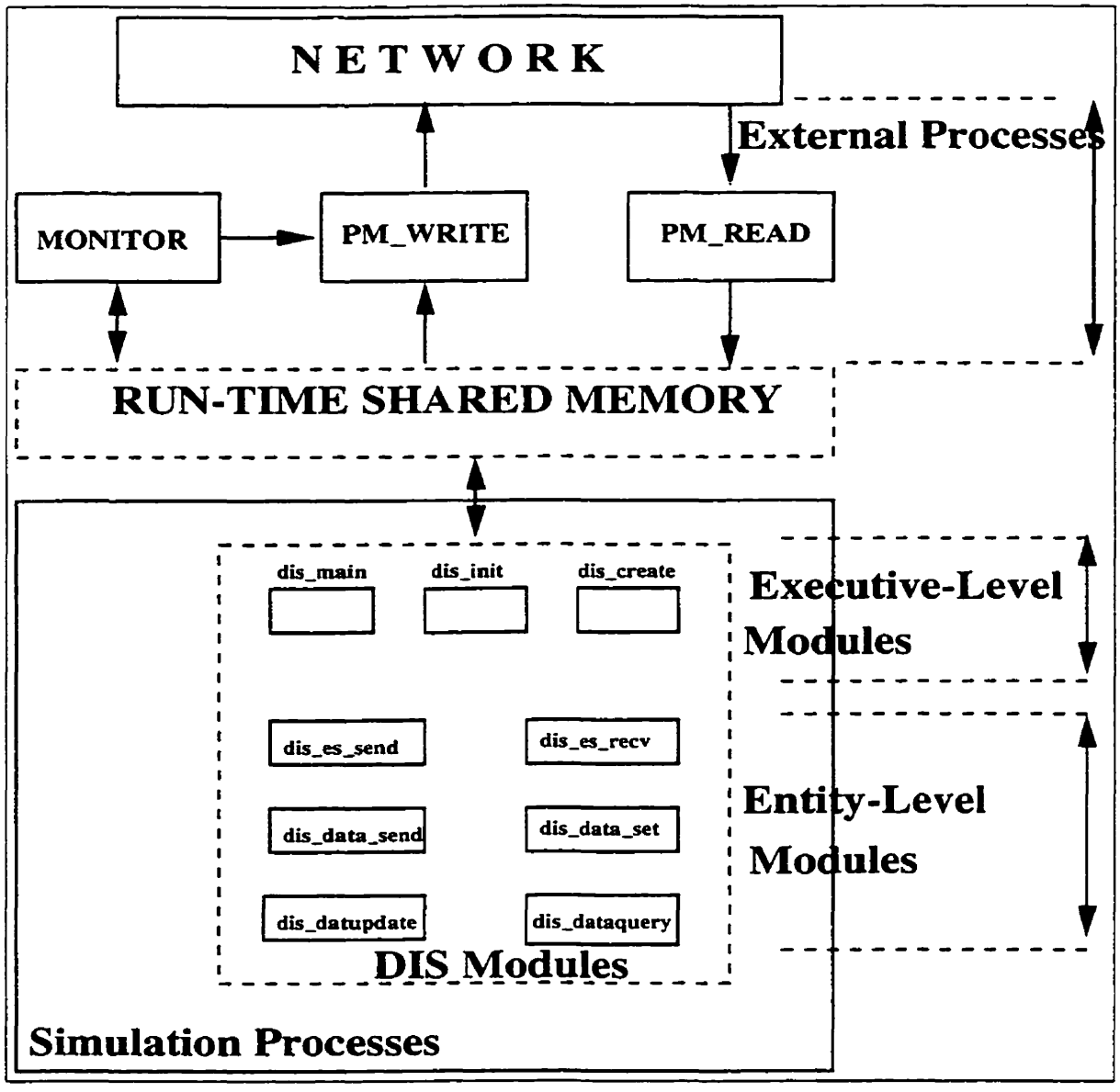


Figure 7.9: An overview of DIS modules

the network. It receives these PDUs to a queue in shared memory, from which they are read by the *dis-main* object and distributed to other modules.

DIS modules may be subdivided into three main categories according to their scope and function. Modules which may only have a single instance, such as *dis-main*, are called Executive-Level DIS Modules. They communicate information that is not particular to a single entity in the simulation or which control or communicate the status of the DIS processes. Modules which are associated with a particular simulation entity (whether the entity is simulated locally or in another simulation) may have multiple instances and are called

Entity-Level DIS Modules. They communicate particular information regarding the entity with which they are associated. The remaining modules are DIS Data Type Conversion, which serve to convert between the raw data contained in the simulation, and the specific data formats required for some DIS PDUs. The interface between the DIS external processes, the DIS modules executing in SIMex-Plus simulation processes, and the network is shown in Figure 7.9.

The external process is launched by the *dis-main* module during initialization. It monitors the status of the simulation and of the DIS module in order to inform other simulation applications of changes in the status which cannot be detected from within SIMex. The *SM-MONITOR* process is responsible for sending all Stop/Freeze and Start/Resume PDUs, according to the status of the simulation (i.e. frozen, unfrozen, or suspended), and the status of the DIS modules (i.e. initialization or run mode). When DIS modules are re-initialized, the *SM-MONITOR* process notifies other simulation applications, so that they may re-initialize any data required to ensure proper communication after the DIS modules resume normal operation. Moreover, when the simulation is terminated, the *SM-MONITOR* process handles all the required cleaning up, including removal of the PDU Manager modules, *PM-WRITE*, *PM-READ*, and removal of the shared memory segment used for communication between DIS modules and the rest of SIM modules.

Chapter 8 Telecommand CDB Labels for MOTS

8.1 Introduction on MOTS

The MSS Operations and Training Simulator (MOTS)[1], being developed by CAE Electronics for the Canadian Space Agency (CSA), will be used to develop procedures for operating the Mobile Servicing System (MSS), and to train astronauts, mission controllers and instructors. Among the many activities to be performed by astronauts, the two most important will be payload handling and spacecraft berthing/de-berthing. The training curriculum will provide astronauts with all the necessary skills to operate the MSS. MOTS simulates, in real-time, the end-to-end MSS Space Segment, replicating its full functionalities, especially that of the *Human-Machine Interface (HMI)*. Operator telecommands are input from control consoles that faithfully emulate the flight article. These telecommands are processed by the arm, joint and end-effector control system models to generate the appropriate control actions. The dynamics simulation model then determines the actual position and orientation of the arm and the end-effector. MOTS also provides the operator with a high fidelity simulation of the camera views and on-orbit lighting conditions.

MOTS software is twofold: first simulation models and second tools to support execution and configuration management of the simulation models. MOTS simulation configuration & load control uses SIMex-Plus [CAELIB., 1995s] capabilities (e.g. CTS & PFU utilities for MOTS Specific Support Environment (SSE))[2]. Primary methods for interfacing with the simulation include hand controllers, D & C panel and Human Computer Interface (HCI) pages via CDB servicing routines as depicted in Figure 8.1. MOTS training scenarios will use virtual reality, 3-D audio, 3-D imaging, artificial proprioceptive feedback and expert systems to meet MSS training requirements. The simulation management user in-

[1] Refer to Appendix C for list of acronyms and their description.

[2] CAELIB Utilities are summarized in Chapter 7 while CAE Space Engineering Workbench is outlined in Appendix A.

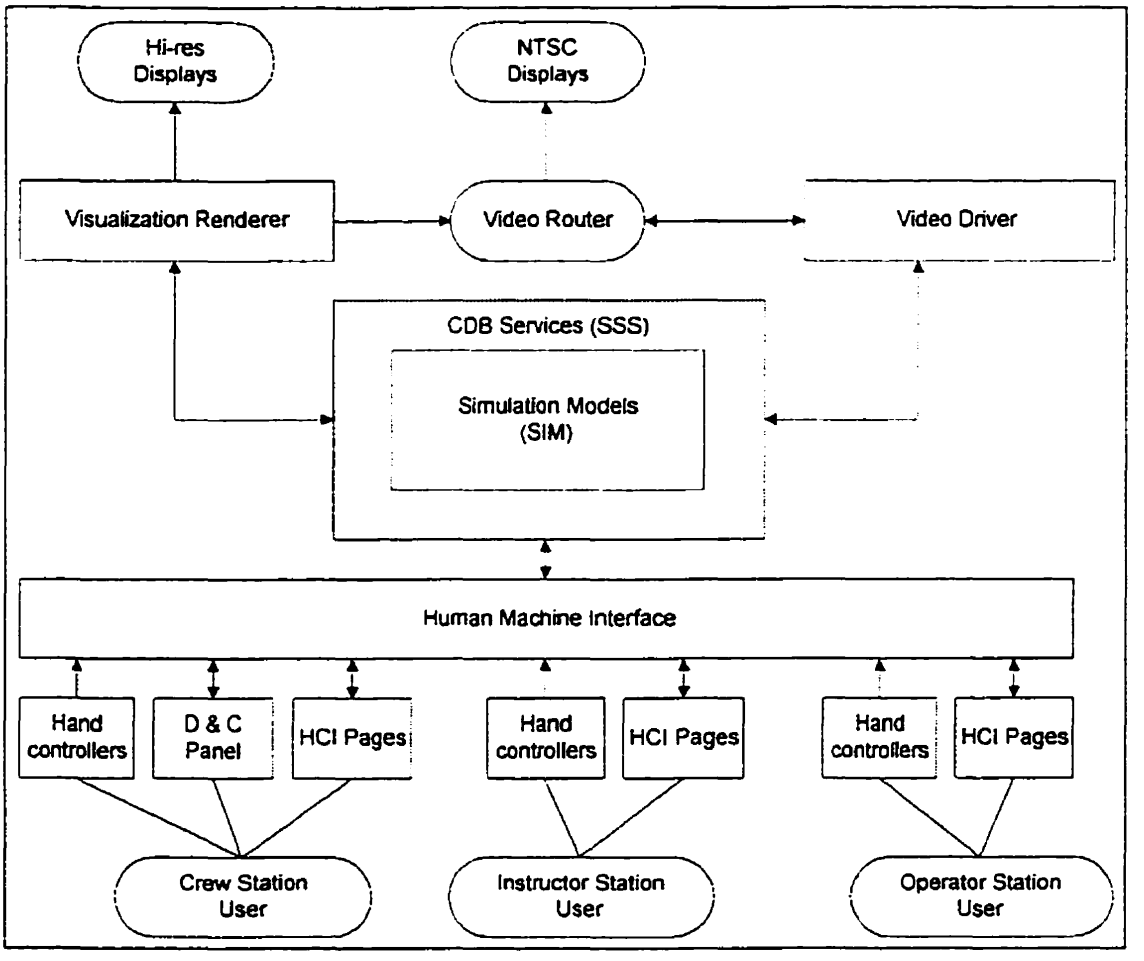


Figure 8.1: MOTS internal interfaces

terface is as summarized in Figure 8.2. Hence, in summary, MOTS facility is a real-time simulation environment of varying degrees of fidelity, along with an aggregate of software tools intended for the support of MSS space operations and training of crew and ground controllers. Main technical design decisions include:

- Follow an equipment-like breakdown as much as possible and use equipment Interface Control Documents to define interfaces between equipment simulation models;
- Use stub models for the equipment that is not fully simulated but which is necessary to close the loop along with re-using MDSF models;
- Malfunctions[3] performed directly at source equipment level, thus ensuring cascading effects; and
- Simulate hardware power-on, software down-loads and power-on self-tests (POST) simulated with programmable time delays.

[3]**Malfunction** simulation is a simulated condition where an object or system fails to function in a normal manner.

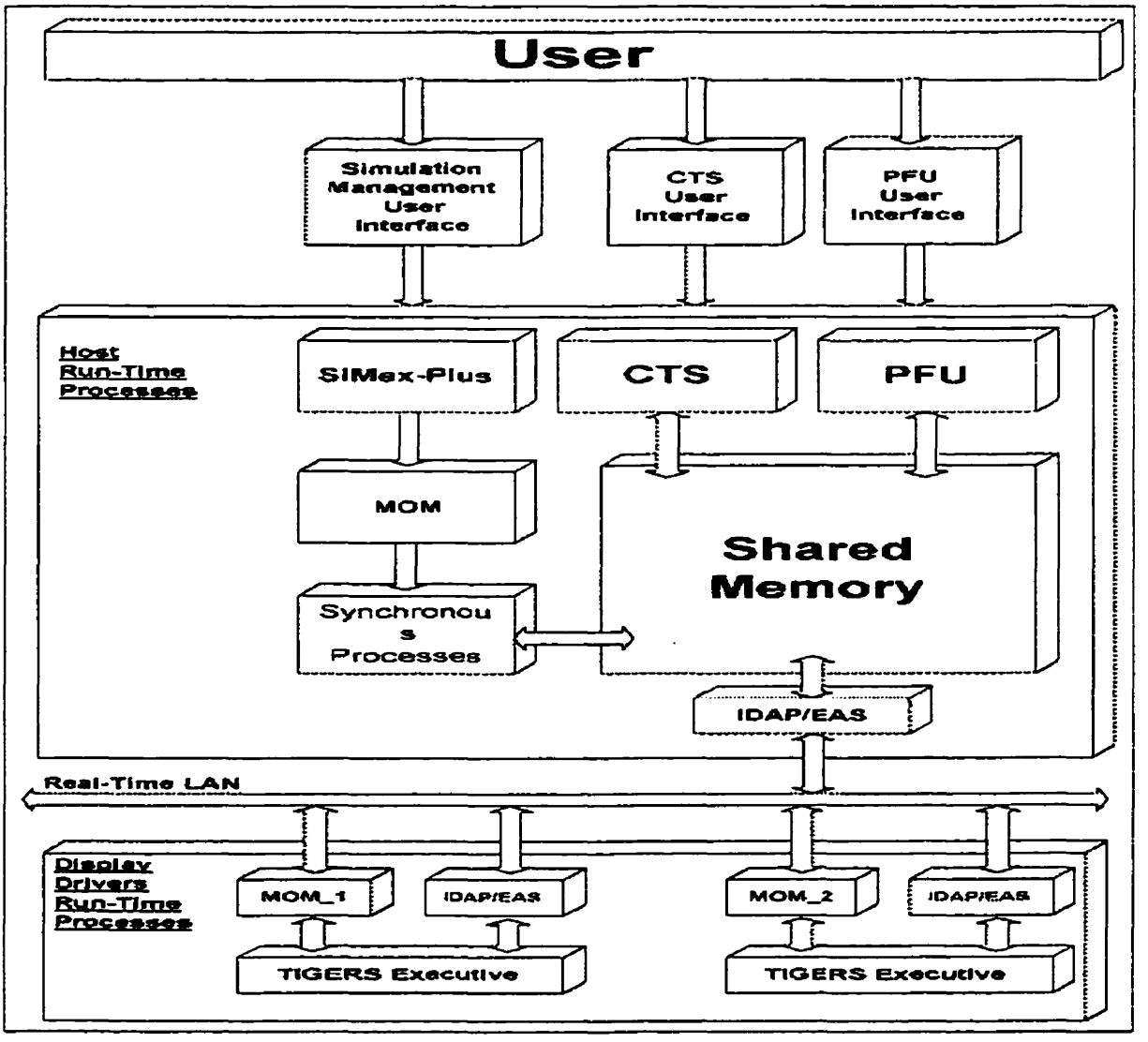


Figure 8.2: MOTS user interfaces

8.2 MOTS Hardware and Data-flow

The MSS is Canada's contribution to the International Space Station Alpha (ISSA). The MSS is a multi-purpose, versatile complex equipped with manipulators, advanced control systems and *"human in the loop"* capability. It will support construction, operation and maintenance aspects of the Space Station and its attached payloads, and will be the primary tool used by astronauts to support Extra Vehicular Activity (EVA) on the Space Station. The MSS is comprised of two robotic manipulator systems, a base system, a mobile transporter, and two robotic and command workstations as shown in Figure 8.3. The first of the two MSS manipulators is the Space Station Remote Manipulator System (SSRMS) which is a

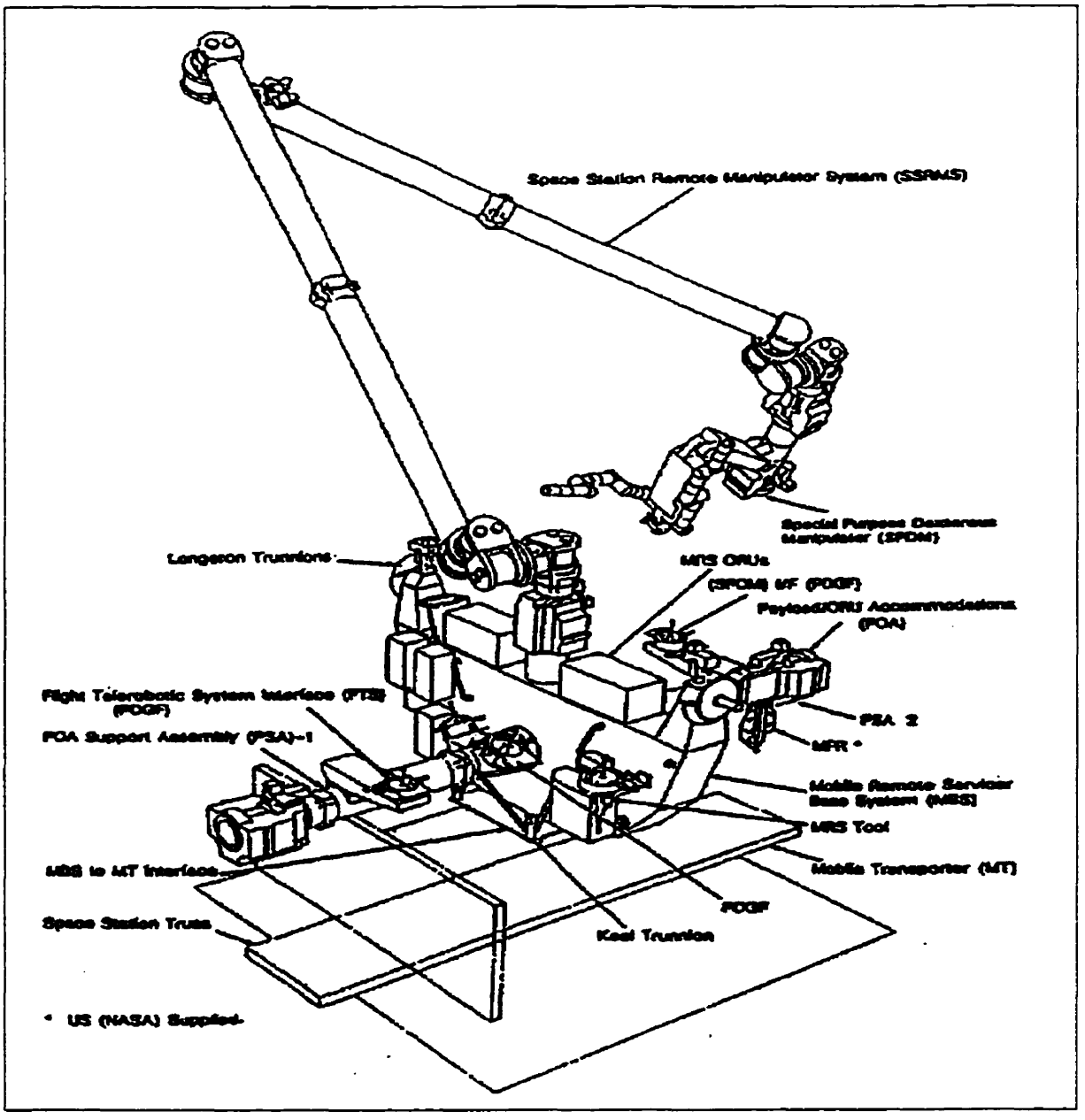


Figure 8.3: Mobile Servicing System (MSS)

self-relocatable, 16.8-m manipulator arm with seven degrees of freedom. The SSRMS will be used for large scale manipulation of payloads; e.g. installation of pressurized modules and truss elements for ISSA assembly. It will also be used to position EVA crew at work-sites for ISSA maintenance. The SSRMS is symmetric relative to its elbow joint; this allows it to be attached and operated from either end, by means of its Latching End Effector (LEE) for grapple fixtures and payloads attachments. The second MSS manipulator is the Special Purpose Dexterous Manipulator (SPDM), which is shown attached to the tip of the SSRMS

in Figure 8.3, is a smaller, dexterous robotic system consisting of an articulated body and a pair of 2 meter long arms, each having seven degrees of freedom. The SPDM will be used for small and precise movements, in general tasks similar to those performed by a space suited extravehicular astronaut. The next MSS component is the Mobile Remote Servicer (MRS) Base System (MBS) which serves as an attachment structure and stable work-base for the SSRMS and SPDM. It will also be used as a temporary attachment point for ISSA payloads and elements. In turn, the MBS is attached to the Mobile Transporter (MT) which is used to move payloads and robotic equipment to strategically located positions along the truss of the ISSA, thus providing mobility to reach various maintenance sites.

Decomposition of a teleprogramming system is crucial for its controllability and observability in real-time. For a teleprogramming system, the allocation of requirements from a Computer Software Configuration Item (CSCI) to its Computer Software Components (CSCs) and Computer Software Units (CSUs) is necessary. Thus, MOTS hardware device interface to CSCI is as shown in Figure 8.4. It has to be noted that MOTS software is divided into three CSCIs:

1. **SIM CSCI:** responsible for simulating the MSS equipment and environment;
2. **VAD CSCI:** responsible for all MOTS visualization functions and user interfaces; and
3. **SSS CSCI:** which essentially consists of the tools and utilities required to build and run simulations.

MOTS equipment-like breakdown into CSCs is essential where each CSC can be broken down to smaller CSCs which are finally broken down into CSUs. In total, there are about 150 MOTS CSUs from 19 CSCs as summarized in Table 8.1. The CSC data-flow overview is shown in Figure 8.5. Table 8.2 shows a sample CSUs breakdown for the CSC SSRMS LEE along with their descriptions and execution rates.

The control of all the MSS activities originates from the Robotic Work Station (RWS) which will provide robotic system telecommand and control to on-orbit crew-members, and will enable the MSS to interface with the ISSA. It will be used to monitor robotic system performance and operations. The RWS contains four types of input/output devices which

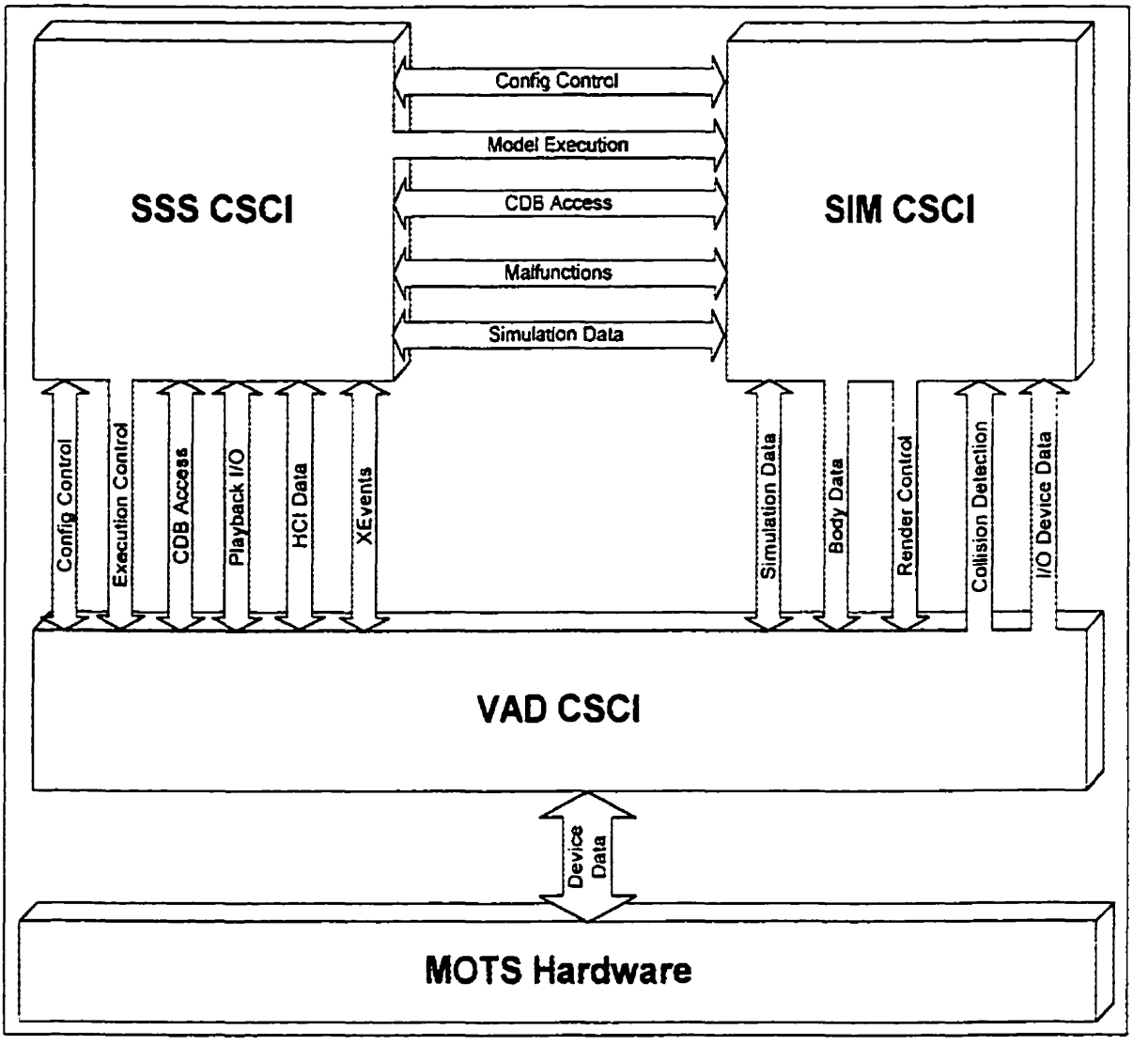


Figure 8.4: MOTS CSCI interfaces overview

enable the control of the various MSS equipment, namely: three video monitors with split-screens, two 3-D hand controllers (translational/rotational), a display and control (D&C) panel to allow the operator to enter telecommands using switches or dials, and a Portable Control Station (PCS) that provides interface with the MSS via HCI pages. The PCS is physically a part of the RWS but is connected to the Space Station, which in turn processes the HCI pages inputs and sends the telecommands to the MSS via the RWS. MOTS provides an end-to-end simulation environment in real-time of the MSS and the Space Station components that interface with it, coupled with monitoring and control functions to allow effective training of crew and ground controllers in the operation of the MSS. MOTS archi-

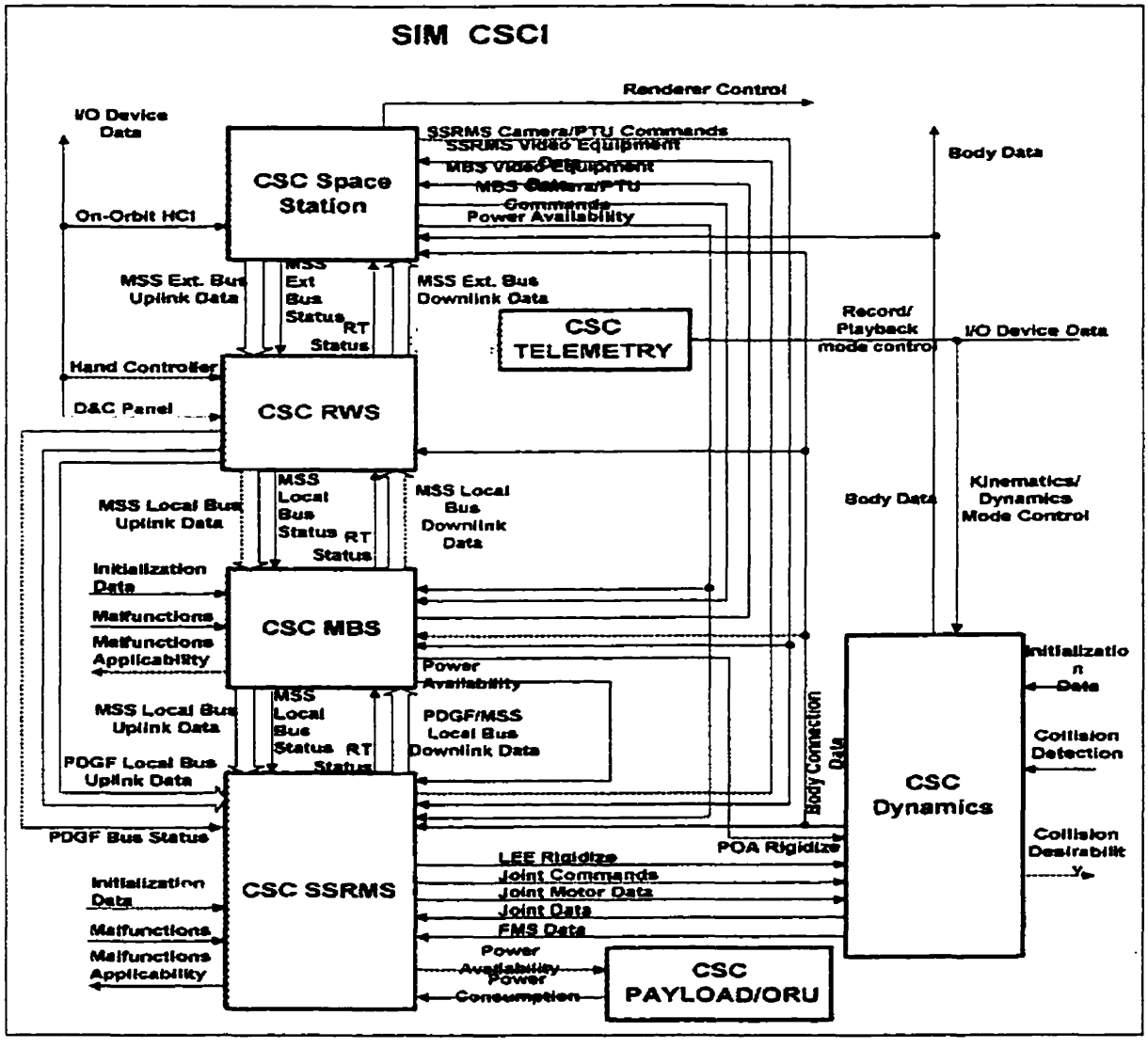


Figure 8.5: CSC data-flow for the SIM CSCI

tecture provides the necessary hardware and software elements to re-create the MSS operational environment through simulation, for the primary purposes of training and procedures development [Cyril *et al.*, 1992].

8.3 CDB Interface to MOTS Components: HCI pages

The main hardware components of the MOTS are a simulation host computer, an Instructor Station (IS), an Operator Station (OS), a Crew Station (CS), two Software Development Workstations (SDW), and a video distribution and recording system as shown in Figure 8.6. The IS is designed to provide the capability for instructors to monitor and control all aspects

CSC Name	CSS Description	Total CSUs
SSRMS ACU	SSRMS Arm Control Unit	16
SSRMS LEE	SSRMS Latching End Effector	19
SSRMS Joint	SSRMS Joints' Unit	8
SSRMS Video	SSRMS Video Unit	13
SSRMS Power Distribution	Power Unit for SSRMS	2
SSRMS Thermal	Thermal Unit for SSRMS	7
MBS MCU	MBS Control Unit	6
MBS POA	MBS Payload/ORU Unit	16
MBS PDGF	MBS Power Data Unit	1
MBS Video	MBS Video Unit	2
MBS Power Distribution	MBS Power Unit	5
MBS Thermal	MBS Thermal Unit	2
RWS	Robotics Work Station	1
Payload/ORU	Payload/ORU Unit	1
SS and C&C Video Control	Space Station Control Unit	5
SS C&C SSRMS Control	Space Station SSRMS Control Unit	1
SS Time	Space Station Time Unit	1
SS Power Distribution	Space Station Power Unit	1
Telemetry	Telemetry Unit	10
Dynamics	Dynamics Unit	23

Table 8.1: MOTS SIM to CSC decomposition

of training sessions on the simulator and to insert malfunctions. A 3-D translational and a 3-D rotational hand-controllers will be provided to control the manipulator. The IS will select one of the MOTS stations to be the simulation session control authority; i.e., only one station can control the MOTS simulation at any time. The IS will provide the instructor the ability to view any arbitrary viewpoint (God's eyes view) of the simulated MSS in addition to the camera views. Also, the instructor will be able to control the manipulator and the MSS simulation via the HCI pages which contain malfunctions, data monitoring, volatile update, scripting, sound, etc. (all shown in Figure 8.7). All HCI pages are interfaced to the MOTS X-Runner and Library via CDB. The CS provides the crew with simulated camera views of the MSS on the video monitors and MSS on-orbit HCI pages on the workstation. Control inputs are provided via the workstation's keyboard and track-ball, and two 3-D hand-controllers (translational/rotational). The CS will also be equipped with a D&C panel which

CSU Name	CSU Description	Rate (Hz)
LEE-EXEC-1	Calls all LEE modules with an execution rate of 2000 Hz	2000
LEE-EXEC-2	Calls all LEE modules with an execution rate of 20 Hz	20
LEE-COMPUTE-STATE	Computes LEU current state	20
LEE-VALIDATE-LEEMMM-CMDS	Validates all LEEMM telecommands that have passed ACU validation	20
LEE-CMD-PRECONDITIONS	Checks the preconditions of the LEEMM telecommands	20
LEE-EXEC-STATUS	Determines the execution status of LEEMM telecommands	20
LEE-SET-MOTOR-DRIVE	Decomposes auto telecommands into sequence of manual telecommands	20
LEE-CONTROL-MODE	Determines the appropriate control mode of LEEMM telecommand	20
LEE-EXECUTE-LEEMM-CMDS	Computes position telecommands in incremental, step or combined mode	125
LEE-MOTOR-POS-AND-RATE	Computes position and rate of motor	125
LEE-SERVO-CONTROLLER	Executes position, rate and current loops of LEE servo controller	2000
LEE-HARDWARE	Simulates the LEE hardware	2000
LEE-STATUS-SWITCHES	Computes LEE devices' switches status	125
LEE-LOAD-CELL	Computes LEE rigidize load cell force	125
LEE-PROCESS-TELEMETRY	Computes LEE telemetry	20
LEE-DATA-LOGGING	Performs high speed data logging and logged data upload	125
LEE-MALFUNC-APPLICABILITY	Determines the applicability of all LEE malfunctions	20
LEE-FMS	Computes sensor force and moment	1000
LEE-POWER	Computes LEE components power	20

Table 8.2: CSC SSRMS LEE to CSU decomposition

will send telecommands to the MSS simulation via switches and push buttons.

The OS provides a capability for operations personnel to do MSS procedures development, operations analysis and HCI pages prototyping. OS will provide control of the simulation session, as well as monitoring of the simulator status. OS will also be equipped with hand-controllers to manually control the manipulator. The SDWs provide a comprehensive

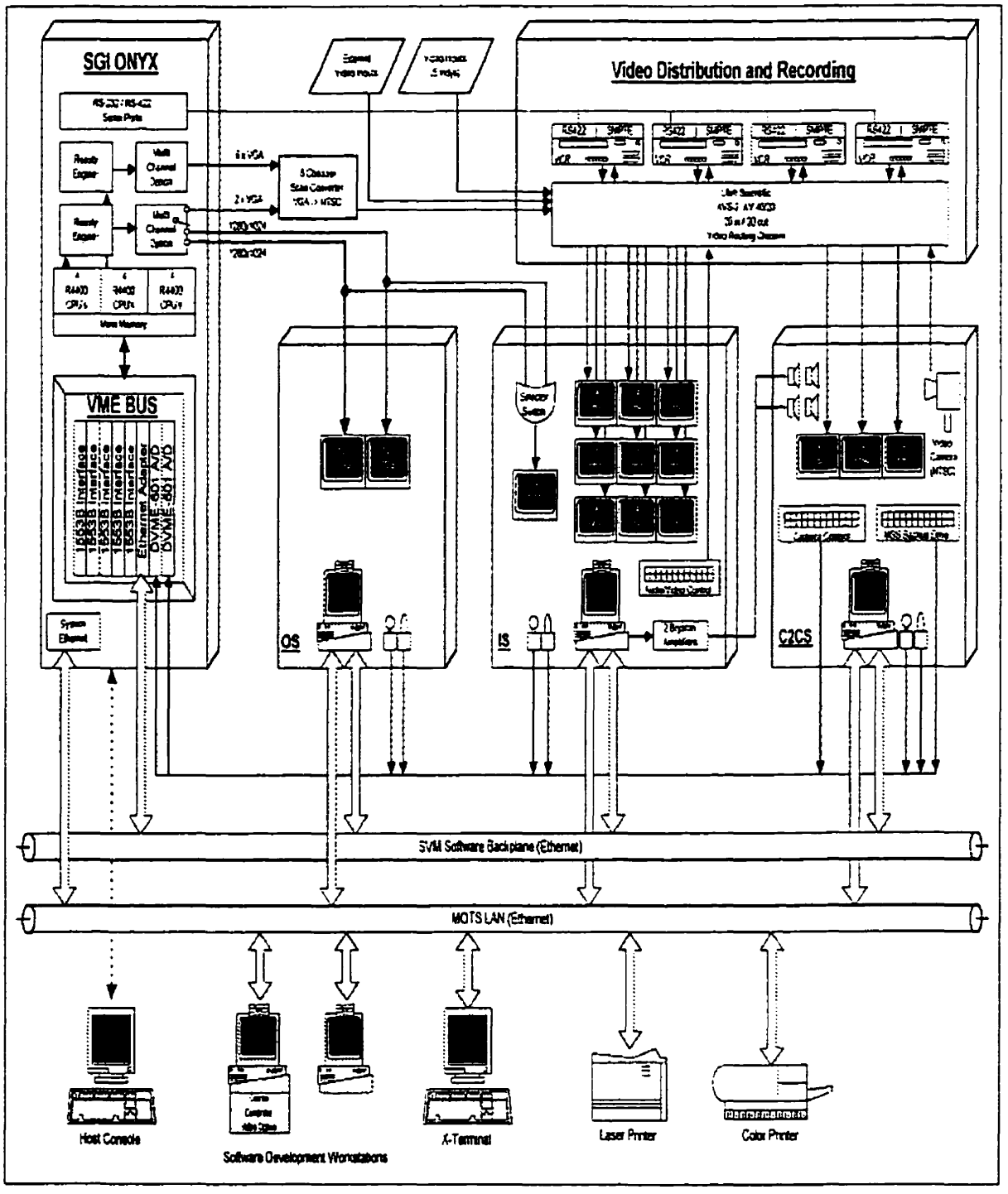


Figure 8.6: MOTS hardware design

software tool-set required to develop, configure and run simulation models. They permit the developer to build and run complete simulations, including models and HCI configurations. MOTS video distribution system is responsible for the distribution of simulated camera views to the video monitors available in the MOTS stations. MOTS record and playback

system, controlled by the HCI pages hierarchy, can record up to four channels of video and audio simultaneously. The tapes can be later replayed, either in real-time, slow motion or frame by frame. Finally, the CS for the MOTS is provided in two phases. In the first phase, a Class II replica of the MSS RWS, referred to as C2CS, is provided. Simulated camera views are presented on NTSC monitors and MSS display pages are presented on the work station. In the second phase, a Class I replica of the RWS, referred to as the Class I Crew Station (C1CS), is integrated into the MOTS. CS provides the capability for training crew, validating on-orbit procedures (with a person in the loop), and prototyping new HCI displays for the MSS.

8.4 CDB Telecommand Data for MOTS Simulation Models

MOTS will provide a simulation of the nominal and malfunction behaviour of the MSS in real-time via CDB interface. Malfunctions can be inserted by the instructor via the simulation control HCI pages, and processed directly by the appropriate simulation model. The dynamics model has the capability to simulate a multi-body system with a chain, tree or closed loop topology [Carr and others, 1990]. The software residing in the RWS, namely the Control Electronics Unit (CEU), will provide an interface between the HCI pages, D&C panel and hand-controller inputs and the simulated MSS elements. It forwards telecommands to the SSRMS and MBS, and receives telemetry data and status information back. The Arm Computer Unit (ACU) represents the central processing unit of the SSRMS. Residing in the ACU is the SSRMS arm control simulation code. It receives arm telecommands from the RWS, validates them, and operates the arm in the selected mode of operation (manual or automatic). Based on the arm commanded status, low level telecommands are issued to control SSRMS joint servos. The ACU also processes Latching End Effector (LEE) and video-based telecommands and forwards them to the appropriate SSRMS equipment models. The ACU model also computes the power consumed by the heaters and the electronics and the heat generated by the ACU hardware. The SSRMS Joint receives low level position and velocity telecommands from the ACU and computes the drive signals for the joint motors. The control model computes the power consumed by the joint electronics, motors, brakes and heaters. The heat generated by the joint hardware is computed as well.

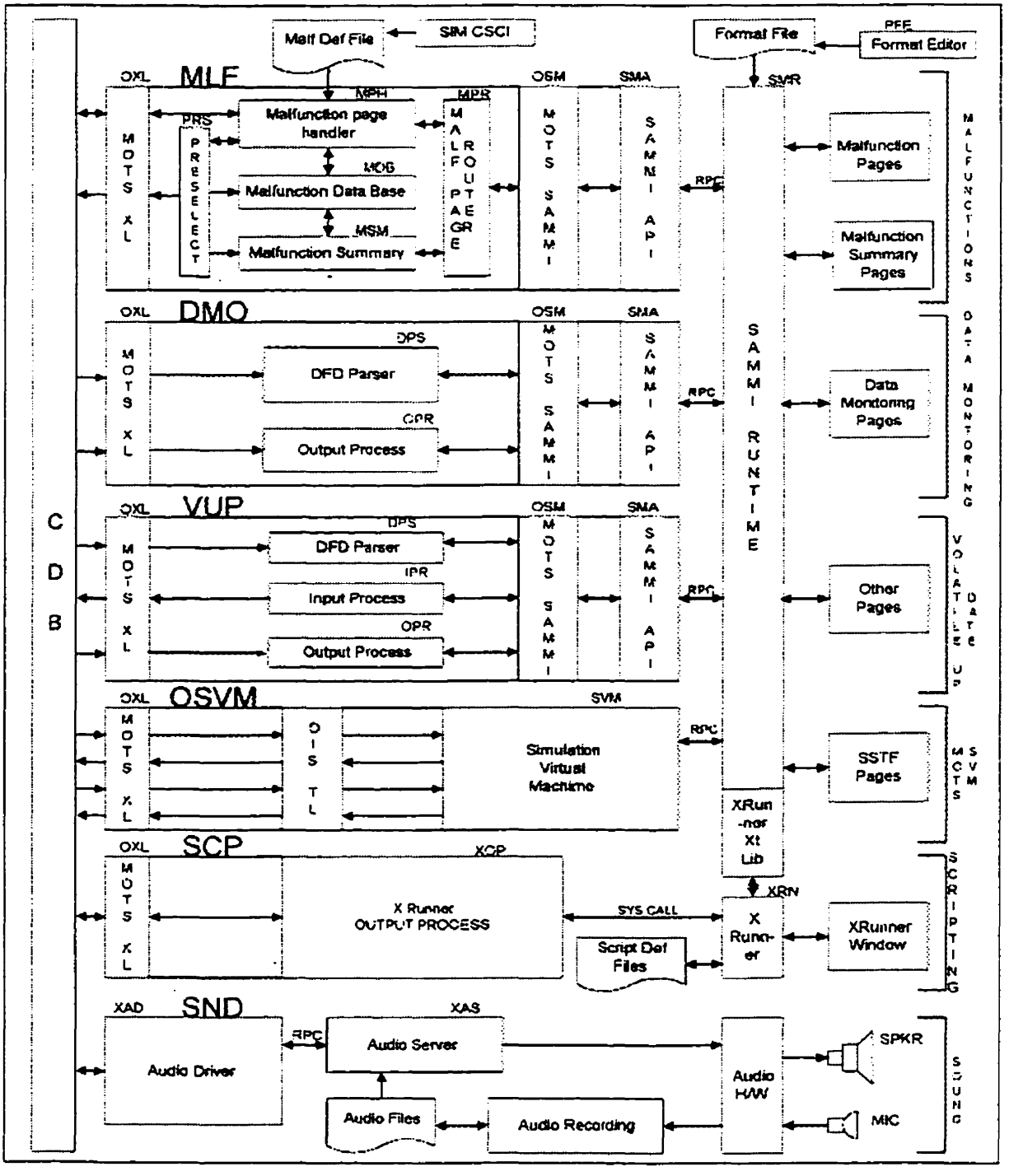


Figure 8.7: MOTS HCI pages

The SSRMS LEE receives telecommands to capture and release payloads, from the ACU. It then performs validation checks based on the status of the LEE hardware and issues position-based telecommands to the LEE servo controller model, which in turn sends the appropriate signals to the simulated LEE motors and drive trains in order to drive the

LEE snare, rigidization, and latch mechanisms. The LEE software will execute LEE-related telecommands in both manual and automatic modes of operation. The LEE model also computes the power consumed and the heat generated by the LEE electronics, heaters, motor modules and video equipment. The MBS Computer Unit (MCU) is equivalent to the ACU model on the SSRMS. It processes CRPCM, video and Payload ORU Accommodation (POA) telecommands from the RWS and forwards them to the appropriate equipment model. It also computes the power consumed and heat generated by electronics and heaters. The Video model is common to both the SSRMS and MBS. It receives telecommands from the ACU or MCU respectively, to control the Video Distribution Units (VDUs), lights and cameras. It also computes the power consumed and heat generated by the video equipment, as well as turning on and off video equipment heaters for temperature control. The MBS POA has the same functionality as the SSRMS LEE with the exception that it receives telecommand from the MCU. The MBS CRPCM is responsible for switching and distributing power to output loads from a common input power feed connected to the MT/MBS interface. It also processes telecommands from the MCU and sends back requested data and health status. All these telecommands data are initialized, stored and updated in real-time in the CDB common shared memory using HCI pages interface.

The operator interacts with the HCI pages using a keyboard and track-ball and is able to issue telecommands to the simulated MSS elements, modify input parameters and monitor output parameters. The actual content and layout of the displays is reconfigurable by the instructor or operator before the start of a simulation session. A sample list of CDB telecommands data is given in Table 8.3. In total, MOTS will have over 10,000 telecommands & telemetry data stored in CDB shared memory to be accessible in real-time by any simulation module.

8.5 MOTS Critical Design Issues & Intended Training Usage

MOTS must provide a faithful simulation of the MSS in order to effectively train astronauts. This is achieved by using simulation models (dynamics and control system) that have already been validated. Also, MOTS will run in a robust real-time environment, developed at CAE, which has proven to provide deterministic and consistent simulation results (see Ap-

Telecommand data	Type	Description
tc-hc-por-x-rate	integer	Linear deflection of the H/C along X-axis
tc-hc-por-y-rate	integer	Linear deflection of the H/C along Y-axis
tc-hc-por-z-rate	integer	Linear deflection of the H/C along Z-axis
tc-hc-por-roll-rate	integer	Angular deflection of the H/C along roll-axis
tc-hc-por-pitch-rate	integer	Angular deflection of the H/C along pitch-axis
tc-hc-por-yaw-rate	integer	Angular deflection of the H/C along yaw-axis
tc-confirm-autoseq	logical	Telecommand to confirm autosequence
tc-desel-line-tracking	logical	Telecommand to deselect line motion
tc-limp	logical	Telecommand to select joint limp
tc-sel-por-ori-offset	real	POR angular target commanded offset
tc-set-fma-limits-force	real	Maximum FMA limits for forces
tc-set-fma-limits-moment	real	Maximum FMA limits for moments
tc-checkout-lee	logical	Telecommand to exercise LEE mechanisms
tc-close-snares	logical	Telecommand to manually close LEE snares
tc-power-on-camera	logical	Telecommand to power-on cameras
tc-power-on-camera-id	integer	Camera ID to be powered-on
tc-power-off-vdu	logical	Telecommand to power-off video units
tc-power-off-vdu-id	integer	Video unit ID to be powered-off
tc-mbs-terminate-diag	logical	To stop diagnostics for MBS
tc-sync-pan-stop	logical	Pan stop telecommand flag for cameras
tc-sync-tilt-down	logical	Tilt down telecommand flag for cameras

Table 8.3: Sample list of MOTS CDB telecommands data

pendix A). Another issue arises from the fact that a computer simulation runs much faster than a real system. Hence, time delays have to be incorporated in the design in order to simulate latencies of the real system. Time delays will be incorporated in the design of control system models and all models that simulate the MSS equipment (powering-up, state transitions, etc.). Most of the robotic tasks, such as assembly and maintenance, on the Space Station will be carried out with only camera views. This puts a requirement on the simulator to provide imagery with a very high resolution and detailed representations of the simulated objects. MOTS visualization models are designed with the purpose of providing the users with very high quality camera views of the simulated MSS elements. CSA is responsible for skills, knowledge, and attitude training for MSS operations and maintenance. This training will be conducted in Canada, within the Canadian MSS Training Facility (CMTF). The CMTF comprises computer-based training (CBT) systems, multimedia learning cen-

tre, and training aids, such as models, mock-ups, and cut-aways. CMTF will also interface with other MSS Operations Complex (MOC) facilities, such as the MOTS and the Space Operations Support Centre (SOSC) in order to meet CSA's MSS training responsibilities. MOTS will be used for familiarization training, MSS-specific comprehensive skills training, integrated EVA crew training, and on-board training. This training will be provided in distinct learning modules to allow customization based on the previous experience of each trainee. MSS trainees will typically be self-motivated and self-directed, with varying backgrounds and experience. Training will be provided using demonstration, prepared training scripts, and discovery methods (e.g. free play). These self-administered, self-paced training methods will require automated feedback to trainees in real-time, in the form of comments, graphs, training cues, etc.

For group learning, simulation could be run from the CMTF multimedia learning centre for training sessions which involve group discussion and problem solving. Training scenarios will use virtual reality, 3-D audio, 3-D imaging, and expert systems to meet MSS training requirements. Training on MOTS will range from simple to complex scenarios (e.g. trainees will control the MSS, first in a single plane, then progress to two and finally three planes, under kinematic and/or dynamic simulations). Static displays of MSS components will be provided as well. The IS will provide all the tools required by the instructor for concurrent monitoring, interaction with and intervention in trainees' learning. Instructor actions are based on principles of adult learning, rather than pedagogy, and consist of: *advising; monitoring; mentoring; and being a subject-matter expert.* The instructor-trainee relationship will be interactive, via over-the-shoulder video, audio, and computer interfaces. The inputs to the workstation are numerous and may include video camera images, feedback from trainee consoles, audio/video data, etc. The instructor will be able to control the information provided to trainees, and can replicate trainee actions. For example, the instructor could reposition the SSRMS to a previous location and the trainee could observe the instructor perform the same task. The IS will also be used to author training scenarios. Instructors will have the utilities necessary for the off-line preparation of training scenarios using prestored libraries of simulation objects, e.g. Space Station configurations, set ups, and payload configurations. Authoring will require minimal programming knowledge on the part

of instructors, and will include on-line help and run-time debugging capabilities. Applicable MOTS data files will be exchanged with CMTF systems for the purpose of course-ware development.

For all CSUs that are called by the Synchronous Scheduler (as discussed in Chapter 7), there can be no external error messaging. In general, synchronous CSUs must have work-arounds for all situations which can result in error. There are two design guidelines to be followed during CSUs design to prevent run-time errors from crashing the simulation: (a) there should be no assumption on the range of values of a CSU input. Values that would create an arithmetic fault should be replaced by an acceptable value, or the computation should be bypassed; and (b) vector and matrix indexing should be verified to prevent writing into an illegal area of the CDB shared memory. Asynchronous CSUs can detect errors returned by the IRIX Operating System (OS), issued by function call return statuses and local errors. When CSUs catch relevant signals returned by the OS, appropriate signal handlers are called for error recovery. Irrelevant signals will be processed by default signal handlers. When CSUs detect status errors from function calls, messages are returned to *stdout* and, where possible error recovery is engaged. Although for the majority of cases *exithn()* is called. CSUs local errors typically have error recovery mechanisms. For example, if a CSU cannot translate a system logical name, a message is returned and a default one is used.

Chapter 9 Experimental and Analytical Results

9.1 Introduction

Although much more work remains to be done in refining the *CDB-based teleprogramming* system, analytical and experimental results have decisively confirmed the validity and effectiveness of the *teleprogramming methodology*. Remote site autonomy at the level of $nt = 100$ seconds is realistically formulated as it reduces the delay by 99% during delayed-teleoperated tasks. Thus, verifying the feasibility of near-optimal teleoperated control system. In more challenging applications, the degree of attainable remote site autonomy will dictate the overall efficiency. A *double-buffering execution management scheme* ensures that the *lag time* η between a remote and virtual workcell 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 even though it delays the first telecommand. 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. Experimental results of a teleprogrammable virtual tracking system while assessing the controllability and observability of the control system are provided. A typical simulation experimental run for teleoperator interface and training is also outlined for a CDB-based MOTS system. Through-out this research, critical issues being observed were:

1. the generation, parsing, translation and execution of telecommands in real-time;
2. the convenience and naturalness of the operator's interaction with the virtual simulator mimicking a remote workcell;
3. the correctness and adequacy of on-line extracted symbolic instructions or telecommands describing the operator's actions while using interface devices as sensors;
4. the stability and reliability of remote site execution management;
5. feasibility and effectiveness of on-line error recovery; and
6. overall efficiency while performing remote tasks.

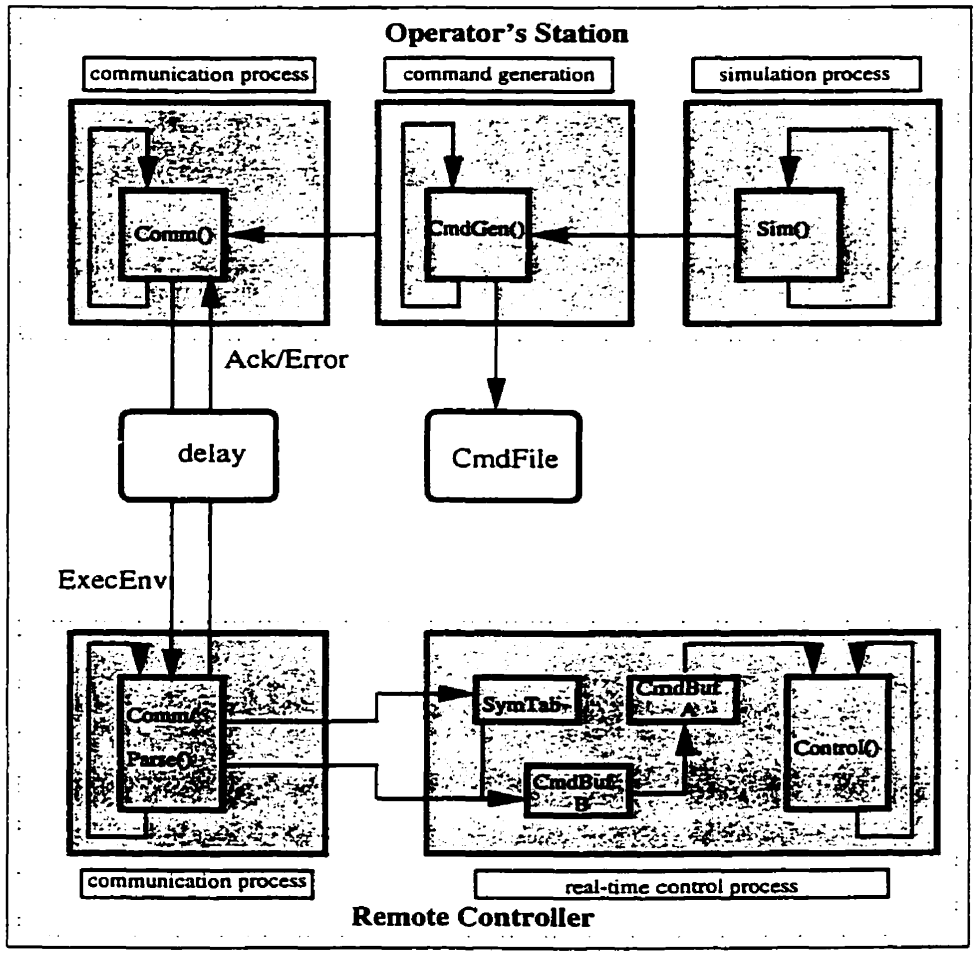


Figure 9.1: Double-buffering execution scheme

9.2 Analytical Validation of a Teleprogramming Paradigm

As noted from Chapter 5, it was found that by following the standard sequential dequeue-parse-execute management scheme, care must be taken to control the *lag time* η_i at time eT_i (i.e. just before executing telecommand ⓘ ; all symbols used are defined in Table 5.2) which depends on the *waiting time* wt_i.

$$wt_i = \begin{cases} rT_i - (eT_{i-1} + t_{i-1}); & \text{if } rT_i > (eT_{i-1} + t_{i-1}) \\ 0; & \text{otherwise} \end{cases} \tag{9.1}$$

$$\eta_i = eT_i - gT_i = t_1 + \tau + \sum_{j=1}^{i}(pt_j + wt_j) \tag{9.2}$$

Equation 9.2 implies that even if the sum of waiting times is bounded, in limit η_i can sig-

nificantly grow to ∞. This indicates that the *lag time* not only increases as the task progresses, but is in fact unbounded. In order to solve this problem, a *double-buffering* execution scheme, as illustrated in Fig. 9.1 is proposed. In this scheme, each dequeued execution environment is translated and placed into a *command buffer*. The remote controller maintains two such buffers (i.e. CmdBuf A and B). While one is being executed, the other is being constructed by parsing and translating the next execution environment. The combination of this parallelism and an artificially introduced *holding time* ht_1, which delays the execution of the first telecommand by ht_1, can be used to control the lag. ht_1 initially increases the *lag time*, but keeps it *constant* and *bounded* from then on. In terms of the above nomenclature, the necessary & sufficient condition to ensure non-increasing lag time η, is:

$$\forall_i : rT_i + pt_i \leq eT_{i-1} + t_{i-1} \tag{9.3}$$

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

$$\forall_i : rT_i \leq eT_{i-1} \tag{9.4}$$

$$\forall_i : pt_i \leq t_{i-1} \tag{9.5}$$

Satisfaction of Equations 9.4 and 9.5 implies satisfaction of Equation 9.3. This requires telecommand ⓘ 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 $(i-1)^{th}$ telecommand finishes executing. Practical considerations allows to assume that the requirement of Equation 9.5 will be satisfied in all situations. Now it remains to show that the condition of Equation 9.4 is guaranteed. The following propositions establishes this result as illustrated in Fig. 9.2.

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

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

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

$$gT_i = \sum_{j=1}^{i-1} t_j; t_0 = 0$$

$$sT_i = \sum_{j=1}^{i} t_j = gT_i + t_i$$

$$rT_i = \sum_{j=1}^{i} t_j + \tau = sT_i + \tau$$

$$eT_i = t_1 + \tau + ht_1 + \sum_{j=1}^{i-1} t_j$$

Recalling that $t_i \leq t_{max}$, we have:

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

Moreover,

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

□

This completes the proof that the double-buffering execution scheme keeps the *lag time* between the virtual and the remote workcells not only bounded, but constant throughout the execution of a *teleprogramming* task. Also notice that, the above execution scheme maintains a lag time of $\eta = 2t_{max}$ even in the presence of no communication delay (i.e. $\tau = 0$). This is a direct consequence of conditions given in Equations 9.4 and 9.5 which are applicable in all cases even if an execution failure occurs after any arbitrary n telecommands.

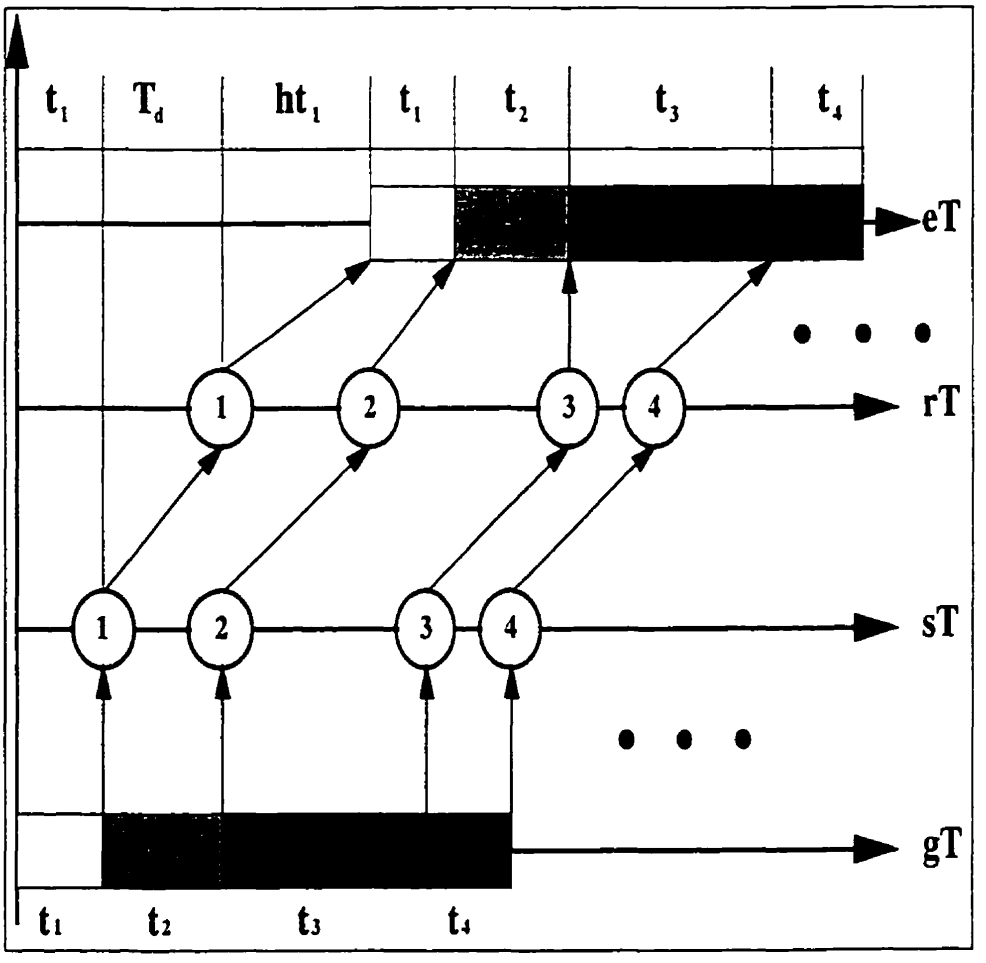


Figure 9.2: Double-buffering execution management scheme

9.3 Justification of Teleprogramming Autonomy Level

The level of autonomy during teleprogramming depends on the number of telecommand data elements or teleprograms available to be communicated to the remote workcell. The communication media can be managed by the CDB *Distributed Interactive Simulation (DIS)* module for real-time interface as discussed in Section 7.7. All telecommand data elements (i.e. subprograms) are stored in the shared memory during run-time. Procedures for their formulation, initialization, management and interfacing to simulation modules are well discussed in Chapter 7. In Chapter 3, the teleprogramming paradigm was discussed while Chapter 5 discusses the formulation of telecommands and their meta-interaction. The main focus in using the teleprogramming control methodology is to overcome the communication delays when operating at a distance such as in space or underwater. Moreover, teleprogramming dramatically improves the operator task performance due to its richness

in feedback based on virtual reality aids which support *human-machine interface (HMI)* or *human-computer interface (HCI)* in real-time. Thus, HCI pages are essential in manipulating telecommands as discussed in Section 8.3 for the case of MOTS. Now, what remains is to justify the effect of time delays in relation to the time taken to perform a given task.

As stated in Sections 1.4 and 3.6, the total time taken to perform a given task is given by T_{total}; where:

$$T_{total} = (1 + \frac{2\tau}{nt})T_{task} \tag{9.6}$$

NB: All symbols are as defined earlier in Sections 1.4 and 3.6.

Thus, with classical teleoperation (i.e. using a *move-and-wait* strategy with $nt = 1$) the total time to perform same task will be T_{total_1}; where:

$$T_{total_1} = (1 + 2\tau)T_{task} = T_{task} + 2\tau T_{task} = T_{task} + T_{delay_1} \tag{9.7}$$

Similarly, using teleprogramming method (with at leat 100 sup-programs or $nt = 100$) the total time to perform same task will now be $T_{total_{100}}$; where:

$$T_{total_{100}} = (1 + \frac{2\tau}{100})T_{task} = T_{task} + \frac{2\tau}{100}T_{task} = T_{task} + T_{delay_{100}} \tag{9.8}$$

It has to be noted that in both cases there is a *time delay* given by T_{delay_1} and $T_{delay_{100}}$ respectively as shown in Equations 9.7 & 9.8; where:

$$T_{delay_1} = 2\tau T_{task} \tag{9.9}$$

$$T_{delay_{100}} = \frac{2\tau}{100}T_{task} \tag{9.10}$$

From above, one can compute the total time savings due to teleprogramming autonomy given by $T_{autonomy}$; where:

$$T_{autonomy} = T_{delay_1} - T_{delay_{100}} = 2\tau T_{task}(1 - \frac{1}{100}) = 0.99 T_{delay_1} \qquad (9.11)$$

Hence, we have **99%** time saving due to teleprogramming autonomy with just 100 telecommand data elements. This criteria will always be used to justify the level of teleprogramming autonomy in improving the task performance. With higher levels of autonomy (i.e. $nt \gg \tau$), the operation will run as in real-time since the effect of communication delays will asymptotically vanish.

9.4 Simulation Results of a Visual Tracking Scheme

To prove whether or not visual tracking can be achieved by the controller and if the predicted positions of the object were accurate, the proposed control methodology and the designed visual robotic tracking controller (of Fig. 6.3) were applied to an experimental system shown in Fig. 6.1. The predictor and the double-loop feedback controller were realized by a computer simulations using MATLAB on a Sun Workstation. A variety of object motion trajectories used are shown in Fig. 9.3. For each motion trajectory a root locus of the closed-loop system of the controller was performed and a closed-loop transfer function with a gain of K equal to 0.6 was selected on the root locus lying inside the unit circle (i.e. implying that the stability criterion is met). In order to study a variety of motion classifications (as discussed in Section 2.7), the simulation was done using the **object** motion trajectories defined by the following models:

1. **Translation motion:** ramp-like (fixed orientation) with initial point at (10,10,5) (Figure 9.3a);

   $x_c = 10 + 0.2t$

   $y_c = 10 + 0.8t$

2. **Rotation motion:** circular-like (about a fixed position) with initial point at (10,10,5) and frequency of 0.5Hz. (Figure 9.3b);

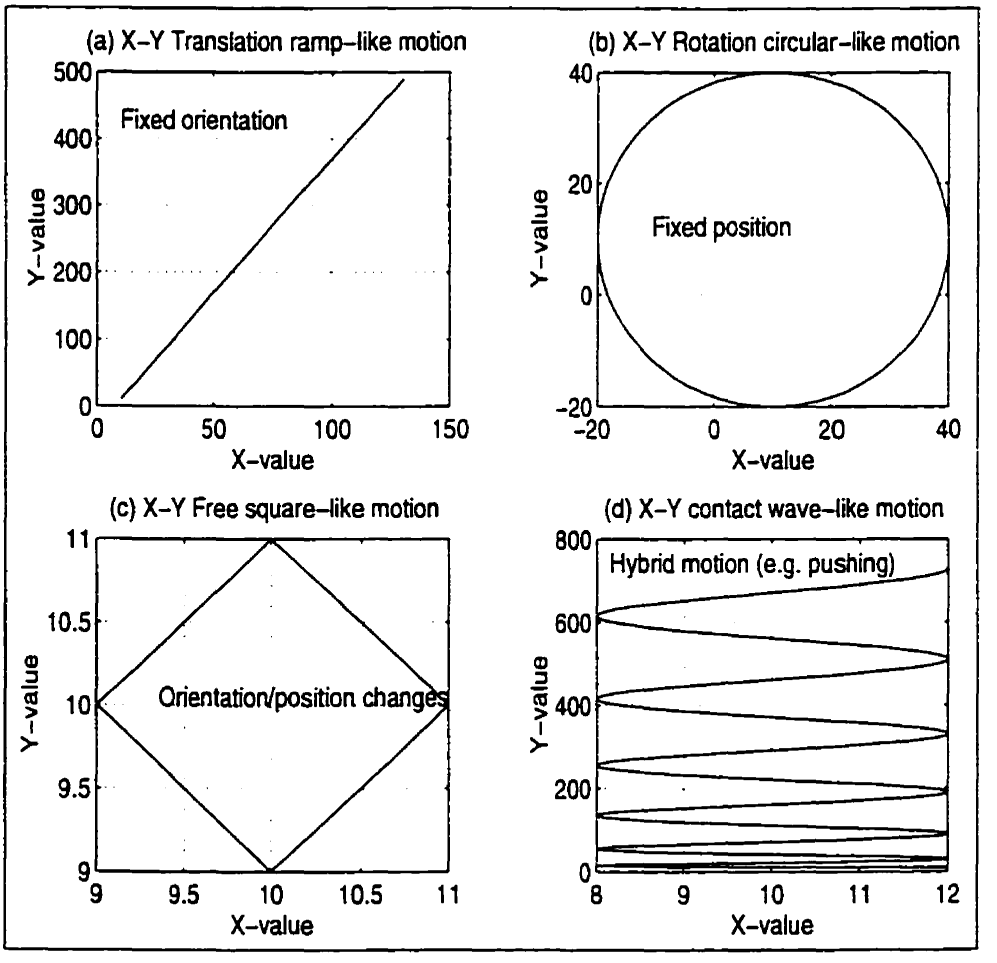


Figure 9.3: Object motion trajectories

$$x_c = 10 + 30sin(\pi ft)$$
$$y_c = 10 + 30cos(\pi ft)$$

3. **Free motion:** square-like (orientation and position changes) with initial point at (10,10,5) and frequency of 0.02Hz (Figure 9.3c);

$$x_c = 10 + 2sin(\pi ft)$$
$$y_c = 10 + 2cos(\pi ft)$$

4. **Contact motion:** wave-like with initial point at (10,10,5) and frequency of 0.02Hz. (Figure 9.3d);

$$x_c = 10 + 2cos(\pi ft)$$
$$y_c = 10 + 0.002t^2$$

In all cases the initial point of the **gripper** was set at (5,5,10) and the simulation interval of 600 seconds was used. For simplicity, results for each case are summarised in the following figures. Fig. 9.4 shows four plots for case 1 (i.e. *translation motion*) where (a) shows the comparison between actual and predicted motions, (b) is the root locus of the closed-loop system, (c) compares the motions of object and gripper along Z-axis versus time and (d) compares the motions of gripper and object in X-Y plane. Fig. 9.5 shows the motion position errors along the three axes versus time and Fig. 9.6 shows the input forces along each axis versus time. The main performance criteria was improved settling time as compared to other existing standard control methods such as PID controllers. The comparison can easily be shown by tuning the control parameters as discussed in Chapter 6.

The accuracy of the predictor depends on the tolerance specified and the complexity of the trajectory of the moving object. In this case, a tolerance of 0.1 units of position was used as a maximum allowable error from the predictor. Moreover, it was found that two prediction step interval gives more accurate tracking results of the moving object. It was also found that a smaller period length increases the number of *waves* of a sinus or cosine function within a certain time period. The waves are then narrower and have the same effect as *spikes*. In the limiting case these spikes act as discontinuities which affects the tracking performance. Similar results for the other three cases (i.e. *rotation, free & contact motions*) are presented in Figures 9.7 through 9.17. However, in the case of free motion, position errors along X, Y, and Z axes are zero (i.e. coincides with the time axis as shown in Figure 9.12). In general the results are satisfactory enough and further efforts to improve the performance of the controller may be studied by introducing the new robust control theories such as H_∞ approach. However, the overall system is stable and is amenable to real-time performance.

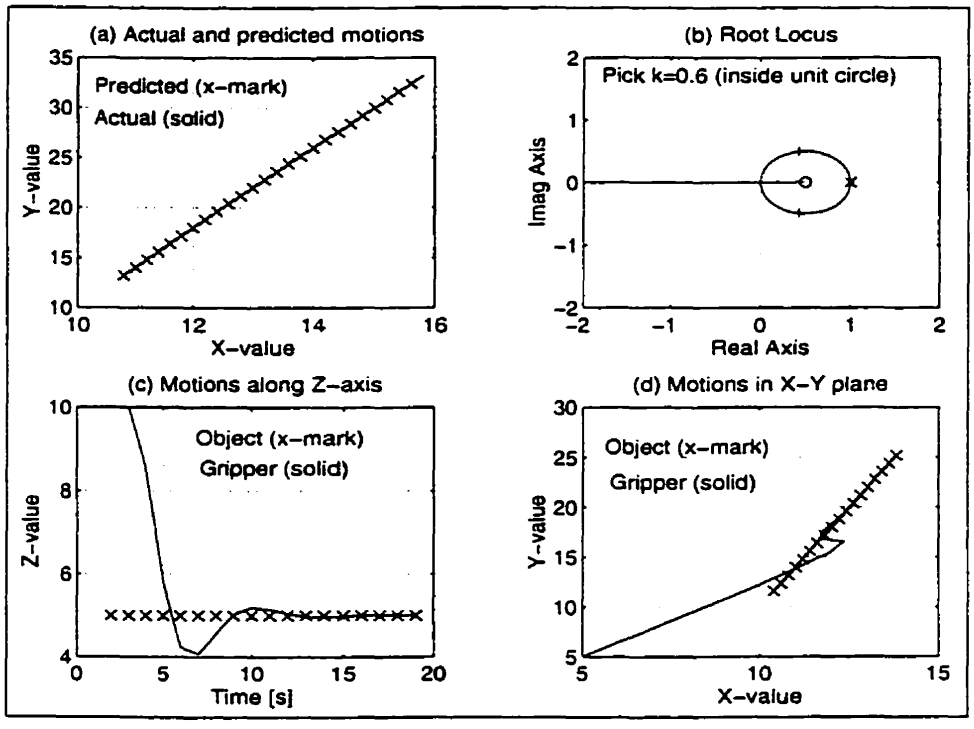


Figure 9.4: Analysis for translation motion: ramp-like

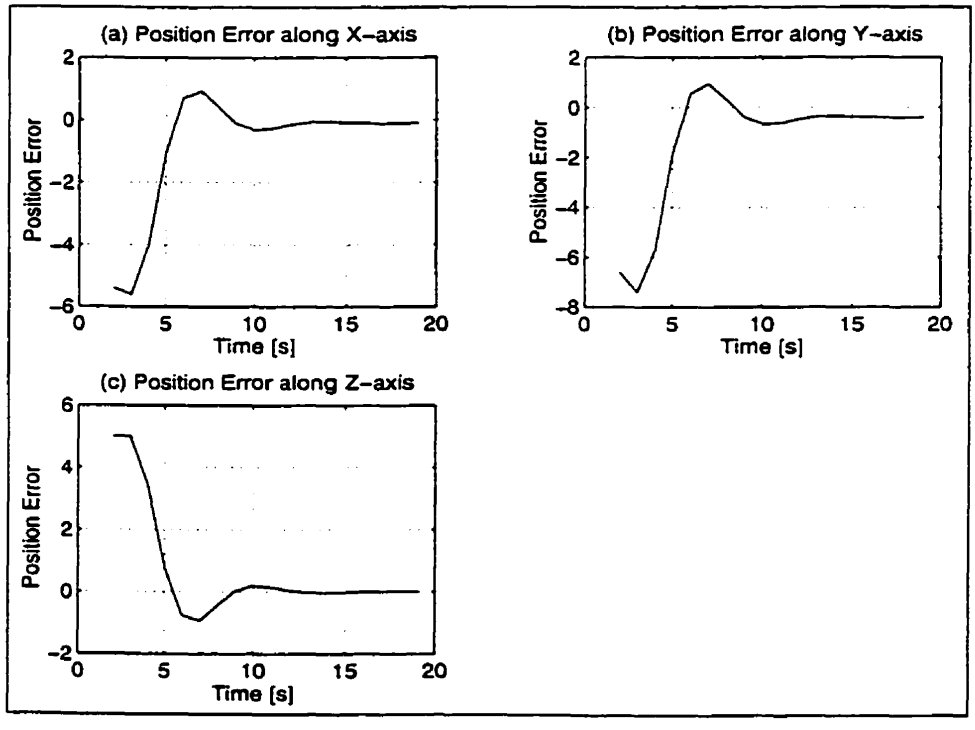


Figure 9.5: Position errors for ramp-like motion

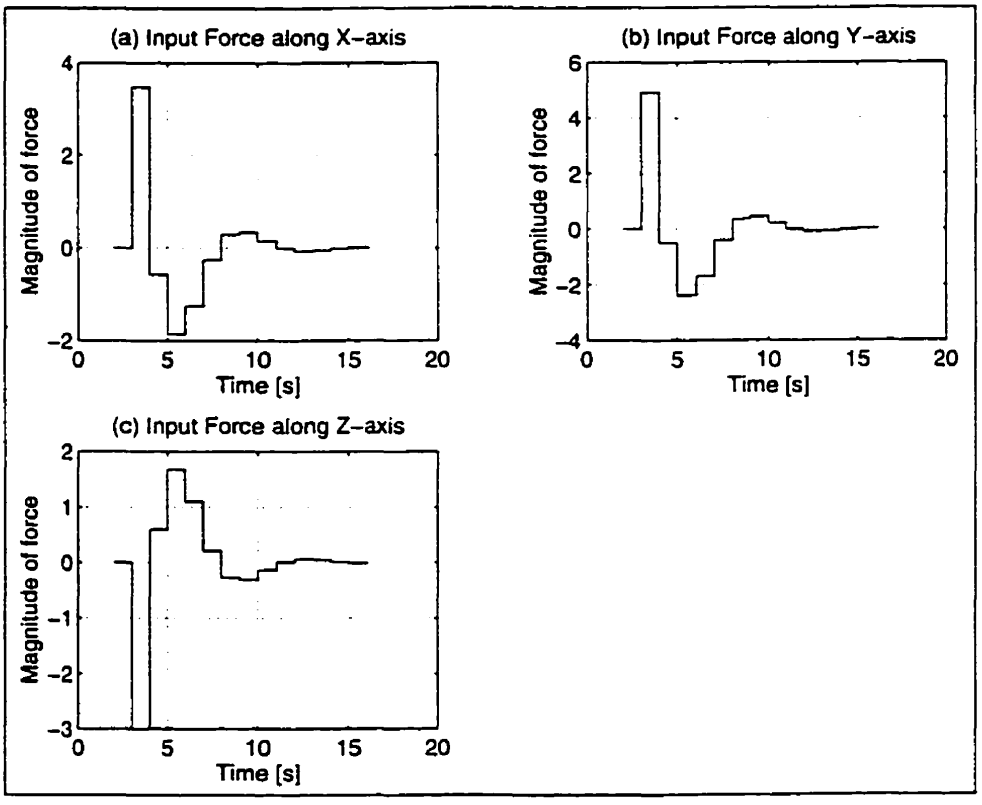


Figure 9.6: Input forces for ramp-like motion

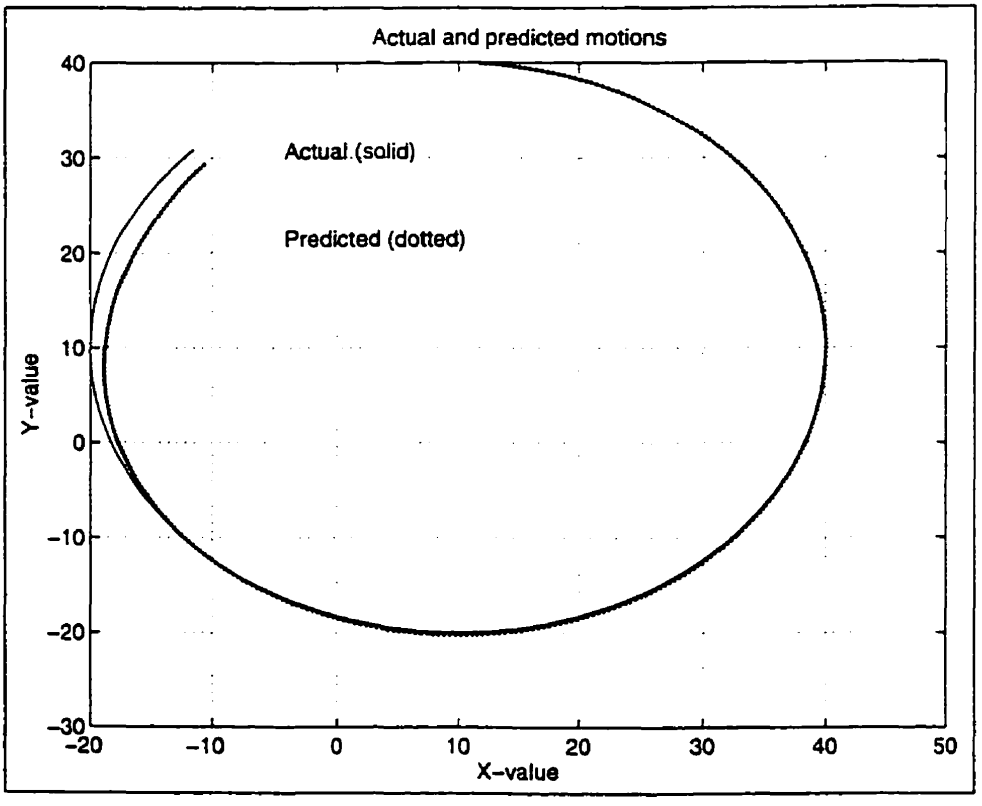


Figure 9.7: Predictor performance for rotary-like motion

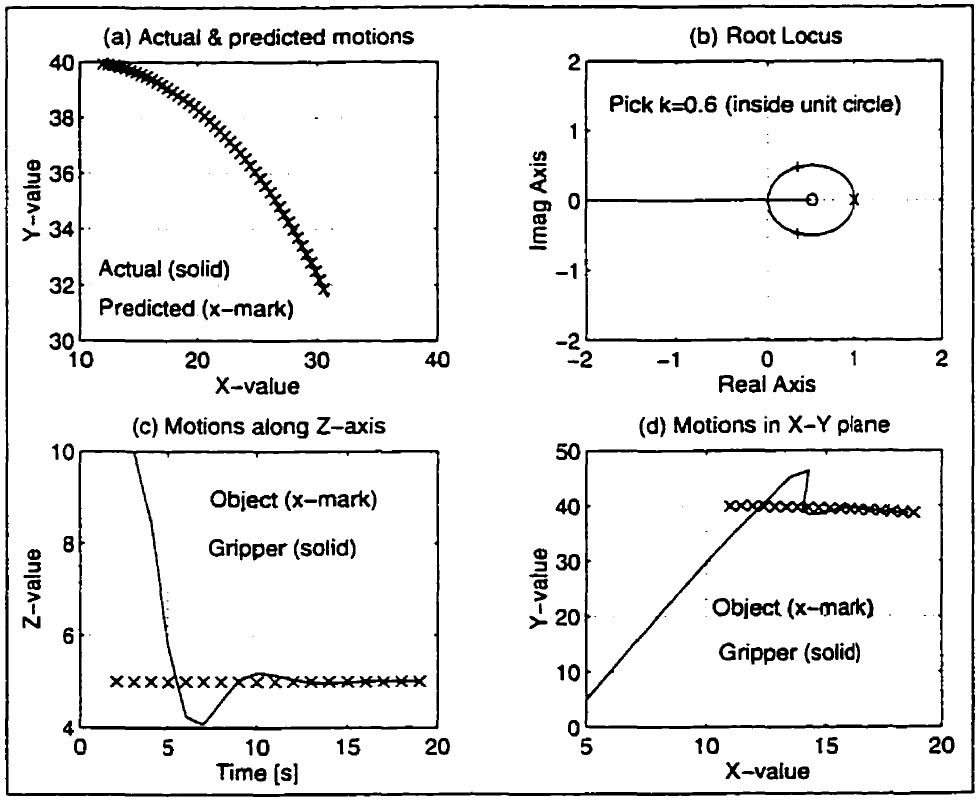


Figure 9.8: Analysis for rotation motion

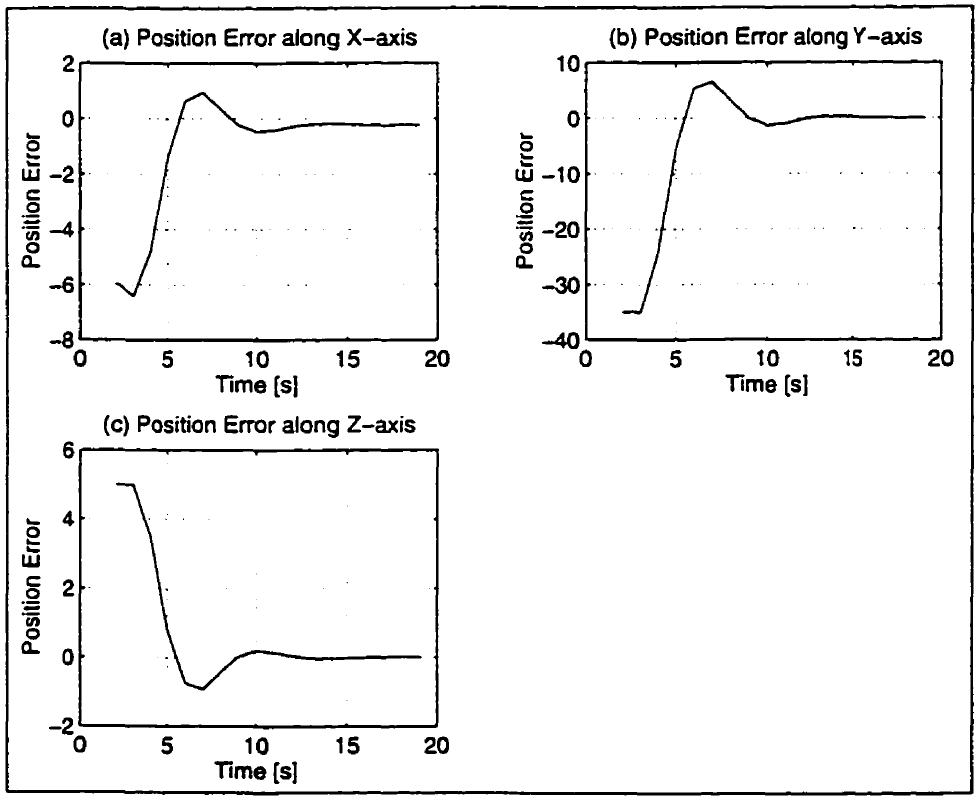


Figure 9.9: Position errors for rotation motion

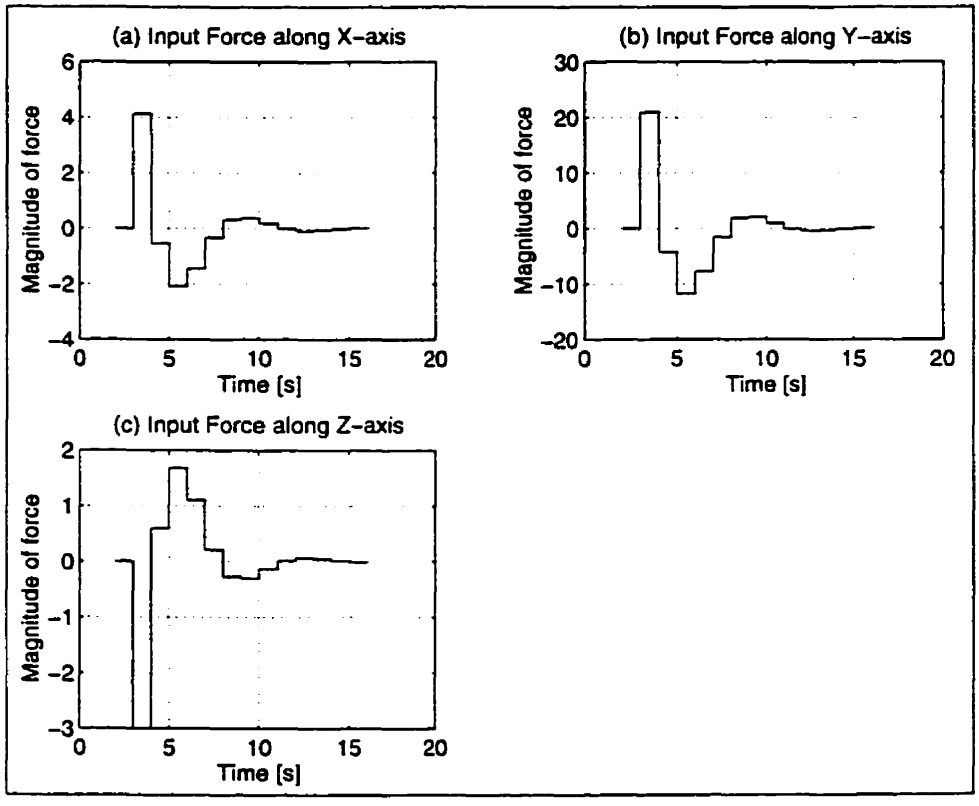


Figure 9.10: Input forces for rotation motion

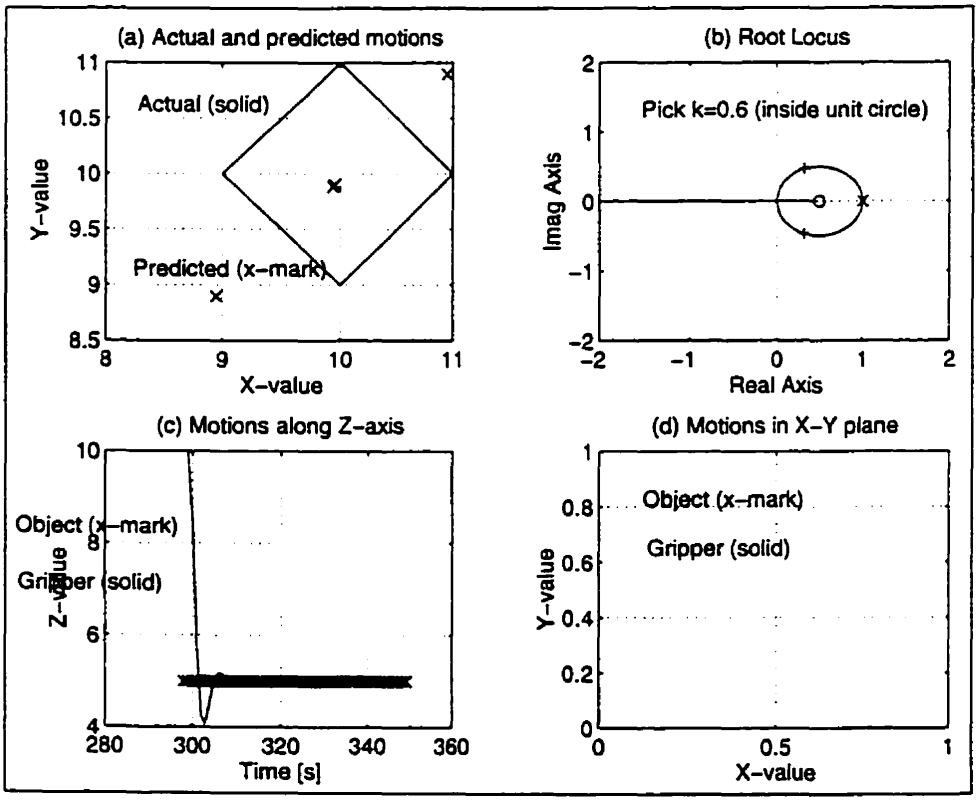


Figure 9.11: Analysis for free motion

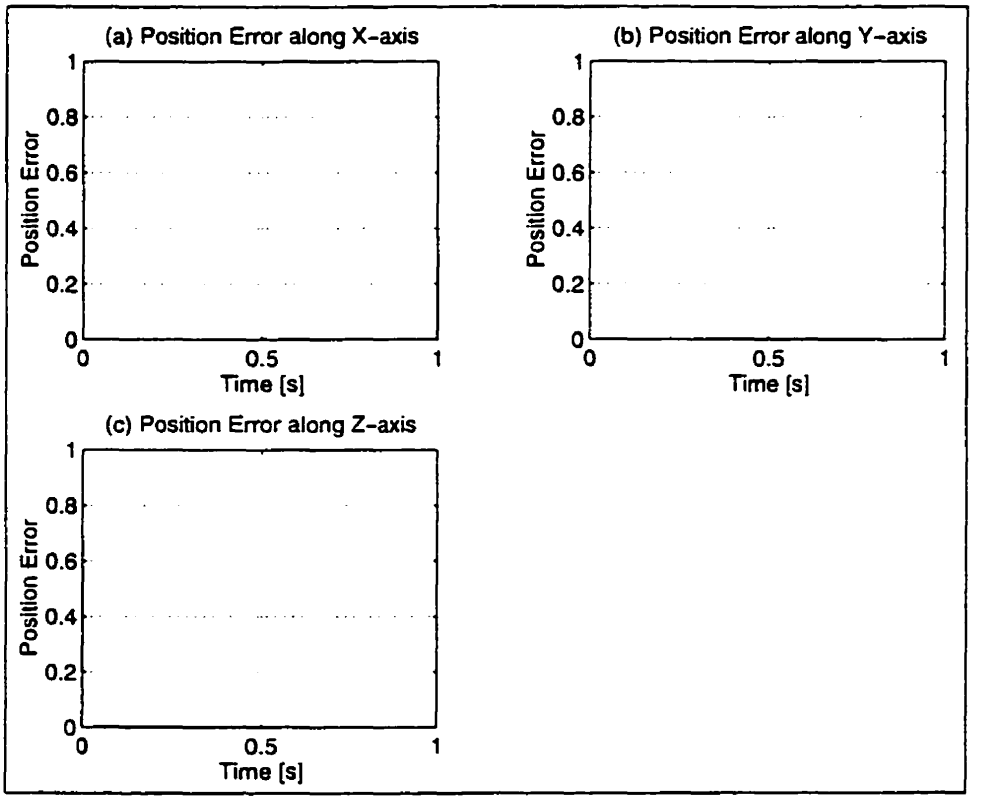


Figure 9.12: Position errors for free motion (zeros in this case)

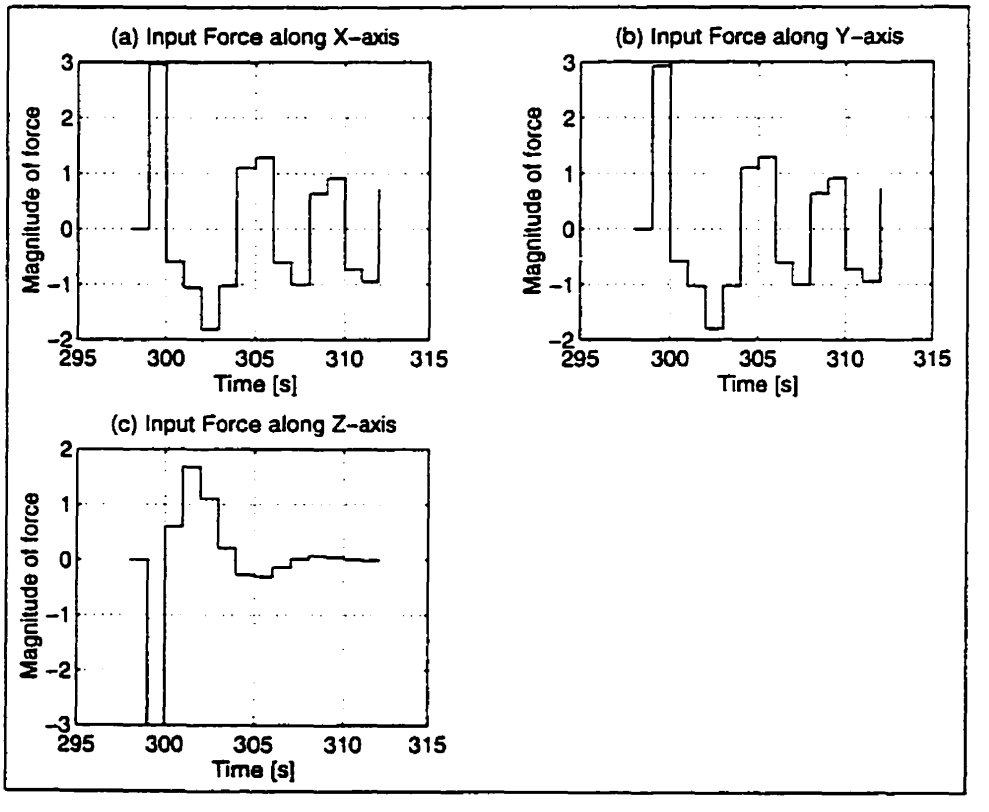


Figure 9.13: Input forces for free motion

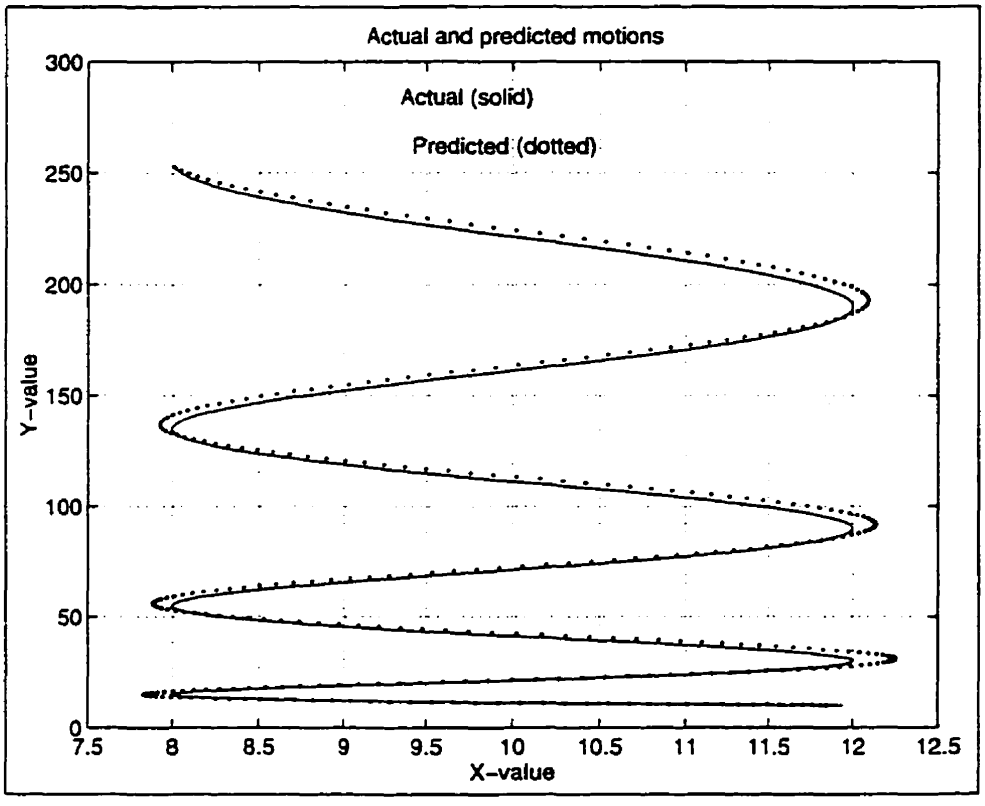


Figure 9.14: Predictor performance for contact motion: wave-like

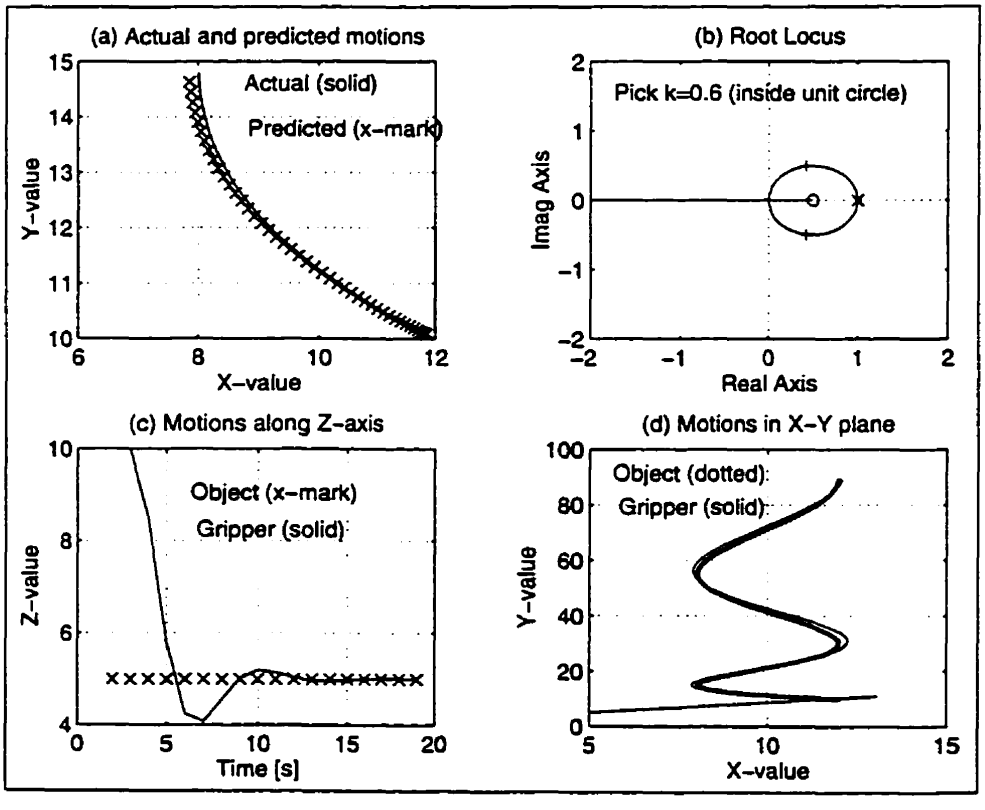


Figure 9.15: Analysis of contact motion

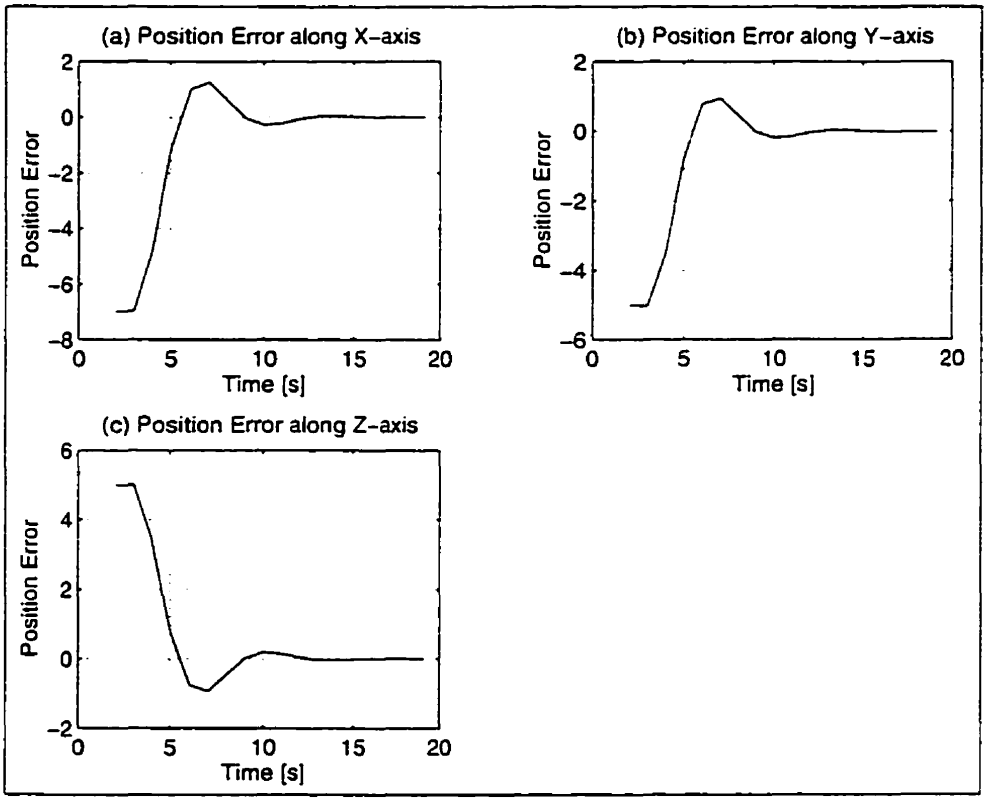


Figure 9.16: Position errors for contact motion

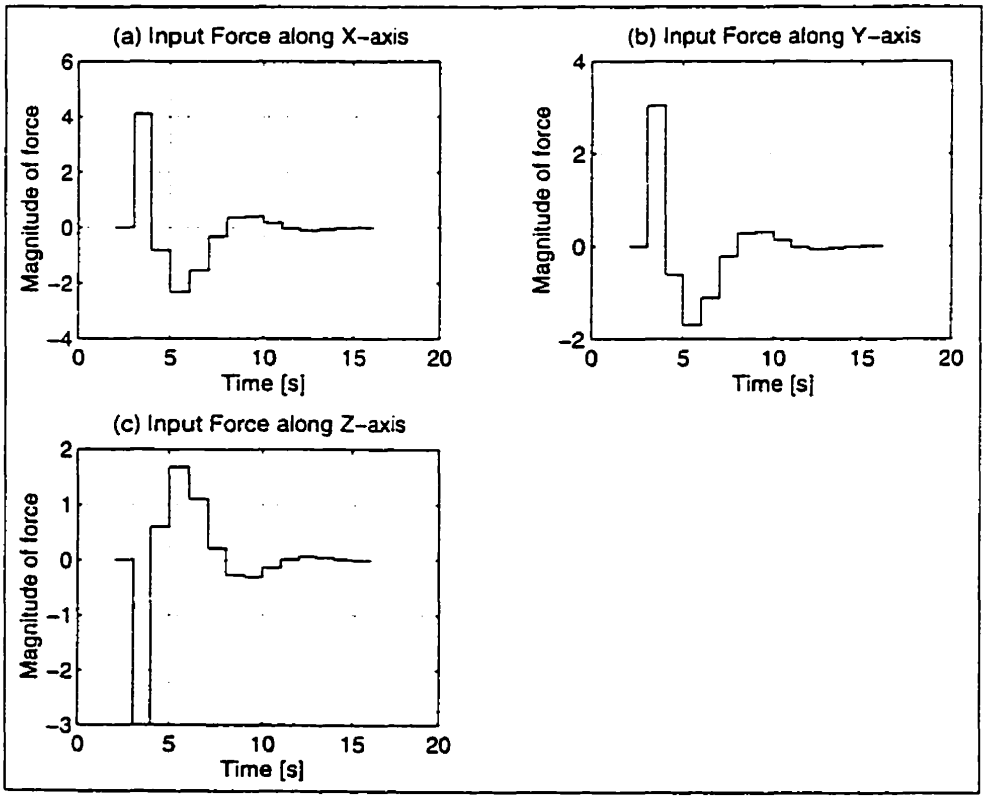


Figure 9.17: Input forces for contact motion

Function	Parameter/Rate
Man-in-the-loop Latency	150 ms to 200 ms
HCI Latency	500 ms
Snapshot Rate	0.2 Hz
Dynamics Equation Integration Rate	1000 Hz
Model Coefficient Update Rate	1 Hz
Control System Update Rate	20 Hz to 2000 Hz
Highest Retained System Frequency	100 Hz
CAE Env. Data Transfer Rate	50 KB at 40 Hz

Table 9.1: MOTS technical performance parameters

9.5 Telecommands for a CDB-based MOTS Simulator

MOTS will play a very important role in preparing astronaut for the operation of MSS. It will provide a faithful simulation of the dynamic and kinematic behaviour, control system algorithms, malfunctions, graphics, thermal properties, power consumption and telemetry of MSS elements. The user will be able to issue CDB telecommands from the MOTS stations to the MSS simulation via the HCI pages and hand-controllers. Camera views of the MSS with different viewpoints will also be shown on video monitors in the MOTS stations. Thus, MOTS will provide a high-fidelity, functional kinematic and dynamic software simulation of the MSS space segment in on-orbit configuration. MOTS will be used for training crew, instructors and ground personnel. MOTS technical performance parameters are as summarized in Table 9.1. As mentioned before, over 10,000 telecommand & telemetry data were formulated and stored in a CDB shared memory where over 150 CSUs will be able to access them in real-time during MOTS training and operation phases (refer to Sections 8.3 through 8.5). This data was mainly deduced from the MOTS system requirements and design specifications. In terms of MOTS simulator, the Dispatcher has the following

capabilities:[1]

- Calls simulation routines in defined order and at regular iteration rates;
- Is **"main"** routine for simulation processes: **sp0**(+ superband), **ap0 & apK** (K > 0);
- Runs SIM modules, and SSS RT modules for snapshot, playback, etc.;
- Superbanding allows iteration rates in excess of the basic simulation rate; and
- Runs a slow asynchronous process (ap0), with stop/start control from the simulation models, allowing for deterministic processing of low priority computations.

MOTS CSC Dispatcher has a **superband** node whose modules are executed a number of times each time a leg is traversed, as opposed to the modules in the other bands which are executed only once. Each level of a tree represents a different time band, and by placing the modules in appropriate strategic bands, their execution rate and execution sequences can be controlled. In this way *real-time synchronization* is not at the level of the dynamic system integration time period, but rather is enforced at the start of each dispatcher time frame period. The dispatcher may have associated with it up to two scheduling trees, i.e. a **critical & non-critical tree**. At the start of each time frame a leg from the critical tree is executed, and if sufficient processing time remains before the beginning of the next time frame a leg from the non-critical tree is then executed. The modules from the critical tree are non-interruptible, and furthermore they can interrupt the execution of the modules from the non-critical tree. The Asynchronous Dispatcher, uses only the non-critical tree. The Synchronous Dispatcher process contains the modules handling output to the VAD CSCI and input from the operator. The dispatcher has only three interfaces: one to PFU to monitor the performance of individual modules in the foreground processes, one to CTS to examine and deposit data in foreground processes, and one to the mother process (MOM) to display message on the system's console (refer to Sections 7.5 & 7.6 for details).

The general performance of all MOTS SIM modules was assessed using the PFU Utility. Moreover, computerized tests were performed for all CDB telecommand data using CTS

[1]For details on CAELIB Dispatcher utility refer to Sections 7.3 & 7.4.

Utility. Finally, best execution rates were defined for each CSU before final scheduling was done. Hence, a preliminary description of the CSUs scheduling, based on the design guidelines described in Chapter 8 and their execution rates, is as follows:[2]

1. **Superband** **1000 Hz:**

- D-FASTLOOP; CSUs belonging to CSC Dynamics:
- D-MALFUNCTION;
- POA-SERVO-CONTROLLER (2X); CSUs belonging to CSC MBS POA:
- POA-HARDWARE (2X);
- POA-EXEC-1;
- POA-MOTOR-POS-AND-RATE (8X);
- POA-EXECUTE-LEEMM-CMDS (8X);
- POA-STATUS-SWITCHES (8X);
- POA-LOAD-CELL (8X);
- POA-DATA-LOGGING (8X);
- LEE-SERVO-CONTROLLER (2X); CSUs belonging to CSC SSRMS LEE:
- LEE-HARDWARE (2X);
- LEE-FMS;
- LEE-EXEC-1 (2X);
- LEE-MOTOR-POS-AND-RATE (8X);
- LEE-EXECUTE-LEEMM-CMDS (8X);
- LEE-STATUS-SWITCHES (8X);
- LEE-LOAD-CELL (8X);
- LEE-DATA-LOGGING (8X);
- JEU-JOINT-CNTRL (2X); CSUs belonging to CSC SSRMS Joint:
- JEU-DATA-LOGGING (8X).

2. **Band X** **20 Hz:**

- ACU-EXEC; CSUs belonging to CSC SSRMS ACU:
- ACU-COMPUTE-STATE;
- ACU-PROCESS-CMDS;
- ACU-PROCESS-LEE-CMDS;
- ACU-MCU-PROCESS-VIDEO-CMDS;
- ACU-PROCESS-TELEMETRY;
- ACU-MALFUNCTION-APPLICABILITY;

[2]The notation nX implies that the module is called n times.

- SSRMS-EXEC;
- SSRMS-HC-INPUTS (2);
- C-SOFTCD-BUFFER;
- ARM-TOP-LEVEL;
- C-SSRMS-MONITOR;
- BI-DIR-ACU-TO-JEU;
- BI-DIR-JEU-TO-ACU;
- LIMP;
- ACU-POWER;
- LEE-EXEC-2; CSUs belonging to CSC SSRMS LEE:
- LEE-COMPUTE-STATE;
- LEE-VALIDATE-LEEMM-CMDS;
- LEE-EXEC-STATUS;
- LEE-SET-MOTOR-DRIVE;
- LEE-CONTROL-MODE;
- LEE-CMD-PRECONDITIONS;
- LEE-PROCESS-TELEMETRY;
- LEE-MALFUNCTION-APPLICABILITY;
- LEE-POWER;
- JEU-COMPUTE-STATE; CSUs belonging to CSC SSRMS Joint:
- JEU-PROCESS-TELEMETRY;
- JEU-MALFUNCTION-APPLICABILITY;
- PHF;
- COP;
- JOINT-POWER;
- VDU-COMPUTE-STATE; CSUs belonging to CSC SSRMS VIDEO:
- VDU-PROCESS-BUS-COMMANDS;
- VDU-PROCESS-SYNC-COMMANDS;
- VDU-PROCESS-VIDEO;
- VDU-PROCESS-BUS-TELEMETRY;
- CAMERA-COMPUTE-STATE;
- CAMERA-PROCERSS-COMMANDS;
- CAMERA-PROCESS-TELEMETRY;
- LIGHT-COMPUTE-STATE;
- SSRMS-VIDEO-COMMANDS-EXEC;
- SSRMS-VIDEO-TELEMETRY-EXEC;
- VIDEO-MALFUNCTION-APPLICABILITY;
- VIDEO-POWER;
- SSRMS-POWER; CSUs belonging to CSC SSRMS Power:
- SSRMS-HEATER-CONTROL;
- MCU-EXEC; CSUs belonging to CSC MBS MCU:

- MCU-COMPUTE-STATE;
- MCU-PROCESS-POA-CMDS;
- MCU-PROCESS-CRPCM-CMDS;
- MCU-PROCESS-TELEMETRY;
- MCU-POWER;
- POA-EXEC-2; CSUs belonging to CSC MBS POA:
- POA-COMPUTE-STATE;
- POA-VALIDATE-LEEMM-CMDS;
- POA-EXEC-STATUS;
- POA-SET-MOTOR-DRIVE;
- POA-CONTROL-MODE;
- POA-CMD-PRECONDITIONS;
- POA-PROCESS-TELEMETRY;
- MBS-PDGF; CSUs belonging to CSC MBS PDGF, Video, Power & Thermal:
- MBS-VIDEO-COMMANDS-EXEC;
- MBS-VIDEO-TELEMETRY-EXEC;
- MBS-POWER;
- MBS-CRPCM-POWER;
- MBS-PROCESS-CRPCM-TELEMETRY;
- MBS-CRPCM-COMPUTE-STATE;
- MBS-CRPCM-PROCESS-FUNCTIONS;
- MBS-THERMAL;
- MBS-BASE-ORIENTATION;
- RWS-CEU; CSUs belonging to CSC RWS, Payload/ORU, & SS:
- PAYLOAD-ORU;
- MSS-VIDEO-EXEC;
- SS-PROCESS-PANEL-COMMANDS;
- SS-VIDEO-PROCESS-ROUTING;
- SS-PROCESS-TELEMETRY;
- SS-COLLECT-CAMERA-DATA;
- SS-SSRMS-CONTROL;
- SS-TIME;
- SS-POWER;
- D-SIM-MODE; CSUs belonging to CSC Dynamics:
- D-FMS;
- D-COLLISION-MON;
- D-SIMTIME;
- D-MONITOR;
- D-ORG-VAR;
- D-DIST-MONITOR;
- D-CONSTRAINT-FORCE;
- TELEMETRY-GENERATION; CSUs belonging to CSC Telemetry:

- TELEMETRY-MODEL-CONTROL;
- TELEMETRY-PLAYBACK.

3. **Band Z** **1 Hz:**

- D-CON-INFO; CSUs belonging to CSC Dynamics:
- D-RT-INIT;
- D-CL-TIP-CORRECTION;
- D-INTERBODY-COORDS;
- D-CL-CONSTRAINT-MATRIX;
- D-CL-NULL-SPACE;
- D-NONLIN-FORCE;
- D-MASS;
- D-CL-RED-MASS;
- D-EIGEN-SOLUTION;
- D-RED-SYSTEM;
- D-SLOW-DONE;
- MBS-THERMAL; CSUs belonging to CSC MBS Thermal:
- MBS-BASE-ORIENTATION;
- SSRMS-THERMAL; CSUs belonging to CSC SSRMS Thermal:
- SSRMS-BASE-ORIENTATION;
- JOINT-THERMAL;
- LEE-THERMAL;
- BOOM-THERMAL;
- ORBITAL-HEAT-RATES; and
- THERMAL-SOLVER.

Chapter 10 Conclusion, Contributions & Future Work

10.1 Conclusion

A *virtuality & reality* of a near-optimal time-delayed teleoperator control system based on the *teleprogramming paradigm* is fully described in this dissertation. Teleoperation is essential while performing tasks at a distance in unstructured environments such as underwater, nuclear contaminations, etc. It therefore becomes imperative to have a human operator present who can guide or supervise, rather than control the robot arm. However, its limiting factor is the communications link between the operator and remote site due to time delays which tend to dis-orient teleoperators and dramatically decrease the system performance. Hence, teleoperator systems tend to be costly, unstable and inefficient due to transmission delays. This is because there is no *supervisory control* from the teleoperator as a result of delayed feedback. This research addressed the application of a *human-machine computer-assisted-interface* under *virtual reality* (i.e. simulation) environments for teleoperator *multi-media* interface using a CDB-based telecommand data in a shared memory. A *CDB-based teleprogramming system* is thus proposed to be used as a telecommand interface to the remote site. During execution, care must be taken to avoid increasing the *lag time* η between the virtual and remote workcell. Hence, a *double-buffering* execution scheme is proposed instead of a classical *dequeue-parse-execute* scheme (i.e. direct sequential management scheme). Some of the advantages of using the system include: reduction of total task time (T_{total}) and overall mission cost during remote manipulations, to achieve safe, efficient and simple operations, to improve the frustrating *move-and-wait* approach (i.e. a back-breaking operation that results in fatigue and frequent errors).

A *CDB-based teleprogramming concept* described in this research provides the necessary bridge between classical teleoperation and fully autonomous remote manipulative capability. Teleprogramming a remote workcell under virtual environments corresponds to

visually, orally, kinesthetically, etc. interacting with a virtual world and on-line, automatically generating a sequence of elementary symbolic telecommands to the remote workcell via a *multi-dimensional* virtual interface devices. Coupled with a small degree of autonomy at the remote site, the system is able to provide for continuous and efficient remote control. The flow of control is interrupted only in case of errors. Error occurrence poses a constraint on the maximum allowable *lag time* η between the virtual and the remote workcell, and thus limits the maximum length of operation time before feedback update that the system can gracefully tolerate.

Virtual reality technology, along with the *teleprogramming concept* have primary applications in underwater and shallow space environments where communication delays preclude direct remote control. However, a CDB-based teleprogramming system can be employed in a non-delayed situations as well such as nuclear plants, biological contaminations, or any other hazardous or unstructured environments. Industrial applications are also viable as such machines can be easily set up under virtual environments by cost-effectively trained operators who can re-program them quickly for a desired task in the simulated environment. In shallow space, a *CDB-based teleprogramming* system can be used 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 centre, vastly reducing both the cost and risks involved during manned missions. Thus, a teleprogrammable CDB-based system (i.e. MOTS) will be used to develop procedures for operating the MSS and to train astronauts, mission controllers and instructors from various workstations (e.g. operator station, instructor station, ground station, on-board station).

10.2 Significance of the Dual Usage of Teleprogramming Methodology

Teleprogramming control methodology is not only meant for space-based systems where we have feedback delays. As shown in Chapter 5 and proven in Section 9.2, this method can be used even in a non-delayed systems. This justifies that space-based technologies can also

be integrated for civil usage. Simulations are a valuable and highly cost effective means to better understand the complexities of long duration space-flight and create systems that can reliably weather the trying demands on crew, complex high technology systems and *human-machine interfaces*, that form a large part of these missions. Results from simulations are used to build prototype vehicles. Unique and versatile facilities already exist that might be adapted for obtaining results for incorporating into a manned space mission vehicle architecture in a timely and economic manner (e.g. International Mars Mission (IMM) [Haule and others, 1991]). The prevailing global political climate should permit the use of surplus super power space or defense assets for civilian uses. In fact, in USA, the federal dual use technology transfer mechanism was initiated a few years ago expressly for this purpose. The thawing of the cold war and consequent reduction of strategic force arsenals among traditional rivals should allow the dual use of U.S. Navy submarines for civilian purposes including it's use as a platform for simulation of long duration manned space missions.

A manned Mars expedition simulation experiment could be an ideal first candidate mission. Systems aboard advanced submarines may have a lot in common with the design of long duration spacecraft including the handling of nuclear fuel and components for power generation and propulsion, environmental control and life support, hard technology driven interior habitat architecture, mission operations procedures, large crew assignments, command and control, as well as more primary habitation issues like crew health and psyche maintenance during the six months to a year long tour of duty, crew productivity enhancement and conflict resolution methods, recreation, nutrition, hygiene and waste management. Though there are substantial differences between navy submarines and civilian spacecraft stemming from the contrastive physical environments in which they operate, their technologies and management, their similarities surely warrant detailed comparative investigation. After all, we may not need to reinvent the wheel regarding many issues, saving precious taxpayer resources in the process.

In a candidate simulation mission, a Mars expedition crew is launched aboard a specially outfitted and programmed Trident class nuclear submarine that attempts to simulate many of the known human factors and environmental parameters on a long duration space mission.

Though free fall conditions are hard to simulate on this platform, *virtual reality* and other stimuli present conditions as might be encountered on a real mission. Programmed, random system failures and malfunctions (i.e. anomalies) are introduced and crew response monitored and evaluated throughout the course of the mission. An attempt is made to develop a civilian command and control code for the crew. At the end of the simulated out-bound trip, the crew are disembarked in a remote region of the globe, posing a hostile environment, such as the coast off Northern Alaska or the Antarctic, where they unload, erect and operate a completely sustainable outpost or perform other tasks for a duration similar to those recommended in prior studies. High bandwidth satellite communications with a built in progressive/regressive Earth-Mars time delay is provided. It may be feasible to use optical links at certain favourable wavelengths to communicate with the submerged vessel.

At the end of their stay and prescribed activity at this outpost, the crew are subjected to the experience of a long duration Earth-bound mission simulation. Emergency evacuation measures are put in place using units of the special tactical armed forces that are placed on alert during the entire course of this mission. Activities in extreme conditions, habit-ability, and human productivity in isolation thus monitored will enhance our understanding of long duration missions. Such a dynamic mission simulation will also help us to ferret out shortcomings and would shed light into the design of several critical *soft* human factors parameters that will have to be dealt within a hard technology environment like that encountered in long duration space missions, where a thorough understanding of human factors as well as the treatment and quality of *human-machine interfaces* will be the major determinant in mission outcome. There are several interesting aspects to this simulation architecture. First of all, this mission simulation can be begun well in advance of space station deployment in 1998. While station activities will focus on human adaptation and performance in free fall, providing high fidelity data on a small sample of astronauts, the submarine simulation, by virtue of the large crew complement and ample enclosed volume can simulate several programmed mission scenarios in parallel. Such a mission could drastically cut short the time otherwise necessary to obtain the statistical data needed for designing and operating a long duration Mars mission spacecraft. Unlike static simulators, this platform would provide a more real setting in a mobile environment, providing dynamic stimuli akin to a real journey.

It may be possible to conduct these missions without much exchange of funds (e.g. between the Navy and NASA of USA), each providing their own hardware and expertise to achieve synergetic results.

As in any other, there are challenges to this architecture as well. The most obvious being, is it possible for the armed forces to really share their assets for elevated civilian, humanitarian purposes like space exploration? If so, why has open literature yet to address these missions regarding relevant habitation parameters, given the fact that the super powers have been plying the ocean depths for decades in these crafts ? A treasure trove of valuable data must exist in the log books and operations manuals of these vessels. However, we cannot compromise national security in handling this information. Traditionally, NASA and the Department of Defense have been at logger-heads over missions and technology transfer matters. Civilian space missions have suffered failures due to a lack of interchange of experience. However, in the recent past, the willingness to cooperate has started to yield significant synergetic and cost effective results. Creative mechanisms could be invented and put in place, so that space and defense related, strategic and sensitive material are safely screened and censored out of this experiment. The access to this knowledge for the design of long duration civilian spacecraft could be a spring board for further cooperation and dual use of space technology. Simulations and the harvest of hard empirical data leading to the rapid and cost effective synthesis of highly effective long duration spacecraft prototypes would follow.

10.3 Contributions

The main contribution of this research work can be summarized as follows:

> *"Formulated a teleprogramming control methodology for the generation, parsing, translation and execution of CDB-based telecommands data elements while overcoming communication delay during remote manipulations (see Chapters 5 & 7); a related teleprogrammable control scheme based on a predictor and a double-loop observer-based feedback is designed with controllability & observability criteria being fulfilled (see Chapter 6). Moreover, a*

double-buffering execution management scheme which ensures a bounded constant lag time between local and remote workcells is introduced (see Sections 5.5 & 9.2). Finally, the meta-interaction of CDB-based telecommands with the simulation modules in real-time is shown to be adequate while using a CDB-based run-time configuration for robotics simulations (see Chapter 8); teleprogramming with modest remote site autonomy is shown to dramatically reduce the effect of communication delays (see Section 9.3)".

A detailed study on the stated problem was carried out in Chapters 1 and 2, and as a result different approaches or ideas were integrated in this research. Chapter 2 summarizes most contributions done by researchers while dealing with the problem from a control point of view using classical teleoperation (i.e. *move-and-wait* strategy). In Chapter 4, the author presented the ubiquity and potential of *virtual reality* in dealing with the problem and promising applicational areas in the future along with their social implications. The originality and novelty of a teleprogramming control paradigm is summarized in Chapters 3, 5 and 6 while the real-time execution management of CDB-based telecommand data elements (or subprograms) is outlined in Chapter 7. A model CDB-based teleprogrammable system (i.e. MOTS) is presented in Chapter 8 where the author initially worked as a CDB-integration specialist. Contribution of this research work is also very well presented in several publications (national and international) as can be deduced from the published papers (two of them ended getting a best paper award in the area of robotics at two different occasions). With this kind of an overall research integration on the stated problem, following are some related contributions in a detailed fashion:

- a 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 local and remote sites is presented;
- an effective approach for providing the operator with realtime feedback information despite the communication delays is formulated; this information is derived by ana-

lyzing the operator's interaction with the virtual world (i.e. simulation) which provides a good approximation to actual *remote presence*.

- a symbolic low level telecommand language 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 is accomplished using CDB labels; thus, on-line reprogramming of a remote workcell is possible and real-time extraction of the corresponding instructional sequences for error recovery;

- a design of a remote site execution strategy, relating to generation, parsing, scheduling and execution management of the incoming instructions, which guarantee a bounded *time lag* based on a *double-buffering* scheme is successfully shown and proven.

- presented a sample design and analysis of a CDB-based simulation system which can be used for teleoperator interface and training using real-time shared memory for telecommand meta-interaction using a DIS module;

- outlined how new computer graphics based technologies can be applied to enhance operator performance via a *human-machine interface* for *multi-media* remote feedback (i.e. visual, audio, tactile, etc.) using CDB labels via HCI pages;

- presented an *observer-based* design for guaranteed stability & controllability to ensure feasibility of near-optimal remote manipulation based on prediction scheme.

10.4 Future Work

The *teleprogramming paradigm* for most telerobotic applications is fully described in this document and can be considered as the basis, upon which more general control methodologies can be developed. However, there are some immediate as well as some of the more far-reaching issues or extensions which need to be addressed:

1. **World modelling for virtual environments**: The issue of constructing a perfect initial model of the remote site is to be considered carefully. However, the necessary technology needed to facilitate interactive off-line construction of the initial remote

environment model exists. What is needed is a relatively low-complexity, practical approach for 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 will participate as a supervisor (i.e. *supervisory control*), resolving necessary ambiguities and ensuring that the extracted model is consistent with the real remote environment.

2. **Special-purpose telecommands and procedures in a virtual world**: Virtual reality motivates the need for special-purpose telecommands in the context of actions, which are more conveniently executed as local high-bandwidth feedback processes at the remote site. Such actions include: fine precision object alignment, grasping and handling of fragile or deformable objects, and high-dexterity dynamically reactive manipulation tasks. Hence, 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. 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 of an iterative subtask. The system can then perform the action repeatedly until some terminating condition is reached.

3. **Error recovery routines**: During error recovery, the state of the virtual graphical simulation is updated to reflect the error state. However, because of the discrepancies between the virtual and the actual remote environment, purely kinematic error status information does not suffice to unambiguously reconstruct the state of the remote workcell. A more sophisticated error recovery mechanism may attempt to monitor the operator's corrective actions and resume autonomous execution. By so doing, it will significantly improve the overall system efficiency as well as operator satisfaction.

4. **On-line model refinement**: Using the above ideas, one may want to take advantage of the interruption during error recovery and try to refine the operator's virtual model of the remote workcell based on the error information packet supplied. The operator's station software can then use this (possibly sparse) local corrective information to refine the geometric relationships in the virtual model.

5. **Virtual editor**: Further extensions of the *teleprogramming control paradigm* under virtual environments, may generalize the idea of having a model-based *virtual editor*. The operator could control multiple robotic and other devices simultaneously through a multiple of available virtual input devices. 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, the operator could interleave task specifications for different agents in the virtual editor, with the execution of the remote site proceeding at a slower, agent-dependent rate. 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.

References

[Adams, 1961] J. Adams, "An investigation of the effects of time lag due to long transmission distances upon remote control: Phase i: Tracking experiments," Tech. Rep. TN D-1211, NASA, Washington, DC, 1961.

[Adams, 1962] J. Adams, "An investigation of the effects of time lag due to long transmission distances upon remote control: Phase ii: Vehicle experiments and phase iii: Conclusions," Tech. Rep. TN D-1351, NASA, Washington, DC, 1962.

[Adnan and Cheatham, 1992] S. Adnan and J. B. Cheatham, "Testbed for remote telepresence research," in *Cooperative Intelligent Robotics in Space III* (J. D. Erickson, ed.), (Boston, MA), pp. 393–400, Nov. 16-18 1992. Vol. 1829.

[Akin and others, 1983] D. Akin *et al.*, "Space applications of automation, robotics and machine intelligence systems: Phase ii, vol.3: Executive summary," contract nasa 8-34381, MIT, Marshall Space Flight Center, 1983.

[Alami and others, 1990] R. Alami *et al.*, "Task level programming for intervention robots," in *IARP 1st Workshop on Mobile Robots for Subsea Environments*, (Monterey, CA), pp. 119–136, October 1990.

[Albus and others, 1986] J. Albus *et al.*, "Nasa standard reference model for telerobot control system architecture," tech. rep., (NASREM) NASA, Robot System Division, 1986.

[Allen and others, 1993] P. Allen *et al.*, "Automated tracking and grasping of a moving object with a robotic hand-eye system," *IEEE Trans. on Robotics & Automation*, vol. 9, pp. 152–165, April 1993.

[Anderson and Davies, 1994] R. Anderson and B. Davies, "Using virtual objects to aid underground storage tank teleoperation," in *IEEE Int. Conf. on Robotics & Automation*, vol. 2, pp. 1421–1427, May 1994.

[Anderson and Spong, 1988] R. Anderson and M. Spong, "Bilateral control of teleoperators with time delay," in *IEEE Int. Conf. on Robotics & Automation*, pp. 131–137, 1988.

[Anderson and Spong, 1989] R. Anderson and M. Spong, "Asymptotic stability for force reflecting teleoperators with time delay," in *IEEE Int. Conf. on Robotics & Automation*, pp. 1618–1625, 1989.

[Apostolos and others, 1992] M. K. Apostolos *et al.*, "Multisensory feedback in advanced teleoperations: Benefits of auditory cues," in *Sensor Fusion V* (P. S. Schenker, ed.), (Boston, MA), pp. 98–105, Nov. 15-17 1992. Volume 1828.

[Appino and others, 1992] P. Appino *et al.*, "An architecture for virtual worlds," *Presence: Teleoperators & Virtual Environments*, vol. 1, pp. 1–17, Winter 1992.

[Archer and Namikas, 1958] E. Archer and G. Namikas, "Pursuit rotor performance as a function of delay of information feedback," *Exper. Psychology*, pp. 325–327, 1958.

[Asada and Izumi, 1987] H. Asada and H. Izumi, "Direct teaching and automatic program generation for the hybrid control of robot manipulators," in *IEEE Int. Conf. on Robotics & Automation*, pp. 1401–1406, 1987.

[Asada and Yang, 1989] H. Asada and B. Yang, "Skill acquisition from human experts through pattern processing of teaching data," in *IEEE Int. Conf. on Robotics & Automation*, pp. 1302–1307, 1989.

[Bailey and others, 1987] R. Bailey *et al.*, "Effect of time delay on manual flight control and flying qualities during in-flight and ground-based simulation," in *AIAA Flight Simulation Technologies Conference*, (New York), p. 2370, 1987.

[Barrette, 1992] R. Barrette, "Wide field of view, full color, high resolution, helmet mounted display," in *Society for Information Display International Symposium*, (Boston), May 1992.

[Bates, 1992] J. Bates, "Virtual reality, art and entertainment," *Presence: Teleoperators & Virtual Environments*, vol. 1, pp. 133–138, Winter 1992.

[Beil and Warrick, 1949] W. Beil and M. Warrick, "Studies in perception of time delay," *American Psychologist*, vol. 4, p. 303, 1949.

[Bejczy and Kim, 1995] A. Bejczy and W. Kim, "Sensor fusion in telerobotic task controls," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 241–249, August, 5-9 1995. Vol. 2.

[Bejczy and others, 1990] A. K. Bejczy *et al.*, "The phantom robot: Predictive displays for teleoperation with time delays," in *IEEE Int. Conf. on Robotics & Automation*, pp. 546–551, May 1990.

[Bejczy, 1980] A. Bejczy, "Sensors, controls and man-machine interface for advanced teleoperation," *Science*, vol. 208, no. 4450, pp. 1327–1335, 1980.

[Bensalah and Chaumette, 1995] F. Bensalah and F. Chaumette, "Compensation of abrupt motion changes in target tracking by visual servoing," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 181–187, August, 5-9 1995. Vol. 1.

[Berkelman and others, 1995] P. Berkelman *et al.*, "Interacting with virtual environments using a magnetic levitation haptic interface," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 117–122, August, 5-9 1995. Vol. 1.

[Berry and others, 1982] D. Berry *et al.*, "Inflight evaluation of control system pure time delays," *Aircraft*, vol. 19, pp. 318–323, 1982.

[Bishop *et al.*, 1994] B. Bishop, S. Hutchison, and M. Spong, "On the performance of state estimation for visual servo systems," in *IEEE Int. Conf. on Robotics & Automation*, vol. 1, pp. 168–173, May 1994.

[Boff and others, 1986] K. Boff *et al.*, *Handbook of perception & Human performance*, vol. I & II. New York: John Wiley, 1986.

[Bolle and Vemuri, 1991] R. Bolle and B. Vemuri, "On three-dimensional surface reconstruction methods," *IEEE Trans. on Pattern Analysis & Machine Intel.*, vol. 13, no. 1, 1991.

[Browse and Little, 1991] R. A. Browse and S. A. Little, "The effectiveness of real-time graphic simulation in telerobotics," in *IEEE Int. Conf. on Systems, Man, & Cybernetics*, pp. 895–898, October 1991. Volume I.

[Bryson and Fisher, 1990] S. Bryson and S. Fisher, "Defining, modeling and measuring system lag in virtual environments," in *SPIE Stereoscopic Displays and Application Conference*, (Bellingham, WA), pp. 98–109, 1990. # 1256.

[Burdea and others, 1992] G. Burdea *et al.*, "A portable dextrous master with force feedback," *Presence: Teleoperators & Virtual Environments*, vol. 1, pp. 18–28, Winter 1992.

[Buzan, 1989] F. Buzan, *Control of telemanipulators with time delay: a predictive operator aid with force feedback.* PhD thesis, M.I.T., 1989.

[CAELIB., 1995s] C. CAELIB., "Caelib on unix systems," Tech. Rep. Release 15, Vol. 1-7, CAE Electronics Ltd., 1995s.

[Caldwell *et al.*, 1994] D. Caldwell, A. Wardle, and M. Goodwin, "Tele-presence: Visual, audio and tactile feedback and control of a twin-armed mobile robot," in *IEEE Int. Conf. on Robotics & Automation*, vol. 1, pp. 244–250, May 1994.

[Campbell, 1988] P. D. Campbell, "Teleoperation and autonomy in space station robotic systems," in *Space Station Automation IV*, (MA), pp. 56–62, Nov. 1988. Vol. 1006.

[Carr and others, 1990] L. Carr *et al.*, "Mdsf - a development and simulation facility for space robotic manipulators," in *Simulators for European Space Programmes Workshop*, (Noordwijk), October 1990.

[Chaumette and Espiau, 1991] F. Chaumette and B. Espiau, "Positioning of a robot with respect to an object, tracking it and estimating its velocity by visual servoing," in *IEEE Int. Conf. on Robotics & Automation*, pp. 2248–2253, 1991.

[Chaumette and Santos, 1993] F. Chaumette and A. Santos, "Tracking a moving object by visual servoing," in *12th World Congress IFAC*, (Sydney), pp. 4098–4104, July 1993.

[Chen, 1989] Y. Chen, "Replacing a pid controller by a lag-lead compensator for a robot - a frequency response approach," *IEEE Trans. on Robotics & Automation*, vol. 5, pp. 174–182, April 1989.

[Cho *et al.*, 1995] Y. Cho, T. Kotoku, and K. Tanie, "Discrete-event-based planning and control of telerobotic part-mating process with communication delay and geometric uncertainty," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 1–6, August, 5-9 1995. Vol. 2.

[Christensen and others, 1991] B. Christensen *et al.*, "Graphical model based control of intelligent robot systems," in *IEEE Int. Conf. on Systems, Man, & Cybernetics*, pp. 1069–1075, 1991. Vol. I.

[Cole and others, 1991] R. E. Cole *et al.*, "Teleoperator performance with virtual window display," in *Stereoscopic Displays & Applications II*, (Bellingham), 1991. Vol. 1457.

[Colgate *et al.*, 1995] J. Colgate, M. Stanley, and J. Brown, "Issues in the haptic display of tool use," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 140–145, August, 5-9 1995. Vol. 3.

[Conklin, 1957] J. Conklin, "Effect of control lag on performance in a tracking task," *Experimental Psychology*, vol. 53, pp. 261–268, 1957.

[Crane, 1983] D. Crane, "Compensation for time delay in flight simulator visual-display systems," in *AIAA Flight Simulation Technologies Conference*, (NY, NASA), pp. 163–171, 1983.

[Crane, 1984] D. Crane, "The effects of time delay in man-machine control systems: implications for design of flight simulator visual-display delay compensation," in *IMAGE 3 Conference*, (NASA Washington, DC), pp. 331–343, 1984.

[Cyril *et al.*, 1992] X. Cyril, N. Kucuk, and L. Wihl, "An engineering and training simulation facility for space robotic manipulators," in *SCS Simulation Multiconference*, (Orlando, Fla), April 1992.

[De Schutter and Leysen, 1987] 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 *Int. Symposium on Robotics Research*, pp. 127–135, 1987.

[De Schutter and Van Brussel, 1988] J. De Schutter and H. Van Brussel, "Compliant robot motion i: a formalism for specifying compliant motion tasks," *International Journal of Robotics Research*, vol. 7, Aug. 1988.

[Desai and Volz, 1989] R. Desai and R. Volz, "Identification and verification of termination conditions in fine motion in presence of sensor errors and geometric uncertainties," in *IEEE Int. Conf. on Robotics & Automation*, pp. 800–807, 1989.

[Deyo and others, 1988] R. Deyo *et al.*, "Getting graphics in gear: graphics and dynamics in driving simulation," *Computer Graphics*, vol. 22, pp. 317–326, 1988.

[Donald, 1986] B. Donald, "Robot motion planning with uncertainty in the geometric models of the robot and environment: a formal framework for error detection and recovery," in *IEEE Int. Conf. on Robotics & Automation*, pp. 1588–1593, 1986.

[Ellis, 1991] S. Ellis, "Nature and origins of virtual environments: a bibliographical essay," *Computer Systems in Engineering*, vol. 2, no. 4, pp. 321–347, 1991.

[Encarnacao and others, 1994] J. Encarnacao *et al.*, "Europian activities in virtual reality," *IEEE Computer Graphics & Applications*, vol. 14, pp. 66–74, Jan. 1994.

[Endo and others, 1995] K. Endo *et al.*, "Trajectory teaching and tracking control by a sequence of image feature points," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 168–173, August, 5-9 1995. Vol. 1.

[Espiau *et al.*, 1992] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *IEEE Trans. on Robotics & Autom.*, vol. 8, pp. 313–326, June 1992.

[Ferrell and Sheridan, 1967] W. Ferrell and T. Sheridan, "Supervisory control of remote manipulation," *IEEE Spectrum*, vol. 4, pp. 81–88, Oct. 1967.

[Ferrell, 1965] W. R. Ferrell, "Remote manipulation with transmission delay," *IEEE Trans. on Human Factors & Electronics*, vol. 6, no. 1, pp. 24–32, 1965.

[Ferrell, 1966] W. R. Ferrell, "Delayed force feedback," *IEEE Trans. on Human Factors in Electronics*, vol. 8, pp. 449–455, Oct. 1966.

[Fisher, 1987] S. Fisher, "Telepresence in dataspace," in *Symposium and Workshop on Spatial Displays and Spatial Instruments*, (Asilomar, CA), 1987.

[Foley, 1987] J. Foley, "Interfaces for advanced computing," *Scientific American*, vol. 257, pp. 126–135, Oct. 1987.

[Fong *et al.*, 1986] C. Fong, R. Dotson, and A. Bejczy, "Distributed microcomputer control system for advanced teleoperation," in *IEEE Int. Conf. on Robotics & Automation*, pp. 987–995, 1986.

[Frank and others, 1988] L. Frank *et al.*, "Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator," *Human Factors*, vol. 30, pp. 201–217, 1988.

[Franklin and others, 1990] G. Franklin *et al.*, *Digital Control of Dynamic Systems*. Addison Wesley, 2 ed., 1990.

[Friedmann and others, 1992] M. Friedmann *et al.*, "Synchronization in virtual realities," *Presence: Teleoperators & Virtual Environments*, vol. 1, pp. 139–144, Winter 1992.

[Funda and others, 1992] J. Funda *et al.*, "Teleprogramming: Toward delay-invariant remote manipulation," *Presence: Teleoperators & Virtual Environments*, vol. 1, no. 1, pp. 29–44, 1992.

[Funda, 1991] J. Funda, *Teleprogramming: Towards Delay-Invariant Remote Manipulation*. PhD thesis, CIS Dept., Moore School, University of Pennsylvania, Philadelphia, PA 19104, August 1991.

[Garant and others, 1995] E. Garant *et al.*, "Three-dimensional modelling for a virtual reality operator training simulator," in *Stockholm PowerTech Conference*, (Stockholm), June 18-22 1995.

[Garvey and others, 1958] W. Garvey *et al.*, "Differential effects of display lags and control lags on the performance of manual tracking systems," *Experimental Psychology*, vol. 56, pp. 8–10, 1958.

[Giralt and others, 1989] G. Giralt *et al.*, "Autonomy versus teleoperation for intervention robots? a case for task level teleprogramming," in *Intelligent Autonomous Systems-2*, (Amsterdam), 1989.

[Giralt and others, 1991] G. Giralt *et al.*, "Remote operated autonomous robots," in *Int. Symposium on Intelligent Robotics*, (India), Jan. 1991.

[Giralt and others, 1993] S. Giralt *et al.*, "Remote intervention, robot autonomy, and teleprogramming: Generic concepts and real-world application cases," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 314–320, 1993.

[Goforth and Dominy, 1988] A. Goforth and R. Dominy, "Ai, automation and the flight telerobotic servicer," in *Space Station Automation IV*, (Cambridge, MA), pp. 2–11, Nov. 7-9 1988. Vol. 1006.

[Grinstein and others, 1992] G. Grinstein *et al.*, "Intelligent virtual interfaces for telerobotics," in *Cooperative Intelligent Robotics in Space III* (J. D. Erickson, ed.), (Boston, MA), pp. 401–408, Nov. 16-18 1992. Vol. 1829.

[Grossman and Taylor, 1978] D. Grossman and R. Taylor, "Interactive generation of object models with a manipulator," *IEEE Trans. on Systems, Man & Cybernetics*, vol. 8, pp. 667–679, Sep. 1978.

[Hacksel and Salcudean, 1994] P. Hacksel and S. Salcudean, "Estimation of environment forces and rigid-body velocities using observers," in *IEEE Int. Conf. on Robotics & Automation*, vol. 2, pp. 931–936, May 1994.

[Hannaford, 1989] B. Hannaford, "A design framework for teleoperators with kinesthetic feedback," *IEEE Trans. on Robotics & Autom.*, vol. 5, no. 4, 1989.

[Hashimoto, 1993] K. Hashimoto, *Visual Servoing*. Singapore: World Scientific, 1993.

[Haule and Malowany, 1994a] D. Haule and A. Malowany, "Real-time teleprogramming of remote robotic workcells," in *SME 5th World Conf. on Robotics Research*, (Boston), pp. 5:31–5:49, Sep. 1994.

[Haule and Malowany, 1994b] D. Haule and A. Malowany, "Teleprogramming control paradigm for remote robotic workcells," in *Canadian Conf. on Electrical & Computer Eng.*, vol. II, (Halifax), pp. 801–805, Sep. 1994.

[Haule and Malowany, 1995a] D. Haule and A. Malowany, "Control scheme for delayed teleoperation tasks," in *IEEE PACRIM Conf. on Communications, Computers & Signal processing*, (Victoria), pp. 157–160, May 1995.

[Haule and Malowany, 1995b] D. Haule and A. Malowany, "Real-time teleprogramming of remote robotic workcells," *SME Trans. on Robotics Research*, vol. SME-94-53, p. (21), (accepted) 1995.

[Haule and Malowany, 1995c] D. Haule and A. Malowany, "Tele-programming methodology: for real-time motion control of remote robotic workcells while overcoming communication delays," *Systems Engineering: Special Issue on Motion Control Systems*, vol. 5, pp. 133–147, August 1995.

[Haule and others, 1991] D. Haule *et al.*, "International mars mission," technical report, International Space University (ISU), Toulouse, France, 1991.

[Hayward, 1995] V. Hayward, "Toward a seven axis haptic device," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 133–139, August, 5-9 1995. Vol. 3.

[Held and Durlach, 1991] R. Held and N. Durlach, *Telepresence, time delay and adaptation*. London: Taylor & Francis Ltd., 1991.

[Helsel and Roth, 1991] S. Helsel and J. Roth, *Virtual Reality Theory, Practice and Promise*. USA: Meckler, 1991.

[Hess, 1984] R. Hess, "Effects of time delays on systems subject to manual control," *Guidance, Control & Dynamics*, vol. 7, pp. 416–421, 1984.

[Hirai and Asada, 1993] S. Hirai and H. Asada, "Kinematics and statics of manipulation using the theory of polyhedral convex cones," *Int. Journal of Robotics Research*, vol. 12, no. 5, pp. 435–447, 1993.

[Hirzinger and others, 1989] G. Hirzinger *et al.*, "Predictive and knowledge-based telerobot control concepts," in *IEEE Int. Conf. on Robotics & Autom.*, pp. 1768–1777, 1989.

[Hoffman, 1989] R. Hoffman, "Automated assembly in a csg domain," in *IEEE Int. Conf. on Robotics & Automation*, 1989.

[Hogan, 1980] N. Hogan, "Mechanical impedance control in assistive devices and manipulators," in *Joint Automatic Control Conference*, (San Francisco), pp. TA10–B, 1980.

[Hui and Gregorio, 1995] R. Hui and P. Gregorio, "The virtual handle," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 127–133, August, 5-9 1995. Vol. 3.

[Ikeuchi and Suehiro, 1994] K. Ikeuchi and T. Suehiro, "Toward an assembly plan from observation, part i: Task recognition with polyhedral objects," *IEEE Trans. on Robotics & Automation*, vol. 10, pp. 368–385, June 1994.

[Inc., 1993a] P. Inc., *3Space User's Manual*. Polhemus, 1993.

[Inc., 1993b] V. R. Inc., *DataGlove Model 4 Operation Manual*. CA: VPL, 1993.

[Ince and others, 1991] I. Ince *et al.*, "Virtuality and reality: A video/graphics environment for teleoperation," in *Int. Conf. on Systems, Man & Cybernetics*, (Charlottsville, VA), pp. 1083–1089, October 1991.

[Isidor and Byrnes, 1990] A. Isidor and C. Byrnes, "Output regulation of nonlinear systems," *IEEE Trans. on Automatic Control*, vol. 35, no. 2, pp. 131–140, 1990.

[Jennings *et al.*, 1989] 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 *IEEE Int. Conf. on Robotics & Automation*, pp. 623–637, 1989.

[Jimenez and others, 1993] S. Jimenez *et al.*, "Predicting the dynamic behaviour of a planetary vehicle using physical modeling," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 345–351, 1993.

[Kaczor *et al.*, 1993] M. Kaczor, C. Laugier, and E. Mazer, "Tele-act: A task level teleprogramming system," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 310–313, 1993.

[Kahaner, 1994] D. Kahaner, "Japanese activities in virtual reality," *IEEE Computer Graphics & Applications*, vol. 14, pp. 75–78, Jan. 1994.

[Kang and Ikeuchi, 1993] S. Kang and K. Ikeuchi, "Towards automatic robot instruction from perception: Recognizing a grasp from observation," *IEEE Trans. on Robotics & Automation*, vol. 9, no. 4, pp. 432–443, 1993.

[Kawamura and others, 1995] S. Kawamura *et al.*, "Development of a virtual sports machine using a wire drive system - a trial of virtual tennis," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 111–116, August, 5-9 1995. Vol. 1.

[Khatib, 1985] O. Khatib, "The operational space formulation in robot manipulator control," in *15th ISIR*, (Tokyo, Japan), Sep. 1985.

[Kim and Bejczy, 1991] W. Kim and K. Bejczy, "Graphics displays for operator aid in manipulation," in *IEEE Int. Conf. on Systems, Man, & Cybernetics*, pp. 1059–1067, 1991.

[Kim *et al.*, 1990] W. Kim, B. Hannaford, and A. Bejczy, "Shared compliance control for time-delayed telemanipulation," in *1st Int. Symposium on Measurement & Control in Robotics*, June 1990.

[Kim *et al.*, 1993] W. Kim, B. Hannaford, and A. Bejczy, "Force-reflection and shared compliant control in operating telemanipulators with time delays," *IEEE Trans. on Robotics & Automation*, vol. 8, pp. 176–185, 1993.

[Kotoku, 1992] T. Kotoku, "A predictive display with force feedback and its application to remote manipulation system with transmission time delay," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 239–246, July, 7-10 1992. Vol. 1.

[Kuniyoshi *et al.*, 1992] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Seeing, understanding and doing human task," in *IEEE Int. Conf. on Robotics & Automation*, pp. 2–9, 1992.

[Latta and Oberg, 1994] J. Latta and D. Oberg, "A conceptual virtual reality model," *IEEE Computer Graphics & Applications*, vol. 14, pp. 23–29, Jan. 1994.

[Lee and Lee, 1992] S. Lee and H. Lee, "Modeling, design and evaluation for advanced teleoperator control systems," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 881–888, July, 7-10 1992. Vol. 2.

[Lee and Lee, 1994] S. Lee and H. Lee, "Design of optimal time delayed teleoperator control system," in *IEEE Int. Conf. on Robotics & Automation*, pp. 3252–3258, 1994.

[Lee *et al.*, 1994] P. Lee, D. Ruspini, and O. Khatib, "Dynamic simulation of interactive robotic environment," in *IEEE Int. Conf. on Robotics & Automation*, vol. 2, pp. 1147–1152, May 1994.

[Lei and Ghosh, 1993] M. Lei and B. Ghosh, "Visually guided robotic tracking and grasping of a moving object," in *IEEE Conf. on Decision & Control*, pp. 1604–1609, 1993.

[Leslie and others, 1966] J. Leslie *et al.*, "Predictor aided tracking in a system with time delay: performance involving flat surface, roll and pitch conditions," Tech. Rep. CR-75399, NASA, Washington, DC, 1966.

[Liang *et al.*, 1991] J. Liang, C. Shaw, and M. Green, "On temperoral-spatial realism in the virtual reality environment," in *UIST'91 4th Annual ACM Symposium on User Interface Software and Technology*, (New York), pp. 19–25, 1991.

[Lindsay and Paul, 1993] T. Lindsay and R. Paul, "Simplifying tool usage in teleoperative tasks," in *SPIE Int. Conf. on Telemanipulator Technology and Space Telerobotics*, (Boston, MA), pp. 288–298, Nov. 15-16 1993.

[Lindsay, 1992] T. Lindsay, *Teleprogramming: Remote Site Robot Task Execution*. PhD thesis, University of Pennsylvania, 1992.

[Loomis, 1992] J. Loomis, "Distal attribution and presence," *Presence: Teleoperators & Virtual Environments*, vol. 1, pp. 113–119, Winter 1992.

[Lozano-Perez and Brooks, 1985] T. Lozano-Perez and R. Brooks, "An approach to automatic robot programming," A.I. Memo 842, Artificial Intelligence Laboratory, MIT, Cambridge, MA, April 1985.

[Lozano-Perez *et al.*, 1983] T. Lozano-Perez, M. Mason, and R. Taylor, "Automatic synthesis of fine motion strategies for robots," in *Robotics Research: 1st Int. Symposium*, Aug. 1983.

[Lozano-Perez *et al.*, 1984] T. Lozano-Perez, M. Mason, and R. Taylor, "Automatic synthesis of fine motion strategies for robots," *Int. Journal of Robotics Research*, vol. 3, no. 1, pp. 3–24, 1984.

[Lozano-Perez, 1981] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Trans. on Systems, Man & Cybernetics*, vol. 11, pp. 681–689, 1981.

[Luh *et al.*, 1980] J. Luh, M. Walker, and R. Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Trans. on Automatic Control*, vol. 25, pp. 468–474, 1980.

[Marapane and others, 1992a] S. Marapane *et al.*, "Graphical simulation and animation environment for flexible structure robots," in *Sensor Fusion V* (P. S. Schenke, ed.), (Boston, MA), pp. 86–97, Nov. 15-17 1992. Vol. 1828.

[Marapane and others, 1992b] S. Marapane *et al.*, "A real and virtual robot head for active vision research," in *Sensor Fusion V* (P. S. Schenker, ed.), (Boston, MA), pp. 60–71, Nov. 15-17 1992. Vol. 1828.

[Mason, 1981] M. Mason, "Compliance and force control for computer controlled manipulators," *IEEE Trans. on Systems, Man & Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.

[McKee and Schenker, 1995] G. McKee and P. Schenker, "Human-robot cooperation for automated viewing during teleoperation," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 124–129, August, 5-9 1995. Vol. 1.

[McMillan *et al.*, 1994] S. McMillan, D. Orin, and R. McGhee, "Efficient dynamic simulation of an unmanned underwater vehicle with a manipulator," in *IEEE Int. Conf. on Robotics & Automation*, vol. 2, pp. 1133–1140, May 1994.

[Merritt and Cole, 1991] J. Merritt and R. Cole, "A rapid-sequential-positioning task for evaluating motion parallax and stereoscopic 3d cues in teleoperator displays," in *IEEE Int. Conf. on Systems, Man, & Cybernetics*, (Charlottsville, VA), pp. 1041–1046, October 1991. Vol. I.

[Miles and Cannon, 1995] E. Miles and H. Cannon, "Utilizing human vision and computer vision to direct a robot in a semi-structured environment via task-level commands," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 366–371, August 1995.

[Milgram and others, 1995] P. Milgram *et al.*, "Telerobotic control using augmented reality," in *IEEE Int. Workshop on Robot & Human Communication*, pp. 906–913, 1995.

[Miner and Stansfield, 1994] N. Miner and S. Stansfield, "An interactive virtual reality simulation system for robot control and operator training," in *IEEE Int. Conf. on Robotics & Automation*, vol. 2, pp. 1428–1435, May 1994.

[Mitsuishi and others, 1995] M. Mitsuishi *et al.*, "Experiments in tele-handling and telemachining at the macro and micro scales, using the internet for operational environment transmission," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 13–20, August, 5-9 1995. Vol. 2.

[Morie, 1994] J. Morie, "Inspiring the future: merging mass communication, art, entartainment and virtual environments," *Computer Graphics*, vol. 28, pp. 135–138, May 1994.

[Noorhosseini and Malowany, 1994a] S. Noorhosseini and A. Malowany, "Robot action planning using state matrix representation," in *Canadian Conf. on Electrical & Computer Eng.*, (Halifax), pp. 789–792, Sep. 1994.

[Noorhosseini and Malowany, 1994b] S. Noorhosseini and A. Malowany, "State matrix representation of assembly and robot planning," *Robotica, Int. Journal of Information and Research in Robotics and Artificial Intelligence*, p. (32), (accepted) 1994.

[Ntuen and others, 1991] C. Ntuen *et al.*, "An experiment in human-robot interaction during task execution," in *IEEE Int. Conf. on Systems, Man, & Cybernetics*, (Charlottsville, VA), pp. 1213–1218, Oct. 1991. Vol. I.

[Ogata and Takahashi, 1994] H. Ogata and T. Takahashi, "Robotic assembly operation teaching in a virtual environment," *IEEE Trans. on Robotics & Automation*, vol. 10, pp. 391–399, June 1994.

[Okapuu-von Veh and others, 1996] R. E. Okapuu-von Veh *et al.*, "Design and operation of a virtual reality operator training system," in *IEEE Winter Power Meeting*, vol. (also to appear in IEEE Trans. Power Systems), pp. paper 96 WM 157–8 PWRS, February 1996.

[Onda and others, 1995] H. Onda *et al.*, "Assembly motion teaching system using position/force simulator - extracting a sequence of contact state transition," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 9–16, August, 5-9 1995. Vol. 1.

[Ouh-young *et al.*, 1989] M. Ouh-young, D. Beard, and F. Brooks, "Force display performs better than visual display in a simple 6-d docking task," in *IEEE Int. Conf. on Robotics & Automation*, pp. 1462–1466, 1989.

[Papanikolopoulos *et al.*, 1993] N. Papanikolopoulos, P. Kholsa, and T. Kanade, "Visual tracking of a moving target by a camera mounted on a robot: a combination of control and vision," *IEEE Trans. on Robotics & Automation*, vol. 9, pp. 14–35, February 1993.

[Papanikolopoulos *et al.*, 1994] N. Papanikolopoulos, B. Nelson, and P. Kholsa, "Six degree-of-freedom hand/eye visual tracking with uncertain parameters," in *IEEE Int. Conf. on Robotics & Automation*, vol. 1, pp. 174–180, May 1994.

[Paul and Ikeuchi, 1995] G. Paul and K. Ikeuchi, "Modelling planar assembly tasks: Representation and recognition," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 17–22, August, 5-9 1995. Vol. 1.

[Paul *et al.*, 1992] R. Paul, T. Lindsay, and C. Sayers, "Time delay insensitive teleoperation," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 247–254, July, 7-10 1992. Vol. 1.

[Paul *et al.*, 1993] R. Paul, C. Sayers, and M. Stein, "The theory of teleprogramming," *Robotics Society of Japan*, vol. 11, no. 6, pp. 14–19, 1993.

[Pausch *et al.*, 1992] R. Pausch, T. Crea, and M. Conway, "A literature survey for virtual environments: Military flight simulator visual systems and simulator sickness," *Presence: Teleoperators & Virtual Environments*, vol. 1, no. 1, pp. 344–363, 1992.

[Pook and Ballard, 1993] P. Pook and D. Ballard, "Recognizing teleoperated manipulations," in *IEEE Int. Conf. on Robotics & Automation*, pp. 578–585, 1993.

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

[Raju and others, 1989] G. Raju *et al.*, "Design issues in 2-port network models of bilateral remote manipulation," in *IEEE Int. Conf. on Robotics & Autom.*, pp. 1316–1321, 1989.

[Ravela and others., 1995] S. Ravela and others., "Adaptive tracking and model registration across distinct aspects," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 174–180, August, 5-9 1995. Vol. 1.

[Ribarsky and others, 1994] W. Ribarsky *et al.*, "Visualization and analysis using virtual reality," *IEEE Computer Graphics & Applications*, vol. 14, pp. 10–12, Jan. 1994.

[Richards and Papanikolopoulos, 1995] C. Richards and N. Papanikolopoulos, "The automatic detection and visual tracking of moving objects by eye-in-hand robotic systems," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 228–233, August, 5-9 1995.

[Rosenblum, 1994] L. Rosenblum, "Research issues in scientific visualization: special report," *IEEE Computer Graphics & Applications*, vol. 14, pp. 61–85, March 1994.

[Salisbury, 1980] J. Salisbury, "Active stiffness control of a manipulator in cartesian coordinates," in *19th IEEE Conference on Decision and Control*, Dec. 1980.

[Sanderson and others, 1988] A. Sanderson *et al.*, "Task planning for robotic manipulation in space," *IEEE Trans. on Aerospace & Electronic Systems*, pp. 619–628, 1988.

[Sayers *et al.*, 1992] C. Sayers, R. Paul, and M. Mintz, "Operator interaction and teleprogramming for subsea manipulation," in *Fourth IARP Workshop on Underwater Vehicles*, Nov. 1992.

[Schebor and Turney, 1991] F. Schebor and J. Turney, "Realistic and consistent telerobotic simulation," in *IEEE Int. Conf. on Systems, Man, & Cybernetics*, (Charlottsville, VA), pp. 889–894, Oct. 1991. Vol. I.

[Schenker *et al.*, 1991] P. Schenker, A. Bejczy, W. Kim, and K. Lee, "Advanced man-machine interfaces and control architecture for dexterous teleoperations," in *IEEE Oceans*, (Honolulu, HI), Oct. 1991.

[Sharma and Hutchison, 1994] R. Sharma and S. Hutchison, "On the observability of robot motion under active camera control," in *IEEE Int. Conf. on Robotics & Automation*, vol. 1, pp. 162–167, May 1994.

[Sheridan and Ferell, 1963] T. Sheridan and W. Ferell, "Remote manipulative control with transmission delay," *IEEE Trans. on Human Factors in Electronics*, pp. 25–29, 1963.

[Sheridan, 1980] T. B. Sheridan, "Computer control and human alienation," *Technology Review*, vol. 83, pp. 60–73, October 1980.

[Sheridan, 1989] T. Sheridan, "Telerobotics," in *IEEE Int. Conf. on Robotics & Automation (Workshop: Integration of AI & Robotic Systems)*, 1989.

[Sheridan, 1992] T. B. Sheridan, *Telerobotics, automation and human supervisory control.* London, England: MIT Press, Cambridge, MA, 1992.

[Sheridan, 1993] T. B. Sheridan, "Space teleoperation through time delay: Review and prognosis," *IEEE Trans. on Robotics & Automation*, vol. 9, no. 5, pp. 592–606, 1993.

[Shimoga *et al.*, 1995] K. Shimoga, A. Murray, and P. Khosla, "A touch reflection system for interaction with remote and virtual environments," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, p. 123, August, 5-9 1995. Vol. 1.

[Shimoga, 1993] K. Shimoga, "A survey of perceptual feedback issues in dexterous telemanipulation: Part i: Finger force feedback; part ii: Finger touch feedback," in *IEEE Virtual Reality Annual Int. Symposium*, (Seattle, WA), pp. 263–279, Sept., 18-22 1993.

[Simon and others, 1994] L. Simon *et al.*, "Autonomous navigation in outdoor environment: Adaptive approach and experiment," in *IEEE Int. Conf. on Robotics & Automation*, pp. 426–432, 1994.

[Smith and others, 1987] J. Smith *et al.*, "The space station assembly-phase: Flight telerobotic servicer feasibility," JPL Reports Vol. I and II, JPL, Sept. 1987.

[Smith and Sarafian, 1986] R. Smith and S. Sarafian, "Effect of time delay on flying qualities: an update," *Guidance, Control and Dynamics*, vol. 9, pp. 578–584, 1986.

[Smith, 1963] K. Smith, "Special review: Sensory feedback analysis in medical research: delayed sensory feedback in behavior and neural function," *American Journal of Physical Medicine*, vol. 42, no. 2, pp. 49–84, 1963.

[Stark and others, 1987] L. Stark *et al.*, "Telerobotics: display, control and communications problems," *IEEE Robotics & Automation*, vol. 3, pp. 67–75, Feb. 1987.

[Stein and others, 1995] M. Stein *et al.*, "A cross-country teleprogramming experiment," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 21–26, August 1995.

[Stein and Paul, 1992] M. Stein and R. Paul, "Kinesthetic replay for error diagnosis in time delayed teleoperation," in *Telemanipulator Technology* (H. Das, ed.), (Boston, MA), pp. 278–287, Nov. 15-16 1992. Vol. 1833.

[Stein and Paul, 1994] M. Stein and R. Paul, "Operator interaction, for time-delayed teleoperation, with a behavior-based controller," in *IEEE Int. Conf. on Robotics & Automation*, vol. 1, pp. 231–236, May 1994.

[Stein, 1993] R. Stein, "Portability issues in teleprogramming: a case study," in *SPIE Int. Conf. on Telemanipulator Technology and Space Telerobotics*, (Boston, MA), pp. 84–95, Sept. 7-9 1993.

[Sturman and Zeltzer, 1994] D. Sturman and D. Zeltzer, "A survey of glove-based input," *IEEE Computer Graphics & Applications*, vol. 14, pp. 30–39, Jan. 1994.

[Sutherland, 1965] I. Sutherland, "The ultimate display," in *IFIP Congress*, pp. 506–508, 1965. Vol. 65.

[Takahashi and Ogata, 1992] T. Takahashi and H. Ogata, "Robotic assembly operation based on task-level teaching in virtual reality," in *IEEE Int. Conf. on Robotics & Automation*, pp. 1083–1088, 1992.

[Tarn and others, 1995] T. Tarn *et al.*, "Function-based control sharing for robotic systems," in *IEEE/RSJ Int. Conf. on Intel. Robots & Systems*, pp. 1–6, August 1995.

[Thayer and others, 1992] S. Thayer *et al.*, "Three-dimensional sensing, graphics and interactive control in a human-machine system for decontamination and decommissioning applications," in *Sensor Fusion V* (P. S. Schenker, ed.), (Boston, MA), pp. 74–85, Nov. 15-17 1992. Vol. 1828.

[Tso and Liu, 1995] S. Tso and K. Liu, "Automatic generation of robot program codes from perception of human demonstration," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 23–28, August, 5-9 1995. Vol. 1.

[Tung and Kak, 1995] C. Tung and A. Kak, "Automatic learning of assembly tasks using a dataglove system," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 1–8, August, 5-9 1995. Vol. 1.

[Voyles and Khosla, 1995] R. Voyles and P. Khosla, "Tactile gestures for human/robot interaction," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 7–13, August, 5-9 1995. Vol. 3.

[Wakita and others, 1992] Y. Wakita *et al.*, "Automatic camera-work control for intelligent monitoring of telerobotic tasks," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 1130–1135, July, 7-10 1992. Vol. 2.

[Wakita *et al.*, 1995] Y. Wakita, S. Hirai, and K. Machida, "Intelligent monitoring system for limited communication path: Telerobotic task execution over internet," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pp. 104–109, August, 5-9 1995. Vol. 2.

[Walters, 1994] W. Walters, *Visual Servoing*, ch. Visual Servo Control of Robots using Kalman Filter Estimates of Pose Relative to Work-pieces. World Scientific, hashimoto, k. (ed.) ed., 1994.

[Wojcik, 1992] Z. Wojcik, "Ground operation of the mobile servicing system on space station freedom," in *Cooperative Intelligent Robotics in Space III* (J. D. Erickson, ed.), (Boston, MA), pp. 75–90, Nov. 16-18 1992. Vol. 1829.

[Yamakita and others, 1992] M. Yamakita *et al.*, "Tele-virtual reality of dynamic mechanical model," in *IEEE/RSJ Int. on Intel. Robots & Systems*, pp. 1103–1110, July 1992.

[Yokokohji *et al.*, 1994] Y. Yokokohji, N. Hosotani, and T. Yoshikawa, "Analysis of maneuverability and stability of micro-teleoperation systems," in *IEEE Int. Conf. on Robotics & Automation*, vol. 1, pp. 237–243, May 1994.

[Yoshimi and Allen, 1994] B. Yoshimi and P. Allen, "Active, uncalibrated visual servoing," in *IEEE Int. Conf. on Robotics & Automation*, vol. 1, pp. 156–161, May 1994.

[Zeltzer, 1992] D. Zeltzer, "Autonomy, interaction and presence," *Presence: Teleoperators & Virtual Environments*, vol. 1, pp. 127–132, Winter 1992.

[Zhu *et al.*, 1992] W. Zhu, H. Chen, and Z. Zhang, "A variable structure robot control algorithm with an observer," *IEEE Trans. on Robotics & Automation*, vol. 8, pp. 14–35, August 1992.

Appendix A Real-time Space Robotics Simulation

A.1 Hardware Configuration

In the development of complex robotic systems[1], simulation plays an important role in providing an environment to evaluate behaviour, capabilities and operation effectiveness of the system under design. Simulations are extensively used for verification of flight systems and will play an indispensable role in aiding mission planning and crew training. The Canadian contribution to the International Space Station program is the Mobile Servicing System (MSS). The MSS main components are two robotic devices: a large, symmetric, relocatable manipulator arm with seven degrees of freedom; and a smaller, dexterous robotic system consisting an articulated trunk and two arms, each having seven degrees of freedom. The Manipulator Development and Simulation Facility (MDSF), developed for the MSS program, is a facility capable of simulating arbitrarily configured robotic manipulators [Carr and others, 1990]. MDSF can be used both for engineering development and for crew training activities [Cyril *et al.*, 1992]. The engineering segment of the facility (i.e. hardware) provides generic capability of modelling and simulating robotic manipulator systems. The requirement to provide a generic simulation facility is driven by the long MSS program life, the modification of requirements throughout the life cycle of the project, and the lack of a unique robotic system configuration. This generic capability is reflected, among other features, in a software reconfigurable control and display station, the user interface which allows a building block approach to robotic system synthesis, the utilities, and the equations of motion.

MSS is a multi-purpose, versatile complex equipped with manipulators, advanced control systems and *human in the loop* capability as shown in Figure A.1. The MSS will support construction, operation and maintenance aspects of the Space Station and its attached pay-

[1]This Appendix contains selected abstracts from [CAELIB., 1995s] plus personal training notes during internal CAELIB course at CAE Electronics.

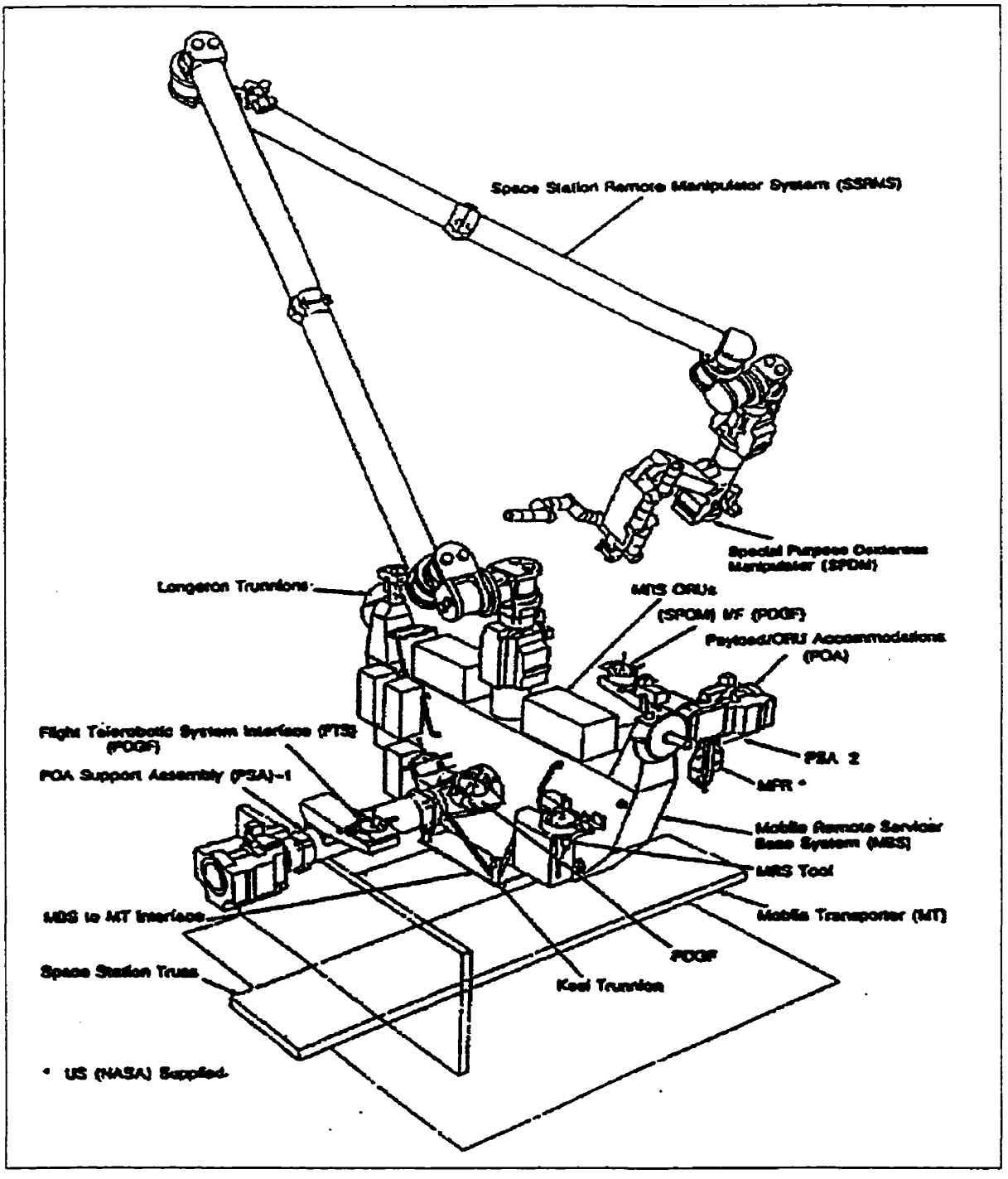


Figure A.1: Mobile Servicing System (MSS)

loads. The main components of MSS are two robotic devices: the Space Station Remote Manipulator System (SSRMS) and the Special Purpose Dexterous Manipulator (SPDM). The SSRMS is a large (17.6m long), self-relocatable manipulator arm with 7 dof used for large scale manipulation of payloads and astronauts. The SSRMS is symmetric relative to

its 4-th (elbow) joint; this allows it to be attached and operated from either end. The SPDM is a smaller, dexterous robotic system consisting of an articulated trunk (body) and a pair of 2m long arms, each having 7 dof. The SPDM is used for small and precise movements, in general tasks similar to those performed by a space suited extra-vehicular astronaut. Simulations will be extensively used during design, development and operation of the MSS. To carry out these simulations the MDSF is being developed at CAE. It encompasses within one facility two kinds of simulator, one for engineering development and the other for training and operational support. Although developed for Space Station applications, MDSF architecture is sufficiently flexible to allow the simulation of other space, terrestrial or underwater robots.

The Real-Time Simulation (RTSIM) CSCI simulates in real-time the dynamics of articulated multi-body robotic systems. It allows to call simulation modules in a specific order, with specific iteration rates, and provides data gathering and snapshot functions. The RTSIM CSCI includes a number of CSCs, all designed to run in real time, synchronised with each other and with external hardware or software components. RTSIM includes the Real-Time Dynamics Simulation (RT-DS) CSC and various executive software components such as the Real-Time (RT) Dispatcher CSC. The RTSIM CSCI decomposition overview is shown in Figure A.2. Communication between the Simulation Control Function (SCF), RT-DS, the visualization system, the user-supplied control system, the background monitoring utilities and any other process that runs concurrently with, and which communicates with the simulation, is achieved through the *Common Data Base* (CDB) as discussed in Chapter 7. The Data Gathering function of RTSIM allows the user to gather data from the CDB at regular intervals. This data is saved to disk, in order to be analyzed later by the Data Reduction and Analysis Function (DRAF). The different software components will reside in the Host and Auxiliary Processing Systems (HPS and APS) or in the Control, Development and Animation Work-centre (CDAW).

A.2 Robotics Real-time Simulations Environment

The *RT-DS CSC* simulates the dynamics of the robotic or teleoperated system defined in the simulation definition. The purpose of *RT Dispatcher CSC* is to call the simulation rou-

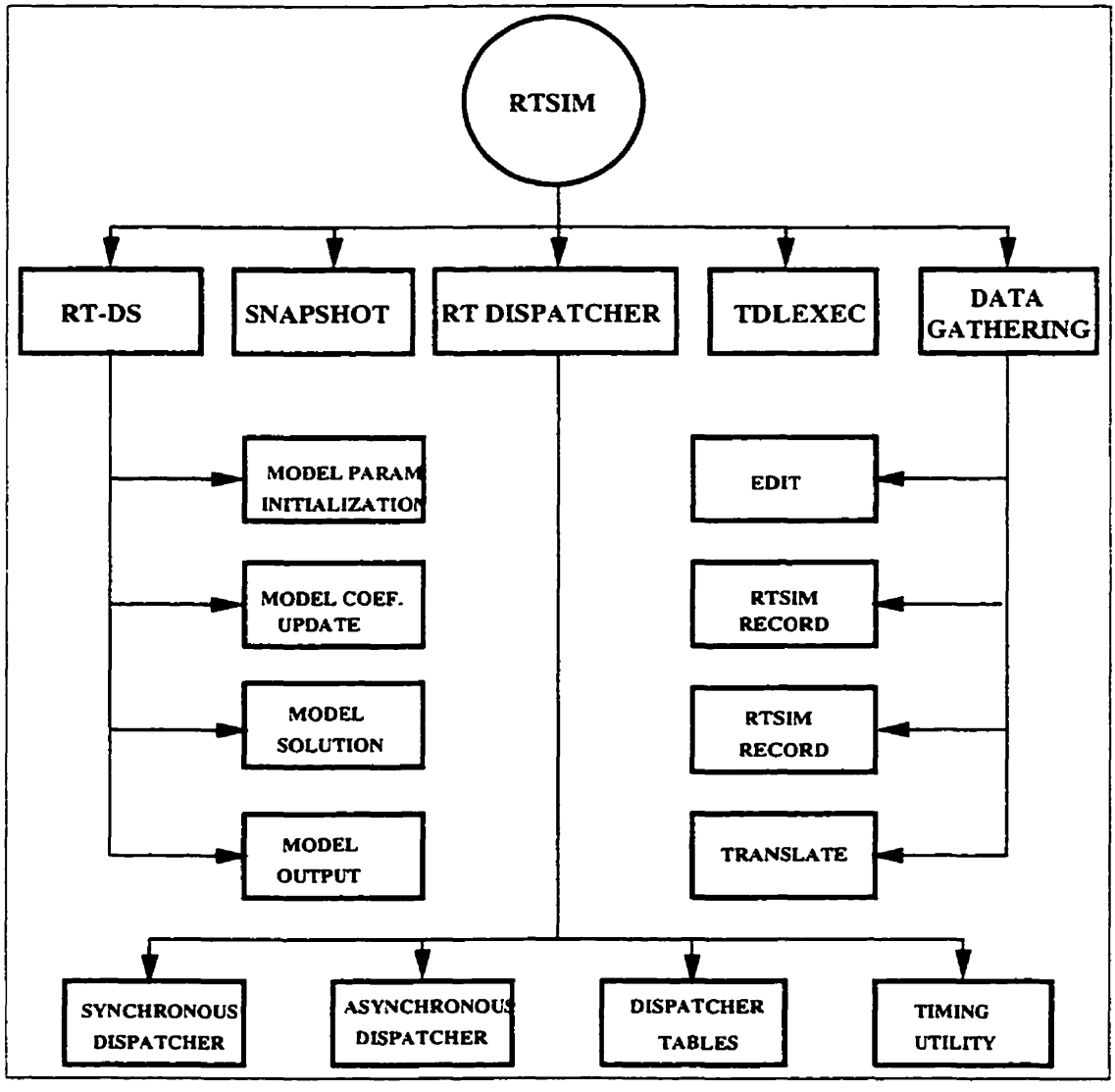


Figure A.2: RTSIM CSCI decomposition

tines in a specific order, within processing bands which execute at specified iteration rates. The *Synchronous Dispatcher* is activated by a clock signal and the simulation process that it controls is non-interruptible. The *Asynchronous Dispatcher* controls the execution of modules whose immediate scheduling is not critical to the operation of the RT simulator. The process can be interrupted because it runs with lower priority than the synchronous process. The *Dispatcher Tables* allow the user to specify how the simulation modules are to be dispatched within each process. The *Timing Utility* allows the monitoring of the CPU time taken by modules, bands and legs of each process. The *Data Gathering CSC* allows the user to gather data generated during the simulation run for later analysis (by DRAF) and/or for

post simulation visualization (i.e. edit, record, translate, etc.). The *Snapshot CSC* allows the user to record the state of a simulation at any given time during its run and to restore the simulation to the recorded state, from which it may be restarted. The *Task Definition Language Executive (TDLEXEC)* processes the telecommands passed to it.

All communication between the CSC simulation modules is through the CDB. The input, output and the state variables of the modules comprising the RT-DS CSC are located in the CDB. When the Data Gathering CSC is recording simulation data, it accesses the variables located in the CDB without interfering with the normal execution of the dynamic simulation modules (i.e. the recording operation is transparent to the simulation modules). The CDB is organized in such a way that the variables are separated in two groups:

- **The restorable area.** This contains variables that describe the state of the simulation. Once restored, the simulation can be restarted from the point where the recording was made using the snapshot.
- **The non-restorable area.** This contains variables that are used by non-simulation modules, such as the RT Dispatcher, Snapshot and Data Gathering.

Communication through CDB also helps the interlocking mechanism between the operator input and the CSC TDLEXEC. In either case, the same CDB variables are used as input to the simulation module; the interlocking mechanism is transparent to the simulation modules. The same data is shared through the CDB by all sub-CSCs, which are executed at different rates on the APS within the synchronous and asynchronous dispatcher processes. The RT Dispatcher CSC is responsible for controlling the execution of all other RTSIM CSCs. RTSIM is implemented using three simulation processes, i.e. (a) **sp0:- main synchronous process;** (b) **ap0:- main asynchronous process;** and (c) **ap1:- low priority asynchronous process.** For more details on process scheduling and management, are presented in Chapter 7.

A.3 Rationale for an Engineering Simulator

Engineering simulator facilitates analysis and design, system level software tests and performance verification. Simulations may be either real-time or non-real time; the former allowing human-in-the-loop interaction for the development and fine tuning of control systems and human interfaces, and the later for complete analysis of modes of vibration beyond the limits of human perception. The MDSF consists of a computer complex and CDAW as shown in Figure A.3. The computer complex consists of a VAX cluster and an auxiliary processing system. The latter, a Convex C220 computer, is used exclusively for the execution of real-time simulation processes. Non-real time simulations may be executed anywhere in the cluster. The CDAW is a reconfigurable workcenter that processes the operator's inputs to the simulated robotic system, and displays, in real-time, the response and dynamic evolution of the system. It consists of hand controllers, a Controls and Displays Workstation (an IRIS 4D/25), a Real-Time Engineering Visualization Workstation (IRIS 4D/120GTX), and a terminal connected to the host processing system. The 6 dof can *lock out* dof to form rotational or translational controllers. The CRT terminal is used for simulation control, task definition, and changing and monitoring parameters.

The requirement for a generic capability has affected every aspect of the MDSF design; it is reflected, among other features, in a building block approach to robotic system synthesis and the definition of simulator configurations, in the user defined robotic control systems, software reconfigurable control and display station, and in the generic mathematical model of kinematics and dynamics of articulated multi-body structures which constitutes the nucleus of the MDSF. There are no limitations on the shape of the bodies, nor on the relative orientation between contiguous articulations, nor on the topological configuration. MDSF algorithms permit general relative motion with up to three translational and three rotational dof. Articulations with 6 dof permit the introduction of separate bodies or structures, with or without attached manipulator systems, co-existing in proximate orbits. This allows the presentation of a payload prior to being captured or after release from one of the manipulators or the approach of an orbiter to the SS. With MDSF, the parameters and the algorithms modeling the associated control systems are user defined. In MDSF the task of defining the

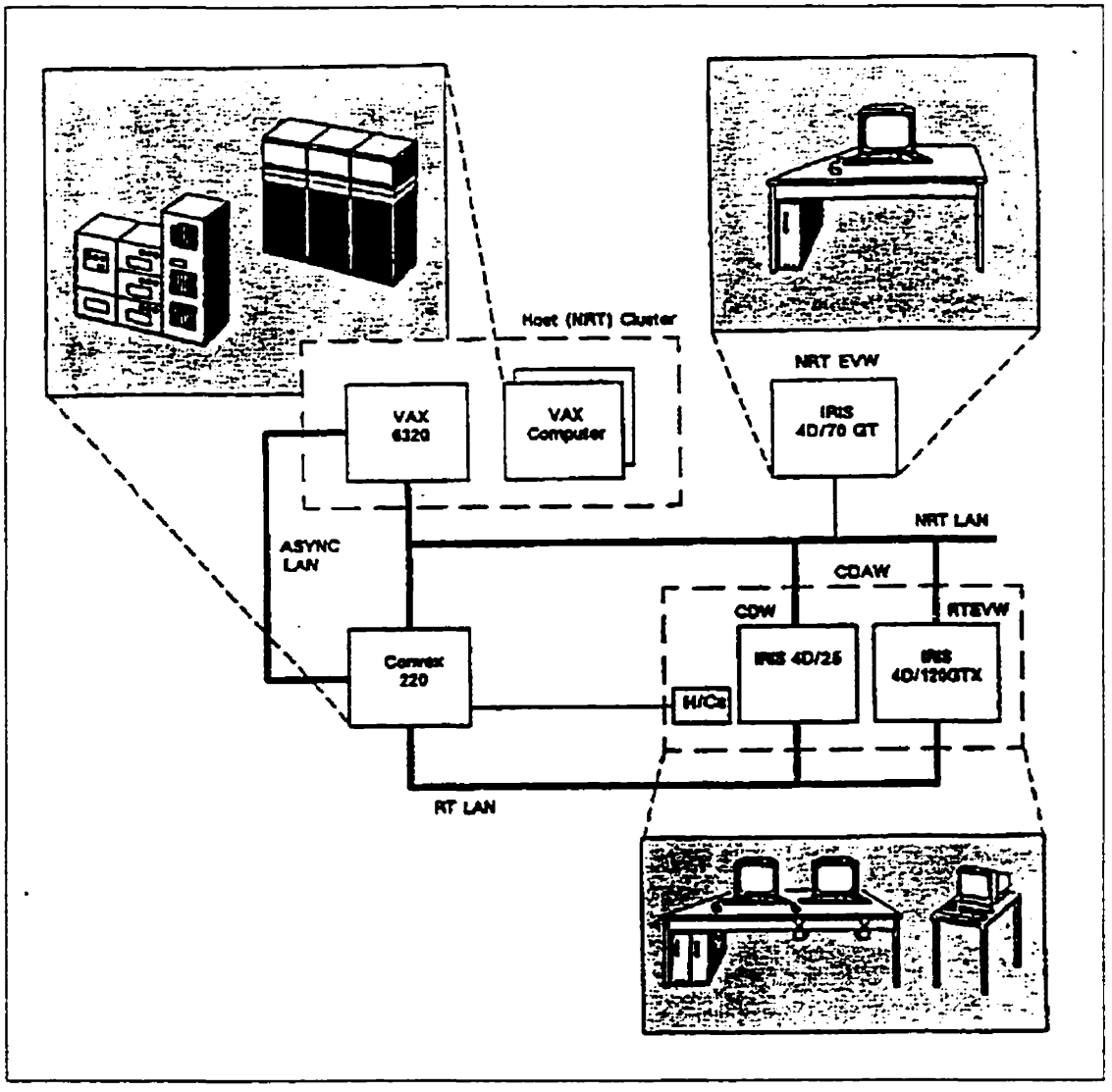


Figure A.3: MDSF - an engineering simulator

system is simplified, i.e. the user can choose to define the system at the level of his expertise through a step-by-step, menu-driven, interactive process. At the disposal of the user is a database of bodies, articulations and systems. The user may wish to create, modify or delete new bodies, articulations or systems.

Simulation definition is a function, built upon CAE's configuration management utility, it allows the user to define, configure and manage simulators of specific robotic manipulator systems. The object of the process is a simulator configuration which contains all the information needed to define, configure, build, manage and control a simulator. The idea of MDSF as a facility that is reconfigured into or supports different simulators is central to

the MDSF architecture. A configuration consists of software elements and relationships between them[2]. The configuration identifier and its elements are stored in a database which has a dependency graph structure where each node represents an element and each link represents a relationship between elements. A relationship describes *build*, *configure* or *load* dependencies. A simulation contains a simulator configuration and may contain robotic simulation task definition, initial conditions, on-line monitoring and data gathering lists and post-processing telecommand files (all stored in a common shared memory as CDB data elements or labels).

Other utilities included in the simulation control function include the Page Monitoring System (PMS) and the Computerized Test System (CTS). PMS allows users to monitor one or several pages of numerical outputs of the simulation processes, and to modify the values of simulation variables. CTS is a utility that supports the integration, testing and debugging of simulation software in either an off-line stand-alone mode, or in on-line mode which interacts with a loaded, running simulation. Input variables can be dynamically ramped to follow a given time history, while outputs may be plotted or monitored to check against expected values. CTS allows users to examine and modify any simulation variable or block of variables, and makes possible full automation of setups and checkout tests. Real-time engineering visualization allows the animation of the dynamics of the system being simulated. Visualization is achieved by a 3D renderer developed for RTEVW. The ability to update the scene rapidly is enhanced by a 3D graphics editor developed for MDSF application. A graphics display function is used for the software emulation of the control and display panels of teleoperated robotic systems. It is based on CAE's TIGERS[3] package. This function allows a user to design a control and display panel which may include levers, sliders, switches, analog gauges, digital displays, warning lights, etc. A graphics editor facilitates this operation. This panel is then linked to the CDB of simulation parameters, and driven in real-time. Both a pointing device and a touch screen are used for operator input.

[2]Refer to Chapter 7 for Software Configuration.

[3]TIGERS is an acronym for *The Integrated Graphics Environment for Real-time Systems.*

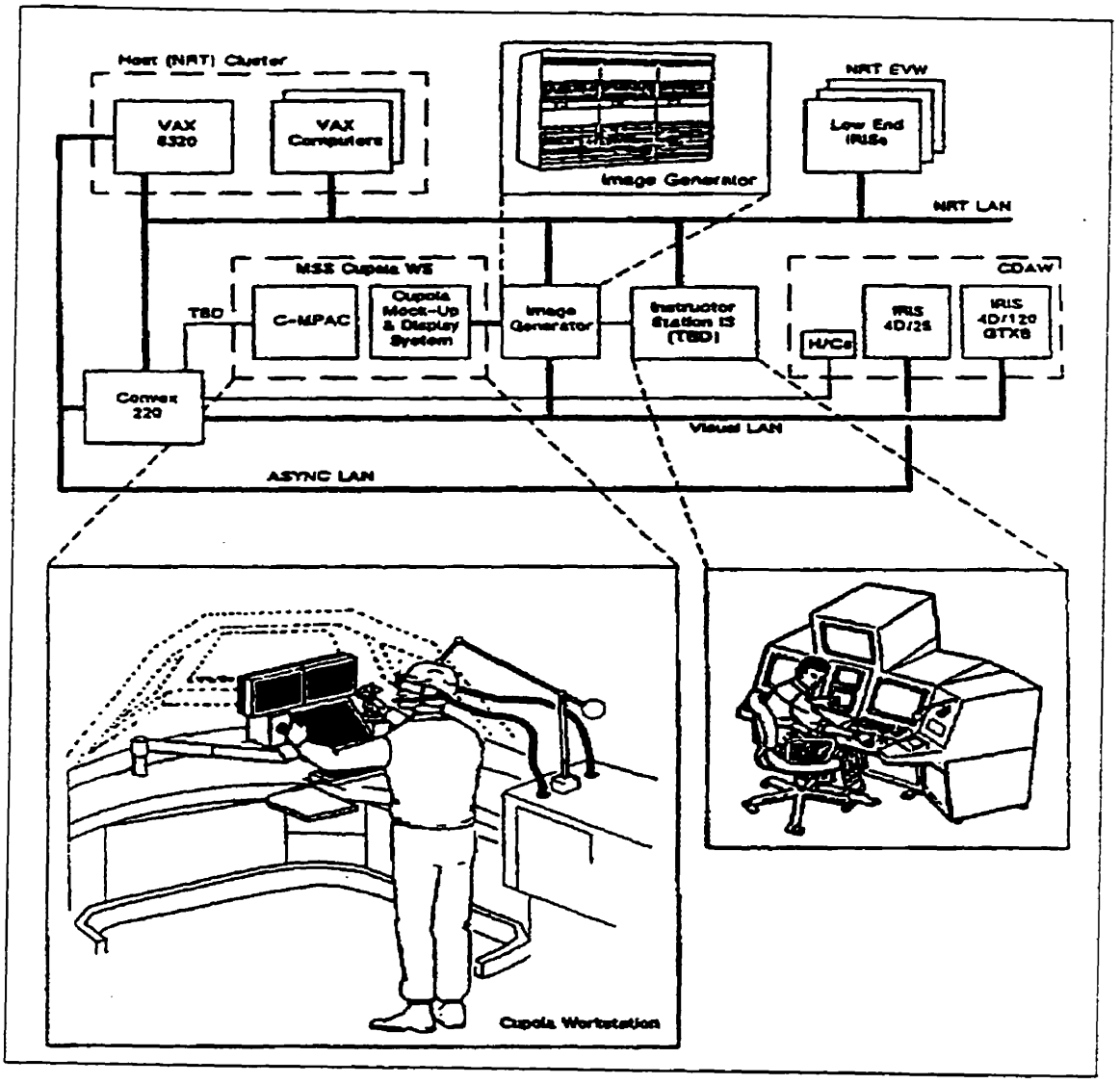


Figure A.4: MDSF - training simulator

A.4 Robotics Training Simulator's Benchmark

Under the Canadian Space Station Program there is a mandate to produce a training simulator for the MSS. The training simulator will be built as an enhancement to the MDSF real-time engineering capability. The enhancement will be achieved by adding upgraded dynamics models, specific implementations of the MSS control systems and HMIs, a crew control station, an instructor facility, and an image generation and display system to the existing MDSF infrastructure. A conceptual diagram of the training simulation facility is shown in Figure A.4. Typical tasks to be carried out by MSS are: Payload/ORU handling; manipulator handling tasks; space station assembly operations and orbiter berthing and de-

berthing by SSRMS. A vast array of systems will be monitored and controlled by crew members in the SS. To minimise the crew load and training required to operate any given system a common set of standards and guidelines are defined for the HCI. The HCI resides on common workstations known as the Multi-Purpose Applications Console (MPAC) allowing crew members to interact with systems on-board. The flight hardware of the MPAC Cupola (MPAC-C) consists of displays, a keyboard, a trackball, hand controllers, and a processor. During the human-in-the-loop operations the hand controller commands can either be applied in Cartesian space to a point of resolution (e.g. the end-effector) or as telecommands to individual joints. The crew training workstation will emulate the visual and hardware interface of the flight system. The workstation will consist of a mockup structure, flat panel CRTs, interactive control devices, and an optical display system for simulated out-the-window scenes. To fulfill the flight MPAC-C functionality the HCI software is simulated as part of the MDSF. The Instructor Station (IS) must allow the instructor to choose the training scenario by: specifying the orbital structures to be included; specifying the orbital and environmental parameters; activating and deactivating malfunctions; and participating in hand-off operations (i.e. coordinated movement between two manipulators, one controlled by the student, the other by the instructor). The hardware necessary to complete the IS is closed circuit TV and two-way communication device, and a visual repeater for the MPAC-C displays.

For visual imagery, station assembly and maintenance tasks will require very high scene content, particularly in the details represented on each object. The display system will require high resolution to match the scene content. Because of the space environment, brightness and contrast requirements will be high as well. Thus, a number of Closed Circuit TV (CCTV) cameras are to be mounted on the SSRMS, SPDM, and SS to provide views to the crew at the workstations. An arrangement to provide a crew with a wide Field of View (FOV) [Barrette, 1992] is required. The Fibre Optic Helmet Mounted Display (FOHMD) is a compact display system which provides the trainee with a bright, high-resolution, full-colour display over his entire FOV. The characteristics of the display presented to the trainee are designed to mimic the performance of the human eye (i.e. for stereo-scopic). At CAE, there more tools and software utilities [CAELIB., 1995s] that are very well suited for real-

time robotics simulations either for operations and/or training. A good example is that of MOTS as discussed in Chapter 8.

Appendix B Program Listings for the Control Scheme

The following THREE pages gives a MATLAB source listing for the predictor algorithm and a double-loop observer-based controller algorithm as discussed in Chapter 6.

```
clear all;
clg;
format long;

N=600;
t=[0:1:N]';

%    ******* Translation Motion: ramp-like (fixed orientation) ******
%    ******* Initial point is (10,10) *****

xc=10+0.2*t;
yc=10+0.8*t;

%    ******* Rotation Motion: circular-like (fixed position) ******
%    ******* Initial point is (10,10) and frequency of 0.005Hz *****

%f2=0.005;
%xc=10+30*sin(f2*pi*t);
%yc=10+30*cos(f2*pi*t);

%    ******* Free Motion: square-like (rotations and translations) ******
%    ******* Initial point is (10,10) and frequency of 0.5Hz *****

%f3=0.5;
%xc=10+2*sin(f3*pi*t);
%yc=10+2*cos(f3*pi*t);

%    ******* Contact Motion: Wave-like motion ******
%    ******* Initial point is (10,10) and frequency of 0.02Hz *****

%f4=0.02;
%xc=10+2*cos(f4*pi*t);
%yc=10+0.002*t.^2;

zc=5+0*t; %  *** same Z-motion for all cases with z-value = 5 for all t ****

sx1=xc(1)*xc(2);
sx2=xc(2)*yc(1);
sx3=0;
sx4=xc(2);

sy1=yc(2)*xc(1);
sy2=yc(2)*yc(1);
sy3=0;
sy4=yc(2);

x11=xc(1)*xc(1);
x12=yc(1)*xc(1);
x13=0;
x14=xc(1);

x21=xc(1)*yc(1);
x22=yc(1)*yc(1);
x23=0;
x24=yc(1);

x41=xc(1);
x42=yc(1);
x43=0;
x44=1;

Xa=1000;
Ya=1000;

for i=2:N/2,

```



```
dxy=sqrt((xc(i)-Xa)^2 + (yc(i)-Ya)^2);

if abs(dxy)<=0.1,  % *** maximum allowable error value
break;
end;

sx1 = sx1 + xc(i)*xc(i-1);
sx2 = sx2 + xc(i)*yc(i-1);
sx4 = sx4 + xc(i);

sy1 = sy1 + yc(i)*xc(i-1);
sy2 = sy2 + yc(i)*yc(i-1);
sy4 = sy4 + yc(i);

x11 = x11 + xc(i-1)^2;
x12 = x12 + yc(i-1)*xc(i-1);
x14 = x14 + xc(i-1);

x21 = x21 + xc(i-1)*yc(i-1);
x22 = x22 + yc(i-1)^2;
x24 = x24 + yc(i-1);

x41 = x41 + xc(i-1);
x42 = x42 + yc(i-1);
x44 = 1;

C=[x11 x12 x14; x21 x22 x24; x41 x42 x44];
X=[sx1 sx2 sx4]';
Y=[sy1 sy2 sy4]';
r=rank(C);

if r==3,
B=inv(C);
A1=B*X; A2=B*Y;
Xa=A1(1)*xc(i) + A1(2)*yc(i) + A1(3);
Ya=A2(1)*xc(i) + A2(2)*yc(i) + A2(3);

end;

end;

if i==N/4,

msg='Convergence Failure. The motion is undeterministic';
disp(msg);
end;

% *** If the above condition is verified, the (i+1)th position
% *** is sent to the control system

Xa1=[Xa]; Ya1=[Ya];
Xaa=A1(1)*Xa+A1(2)*Ya+A1(3);
Yaa=A2(1)*Xa+A2(2)*Ya+A2(3);
Xa2=[Xaa]; Ya2=[Yaa];

for j=i:N,

Xa1=[Xa1;Xa];
Ya1=[Ya1;Ya];
Xa2=[Xa2;Xaa];
Ya2=[Ya2;Yaa];

for k=j-i+3:j,

sx1 = sx1 + xc(k)*xc(k-1);
sx2 = sx2 + xc(k)*yc(k-1);
```

```
133   sx4 = sx4 + xc(k);
134
135   sy1 = sy1 + yc(k)*xc(k-1);
136   sy2 = sy2 + yc(k)*yc(k-1);
137   sy4 = sy4 + yc(k);
138
139   x11 = x11 + xc(k-1)^2;
140   x12 = x12 + yc(k-1)*xc(k-1);
141   x14 = x14 + xc(k-1);
142
143   x21 = x21 + xc(k-1)*yc(k-1);
144   x22 = x22 + yc(k-1)^2;
145   x24 = x24 + yc(k-1);
146
147   x41 = x41 + xc(k-1);
148   x42 = x42 + yc(k-1);
149   x44 = 1;
150
151   end;
152
153   C1=[x11 x12 x14; x21 x22 x24; x41 x42 x44];
154
155   % *** This is singular and the entries in 2nd column and 2nd row
156   % *** are linearly dependent
157
158   X=[sx1 sx2 sx4]';
159   Y=[sy1 sy2 sy4]';
160   B1=inv(C1);
161   A1=B1*X; A2=B1*Y;
162   Xa=A1(1)*xc(j) + A1(2)*yc(j) + A1(3);
163   Ya=A2(1)*xc(j) + A2(2)*yc(j) + A2(3);
164   Xaa=A1(1)*Xa+A1(2)*Ya+A1(3);
165   Yaa=A2(1)*Xa+A2(2)*Ya+A2(3);
166
167   end;
168
169   %   ******* Print to a postcript file one after the other ******
170
171   print line0.ps
172   %print circular3.ps;
173   %print square3.ps;
174   %print wave3.ps;
175
176   clg;
177
178   subplot(221); % *** for actual and predicted motions comparison
179
180   plot(xc(i+1:30),yc(i+1:30),'r',Xa2(1:30-i),Ya2(1:30-i),'xb'); grid
181   title('(a) Actual and predicted motions');
182   xlabel('X-value');
183   ylabel('Y-value');
184   gtext('Predicted (x-mark)');
185   gtext('Actual (solid)');
186   pause(5);
187
188   K=0.6;
189   Kd=1.5; % *** We have PD feedback;
190   T=1;
191   Td=T*Kd;
192   Ke=1/(2*T);
193   k1=T^2*K/2;
194   k2=-2+T*K*Td+k1;
195   k3=1-T*K*Td+k1;
196
197   num=[(T*Td+T^2/2) (-T*Td+T^2/2)];
198   den=[1,-2,1];
```



```
199
200   subplot(222); % *** root locus of closed-loop system
201
202   rlocus(num,den);
203   title('(b) Root Locus');
204   [k,poles]=rlocfind(num,den);
205   disp(k);
206   gtext('Pick k=0.6 (inside unit circle)');
207   pause(5);
208
209   %ned=[k1 k1];
210   %ded=[1 k2 k3];
211   %i=0:19; t=i*T;y=dstep(ned,ded,20);stairs(t,y);grid;
212   %pause;
213   %td=[0:60]';
214   %Xc=2+4*td; Yc=5+2*td.^1.2; Zc=-10+0*td;
215   %Xc=5+5*sin(0.1*pi*t); Yc=5+5*cos(0.1*pi*t);
216   %Zc=2+0*t;
217
218   Ux=[0]'; Uy=[0]'; Uz=[0]';
219   X1=[5]'; X2=[0]'; X22=[0]'; Ex=[0]';
220   Y1=[5]'; Y2=[0]'; Y22=[0]'; Ey=[0]';
221   Z1=[10]'; Z2=[0]'; Z22=[0]'; Ez=[0]';
222
223   for m=2:N-1-2,
224
225   X11=X1(m-1)+T*X2(m-1)+((T^2)/2)*Ux(m-1);
226
227   Y11=Y1(m-1)+T*Y2(m-1)+((T^2)/2)*Uy(m-1),
228
229   Z11=Z1(m-1)+T*Z2(m-1)+((T^2)/2)*Uz(m-1),
230
231   X21=X2(m-1)+T*Ux(m-1)+Ke*(X11-X1(m-1)-((T^2)/2)*Ux(m-1)-T*X2(m-1));
232
233   Y21=Y2(m-1)+T*Uy(m-1)+Ke*(Y11-Y1(m-1)-((T^2)/2)*Uy(m-1)-T*Y2(m-1));
234
235   Z21=Z2(m-1)+T*Uz(m-1)+Ke*(Z11-Z1(m-1)-((T^2)/2)*Uz(m-1)-T*Z2(m-1));
236
237
238   %Ex1=X21-X221;
239   %Ey1=Y21-Y221;
240   %Ez1=Z21-Z221;
241
242   Ux1=K*(Xa2(m)-X11-Td*X21);
243
244   Uy1=K*(Ya2(m)-Y11-Td*Y21);
245
246   Uz1=K*(zc(m+2)-Z11-Td*Z21);
247
248   X1=[X1;X11];
249   X2=[X2;X21];
250   Ux=[Ux;Ux1];
251   %X22=[X22;X221]; Ex=[Ex;Ex1];
252
253   Y1=[Y1;Y11];
254   Y2=[Y2;Y21];
255   Uy=[Uy;Uy1];
256   %Y22=[Y22;Y221]; Ey=[Ey;Ey1];
257
258   Z1=[Z1;Z11];
259   Z2=[Z2;Z21];
260   Uz=[Uz;Uz1];
261   %Z22=[Z22;Z221]; Ez=[Ez;Ez1];
262
263   end;
264
```

```
265  subplot(223); % *** Motions of gripper and object along Z-axis
266
267  plot(t(i-1:20),zc(i-1:20),'xg', t(i-1:20), Z1(1:20-i+2),'r'); grid
268  title('(c) Motions along Z-axis');
269  xlabel('Time (s)');
270  ylabel('Z-value');
271  gtext('Object (x-mark)');
272  gtext('Gripper (solid)');
273  pause(5);
274
275  subplot(224); % *** Motions of gripper and object along X-Y plane
276
277  plot(xc(i-1:20),yc(i-1:20),'xb',X1(1:20-i+2),Y1(1:20-i+2),'g'); grid
278  title('(d) Motions in X-Y plane');
279  xlabel('X-value');
280  ylabel('Y-value');
281  gtext('Object (x-mark)');
282  gtext('Gripper (solid)');
283  pause(5);
284
285  %   ******* Print to a postcript file one after the other ******
286
287  print line1.ps;
288  %print circular1.ps;
289  %print square1.ps;
290  %print wave1.ps;
291
292  clg;
293
294  Ex=X1(1:50-i+2)-xc(i-1:50);
295
296  Ey=Y1(1:50-i+2)-yc(i-1:50);
297
298  Ez=Z1(1:50-i+2)-zc(i-1:50);
299
300
301  subplot(221); % *** Position error between gripper and object
302
303  plot(t(i-1:20),Ex(1:20-i+2),'r'); grid
304  title('(a) Position Error along X-axis');
305  ylabel('Position Error');
306  xlabel('Time (s)');
307  pause(5);
308
309  subplot(222); % *** Position error between gripper and object
310
311  plot(t(i-1:20),Ey(1:20-i+2),'b'); grid
312  title('(b) Position Error along Y-axis');
313  ylabel('Position Error');
314  xlabel('Time (s)');
315  pause(5);
316
317  subplot(223); % *** Position error between gripper and object
318
319  plot(t(i-1:20),Ez(1:20-i+2),'g'); grid
320  title('(c) Position Error along Z-axis');
321  ylabel('Position Error');
322  xlabel('Time (s)');
323  pause(5);
324
325  %   ******* Print to a postcript file one after the other ******
326
327  print line2.ps;
328  %print circular2.ps;
329  %print square2.ps;
330  %print wave2.ps;
```



```
331
332  clg;
333
334  subplot(221); % *** Input forces
335
336  stairs(t(i-1:15-2+i),Ux(1:15)); grid
337  title('(a) Input Force along X-axis');
338  xlabel('Time (s)');
339  ylabel('Magnitude of force');
340  pause(5);
341
342  subplot(222); % *** Input forces
343
344  stairs(t(i-1:15-2+i),Uy(1:15)); grid
345  title('(b) Input Force along Y-axis');
346  xlabel('Time (s)');
347  ylabel('Magnitude of force');
348  pause(5);
349
350  subplot(223); % *** Input forces
351
352  stairs(t(i-1:15-2+i),Uz(1:15)); grid
353  title('(c) Input Force along Z-axis');
354  xlabel('Time (s)');
355  ylabel('Magnitude of force');
356  pause(5);
357
358  %   ******* Print to a postcript file one after the other ******
359
360  print line3.ps;
361  %print circular3.ps;
362  %print square3.ps;
363  %print wave3.ps;
364
365  end;
366
```

Appendix C List of Acronyms

Acronym	Description
ACA	Arm Control Algorithm
ACDB	Arm Control Data Bus
ACS	AVU Control Software
ACU	Arm Computer Unit
A/D	Analog/Digital
AHS	ACU Host Software
AIFF	Audio Interface File Format
APPC	Arm Pitch Plane Change
APS	Auxiliary Processing System
ASCII	American Standard Code for Information Interchange
ASTM	AVF Supported Tracking Mode
ATF	Automatic Trajectory File
AUV	Autonomous Underwater Vehicle
AVF	Artificial Vision Function
AVTEL	Audio/Video Teleconferencing
AVU	Artificial Vision Unit
BC	Bus Controller
BCDB	Backup Control Data Bus
BDU	Backup Drive Unit
BDUCS	BDU Control Software
BDUS	Backup Drive Unit Software
BGF	Base Grapple Fixture
BIT	Built-In Test
BITE	Built-In Test Equipment

Acronym	Description
C1CS	Class 1 Crew Station
C2CS	Class 2 Crew Station
C&C	Command & Control
C&DH	Command & Data Handling
C&T	Communications & Tracking
C&W	Caution & Warning
CAE	CAE Electronics Limited
CAI	Computer Assisted Interface
CAS	Common Attachment System
CCC	Control Centre Complex
CCTV	Closed Circuit Television
CDAW	Control, Development and Animation Workcentre
CDB	Common Data Base
CDBP	Common Data Base Processor
CEU	Control Electronics Unit
CI	Configuration Item
CLA	Camera/Light Assembly
CLPA	Camera/Light/Pan-tilt unit Assembly
CMTF	Canadian MSS Training Facility
COTS	Commercial Off-The-Shelf
CPAF	Collision, Protection, and Avoidance Function
CPU	Central Processing Unit
CRPCM	Canadian Remote Power Control Modules
CS	Crew Station
CSA	Canadian Space Agency
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSR	Camera Status Reader

Acronym	Description
CTS	Computerized Test System
D&C	Display & Control
DG	Data Gathering
DFC	Digital Force Control
DFD	Data Flow Diagram
DFR	Dead Face Relay
DMC	Data Management Controller
DMCS	Dextrous Manipulator Control Software
DOF	Degree Of Freedom
DRAF	Data Reduction and Analysis Function
EDP	Embedded Data Processor
EEOCS	End Effector Operating Coordinate System
EPS	Electrical Power System
EVA	Extra Vehicular Activity
FIFO	First In First Out
FOV	Field Of View
FMA	Force Moment Accomodation
FMS	Force Moment Sensor
FTS	Flight Telerobotic System
GFE	Government Furnished Equipment
GLETS	Global-Local Environment Telerobotics Simulator
GN&C	Guidance Navigation & Control
GS	Ground Segment
GSA	Graphical Simulation and Animation
GUI	Graphics User Interface
H/C	Hand Controller (or HC)
HCA	Hand Controller Assembly
HCI	Human Computer Interface

Acronym	Description
HMI	Human Machine Interface
HPS	Host Processing System
HWCI	HardWare Configuration Item
IG	Image Generator
I/O	Input/Output
IOC	Input/Output Controller
IOP	Increment Operations Plan
IOS	Instructor Operator Station
IS	Instructor Station
ISR	Interrupt Service Routines
ISSA	International Space Station Alpha
ISSAP	ISSA Program
JCS	Joint Control Software
JDA	Joint Drive Assembly
JPA	Joint Pivotal Assembly
JEU	Joint Electronics Unit
JMM	Joint Motor Module
JPA	Joint Pivotal Assembly
JPC	Joint Power Conditioner
LCS	LEE Control Software
LEE	Latching End Effector
LEEMM	LEE Motor Modules
LEU	LEE Electronics Unit
LMM	Latch Motor Module
LPC	LEE Power Conditioner
MAM	Manual Augmented Mode
MBS	MRS Base System
MCCF	MSS Command and Control Facility

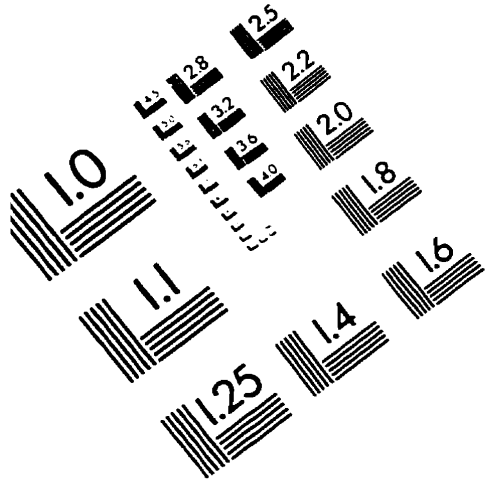
Acronym	Description
MCE	MSS Control Equipment
MCS	Manipulator Control System
MCS	MBS Control Software
MCU	MBS Control Unit
MDA	Motor Drive Amplifier
MDM	Multiplexer/Demultiplexer
MDSF	Manipulator Development and Simulation Facility
MHS	MBS Host Software
MLB	MSS Local data Bus
MMD	MSS Maintenance Depot
MO	Movable Object
MOC	MSS Operations Complex
MOCS	MSC Operating Coordinate System
MOM	MOther Main process
MOTS	MSS Operations and Training Simulator
MRCS	MSS Robotic Control Station
MRS	Mobile Remote Servicer
MSS	Mobile Servicing System
MT	Mobile Transporter
MTCL	MT Capture Latch
MTE	Mobile Transporter Element
MVS	MSS Video Subsystem
NASA	National Aeronautics Space Administration
N/A	Not Applicable
NRTSIM	Non-Real-Time Simulation Model
OCAS	Operator Commanded Auto Sequence
OCJM	Operator Commanded Joint Mode
OCPM	Operator Commanded POR mode

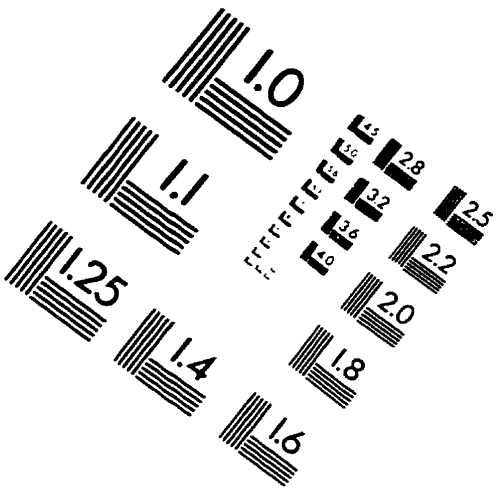
Acronym	Description
OCS	Operations Control Software
OMCS	Operation and Management Control Software
ORU	Orbital Replaceable Unit
OS	Operator Station
O/S	Operating System
OTCM	ORU/Tool Changeout Mechanism
OTS	Off-The-Shelf
OVP	Online Validation Program
PCR	Portable Control station for Robotics
PCS	Portable Computer System
PDGF	Power Data Grapple Fixture
PFM	Pulse Frequency Modulation
PFU	Program perFormance Utility
PHSM	Position Hold Submode
PJAM	Pre-stored Joint Autosequence Mode
PLB	PDGF Local data Bus
PMS	Page Monitoring System
POA	Payload/ORU Accomodations
POR	Point Of Resolution
POST	Power On Self Test
PPAM	Pre-stored POR Autosequence Mode
PSA	POA Support Assembly
PTU	Pan Tilt Unit
PVS	PFM Video Selector
PWM	Pulse Width Modulator
RDC	Resolver to Digital Converter
ROV	Remotely Operated Vehicle
RPC	Remote Procedure Calls

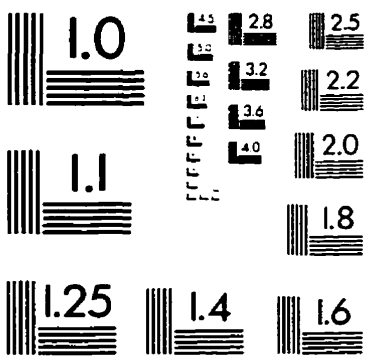
Acronym	Description
RRW	Remote Robotic Workcell
RT	Remote Terminal
RT-DS	Real-Time Dynamics Simulation
RTEVF	Real-Time Engineering Visualization Function
RTSIM	Real-Time Simulation Model
RWS	Robotics WorkStation
SACS	SSRMS Arm Control Software
SCF	Simulation Control Function
SCLC	Simulation Configuration and Load Control
SCU	Sync and Control Unit
SDW	Software Development Workstation
SED	SED Systems Inc.
SJRM	Single Joint Rate Mode
SIM	SImulation Models CSCI
SIMEX	SIMulation EXpert
SMC	Station Management Controller
SOFT-CD	Soft Controls and Displays
SOP	System Operation Procedure
SOSC	Space Operations Support Centre
SOW	Statement Of Work
SPA	Servo Power Amplifier
SPDM	Special Purpose Dexterous Manipulator
SRD	System Requirements Document
SRS	Software Requirements Specification
SRT	Safing Remote Terminal
SS	Space Segment
SSBA	Space Station Buffer Amplifier
SSD	System Specifications Document

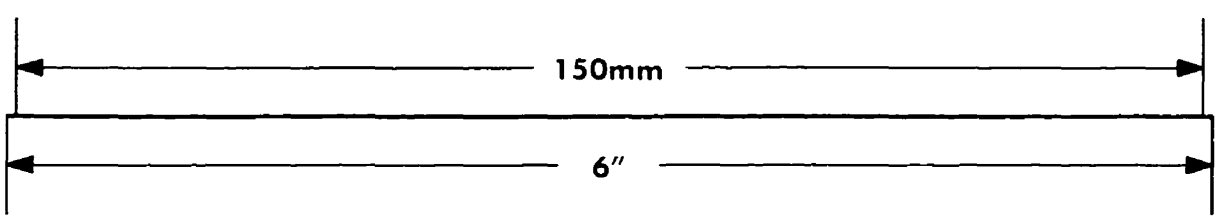
Acronym	Description
SSRMS	Space Station Remote Manipulator System
SSTF	Space Station Training Facility
SSS	Simulation Support System
TBD	To Be Determined
TCF	Thermal Control Function
TDL	Task Definition Language
TDLEXEC	TDL Executive
TIT	Teleoperator Interface and Training
TVC	TeleVision Camera
UIL	User Interface Language
ULC	Unpressurized Logistics Carrier
VAD	Visualization And Display
VDU	Video Distribution Unit
VGS	Video Graphics Software
VOTE	Virtual Operations Training Environment
VR	Virtual Reality
WHS	Workstation Host Software

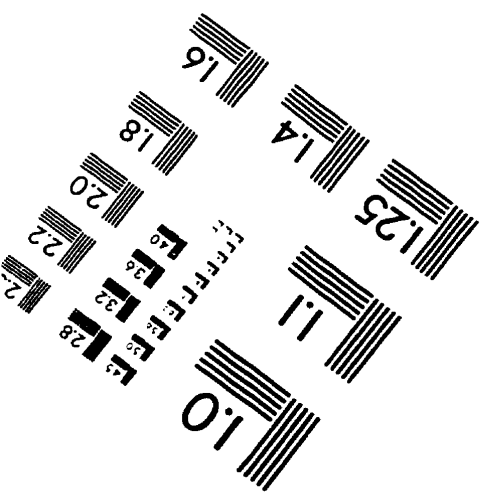
IMAGE EVALUATION
TEST TARGET (QA-3)

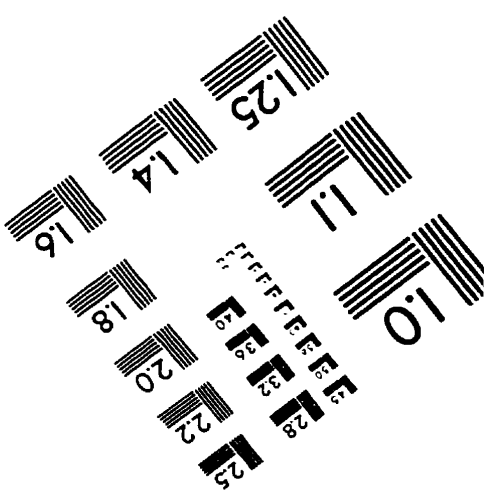












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989