University of Alberta

A Study of Teleoperation of Robotic System via the Internet

By

Yan Liu ©

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2001

0-612-60458-6

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A Study of Teleoperation of Robotic System via the Internet** submitted by Yan Liu in partial fulfillment of the requirements for the degree of **Master of Science.**

[illegible]

Dr. Max Q. -H. Meng, Supervisor

[illegible]

Dr. Qing Zhao

[illegible]

Dr. Yanping Lin

Date: Jan 15, 2001

Abstract

Internet-based teleoperation has been a hot research topic in robotics and automation field in recent years. In this paper, we describe a study on a teleoperated robot control system developed in the ART (Advanced Robotics and Teleoperation) Lab at the University of Alberta. When using the system, a remote operator can use a general-purpose computer with Internet connection and a World Wide Web (WWW) browser to remotely operate a robot over the Internet.

A control architecture that combines computer and robot is constructed. This system is divided into two primary parts. The client part is executed on the remote operator's computer and the server part resides on the server workstation in the ART Lab. The two parts are connected via the Internet. A graphical user interface on the remote computer screen enables the remote operator to send control commands to the robot. Communication coordination between the client and the server is developed using Java servlet. Since the real-time control through the Internet typically suffers from random time-delay and bandwidth constraints, many traditional control methods will have stability and obstacle avoidance related problems. To deal with the problems, event-based control methods for planning and control are applied in the designed Internet-based telerobotic systems, which can deal with the random time-delay constraints successfully by adopting a non-time based reference system in the control algorithm design.

Acknowledgement

I would like to thank my supervisor, Dr. Max Meng, for his guidance and support throughout this work and especially for teaching me the altitude, method and process of research.

I would like to thank Yong Gao in the Department of Computing Science at the University of Alberta for his advice and suggestions about the project. I would also like to thank all members of the Advanced Robotics Teleoperation (ART) Lab at the University of Alberta. Thanks to all those people who shared their time, thoughts with me. Finally, a special thank goes to my family.

Table of Contents

1. Introduction

1.1. Introduction 1

1.2. Thesis Objectives 2

1.3. Contributions of the Thesis 2

1.4. Organization of the Thesis 4

2. Teleoperation Systems

2.1. Introduction 7

2.1.1. Definition 7

2.1.2. Previous Work 8

2.2. Teleoperation through the Internet 9

2.2.1. Introduction 9

2.2.2. System Architecture 11

2.2.3. Interface and Client - Server Communications 12

2.2.4. Control 14

3. Interface and Communications

3.1. Introduction 16

3.2. Internet, World Wide Web (WWW) and HTML 19

3.3. Interface from HTML, VRML and Java 20

3.4. Java Servlet 24

3.4.1. Communication Goal 24

3.4.2. Java Servlet Implementation........ 25

3.5. Java Native Interface (JNI) Implementation 28

3.6. Java Servlet and JNI Work Together........ 34

3.7. Communication Using Sockets........ 39

3.8. User Authentication........ 39

3.9. Get Image from Server........ 41

3.9.1. Using Applet........ 41

3.9.2. Using Servlet........ 42

4. Control

4.1. Introduction........ 44

4.2. Control System........ 46

4.2.1. Controller and Servo Motor........ 46

4.2.2. Control Panel........ 46

4.3. Kinematics........ 48

4.3.1. Manipulator Kinematics........ 48

4.3.2. Manipulator Inverse Kinematics........ 49

4.4. Dynamics........ 50

4.5. Control Algorithms........ 52

4.5.1. Introduction........ 52

4.5.2. Motion Trajectory Planning........ 54

4.5.3. Cartesian Space Decomposition of Event-Based Plans........ 57

4.5.4. Control Law........ 59

4.6. Simulation Results........ 61

4.6.1. Case 1: Straight Line.. 61

4.6.2. Case 2: Broken Line.. 64

4.6.3. Case 3: Circle... 66

4.6.4. Case 4: Straight Line with an Unexpected Obstacle........................ 68

5. Conclusions

5.1. Summary... 71

5.2. Future Research.. 71

Bibliography.. 73

Appendix A.. 80

Appendix B.. 81

List of Figures

2.1 Simplified Internet-based Control System Architecture........................... 9

2.2 Popular Internet-based Robotic Control System Architecture.................. 11

2.3 Communication of Interactive Machine Control Website......................... 13

3.1 System Architecture... 17

3.2 Interface of HTML, Java, and VRML... 24

3.3 Communication System Configuration.. 24

3.4 Interface for User to Send Control Command 27

3.5 Web Page Sent Back by Using Servlet.. 28

3.6 Result of Java and C Working Together... 33

3.7 Interface for Using Servlet to Invoke Java Native Interface....................... 38

3.8 Response of Java Servlet Invoking Java Native Interface......................... 38

3.9 Login Interface1 40

3.10 Login Response1 40

3.11 Login Response2 41

3.12 Login Interface2 41

3.13 Get Image from Server Using Applet 42

3.14 Get Image from Server Using Servlet 43

3.15 Response of Getting Image from Server Using Servlet 43

4.1 Interface for Controlling SSC Controller 47

4.2 Model of Two-Link Robotic Manipulator 49

4.3 Control System Architecture 53

4.4 Velocity and Acceleration Profiles of the Time-Optimal Motion Plan 57

4.5 Decomposition of Straight Line and Circular Path 58

4.6 The Event-Based Error Definition 60

4.7 Straight Line Path Tracking 63

4.8 Broken Line Path Tracking 65

4.9 Circle Path Tracking 67

4.10 Straight Line Path Tracking with Unexpected Obstacle 70

Appendix1 System Architecture for Image Transfer.................................... 80

Appendix2 System Architecture for Control Algorithm.................................. 80

Chapter 1

Introduction

1.1 Introduction

New applications that can benefit from teleoperation systems offer challenging remote control development. Today, robot system is not only applied in factory automation and space exploration applications, but also in home healthcare, rehabilitation and surgical operation assistance tasks. In teleoperated systems, the control commands and remote sensing measurements are transmitted via various media, such as radio, micro-wave, and computer network. The rapid evolution of communication systems, related software and programming languages provide simple and feasible solutions to teleoperation system development. Internet is becoming the preferred platform for interactive communications. The Internet has the advantage of worldwide availability, which makes it possible to control the robot from anywhere in the world where the Internet is available.

A number of Internet-based teleoperation systems have been developed [12, 15, 16, 64]. They are all developed by using the Internet as the communication medium, but the technical details of the sites' implementations are different.

Recently, not only the teleoperation systems have been developed, but also many aspects of the remote control systems of the telerobotic manipulators have been under study. Issues concerning communication channels, communication propagation delays, bandwidth limitations and telepresence have all been dealt with to various degrees [5]. Because most teleoperation systems use the Internet as the communication medium for the control of remote robots, many research projects have been targeted to deal with the shortcomings of the Internet based control [52-54].

1.2 Thesis Objectives

By using the Internet, a robotic teleoperation system has been developed. Instead of developing a highly sophisticated, multi-million-dollar test-bed, we opted for a simple and reliable two-link robot manipulator. Combined with an intuitively operable man-machine-interface, the system gives any WWW user access to the teleoperation of the robot. For the users, the only requirement is an installed WWW browser on their computers.

To develop an advanced Internet-based robotic teleoperation system, many newly available techniques, such as Java servlet, Java Native Interface, and event-based control algorithms are studied and applied. The research reported in this thesis covers the user interface, communications, and control algorithms.

1.3 Contributions of the Thesis

The Internet-based robot control system reported in this thesis implies industrial, military and medical applications. The contributions of this thesis can be summarized as follows.

- An Internet-based robot control system, with a client-server architecture is developed. This system can be used to control various physical devices over the Internet.
- A novel control and trajectory-planning algorithm based on event-based control theory is used in the control system and verified via simulations on a two-link

robot manipulator. The simulation results show that the system has good performance in dealing with unexpected events, such as inevitable random time-delays of the Internet.

- We designed a control panel for controlling the Serial Servo Controller so that it is easier for us to take a local test on the control system. We construct this control software by using Visual Basic 6.0.
- We developed the formula of the kinematics and dynamics of the two-link robot manipulator based on Hamilton Principle and developed a new event-based optimal trajectory planning algorithm based on Pontryagin maximum principle. We developed formula on the decompositions of the event-based plan in Cartesian space for straight line and circular path segments.
- In developing the user interface, VRML, Java applet and HTML are used together to accommodate better client-server interactions. Most current web-based telerobotic interfaces only use HTML forms to assert user control on a robot. The availability of a virtual reality model makes the robot control easier to understand by novice users and allows experiments to be prepared in virtual reality before accessing the real robot.
- The communications between the client and the server are carried out by using servlet. A servlet is an extension to a server that enhances the server's functionalities. Servlets offer a number of advantages over CGI. Most current Internet-based teleoperation systems are developed using CGI for the communication part.

- We developed a login interface with servlet and HTML programs to greet its users by name and log each user's logins. So the servlet can be used for user activity tracking.
- We developed Java applets and Java servlets to get images from server. So it is possible to transfer images from the robot side to the remote user.
- We carried the research on client-server communications using sockets. The communications work well, but the client-server applications using TCP socket are lower-level network communications.
- In our system, Java Native Method is used to call non-Java codes. In robotic teleoperation, to get better control performance, time-critical sections of code are necessary. So the control program is written in C language instead of Java. To the best of our knowledge this is the first time that servlets and Java Native Method (JNI) are used together on Internet-based robotic teleoperation tasks. It is a real challenge to invoke servlets from the client interface and at the same time make servlets work together with JNI. Thus, an advanced control algorithm which is written in C can be used in Internet-based robotic teleoperation systems.

Overall, we developed an Internet-based robotic system architecture with advanced techniques, such as servlets, Java Native Method. Many current Internet-based robotic systems do not use these advanced techniques.

1.4 Organization of the Thesis

Chapter 2, Teleoperation Systems, is an overview of existing teleoperation systems and Internet-based teleoperation. Teleoperation architectures and active research areas are also covered. On the Internet-based teleoperation, popular methods to develop an interface and to realize the communications between the client and the server are discussed. The control methods used in Internet-based teleoperation are analyzed and their disadvantages are presented. This chapter is used to derive the design objectives and the research topic for our thesis project.

Chapter 3, Interface and Communications, describes the implementation of interface and communications of the Internet-based robot control system in the ART lab. System architecture including hardware and software is presented. The methods used for developing interface and communications are discussed and determined. We performed the research on the following areas: how to develop an interface using Java, VRML and HTML; how to perform the communications between the client and the server using Java servlets; how to make Java work with C programs using Java Native Interface; and how to perform the client-server communications using sockets. We developed a login interface with servlet and HTML programs for user logging and tracking. We developed Java applets and Java servlets to get images from the server.

Chapter 4, Control, describes the implementation of the control system and the applications of event-based planning and control scheme in the system. Control system architecture and control realization are described. We designed a control panel to take local test on the control system using Visual Basic 6.0. We developed the formula of the kinematics and dynamics of two-link robot manipulator so that simulation research can be carried out. We developed a new event-based optimal trajectory planning algorithm based on Pontryagin maximum principle. We developed formula on the decomposition of the event-based plan in Cartesian space for straight lines and circular path segments.

The control law used in the system is described, which is based on the event-based control theory. The simulation results of this control system are presented.

Chapter 5, Conclusions, provides concluding remarks and future research.

Chapter 2

Teleoperation Systems

2.1 Introduction

2.1.1. Definition

There are many places that are unsuitable for human presence, such as deep ocean, outer space, nuclear sites, etc. However, we still have to perform many tasks in these places.

As a result, in many cases, these tasks are performed remotely by sending a robot to do the tasks instead of a human, while the robot works under the guidance of a human operator. The human operator supervises the robot through a networked computer. The robot executes tasks based on the information received from the human operator plus its own autonomous sensing and intelligence.

Telerobotics has been an expensive technology. "Remote" telerobotic experiments have used high bandwidth communication links, typically with real-time video and high bandwidth control and sensing links. Therefore, most of the applications have been tasks which could only be performed using telerobotics, or where environmental conditions make human operation hazardous.

A teleoperation system has various components and requires knowledge from many different areas. The major issues that affect the teleoperation performance are:

1. display and sensing;

2. time delay;

3. bandwidth of communication;

4. continuous manual control;

5. operation safety and teleoperation error;

6. autonomy

Modern development trends in telerobot control technology are aimed at amplifying the advantages and alleviating the disadvantage of the human element in control by the development and use of advanced sensing, graphics displays, intelligent computer controls, new computer-based man-machine interface devices and techniques in the information and control channels.

2.1.2. Previous Work

Remotely operated mechanisms have long been desired for use in inhospitable environments such as radiation sites, undersea and space exploration. Recently, teleoperation is being considered for medical diagnosis [4], manufacturing [14] and micromanipulation [35].

The first "teleoperated robots" were developed over 30 years ago. Teleoperation began with very simple mock-ups in nuclear power plants, progressing to more versatile setups for teleoperation of robots in space. Over the last 20 years, the development of intuitively operable teleoperation tools has continued to play an important role in the development of robotics in general. The basic objectives have remained the same, even though the methods and technical limitations have changed [15, 16].

Most traditional telerobots have closed loop control with vision feedback, force feedback or both and the operator closing the loop. Controllers require good concentration because they are in direct control of the manipulator.

2.2 Teleoperation through the Internet

2.2.1. Introduction

The Internet and the WWW system provide a cheap and flexible networking environment for teleoperation (Figure 2.1).

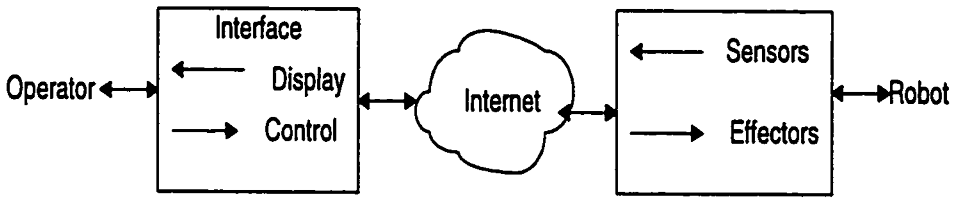


Figure 2.1: *Simplified Internet-based Control System Architecture*

Because Internet-based teleoperation accepts certain bandwidth restrictions, so new classes of application for telerobotics will be proposed:

Robotics training: by using Internet-based robot, university can share robot for education. This technique provides more effective access to a real robot for students from remote sites.

System development, off-line programming and maintenance: one factor inhibiting robotics applications is that programming expertise is often rare. The special skills required for it are seldom found among end-users. Therefore, this technology could be used to provide such skills at remote sites. A robotics system supplier could provide service remotely without having to travel to the customer's site.

Remote inspection and monitoring: this technique could be used for remote inspection and monitoring, particularly for nuclear and other industrial safeguards monitoring.

Space and Entertainment: there is great potential utility of robots in space for construction and repair operations and in entertainment in future.

Most of traditional teleoperation systems require complex hardware at human interface. However, Internet-based teleoperation can provide widespread access by using friendly interface available under the standard HTML language.

From [49], we know that in September 1994, an ASEA IRb-6 robot was connected to the Internet through a Web Server at the University of Western Australia so that anyone with Web access could control the robot. Just four weeks earlier, Ken Goldberg [19] at the University of California, Berkeley had connected a SCARA type robot to the Internet.

Other related research:

Ken Goldberg's Telegarden, http://www.usc.edu/dept/garden/

Railroad at the University of ULN, Denmark, http://rr-vs.informatik.uni-ulm.de/rr/

A web interface for NASA's Sojourner Rover, http://mars.graham.com/wits/

Lunar Rover Initiative, http://www.frc.ri.cmu.edu/projects/lri/

CyberCut, http://CyberCut.berkeley.edu/

Mercury Project, http://www.usc.edu/dept/raiders/

Drinking Maiden, http://digimuse.usc.edu/robot/

PumaPaint, http://yugo.mme.wilkes.edu/~villanov/

Robotic Telescope Observatory site, http://www.eia.brad.ac.uk/rti/

Ken Taylor's Telerobot, http://telerobot.mech.uwa.edu.au/

Interactive Machine Control Website, http://www.aml.gvsu.edu/~ihcws/process.html

The NASA Space Telerobotic Program web site lists over 20 "real robots on the web", http://ranier.oact.hq.nasa.gov/telerobotics_page/realrobots.html.

2.2.2. System Architecture

Most Internet-based robotic control systems use system architecture shown in Figure 2.2.

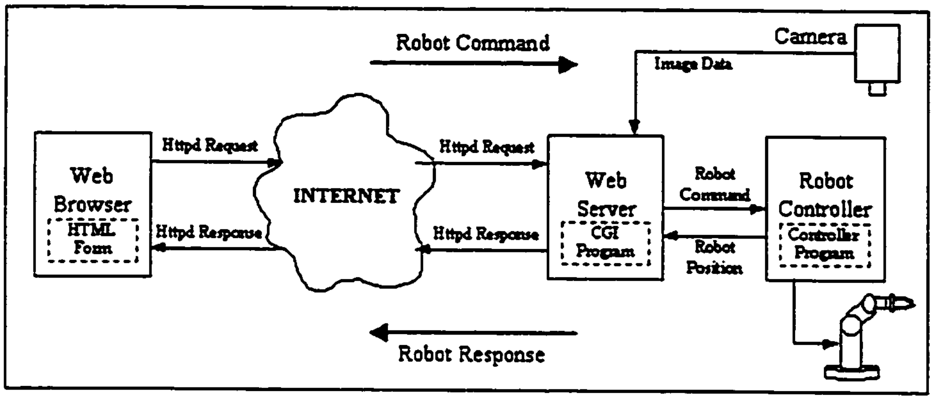


Figure 2.2: *Popular Internet-based Robotic Control System Architecture*

By using this architecture, user's computer in the world can communicate with the server via the Internet. The only requirement on user's computer with web browser is linking to the Internet. World Wide Web users can use PC, Macintosh, workstation, etc. The server receives client's command from world wide network and then sent it to the special device. A high-performance computer should be used as server so that it can provide services for many users simultaneously. The communication between server and robot is through a wireless network interface or serial connection.

Interface is developed by using HTML forms to assert user control on a robot. CGI (Common Gateway Interface) program is used to perform the communication between client and server. Software design for the system is divided into three parts. The first is the assignment of the server, the next is the communication between client and server, and the third is the user interface.

2.2.3. Interface and Client - Server Communications

2.2.3.1. Common Gateway Interface (CGI)

CGI is a mechanism that allows Web clients to execute programs on a Web server and to receive their output. CGI applications are often used to produce HTML pages on the fly; they are also used to process the input from an HTML form. The CGI is implemented in the Web server.

In [49], users of robot submit requested moves by filling in fields on an HTML form or by clicking images of the workspace. The user's browser submits the form details as a CGI request to the web server, which receives the request and launches a CGI script to carry out the requested robot move. Once a move is complete new images are taken of the workspace and a new form with the latest images and robot position is returned via CGI to the user.

Ken Goldberg's Telegarden project use HTML forms and CGI to control the motions of robots. Users enter three-dimensional coordinates into HTML form and press a "Submit" button sending the command to the robot's web server. The server utilizes a CGI program to decode the received instructions, passing them on to a daemon program that positions the robot.

There are many other systems where program is invoked by the WWW server using CGI scripts (e.g. [2, 8, 18, 19, 36, and 41]).

Many current web-based telerobotic interfaces use HTML forms to assert user control on a robot [8, 49, and 55]. But [60] utilizes a Java applet to control a robot during the task of painting on a canvas.

The use of Java allows the sophistication of the user interface to be raised to the level required for satisfactory control.

2.2.3.2. Socket Connection

In Interactive Machine Control Website, socket communication is used between the client and the server (Figure 2.3).

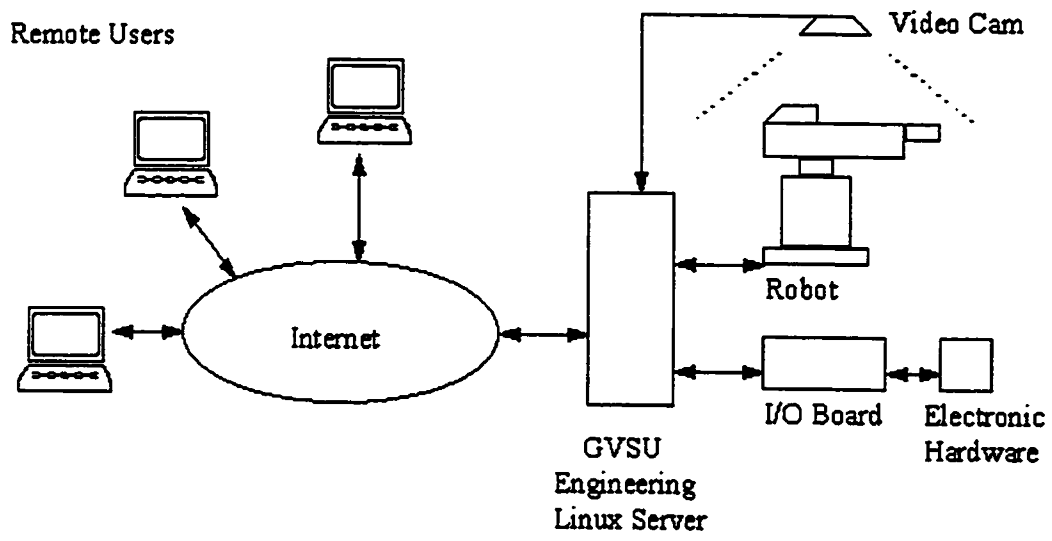


Figure 2.3: *Communication of Interactive Machine Control Website*

The java client requests a socket at the specified port when a user opens the HTML Robot page. If a socket is available (no current users), a socket is provided. The timer starts now. The socket established provides un-interrupted two-way communication between the user and hardware. When a button is pressed on the java applet, a string is sent to the server and the timer is reset, and the server processes the string. For each command string sent, the timer is reset. This establishes a checking mechanism between the server and client.

In addition, in [18], the manipulation simulator exists at the remote site, and communicates with Java process at local site through a socket connection.

In [60], a small proxy server is implemented to accept a communication socket from the outside world, and if valid, passes the command packet to the true robot server daemon program running on a more secure machine.

The client-server application using TCP socket is lower-level network communication. Firewall security prevents socket communication between the client applet and the server.

2.2.4. Control

In many Internet-based robot control systems, robot is controlled by using serial connection with server (e.g. [55, 64]).

Most traditional telerobots have closed loop control with vision feedback, force feedback or both and the operator closing the loop. These approaches work fine when there is little delay in communication, however once transmission delay is introduced the systems become unstable. Previous work on teleoperation with time-delay has focused on constant delay systems [1, 22, and 33].

Previous work on control algorithms for time-delay systems has focused on compensating delays that are either constant or known [7, 21, 23, and 31]. But they are not applicable to Internet-based teleoperation, since the time delay via the Internet may vary a lot and unpredictable. Internet delays are modeled as random variables. A solution to a random time-delay could consider its maximum to design a worst-case controller.

But a control algorithm designed for a maximum delay can not stabilize the system when the delay varies from 0 to this maximum delay [17]. In [34], a few qualitative experiments have been carried out to gain some insight into the problem of Internet-based telerobotic systems. The experiments highlight the characteristics of the Internet, and the effects on the performance of a teleoperation system. But the design of a suitable control law for Internet-based control systems still requires the integration of some specific techniques for handling the problems of random time-delay. To overcome this problem, a method employed by [18] is that he plans motions of objects off-line on a real world simulator, and sends the planned motions to a remote site afterward. But this method is not so good as the one that we can manipulate objects at a remote site on line. In addition, another main performance problem is due to packet losses. So far, the issue of the packet losses has been addressed by showing that some packet loss can be compensated by using n-step predictor [26]. However, such predictor requires the knowledge of the models of the local and remote processes, and the robustness of this approach is not clear. A less demanding approach consists of taking advantage of the availability of the statistic distribution of the losses, and of modeling the effect of the losses in terms of convergence of the overall system to the desired working point. This method is usually not acceptable for some applications such as telemedicine. So in our system we use a new method – event-based control method to deal with the random time-delay and packet losses existing in the Internet.

Chapter 3

Interface and Communications

3.1 Introduction

In teleoperation systems, the human operator and the robot interact with each other and carry out the required tasks cooperatively. In order to operate the robot, it is necessary to obtain the information about the robot, and to give commands to the robot through communication links. The Internet offers the infrastructure for communication information flow to enable this kind of interactive operations (Figure 3.1).

In order to interface this service to the Internet, two separate software packages have been developed. The client part is on the operator's computer. The server part resides and executes on a host machine in remote robot site running Java Web Server 1.1.3. In the system, the server is connected to the Internet to accept the incoming requests, and it also serves as a control center. So it can send commands to the controller of robot after it receives the request from the operator. The host machine is a networked PC with Windows NT, 4.0. Robot controller is attached to a networked host machine. In every interaction cycle, the server feeds control result to the client and the client issues control commands to the server.

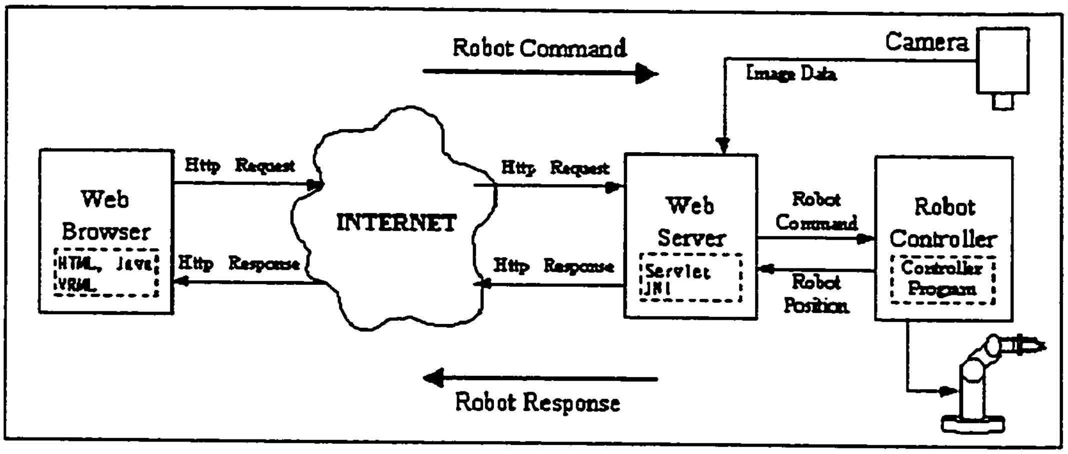


Figure 3.1: *System Architecture*

When the operator uses a Java enabled browser to retrieve the robot's control web page, the operator will be presented with a Java applet. This applet will run in the client side.

There are two popular methods to develop the interface: web-based and interactive application. A web-based interface is a platform-independent HTML form that works with a server side CGI program or servlet. Web browser forms allow the designer to distribute the interface in a platform independent manner with little or no programming on the interface side. The bulk of the processing behind an HTML interface is handled by the CGI or servlet programs on the server side. These programs can involve sophisticated access control subsystems that will decode the interface inputs (parameter and control command) and pass them to the actual control programs for the robot. By using this method, interface can be created easily and there are multiple platforms to which it can be distributed. However, it suffers from the "set-submit" cycle. On the other hand, with an interactive application, the user interacts with an executing program. The program is written and compiled to a specific hardware platforms and utilizes the communication capabilities of the platform to connect to a server program. The interface program can then be released to users having the same hardware and operating system platform. This approach, therefore, only benefits those users with the same platform. Applications require a greater effort to design and code the interface when compared to HTML forms, but they benefit from the ability to be more complex, supporting interactive tasks in a improved fashion. Applications have the added benefit of distribute processing. The client-side application deals with the interface and interaction with the user, and the server-side controls the robot.

Our system combines positive aspects of these two approaches. It presents an architecture-neutral interface through an interactive program running on the client side. It is easily accessible because it is web-based. This combination is accomplished using Java applets and HTML.

Applets and servlets handle the operator-system interface by accepting the control commands and displaying all information needed by the operator. Using the information of the in-call, a connection is opened to the robot control daemon. In this web page, we develop a text area to allow users to leave comments about the execution quality or whatever they want to communicate.

When developing this Internet-based robot control system, programming language C, Java (Java Servlet, Java Native Interface), VRML, HTML, Visual Basic are used. Because of Java's platform-independent feature, it allows us to abstract from individual system platforms while deploying our programs into any general-purpose computers with Internet connection and a WWW browser.

Measures to prevent uninvited visits should be included in such systems, while in the latter section of this thesis, we will lay much emphasis on applying event-based control algorithms on this system to prevent unexpected time-delay. So when a connection is made, the system first checks for authentication using user and password interface developed using servlets. This prevents unauthorized control of the robot hardware, so as to limit our prospective end-users within the scope of a safe group. This is particularly important as we move towards devices with the capacity of physical manifestations of energy in remote environment.

The key visual information originates from cameras. The camera is used to collect images of the robot's work environment. The challenges are focus at the issues of acquiring and conveying stereovision information and of connecting all this activity to the operator in order to create a proper visual environment for the operator. In the system, cameras are set on server side. When an image capture request is received, the camera digitizes an image, converts and compresses it into a GIF. The file is output into a temporary space in server. Image capturing request is handled by servlets, and the requested image will be displayed in the HTML document passed back to the corresponding user.

3.2 Internet, World Wide Web (WWW) and HTML

The Internet can be considered as a strongly connected network of computers, communicating with each other using packet-switched protocol. The use of the Internet to support the communications between the operator and the robot is quite attractive due to its worldwide availability, which makes possibility to control the robot from everywhere in the world. Since the packet exchange in the Internet is affected by the packets' routes and handling policies at each node they traverse, the communication time delay is a random variable [34]. So advanced control scheme is needed to overcome random time-delay existing in the Internet.

World Wide Web is a global, interactive, dynamic, cross-platform, distributed, graphical hypertext information system that runs over the Internet. WWW, including the HTML (Hyper Text Markup Language) language and the HTTP protocol, provides a standard graphical interface to the Internet. We want to remotely control robot through the Internet. So we should develop HTML documents used as a web page and an interface for control operator. After control command is issued, new HTML documents are generated by servlets running on the web server and send back to the client.

3.3 Interface from HTML, VRML and Java

The system we developed will be used to remotely operate a robot through the Internet. In this system, it is crucial that the user interface is designed in a way that optimally presents information about the remote site. The trade-off that must be addressed is due to the finite bandwidth limitations of the communication channel, which will restrict the amount of information that can be presented to the operator. So we develop operator-computer interface using Java, VRML and HTML.

Since the Internet is used as the communication medium, it seems obvious that communication software should be developed in a language both Internet oriented and with multi-platform support. Java is the language which best satisfies these needs. Java has networking capacities. Java is suitable for writing programs that use and interact with the resources on the Internet and the World Wide Web. Java-compatible browser uses this ability of Java to transport and run applet over the Internet. Java allows us to abstract from system architecture, since the program written in this language will run in any platform with a World Wide Web (WWW) browser. A potential operator only has to correctly specify the Uniform Resource Locator (URL) address of the robot's control web page to operate robot remotely. But Java does not have access to such devices as serial ports due to security reasons.

VRML (Virtual Reality Modeling Language) is a language used to make 3D worlds. It is designed to fit into the existing infrastructure of the Internet and the WWW. It provides the technology that integrates 3D, 2D, text, and multimedia into a coherent model. When these media types are combined with scripting languages and Internet capabilities, an interactive application is possible. It provides a simple way to create dynamic worlds and sensory-rich virtual environments on the Internet.

By using this interface, while the video feed is being sent to the operator from the remote site, the operator's computer also updates VRML graphical model of manipulator. The reason why use VRML graphical model and video feed back together is that transferring data is easier and faster than transferring image through the Internet. By using this VRML graphical model, before control command is sent to the real robot, the operator can send command to the VRML model first and see what happened. So the operator can make sure if the robot can arrive a specific point and if the robot will meet obstacles in following a trajectory. We can use this interface as a simulator to emulate manipulation process of our real manipulator. Using VRML compared to a video feedback channel has the advantage of consuming less bandwidth. It demands only extremely narrow-band connection and does not bother about long time delays.

Although the virtual implementation could never replace the real experimentation, it turned out to be much more interactive than the real one. This property comes from the fact that every component of the virtual system is downloaded and executed on the local host. Hence, once everything is downloaded (i.e., web page, Java applets, and VRML scene), the user gets a very high execution speed depending only on his computer and not upon the network performance. This very high interactivity makes it easier for users to control the system. Moreover, it also provides a powerful tool for preparing real experiments.

Work Environment:

1. Netscape Communicator 4.0, used as a web browser.

2. Cosmo Player 2.1.1, which supports VRML 2.0 programming.

3. JDK 1.2.2, which supports Java programming.

VRML system requires that VRML worlds change dynamically in response to user inputs, external events, and the current state of the world. Event processing is performed by a program or script. Files containing Java or JavaScript code may be used to implement programmed behavior. The program can perform a wide variety of actions including sending out events (and thereby changing the scene), performing calculations, and communicating with servers elsewhere on the Internet.

The reason why Java is used in our system:

1. complicated calculations and network access are needed.

2. separate threads can be created so that lengthy calculation is performed in a separate thread. When the calculation is proceeding, the user can continue interact with the world.

Using VRML with HTML pages and Java applets can be very effective.

There are different ways that VRML, HTML, and Java may be combined

1. VRML file inside an HTML file.

2. Java code inside a VRML file: using a Script node that refers to compiled Java code.

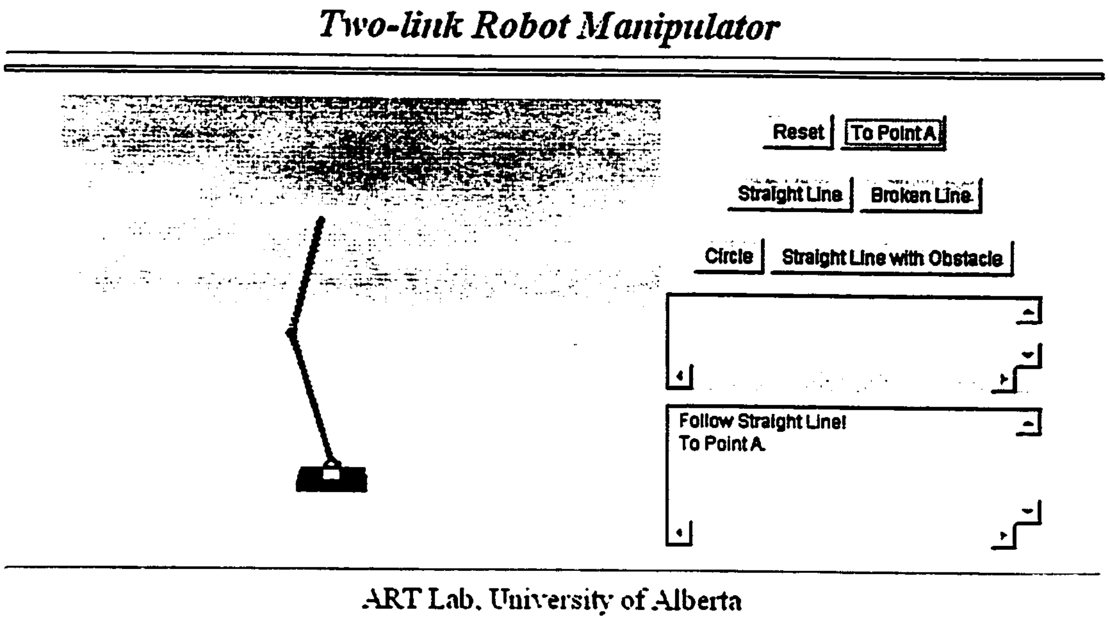
3. Java applet communicating with a VRML browser: known as the External Authoring Interface (EAI).

4. Java classes corresponding to VRML nodes.

5. Java applet inside an HTML file.

In our system, we use methods 1 and 5 to make VRML, HTML and Java work together.

The following interface comes from HTML, Java, and VRML working together (Figure 3.2). By using this interface, operator can send command to virtual robot while sending this command to the real robot, and can get response from the virtual robot and sent-back message came from the real robot. The upper text area is used for user to leave comments about the execution quality or whatever they want.


Two-link Robot Manipulator
Reset
To Point A
Straight Line
Broken Line
Circle
Straight Line with Obstacle

Follow Straight Line!
To Point A

ART Lab. University of Alberta

Figure 3.2: *Interface of HTML, Java, and VRML*

3.4 Java Servlet

3.4.1. Communication Goal

When using the system, a remote operator just needs a general-purpose computer with Internet connection and a World Wide Web (WWW) browser to remotely operate the robot through the Internet.

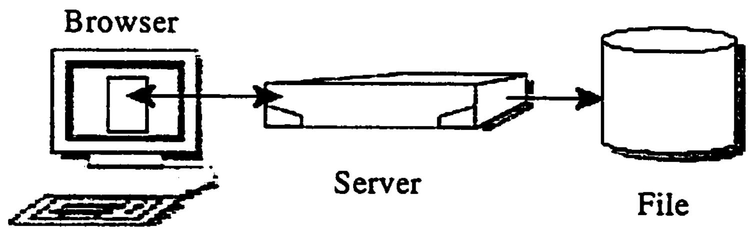


Figure 3.3: *Communication System Configuration*

The system includes the communication (1) between client (with browser) and server and the communication (2) between server and the control program file for robot (Figure 3.3).

HTML file←---→servlet file ←---→C file

(1) (2)

The first communication is performed by using servlet and the second one is performed by using Java Native Interface.

3.4.2. Java Servlet Implementation

3.4.2.1. Why Use Servlets

Paper [49] says, "recently it has become possible to perform processing on the remote computer with Java Applets." Using Java Applets is his dream. Now we know that servlet is better than applet in client-server application. But there is no servlet at all at that time. And paper [2] says, "Java servlet or Remote Method Invocation (RMI) technology might be used in the future in place of the CGI calls to enable more efficient communication."

The rise of server-side Java applications is one of the latest and most exciting trends in Java programming. Java is inherently suited for large client/server applications. Java servlet are a key component of server-side Java development. A servlet is an extension to a server that enhances the server's functionality. Servlets allow developers to extend any Java-enabled server, such as a web server. Servlets offer a number of advantages:

1. Portable: Servlets are portable across operating systems and across server implementations. You can develop a servlet on a Windows NT machine running the Java Web Server and later deploy it effortlessly on a Unix server running Apache.

2. Efficient: With traditional CGI, a new process is started for each HTTP request. With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread. In traditional CGI, if there are N simultaneous request to the

same CGI program, then the code for the CGI program is loaded into memory N times. With servlets, there are N threads but only a single copy of the servlet class.

3. Convenient: Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, and many other utilities.

4. Powerful: Java servlets let you easily do several things that are difficult or impossible with regular CGI, such as servlets can talk directly to the Web server. Servlets are well suited for enabling client/server communication.

5. Safe: Servlets support safe programming practices on a number of levels. Because servlets are written in Java, they inherit the strong type safety of the Java language. A server can further protect itself from servlets through the use of a Java security manager.

3.4.2.2. Support for Servlets

Java servlet is an extension to Java and they are not part of the core Java API. Although they may work with any Java Virtual Machine (JVM), servlet classes may not be bundled with all JVMs. To develop servlets, the Java Servlet Development Kit (JSDK), with which the servlet API is bundled, can be used, together with Java Development Kit (JDK) version 1.1 and above. In our system, we use JSDK 2.0, and Java 2 SDK v1.2.2. We need JSDK to compile our servlets. In addition to JSDK and JDK, we need a servlet engine, so that we can test and deploy our servlets. The choice of servlet engine depends in part on the web server used in the system. In our system, we use Sun's Java Web Server 1.1.3 written in Java, which is unofficially considered the reference implementation for how a servlet engine should support servlets.

3.4.2.3 How Servlets Work

System Configuration:

1. Java Servlet Development Kit (JSDK 2.0)

2. Java Development Kit (Java 2 SDK v1.2.2)

To develop servlets, these two software are necessary.

3. Java Web Server 1.1.3

It is used as a servlet engine.

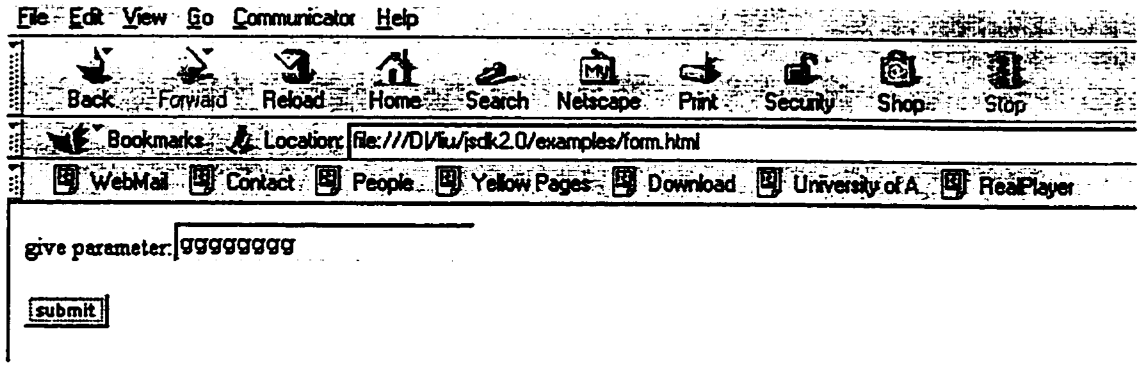
4. Netscape® Communicator 4.7

A browser used to activate the servlet by typing the URL (such as http://localhost: 8080/servlet/Hellow) into browser.

Procedure:

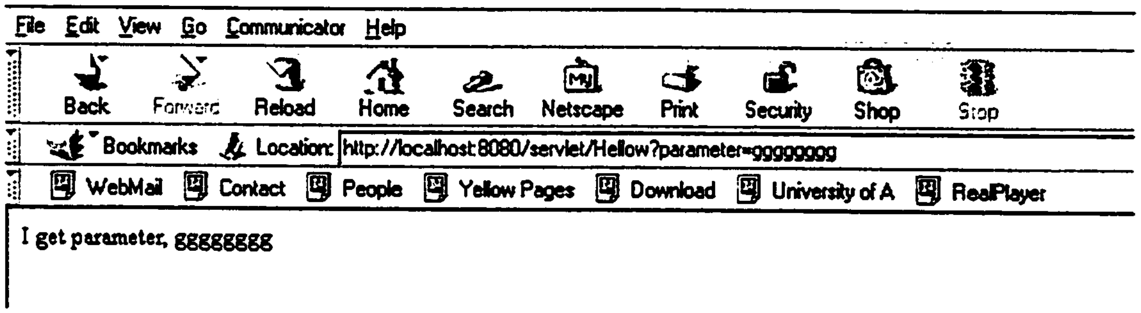
An HTML file and a servlet program work together to realize the communication between client and server. HTML file will provide an interface for user to send control command or parameter. The HTML form has one text input field and a "Submit" button. When the end user clicks the "Submit" button, the corresponding servlet is invoked to process the end user input and the servlet returns an HTML page that displays the text showing that the servlet has received and processed the input parameter.

These two program files are in server side. Especially, the compiled servlet program, that is a *.class* file, should be put in directory: JavaWebServer1.1.3\servlets. So that browser can activate the servlet.


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///D|/liu/jsdk2.0/examples/form.html
WebMail Contact People Yellow Pages Download University of A RealPlayer
submit

give parameter: gggggggg

Figure 3.4: *Interface for User to Send Control Command*


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:8080/servlet/Hellow?parameter=gggggggg
WebMail Contact People Yellow Pages Download University of A RealPlayer

I get parameter, gggggggg

Figure 3.5: *Web Page Sent Back by Using Servlet*

By using these two programs, server gets input from a client and shows to the client that the server has received this input. The servlet retrieves the input and can use it later. Now the communication between a client and a server is realized (Figure 3.4 and Figure 3.5). And then, how can the server invoke control program written in C programming language?

3.5 Java Native Interface (JNI) Implementation

The Java language and its standard API are rich enough to write full-fledged applications. But in some cases you must call non-Java code; for example, if you want to access operating-system-specific features, interface with special hardware devices, reuse a preexisting, non-Java code base, or implement time-critical sections of code.

Interfacing with non-Java code requires dedicated support in the compiler and in the Virtual Machine, and additional tools to map the Java code to the non-Java code. The standard solution for calling non-Java code that is provided by JavaSoft is called the Java Native Interface. JNI is a fairly rich programming interface that allows you to call native methods from a Java application.

Currently, JNI is designed to interface with native methods written only in C or C++.

Why use Java Native Interface in our system? In the Internet-based robotic control system, to get better control performance, using time-critical sections of code is necessary. So in our system, control program is written in C language instead of Java. Servlets are written in Java and control program is written in C programming language. How can these two programs work together? The only answer is to use Java Native Interface (JNI). The JNI is designed to handle situations where you need to combine Java applications with native code (C, C++ code). As a part of the Java virtual machine implementation, the JNI is a two-way interface that allows Java applications to invoke native code and vice versa.

"Java ←--→Java Virtual Machine (JNI) ←--→ Native code" shows the role of the JNI.

When an application uses the JNI, it risks losing two benefits of the Java platform. First, Java applications that depend on the JNI can no longer readily run on multiple host environments. Second, while the Java programming language is type-safe and secure, native languages such as C or C++ are not. A misbehaving native method can corrupt the entire application.

Because applications are written in Java language, as well as in native (C, C++, etc.) programming language, two programming environments are involved.

The Java platform is a programming environment consisting of the Java Virtual Machine (JVM) and the Java Application Programming Interface (API). Java applications are written in Java language, and compiled into a machine-independent binary class format. A class can be executed on any Java virtual machine implementation. Any implementation of the Java platform is guaranteed to support the Java programming language, virtual machine, and API.

The term host environment represents the host operating system, a set of native libraries, and the CPU instruction set. Native applications are written in native programming languages such as C and C++, compiled into host-specific binary code, and linked with native libraries. Native applications and native libraries are typically dependent on a particular host environment. A C application built for one operating system typically does not work on other operating systems.

Java platforms are commonly deployed on top of a host environment. The Java Runtime Environment supports the Java platform on existing operating systems such as Windows.

In order to develop Java Native Interface programs, we should first establish the system environment:

1. Java 2 SDK, v1.2.2
2. Microsoft Visual C++ 5.0
3. MS-DOS

Secondly, we should follow the correct procedure. The procedure of using Java 2 SDK release to write a Java application that calls a C program is:

1. Create a class (*classname.java*) that holds native methods declarations.
2. Use *javac* to compile the *classname.java* source file, resulting in the class file *classname.class.*
3. Use *javah* to create *classname.h* header file.
4. Use *javah –stubs classname* to create *classname.c* stubs files.
5. Write C implementation (*classname.c*) of native methods.
6. Compile the C implementation into a native library. Use C compiler and linker available on Visual C++5.0 to create *classname.dll. classname.dll* is a Dynamic-Link Library that contains the C implementations of native methods.
7. Use the native methods in a Java program. Write a Java program *Test.java* that incorporates *classname.class.*
8. Compile *Test.java* under *classname.class* involved, resulting in the class file *Test.class.*
9. Bring up a DOS window and use *java* to run the Test program. Both the class file (*Test.class*) and the native library (*classname.dll*) are loaded at runtime.

The following programs show how Java and native method work together.

1. *NativeMethods.java*, that holds native methods declarations

```
class NativeMethods
{
                      ⋮
    protected String message;
```

```
        public native void setMessage(String msg);
            public native int printMessage();
            static {
                    System.loadLibrary("NativeMethodsImp");
    }
```

The NativeMethod class declares two methods called setMessage () and printMessage (). The native keyword tells Java that these methods will be implemented in another language. A static program block is included. A static program block runs automatically when a class is run. The static program block calls Java's loadLibrary () method. The NativeMethodsImp library will contain the implementations of the setMessage () and printMessage () native methods which are written in C language.

2. *NativeMethodsImp*.c, that is the C implementation of native methods

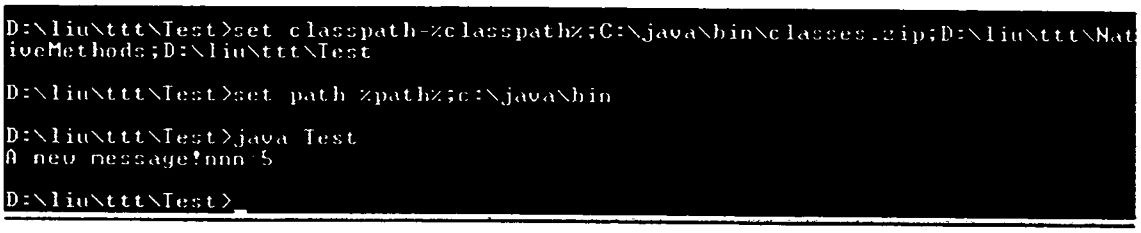
```
                    :

        void NativeMethods_setMessage(struct HNativeMethods* this,
                    struct Hjava_lang_String* msg)
            {
                    ClassNativeMethods* data = unhand(this);
                    data->message = msg;
              }


          int NativeMethods_printMessage(struct HNativeMethods* this)
      {
          ClassNativeMethods* data = unhand(this);
```

```
            javaStringPrint(data->message);

            return 5;

        }

              :
```

3. *Test.java*, the program that uses native methods

```
class Test

{

                    :

   public static void main(String args[])

   {

      NativeMethods nativeMethods = new NativeMethods();

      nativeMethods.setMessage("A new message!");

      int nnn=nativeMethods.printMessage();

              System.out.println("nnn="+nnn);

   }

}
```

Given correct path and classpath, run Test.java program, and we get correct output.



```
D:\liu\ttt\Test>set classpath-%classpath%;C:\java\bin\classes.zip;D:\liu\ttt\Nat
iveMethods;D:\liu\ttt\Test

D:\liu\ttt\Test>set path %path%;c:\java\bin

D:\liu\ttt\Test>java Test
A new message!nnn 5

D:\liu\ttt\Test>_
```

Figure 3.6: *Result of Java and C Working Together*

The result shows that Java really calls C functions (Figure 3.6).

3.6 Java Servlet and JNI Work Together

By using servlets, the communication between client and server is realized (Communication(1)). After user enters control command in HTML form, the invoked servlet is to process the end user's input and return an HTML page.

By using Native Method Interface, that Java program calls C program is realized (Communication (2)). From above illustration, these two programs are in the same computer, and there is no network communication involved. The result is shown in MS DOS window.

client←---→server (servlet)	**Java program←---→C file**
Communication(1)	**Communication (2)**

How can realize that C program file is called from client side?

The answer here is using Native Methods with servlets. Although we can not find any literature that shows how servlet works with Native Methods, we still go ahead to try. Servlet is extension to Java and it has special structure for client-server communication.

The reason why we believe that servlet can work with Native Methods is that servlet is Java program, and Java program can work with Native Methods, although servlet has special structure for client-server communication.

Native code should not be used except when absolutely necessary, since if the native code run by a servlet goes south, the entire server goes down with it. The security protections in Java can not protect the server from native code crashes. Native code also limits the platform independence of a servlet.

How a servlet accesses native methods depends on the web server and Java Virtual Machine (JVM) in which it is running. To make a servlet accesses native methods, web server and JVM must support the standard Java Native Interface (JNI).

Servlet runs inside request/response-oriented server. Although it has special architecture that suits for the communication between client and server, servlet is still a Java program. So we try to combine servlet with Java progarm that calls C programs.

The following programs show how Java native methods work with servlets, so that client can call C file in server through servlets.

1. servlet

```
                 ⋮
public class Hellohh extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
                 throws ServletException, IOException {
                 ⋮
  PrintWriter out = res.getWriter();
  String parameter = req.getParameter("parameter");
    NativeMethods nativeMethods = new NativeMethods();
    nativeMethods.setMessage("A new message!");
    int nnn=nativeMethods.printMessage();
         String str="java";
  out.println("<HTML>");
                    ⋮
  out.println("<BODY>");
```

```
   out.println("12,june,I get parameter," + parameter);
    out.println("out,"+str);
   out.println("vvv,"+nnn);
   out.println("</BODY></HTML>");
  }
   public void doPost(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {
   doGet(req, res);
  }
}
```

2. Java program that holds native methods declarations

```
class NativeMethods
{
                        ⋮
    public native void setMessage(String msg);
        public native int printMessage();
    static {
                    System.loadLibrary("NativeMethodsImp");
    }
}
```

3. C implementation of native methods

```
                        ⋮
void NativeMethods_setMessage(struct HNativeMethods* this,
                    struct Hjava_lang_String* msg)
{  ClassNativeMethods* data = unhand(this);
   data->message = msg;
}
int NativeMethods_printMessage(struct HNativeMethods* this)
{
   ClassNativeMethods* data = unhand(this);
   javaStringPrint(data->message);
   return 5;
}
```

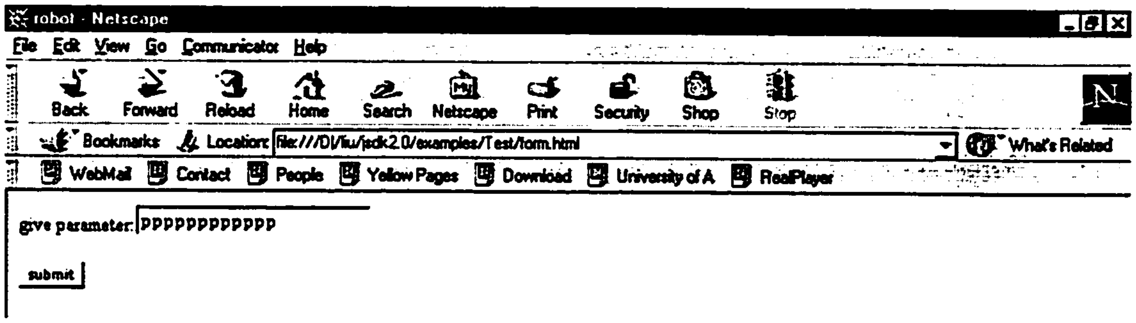
4. HTML file that calls servlet

```
<html>
                    ⋮
<body>
<form METHODS=GET ACTION="http://localhost:8080/servlet/Hellohh">give
parameter:<input TYPE=TEXT NAME="parameter">
<p><input TYPE=SUBMIT value=submit></form>
</body>
</html>
```

There is a common Java Virtual Machine bug that does not allow native code to be loaded by a class that was loaded with a custom class loader (such as the class loader that loads servlets from the default servlet directory). Servlets using native code may therefore need to reside in the server's classpath (server_root/classes).

The directory where the dynamic load library (DLL) that contains the native code is placed depends on the web server and Java Virtual Machine. Some servers have specific locations where they look for libraries. In our system, Java Web Server looks for dynamic load library in server_root\lib. If the server does not provide a specific directory, try placing the library in a JVM-specific location or an operating system-specific location.

The following two web pages show the result of calling C program resided in server side from client through servlet and native methods (Figure 3.7 and Figure 3.8).

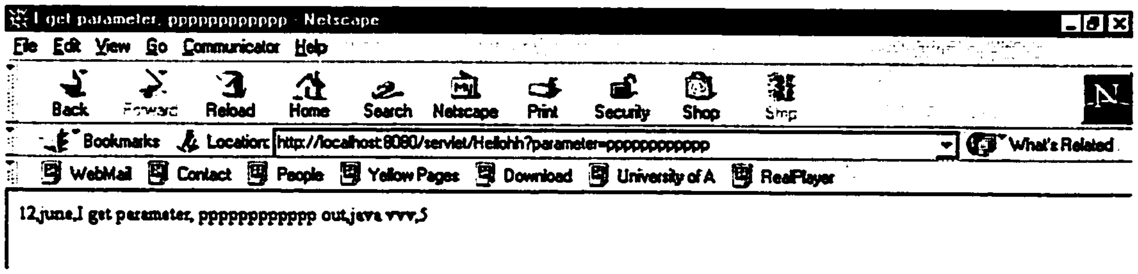

robot - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location:
What's Related
WebMail Contact People Yellow Pages Download University of A RealPlayer
submit

file:///D/liu/jsdk2.0/examples/Test/form.html

give parameter: pppppppppppp

Figure 3.7: *Interface for Using Servlet to Invoke Java Native Method*

In the above web page, if "submit" button is pressed, the related servlet will be invoked. then the following page will be obtained.


I get parameter, pppppppppppp - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location:
What's Related
WebMail Contact People Yellow Pages Download University of A RealPlayer

http://localhost:8080/servlet/Hellohh?parameter=pppppppppppp

```
12,june,I get parameter, pppppppppppp out,java vvv,5
```

Figure 3.8: *Response of Java Servlet Invoking Java Native Interface*

From the returned web page, we can see that

1. servlet can return any text (12, june, I get parameter, out, vvv) to client in returned web page.

2. input parameter from client is extracted by servlet, and it use the parameter in returned web page (pppppppppppp) and it can use it later.

3. servlet can return string value set in it (such as string java) to client in returned web page.

4. most important, servlet can call native method (printMessage())written in C through object and return the result to client in returned web page(5, that is the return value of calling native method printMessage()).

3.7 Communication Using Sockets

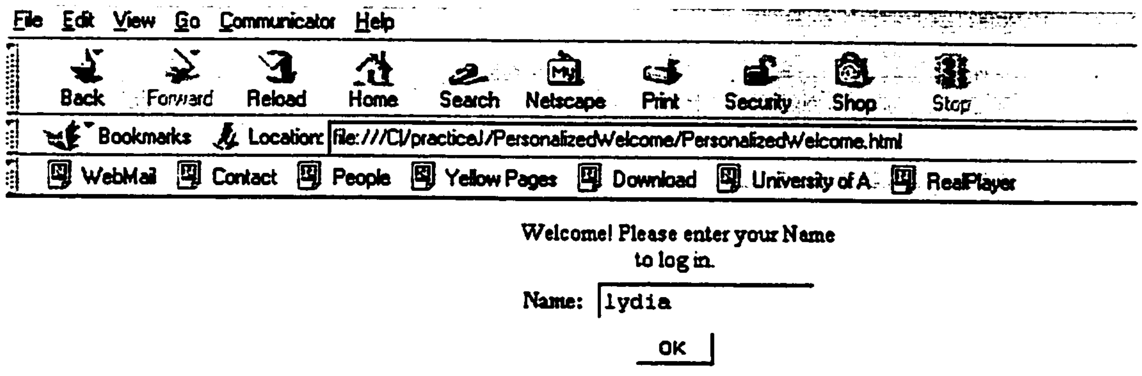
A client-server application can also be realized by using lower-level network communication. TCP provides a reliable, point-to-point communication channel that client-server application on the Internet uses to communicate with each other.

To communicate over the Internet, a client program and a server program are developed which establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection. Client and server must have some protocol by which they can speak to each other. The protocol that client and server use to communicate depends entirely on the communication required. Program for server, client, and protocol are all written in Java application. To perform communication, start the server program first and then run the client program.

3.8 User Authentication

We developed a servlet that works with an HTML program to greet its users by name and remember when each last logged in. So this servlet can be used for user tracking.

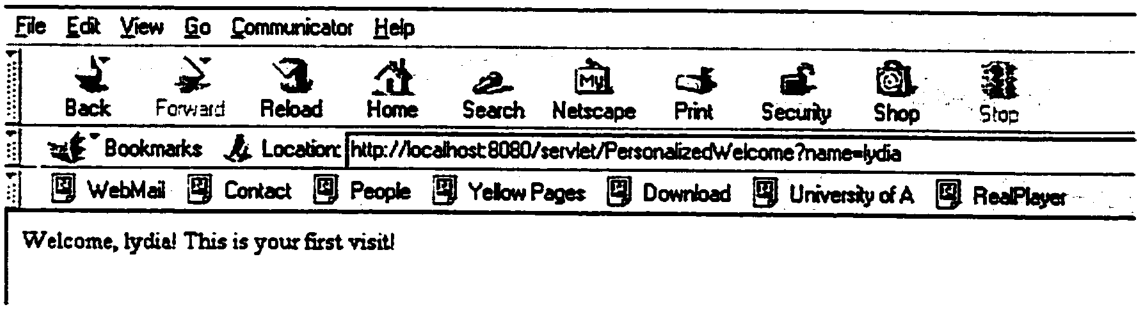
Enter user name to log in. If it is the first time for the user to log in, then a message "Welcome, (user name)! This is your first visit!" will be sent back to the user as response (Figure 3.9 and Figure 3.10). If it is not the first time for the user to log in, then another message will be sent back to the user as response. This message shows the time that the user logged in last time (Figure 3.11).


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: file:///C|/practiceJ/PersonalizedWelcome/PersonalizedWelcome.html
WebMail Contact People Yellow Pages Download University of A RealPlayer
OK

Welcome! Please enter your Name to log in.

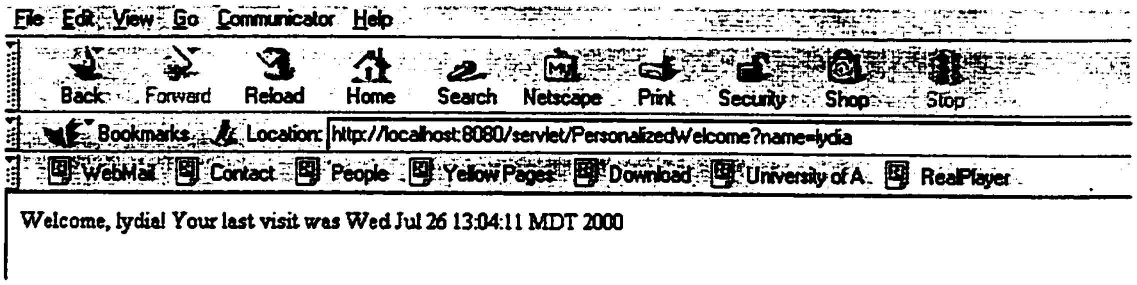
Name: lydia

Figure 3.9: *Login Interface1*


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:8080/servlet/PersonalizedWelcome?name=lydia
WebMail Contact People Yellow Pages Download University of A RealPlayer

Welcome, lydia! This is your first visit!

Figure 3.10: *Login Responce1*

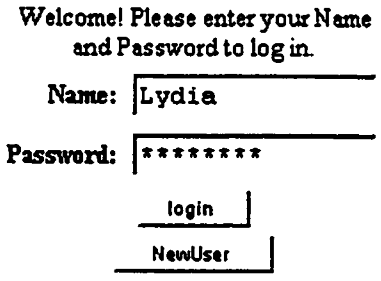

File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location:
WebMail Contact People Yellow Pages Download University of A. RealPlayer

http://localhost:8080/servlet/PersonalizedWelcome?name=lydia

Welcome, lydia! Your last visit was Wed Jul 26 13:04:11 MDT 2000

Figure 3.11: *Login Responce2*

In order to restrict access to our web page, we can ask user to input password to log in (Figure 3.12).


Welcome! Please enter your Name and Password to log in.

Name: Lydia

Password: ********

Figure 3.12: *Login Interface2*

3.9 Get Image from Server

3.9.1. Using Applet

In order to control the robot effectively, the operator needs continuous image feedback from remote site. So, images should be transferred from server to client through the Internet automatically, which has achieved by developing a special designed Java program to avoid manually reloading of images.

After captured plain steam images are converted into Gif-files and stored in the server, Java applet developed continuously display the captured images to user. Image is displayed one by one. After an image is displayed, a fixed time period later the next one.

To deal with complicate Internet world and provide a more flexible operation environment for prospective users, system adds image transfer button on control panel, which user may use whenever needed. The time spice between transmissions is shown on the screen.

This fixed time spice between two images can be changed in program or by user pressing "Fast" and "Slow" button (Figure 3.13).

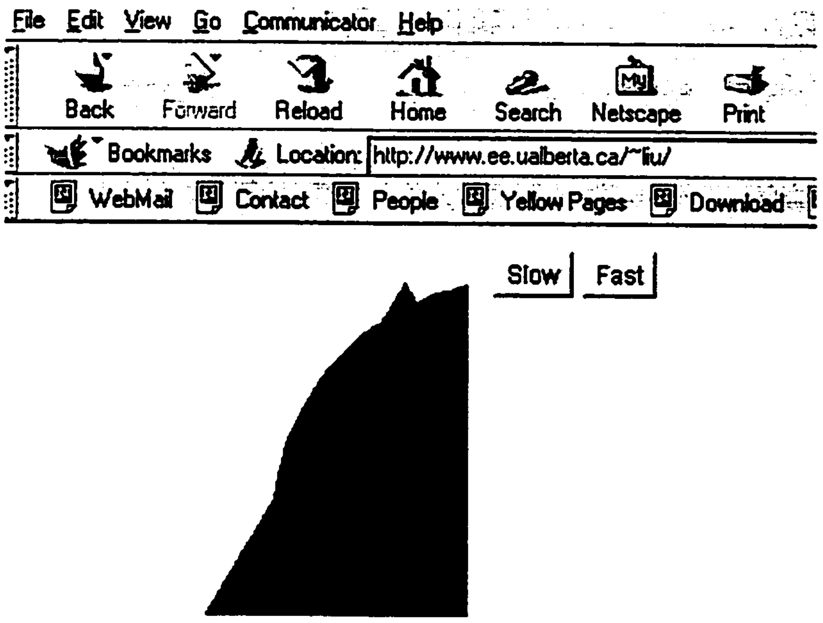
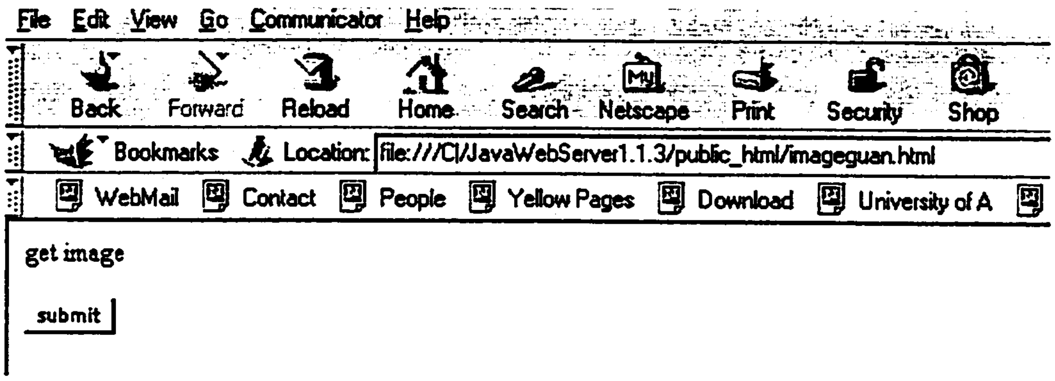

File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print
Bookmarks Location: http://www.ee.ualberta.ca/~liu/
WebMail Contact People Yellow Pages Download
Slow Fast

Figure 3.13: *Get Image from Server Using Applet*

3.9.2. Using Servlet

The restriction of using applet to transfer images from server is that the applet involved must be trusted applets. The security features of applets in web browsers allow communication only to the server machine from which the applet code was transferred. To get a better solution for transferring images, we developed servlet program. The web server monitors requests for images through the use of a servlet program that delivers the current video image to the user (Figure 3.14).


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location: file:///C|/JavaWebServer1.1.3/public_html/imageguan.html
WebMail Contact People Yellow Pages Download University of A

get image

submit

Figure 3.14: *Get Image from Server Using Servlet*

In the above web page, after "submit" button is pressed, ViewFile servlet will be invoked, and the requested image will be sent back to user (Figure 3.15).

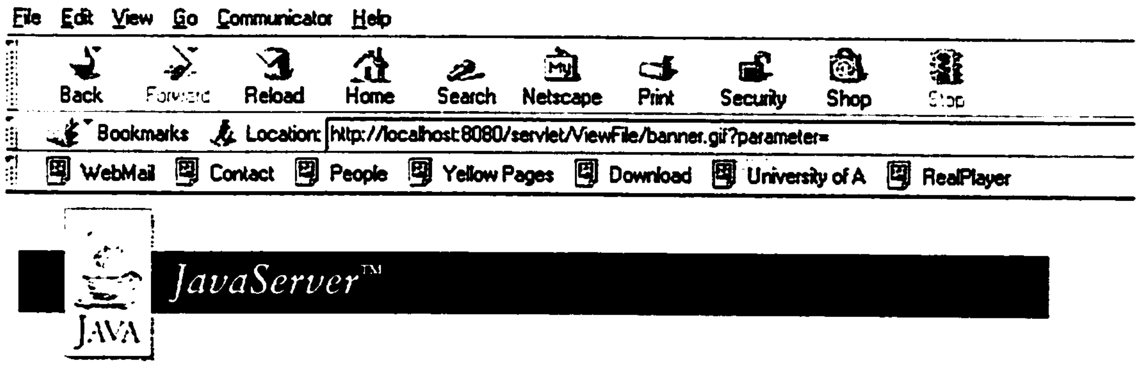

File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:8080/servlet/ViewFile/banner.gif?parameter=
WebMail Contact People Yellow Pages Download University of A RealPlayer
JavaServer™
JAVA

Figure 3.15: *Response of Getting Image from Server Using Servlet*

Chapter 4

Control

4.1 Introduction

Control of the robot is implemented through sending interpreted command lines to the robot controller attached to the online host machine. A proprietary window DLL allows high-level communications with the robot.

When a movement request is received, the server interprets the data in the fields and sends an appropriate movement command to the robot controller. First, a daemon was set up to handle requests involving the robot and auxiliary hardware. The second group of software controls the interface from the Internet to this daemon. The form fields containing the robot instructions are passed as a string to the servlets.

Servlet handles the task requests. Using the information of the in-call, a connection is opened to the robot control daemon. The server-end daemon program may check the validation of command format first. Once the connection is established and check is passed, a request is made to move the robot to the desired locations. After the result of that request is received, the servlet can dynamically lay out the HTML page by extracting information from the data packets sent by control daemon.

On Internet-based control, critical issue is random time-delay. Previous work on teleoperation with time-delay has focused on systems with constant time-delays. But they are not applicable to Internet-based teleoperations, since Internet delays are modeled as random variables. A solution to a random time-delay could use its estimated maximum delay time to design a worst-case controller. But a control algorithm designed for a maximum delay will sacrify in performance when the delay time varies from 0 to this maximum delay. In [43], a few experiments have been carried out to gain some insights into the problems of Internet-based telerobotic systems. The experiments highlight the characteristics of Internet, and the effects on the performance of a teleoperation system. But the design of a suitable control law for Internet-based control systems still requires the integration of some specific techniques for handling the problems of random time-delays.

One effective approach to deal with this problems is the event-based control methods for planning and control which have been successful in reducing the effects of the random time delays by adopting a non-time based reference system in the control algorithm design. The basic idea of the theory is to introduce the concept of an event-based action reference parameter. This parameter, directly relevant to the real-time sensor measurements and the tasks, is independent of time. Thus a non-time based clock driven by sensory measurements is created. Since the new action reference is not directly related to time, it is independent of the random time-delay.

The remote control may be divided into two control modes: low-level mode and high-level mode. In low-level mode, one can control the robot by sending primitive commands and necessary parameters. The robot will execute the command without any intelligence, that is, what the user sends controls the robot directly. In high-level mode, the web users only send high level commands, then the robot will perform the task by using autonomous capacity and local intelligence. The research on building autonomous intelligence of robots, such as collision avoidance, path planning, and object recognition can be applied to enhance the robot's capabilities. Due to the latency of the Internet and the safety of a robot, the high-level mode is essential for the Internet applications. With the local autonomous intelligence of the robots, it is easy for the systems to accomplish teleoperation tasks over the Internet.

4.2 Control System

4.2.1. Controller and Servo Motor

HS-300 Single Servo and Mini SSC II Serial Servo Controller are used in our control system.

HS-300 Single Servo is a self-contained rotational positioning assembly. It is made up of a DC motor, gear reduction, output shaft with position feedback, and a control PC board all built into a small rectangular enclosure. It requires a SSC servo controller to provide the positioning pulses.

The Mini SSC II Serial Servo Controller can control up to eight servos through a computers serial port, using simple instructions at 2400 or 9600 bps.

SSC has the ability to accept serial commands and position servomotors simultaneously.

Communication between the PC and the manipulator controller was performed using serial connection.

4.2.2. Control Panel

In order to take a local test on the control system, we design a serial interface for controlling the Mini SSC II Serial Servo Controller. We construct the serial control software by using Visual Basic 6.0, because this version comes with the MSComm control from Microsoft that makes serial applications programming extremely easy. Here's a screen capture of control interface (Figure 4.1).

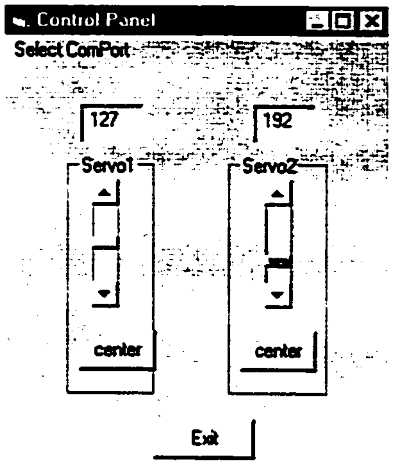


Figure 4.1: *Interface for Controlling SSC Controller*

By using this interface we get complete control of the Mini SSC II Serial Servo Controller, because this interface provide serial communication between PC and SSC. We first introduce how this interface works.

In this interface, the scroll bars are used to control servomotors. The scroll bars let us change the servo position data to be sent to the Mini SSC by sliding each bar up or down. Each individual scroll bar will send a unique number to the Mini SSC to address individual servos.

The command buttons named "center" perform the following tasks:

1. Send data to the Mini SSC II to center the servos.
2. Change the text in the text boxes above each scroll bar.
3. Move the scroll bar position indicator to the middle.

The command button named "Exit" performs the tasks:

1. Close the open com port
2. Exit the application.

Text boxes give me a method to see the actual values of the scroll bars, and contain the servo position data that is being sent through the serial port to the Mini SSC controller.

The menu named "Select ComPort" can be used to select different serial ports.

Next we introduce how the serial control software is developed. To get the interface, we design an executable file. When the main form is run in a VB application it will perform the following tasks:

1. Set up Com Port operating parameters.
2. Select which port to use.
3. Make the com port available.

When writing scroll bar code, make sure that each scroll bar send out:

1. The required (sync) byte that the Mini SSC waits for.
2. The (servo #) that we want to move.
3. The position data that will be sent to the Mini SSC.

4.3 Kinematics

4.3.1. Manipulator Kinematics

Consider two-link robotic manipulator shown in Figure 4.2.

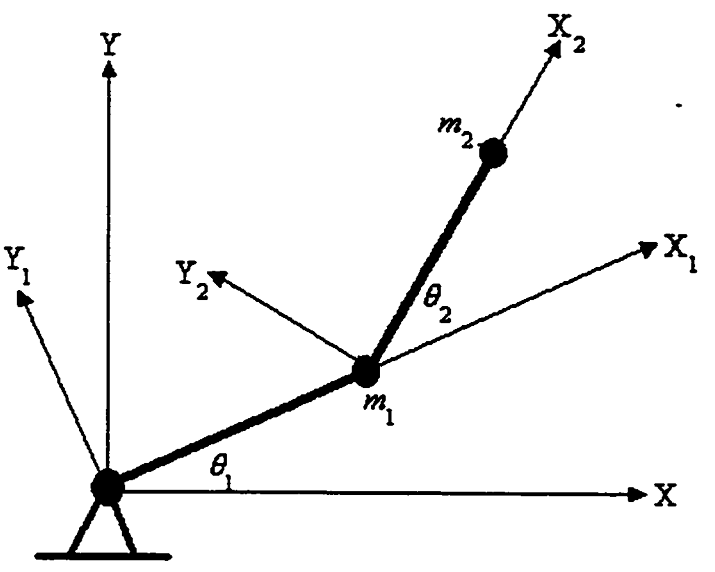


Figure 4.2: *Model of Two-link Robotic Manipulator*

The relationship between the end point and the rotary angles of joints are

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

Given joint displacements, the position of the end point of robot arm can be determined.

4.3.2. Manipulator Inverse Kinematics

Consider two-link robotic manipulator shown in Figure 4.2.

Given the position of the end effector of robot arm, the joint variables that achieve the specified position can be found using the following inverse kinematics:

$$\begin{aligned} \theta_1 &= \alpha_1 + \beta_1 \\ &= \sin^{-1}\frac{y}{\sqrt{x^2+y^2}} + \cos^{-1}\frac{L_1^2 + x^2 + y^2 - L_2^2}{2L_1\sqrt{x^2+y^2}} \end{aligned}$$

$$\theta_2 = \sin^{-1}\frac{y\cos\theta_1 - x\sin\theta_1}{L_2}$$

if $\sqrt{x^2+y^2} < L_1^2 + L_2^2$, then, $\theta_2 = -\pi + |\theta_2|$

4.4 Dynamics

Consider two-link robotic manipulator shown in Figure 4.2.

The structural parameters of robotic manipulator are:

Length: $L_1 = 1\,\mathrm{m}$, $L_2 = 1\,\mathrm{m}$.

Mass: $M_1 = 2\,\mathrm{kg}$, $M_2 = 1\,\mathrm{kg}$.

Mass of joint motor: $m_1 = 2\,\mathrm{kg}$.

Mass of payload: $m_2 = 1\,\mathrm{kg}$.

Let $[\,x_1, 0\,]$ be the vector of the arbitrary point in link one and $[\,x_2, 0]$ be the arbitrary point in link two. $\mathbf{r}_1$ and $\mathbf{r}_2$ are vectors of point x_1 and x_2 in the inertia coordinate reference OXY, respectively. We have

$$\mathbf{r}_1 = \mathbf{A}_1\begin{bmatrix} x_1 \\ 0 \end{bmatrix} \text{ and } \mathbf{r}_2 = \mathbf{A}_1\begin{bmatrix} L_1 \\ 0 \end{bmatrix} + \mathbf{A}_2\begin{bmatrix} x_2 \\ 0 \end{bmatrix}$$

$\mathbf{A}_1$ and $\mathbf{A}_2$ are rotary transformation matrices from body coordinate references $\mathrm{O_1X_1Y_1}$ and $\mathrm{O_2X_2Y_2}$ to coordinate reference OXY.

The kinetic energy of the robotic manipulators is as the follow:

$$\begin{aligned} \mathrm{T} &= \frac{1}{2}\int_0^{L_1} \rho_1 \dot{\mathbf{r}}_1^{\mathrm{T}} \dot{\mathbf{r}}_1 dx_1 + \frac{1}{2} m_1 \dot{\mathbf{r}}_1^{\mathrm{T}} \dot{\mathbf{r}}_1 \Big|_{x_1 = L_1} + \frac{1}{2}\int_0^{L_{21}} \rho_2 \dot{\mathbf{r}}_2^{\mathrm{T}} \dot{\mathbf{r}}_2 dx_2 + \frac{1}{2} m_2 \dot{\mathbf{r}}_2^{\mathrm{T}} \dot{\mathbf{r}}_2 \Big|_{x_2 = L_2} \\ &= \frac{1}{2}(\frac{1}{3}M_1 + m_1 + M_2 + m_2)L_1^2\dot{\theta}_1^2 + \frac{1}{2}(\frac{1}{3}M_2 + m_2)L_2^2(\dot{\theta}_1 + \dot{\theta}_2)^2 \\ &\quad + (\frac{1}{2}M_2 + m_2)L_1L_2\cos\theta_2(\dot{\theta}_1 + \dot{\theta}_2)\dot{\theta}_1 \end{aligned}$$

The gravity potential energy is:

$$G = \int_0^{L_1} \rho_1[0 \quad 1]\mathbf{r}_1 g dx_1 + \int_0^{L_2} \rho_2[0 \quad 1]\mathbf{r}_2 g dx_2$$
$$= (\frac{1}{2}M_1 + m_1 + M_2 + m_2)L_1 g \sin\theta_1 + (\frac{1}{2}M_2 + m_2)L_2 g \sin(\theta_1 + \theta_2)$$

Applying Hamilton Principle,

$$\mathbf{M}\ddot{\boldsymbol{\theta}} + \mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta}) = \boldsymbol{\tau} \tag{4.4.1}$$

where $\boldsymbol{\theta} = [\theta_1 \quad \theta_2]^T$ and $\boldsymbol{\tau} = [\tau_1 \quad \tau_2]^T$. **M** is the inertia matrix, which is symmetric positive defined. **C** is the vector that describes the centrifugal and Goriolis forces. **G** is the gravity vector. Let

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12}^T \\ m_{12} & m_{22} \end{bmatrix}$$
$$= \begin{bmatrix} \begin{matrix}(\frac{1}{3}M_1 + m_1 + M_2 + m_2)L_1^2 + (\frac{1}{3}M_2 + m_2)L_2^2 \\ + (M_2 + 2m_2)L_1 L_2 \cos\theta_2\end{matrix} & (\frac{1}{3}M_2 + m_2)L_2^2 + (\frac{1}{2}M_2 + m_2)L_1 L_2 \cos\theta_2 \\ (\frac{1}{3}M_2 + m_2)L_2^2 + (\frac{1}{2}M_2 + m_2)L_1 L_2 \cos\theta_2 & (\frac{1}{3}M_2 + m_2)L_2^2 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} c_1(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \\ c_2(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \end{bmatrix} = \begin{bmatrix} -(\frac{1}{2}M_2 + m_2)L_1 L_2 \sin\theta_2 (2\dot{\theta}_1 + \dot{\theta}_2)\dot{\theta}_1 \\ -(\frac{1}{2}M_2 + m_2)L_1 L_2 \sin\theta_2 \dot{\theta}_1^2 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} g_1(\boldsymbol{\theta}) \\ g_2(\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} (\frac{1}{2}M_1 + m_1 + M_2 + m_2)L_1 g \cos\theta_1 + (\frac{1}{2}M_2 + m_2)L_2 g \cos(\theta_1 + \theta_2) \\ (\frac{1}{2}M_2 + m_2)L_2 g \cos(\theta_1 + \theta_2) \end{bmatrix}$$

We have

$$\begin{bmatrix} m_{11} & m_{12}^T \\ m_{12} & m_{22} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} c_1(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \\ c_2(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \end{bmatrix} + \begin{bmatrix} g_1(\boldsymbol{\theta}) \\ g_2(\boldsymbol{\theta}) \end{bmatrix} = \boldsymbol{\tau}(t) \tag{4.4.2}$$

Choose the position of end point in the Cartesian coordinate reference as the observable output, that is

$$\mathbf{Y} = [x \quad y]^{\mathrm{T}} \tag{4.4.3}$$

The relationship between the end point and the rotary angles of the joints are

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \tag{4.4.4}$$

The Jacob matrix is

$$\mathbf{J} = \begin{bmatrix} -L_1 \sin\theta_1 - L_2 \sin(\theta_1 + \theta_2) & -L_2 \sin(\theta_1 + \theta_2) \\ L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) & L_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \tag{4.4.5}$$

4.5 Control Algorithms

4.5.1. Introduction

In [54], the event-based controller design was first introduced. The basic idea of the event-based planning and control theory is to introduce a new motion reference variable different from time, but directly related to the sensory measurement of the system. Instead of time, the desired system output is parameterized by the new motion reference variable. The motion reference variable is designed to efficiently carry the sensory information needed for the planner to adjust or modify the original plan to form a desired output. As a result, for any given time instant, the action plan is a function of the system output. This creates a mechanism to adjust and modify the plan based on the sensory measurement.

Figure 4.3 exhibits the underlying structure of robot control system. The function of "Motion Reference" is to compute the current value of the motion reference variable s on line based on the system output measurement. The planner then gives a desired value to the system according to the motion reference s. It can be seen that the planning becomes an investigation/decision component in the sense of feedback. Therefore, the event-based planning and control scheme has the ability to deal with unexpected or uncertain events.

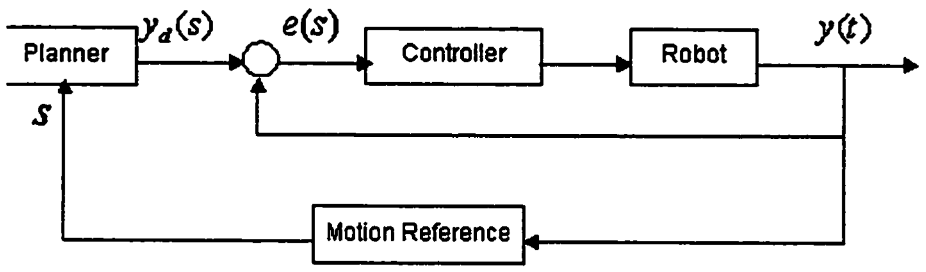


Figure 4.3: *Control System Architecture*

Since the new action reference is not directly related to time, it is independent of time delay. As a result the communication time delay will have no effect on the robotic operation. In addition, the value of the motion reference variable is calculated at the same rate as the feedback control. That is, the original plan is adjusted and modified at a very high rate. A trajectory is submitted to the planner for execution by the robot. Execution proceeds normally, except when obstacle appears. If during a trajectory tracking motion, an unexpected obstacle stopped the robot motion, because path-based motion reference is computed based on robot position measurement, the path-based motion reference stopped increasing as well. As a result, the desired position and velocity of the robot remain constant. Therefore the errors were unchanged. As long as the obstacle can stand a small force generated by the robot velocity error, the robot does not move. Once the obstacle is removed, the velocity error starts the robot motion again. The robot itself completes the rest of planned motion without replanning.

4.5.2. Motion Trajectory Planning

In a robot trajectory-tracking problem, the major event is the tracking of a given path. The most natural reference for the motion is the distance traveled, s, along a given path, S. If s is chosen as the reference, the motion along a given geometric path can be written as

$$\frac{ds}{dt} = v \text{ and } \frac{dv}{dt} = a \tag{4.5.1}$$

where v and a are velocity and acceleration, respectively, along the given path S.

Based on the results of kinematics and dynamic workspace analysis [24, 25], the trajectory constraints could be states as

$$\begin{aligned} &|v| \le v_m, && \text{velocity constraint} \\ &|a| \le a_m, && \text{acceleration constraint} \\ &\left|\frac{da}{dt}\right| \le k, && \text{constraint for jerk - free motion} \end{aligned} \tag{4.5.2}$$

During a motion the arc length s is a function of t. Thus v and a can be described as a function of s, instead of t, that is, $v = V(s)$, $a = A(s)$. Thus

$$a = \frac{dv}{dt} = \frac{dv}{ds}\frac{ds}{dt} = v\frac{dv}{ds} = \frac{1}{2}\frac{d(v^2)}{ds}$$

Let $w = v^2$, we have the trajectory planning

$$\frac{dw}{ds} = 2a \text{ and } \frac{da}{ds} = u \tag{4.5.3}$$

The corresponding constraints are

$$\begin{aligned} &|w| \le w_m, && \text{velocity constraint} \\ &|a| \le a_m, && \text{acceleration constraint} \\ &|u| \le u_m, && \text{jerk - free constraint} \end{aligned} \tag{4.5.4}$$

Phase space (velocity versus position) is used to describe motion trajectories. The trajectory phase space planning, that is, the event-based trajectory planning is to find the velocity profile as a function of path or position, i.e., $v = V(s)$, subject to the kinematics and dynamic constrains.

For any given initial condition s_0, s_f, v_0, and v_f, the trajectory plan is not unique. Using various criteria, different event-based optimal plans could be obtained. The time optimization technique can be applied to obtain the event-based optimal trajectory.

The time T to complete a motion is

$$T = \int_{t_0}^{t_f} dt = \int_{s_0}^{s_f} \frac{1}{v} ds = \int_{s_0}^{s_f} \frac{1}{\sqrt{w}} ds$$

Define $x_1 = w$, $x_2 = a$, $c_1 = x_1 - w_m$, $c_2 = -x_1 - w_m$, $c_3 = x_2 - a_m$, $c_4 = -x_2 - a_m$,

and $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\mathbf{F} = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\frac{d\mathbf{X}}{ds} = \mathbf{X}'$, then

$$\mathbf{X}' = \mathbf{F}\mathbf{X} + \mathbf{B}u \tag{4.5.5}$$

with constraints $\mathbf{C} \le 0$, where $\mathbf{C} = [c_1 \quad c_2 \quad c_3 \quad c_4]^T$.

Now the preceding motion planning problem becomes an optimal control problem. It can be states as follows:

$$\underset{u}{Min}\, J, \; J = \int_{s_0}^{s_f} x_1^{-\frac{1}{2}} ds \tag{4.5.6}$$

$$\text{subject to} \begin{cases} \mathbf{X}' = \mathbf{F}\mathbf{X} + \mathbf{B}u \\ \mathbf{C} \le 0, \quad |u| \le u_m \end{cases} \tag{4.5.7}$$

with $\mathbf{X}(0) = 0$, $\mathbf{X}(s_f) = 0$.

The Pontryagin maximum principle [6] is used to solve this problem. A closed-form solution that is essential for real-time implementation is obtained. After simplification, we get time-optimal trajectory. The time-optimal solution is obtained as

$$u=\begin{cases} u_m & s_0 \le s \le s_1 \\ 0 & s_1 < s \le s_2 \\ -u_m & s_2 < s \le s_f \end{cases} \tag{4.5.8}$$

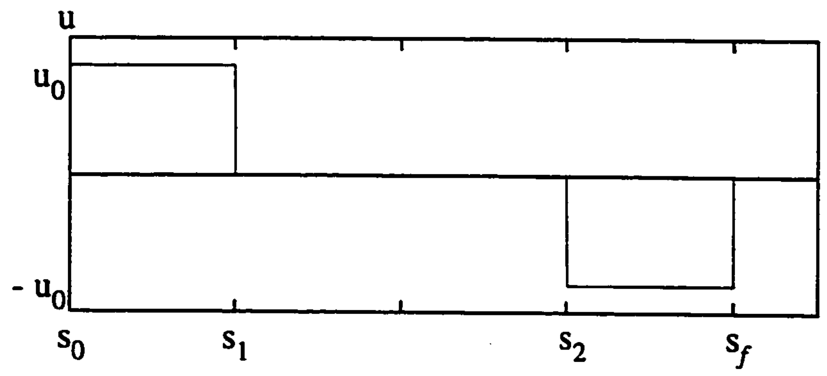
$$a=\begin{cases} u_m(s-s_0) & s_0 \le s \le s_1 \\ \dfrac{-2a_m}{s_f - s_0 - 2a_m/u_m}(s-\dfrac{s_0+s_f}{2}) & s_1 < s \le s_2 \\ u_m(s-s_f) & s_2 < s \le s_f \end{cases} \tag{4.5.9}$$

$$w=\begin{cases} u_m(s-s_0)^2 & s_0 \le s \le s_1 \\ \dfrac{-2a_m}{s_f - s_0 - 2a_m/u_m}(s-s_0-\dfrac{a_m}{u_m})+\dfrac{a_m^2}{u_m} & s_1 < s \le s_2 \\ u_m(s-s_f)^2 & s_2 < s \le s_f \end{cases} \tag{4.5.10}$$

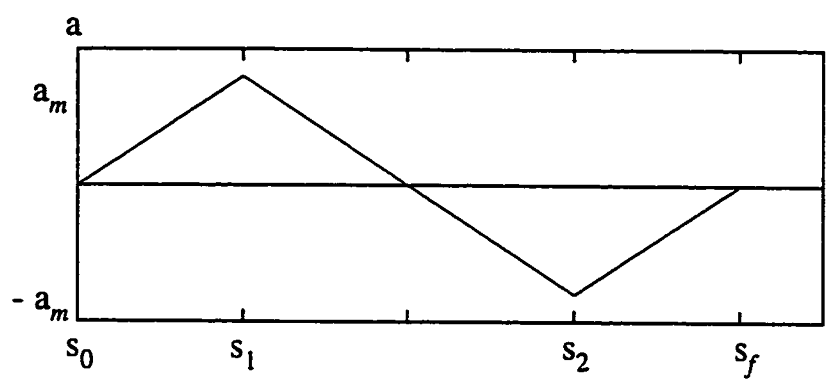
where

$$s_1 = s_0 + \frac{a_m}{u_m},\ s_2 = s_f - \frac{a_m}{u_m}$$

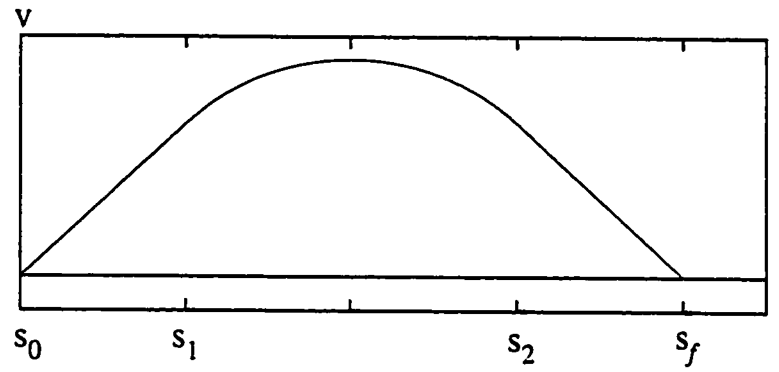
Velocity and acceleration profiles of the time-optimal motion plan are shown in Figure 4.4.



(a)



(b)



(c)

Figure 4.4: *Velocity and Acceleration Profiles of the Time-optimal Motion Plan*

The desired velocity and acceleration are zero at the initial and end points. In order to start the motion, some initial error has to be introduced.

4.5.3. Cartesian Space Decomposition of Event-Based Plans

In order to get the task space plan:

$$\begin{cases} \dot{x} = V_x(s) \\ \dot{y} = V_y(s) \end{cases} \tag{4.5.11}$$

and

$$\begin{cases} \ddot{x} = A_x(s) \\ \ddot{y} = A_y(s) \end{cases} \tag{4.5.12}$$

$v = V(s)$ and $a = A(s)$ will be decomposed in the Cartesian space according to the given path.

Any geometric path given in the task space can be approximated by a combination of several straight lines and circular arcs. So it is necessary to find the decompositions for the straight line and the circular path segment.

4.5.3.1. Straight Line

For the case of straight line, shown in Figure 4.5(a), let (m, n) be the direction cosine, the decomposition of velocity and the acceleration are as follows

$$\begin{cases} \dot{x} = mV(s) \\ \dot{y} = nV(s) \end{cases} \text{and} \begin{cases} \ddot{x} = mA(s) \\ \ddot{y} = nA(s) \end{cases} \tag{4.5.13}$$

where $m = \cos\alpha$ and $n = \sin\alpha$.

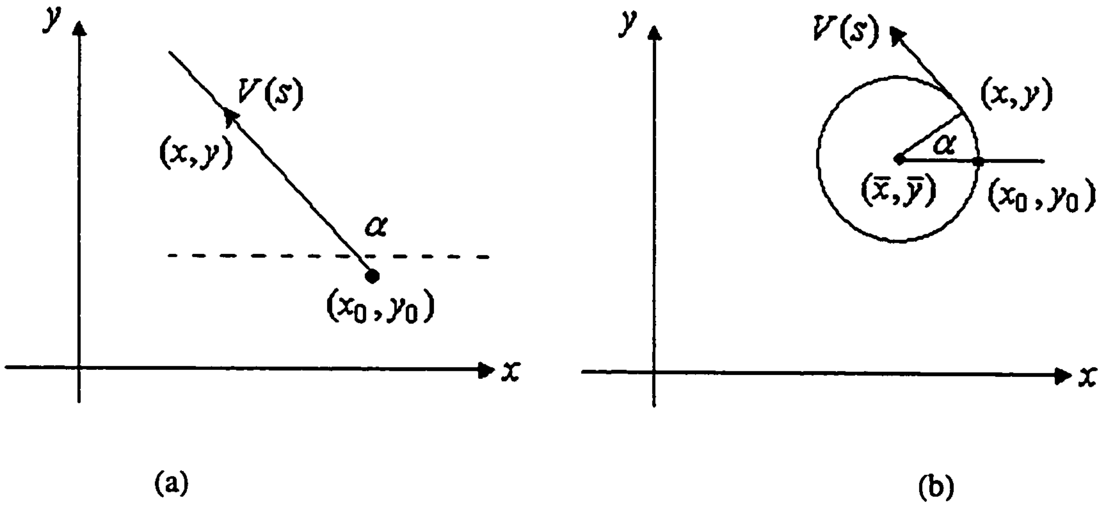


Figure 4.5: *Decomposition of Straight Line and Circular Path*

4.5.3.2. Circular Path

Assume that the center of the circle is $(\bar{x}, \bar{y})$ and the radius is r, the equation of the circle is

$$\begin{cases} x = \bar{x} + r\cos(s/r) \\ y = \bar{y} + r\sin(s/r) \end{cases}$$

where s is the arc length from the initial point (x_0, y_0) to point (x, y) (Figure 4.5(b)). Thus the decomposition of velocity and the acceleration are

$$\begin{cases} \dot{x} = -\sin(s/r)\dfrac{ds}{dt} = -\dfrac{y-\bar{y}}{r}V(s) \\ \dot{y} = \cos(s/r)\dfrac{ds}{dt} = \dfrac{x-\bar{x}}{r}V(s) \end{cases} \tag{4.5.14}$$

$$\begin{cases} \ddot{x} = -\dfrac{\dot{y}}{r}V(s) - \dfrac{y-\bar{y}}{r}\dot{V}(s) = -\dfrac{x-\bar{x}}{r^2}(V(s))^2 - \dfrac{y-\bar{y}}{r}A(s) \\ \ddot{y} = \dfrac{\dot{x}}{r}V(s) + \dfrac{x-\bar{x}}{r}\dot{V}(s) = -\dfrac{y-\bar{y}}{r^2}(V(s))^2 + \dfrac{x-\bar{x}}{r}A(s) \end{cases} \tag{4.5.15}$$

4.5.4. Control Law

A new event-based error definition and computation scheme is introduced and combined with nonlinear feedback control law, which linearizes and decouples the controls in task space.

By using the theory of linearization state feedback, there exist a diffeomorphic state transformation $T(\boldsymbol{\theta})$ and a nonlinear feedback law $\boldsymbol{\tau}(t) = \boldsymbol{\alpha} + \boldsymbol{\beta}\mathbf{v}$ that linearizes and decouples the robot dynamics. The diffeomorphic state transformation $T(\boldsymbol{\theta})$ is given by

$$\mathbf{Y} = T(\boldsymbol{\theta}) = [x \quad y]^{\mathrm{T}} \tag{4.5.16}$$

and the nonlinear feedback law is

$$\boldsymbol{\tau}(t) = \boldsymbol{\alpha}(x) + \boldsymbol{\beta}(x)\mathbf{v} \tag{4.5.17}$$

with

$$\boldsymbol{\alpha} = -\mathbf{M}(\boldsymbol{\theta})\mathbf{J}^{-1}[\dot{\mathbf{J}}\dot{\boldsymbol{\theta}} - \mathbf{J}\mathbf{M}^{-1}(\boldsymbol{\theta})(\mathbf{C}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta}))]$$

$$\boldsymbol{\beta} = \mathbf{M}(\boldsymbol{\theta})\mathbf{J}^{-1}$$

Let

$$\mathbf{v} = \ddot{\mathbf{Y}}^d(t) + K_v\dot{\mathbf{e}}(t) + K_p\mathbf{e}(t) \tag{4.5.18}$$

Thus

$$\tau = \mathbf{M}(\theta)\mathbf{J}^{-1}[\ddot{\mathbf{Y}}^d(t) + K_v\dot{\mathbf{e}}(t) + K_p\mathbf{e}(t) - \dot{\mathbf{J}}\dot{\theta}] + \mathbf{C}(\theta,\dot{\theta}) + \mathbf{G}(\theta) \tag{4.5.19}$$

where

$$\mathbf{e}(t) = \mathbf{Y}^d(t) - \mathbf{Y}(t) \text{ and } \dot{\mathbf{e}}(t) = \dot{\mathbf{Y}}^d(t) - \dot{\mathbf{Y}}(t)$$

$\mathbf{Y}^d(t)$ is the desired end point trajectory vector and $\mathbf{Y}(t)$ is the actual position of the end point. K_v and K_p are the Gain matrixes.

For a time-based plan, the reference base of input and measurement is time t. For any time instance t, a measurement $\mathbf{Y}(t)$ and $\dot{\mathbf{Y}}(t)$; a desired input $\mathbf{Y}^d(t)$ and $\dot{\mathbf{Y}}^d(t)$; and errors $\mathbf{e}(t)$ and $\dot{\mathbf{e}}(t)$ can be obtained. However, for an event-based plan, the time t is no longer a reference base. The input of the system is parameterized by the event-based motion reference s. According to the new motion reference s, the error $\mathbf{e}(t)$ and $\dot{\mathbf{e}}(t)$ must be redefined in order to get an event-based control law.

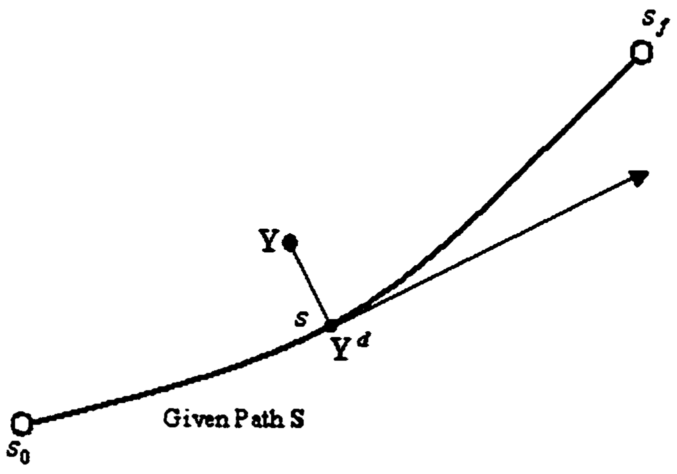


Figure 4.6: *The Event-based Error Definition*

In essence, for a digital sampled data control system we could determine the corresponding $\mathbf{Y}^d(t)$ and $\dot{\mathbf{Y}}^d(t)$ for each sampling time by computing the desired velocity and then integrating the velocity to determine the corresponding desired position. From Figure 4.6, $\mathbf{Y} = [x \quad y]^T$ is a measurement, and the point s corresponds to a point in the given path that has the minimum distance from point $\mathbf{Y}$ to the given path, that is the orthogonal projection of $\mathbf{Y}$. The Cartesian coordinate of point s is considered as a desired position $\mathbf{Y}^d(s)$.

Based on path s, a desired velocity $\dot{\mathbf{Y}}^d(s)$ and the desired acceleration $\ddot{\mathbf{Y}}^d(s)$ can be obtained from the event-based plan. The new error definitions are

$$\mathrm{e}(s) = \mathbf{Y}^d(s) - \mathbf{Y}(s) \text{ and } \dot{\mathrm{e}}(s) = \dot{\mathbf{Y}}^d(s) - \dot{\mathbf{Y}}(s)$$

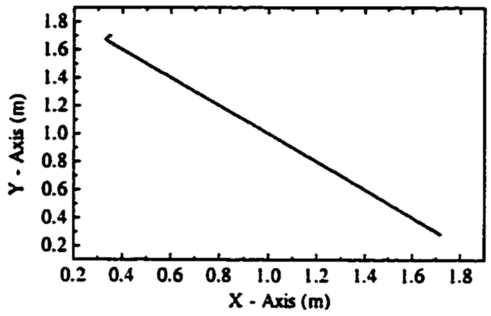
It can be seen that the new error definitions minimize the position error and make all errors independent of time.

4.6 Simulation Results

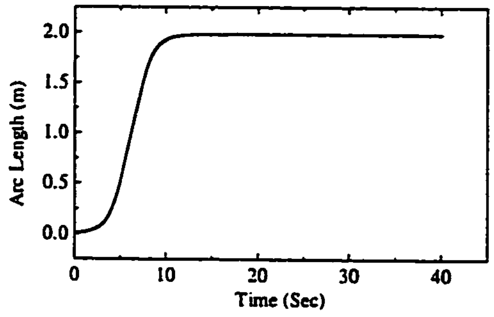
Trajectory tracking of event-based planning and control scheme with time-optimal motion plan has been tested on a 2-DOF robot manipulator shown in Figure 4.2. Simulation results are shown in this section.

4.6.1. Case 1: Straight Line

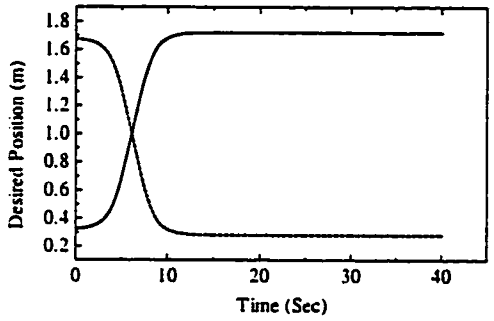
Figure 4.7 shows the performance plots for straight-line tracking using time optimal event-based plan. The initial point is $(0.32 \quad 1.68)$ and the final point is $(1.72 \quad 0.28)$. It is tilted at 135°. The desired trajectory is 1.98m long. $u_m = 0.8 \cdot 1/s^2$ and $a_m = 0.16 m/s^2$. The gains are $k_p = 300$ and $k_v = 150$.



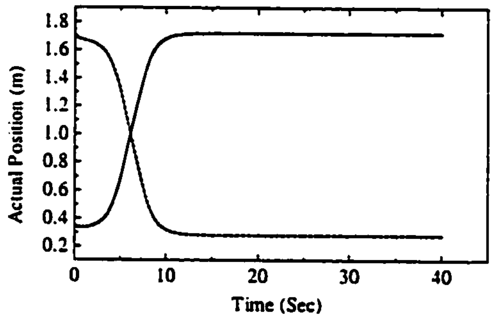
(a) The actual and desired trajectory Solid line: desired, dash line: actual



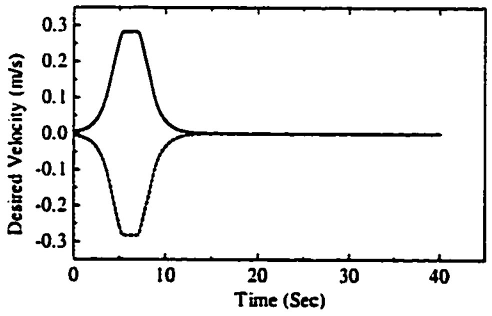
(b) *The distance traveled, s along the given path*



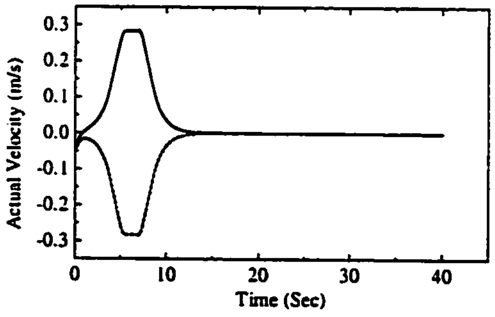
(c) *Desired XY coordinate position Solid line: X-Axis, dash line: Y-Axis*



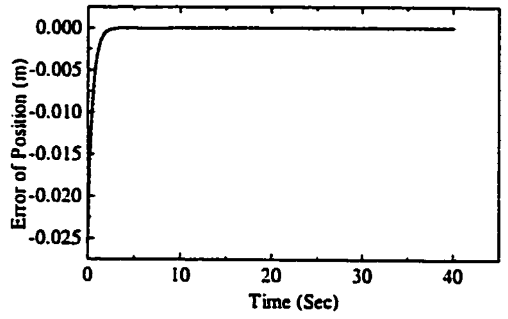
(d) *Actual XY coordinate position*

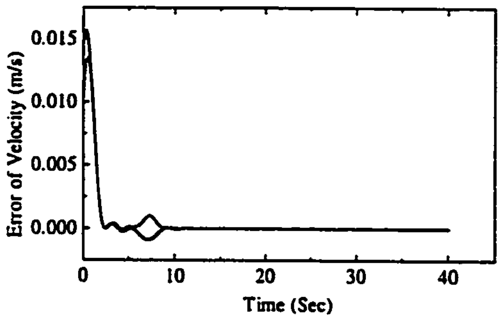


(e) *Desired XY coordinate velocity Solid line: X-Axis, dash line: Y-Axis*



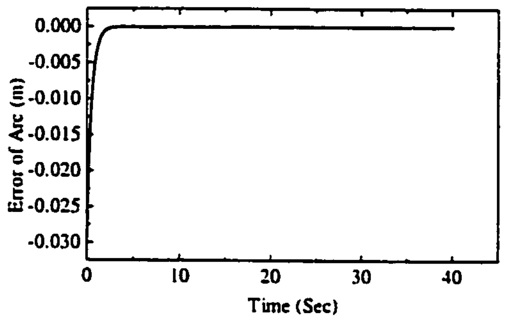
(f) *Actual XY coordinate velocity*

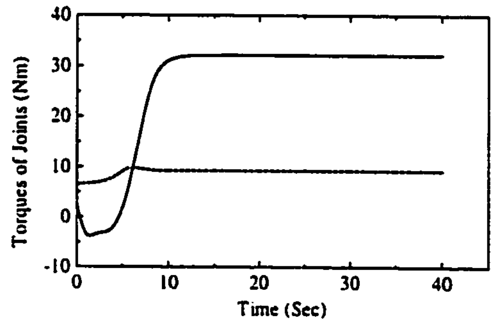




(g) *Error of XY coordinate position*
Solid line: X-Axis, dash line: Y-Axis

(h) *Error of XY coordinate velocity*





(i) *Error of s and* s_d
s_d, s: *desired and actual position of robot*

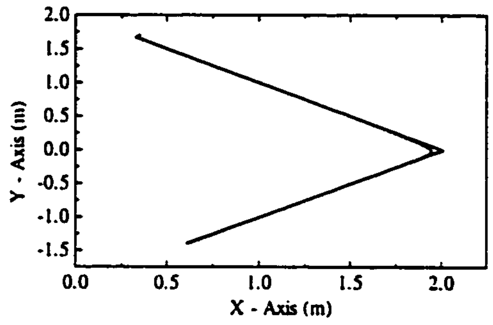
(j) *Driven torques*
Solid line: Joint 1, dash line: Joint 2

Figure 4.7: *Straight Line Path Tracking*

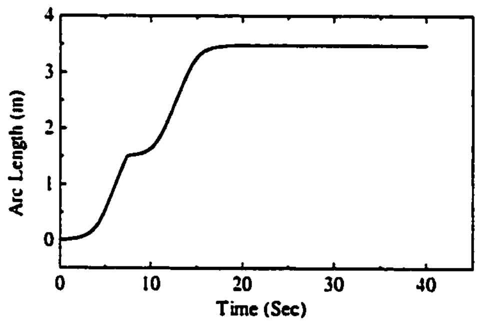
According to the simulation results, the maximum tracking error is less than 0.025m, see Figure 4.7(g) and (i). The velocity errors are less than 0.015m/s, see Figure 4.7(h). Both of two errors tend to zero as the increasing of time, that is, the control law can guarantee the asymptotic stability of the trajectory tracking. The reason for obtaining asymptotic stability is that the time t is no longer a motion reference base. The distance-traveled plot gives the profile of s versus time t, see Figure 4.7(b). It is seen that s is a monotone increasing function of t.

4.6.2. Case 2: Broken Line

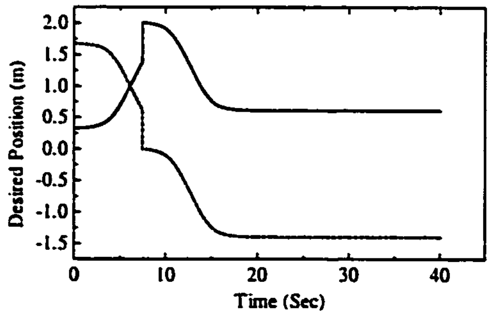
In this case, the simulations for broken-line are given (Figure 4.8). The initial point is $(0.32 \quad 1.68)$ and tilted at 135°. When it moves to the point $(2.0 \quad 0.0)$ along a straight line, it turns left with 90° and tilted at 225°. Its final point is $(0.6 \quad -1.4)$. The length of the first line segment is 1.5m and the second one is 1.98m. $u_m = 0.8 \cdot 1/s^2$ and $a_m = 0.16 m/s^2$. The control gains are the same as Case 1. From Figure 4.8(g) and (i), we can see the maximum error at the turning points. Although it may reach 0.25m, it tends to zero rapidly. The whole trajectory tracking is stable.



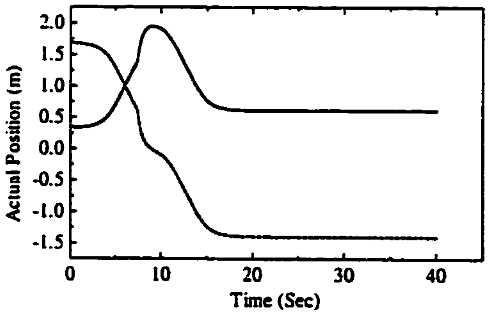
(a) *The actual and desired trajectory Solid line: desired, dash line: actual*



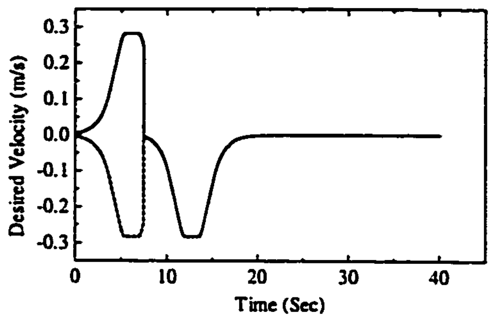
(b) *The distance traveled, s along the given path*



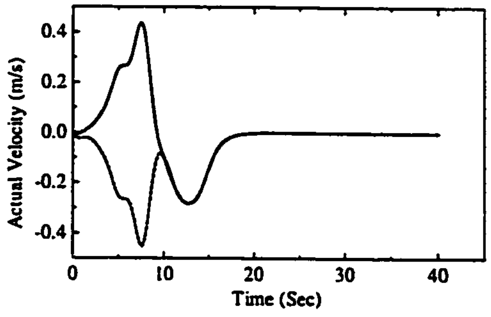
(c) *Desired XY coordinate position Solid line: X-Axis, dash line: Y-Axis*



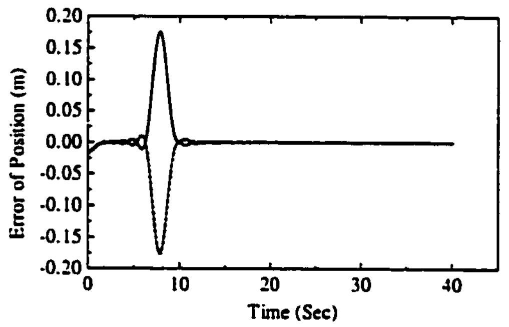
(d) *Actual XY coordinate position*



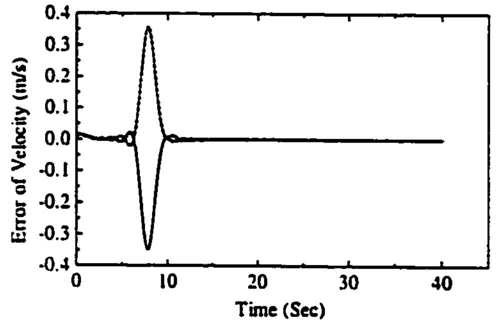
(e) *Desired XY coordinate velocity*
Solid line: X-Axis, dash line: Y-Axis



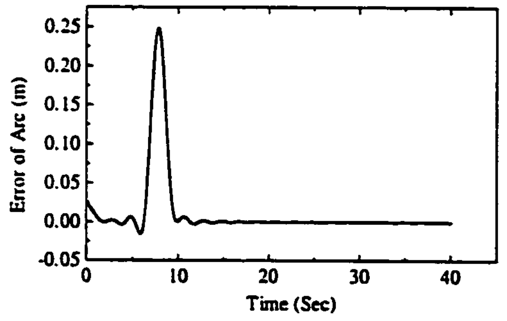
(f) *Actual XY coordinate velocity*



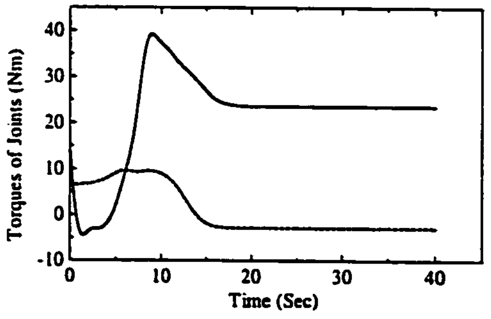
(g) *Error XY coordinate position*
Solid line: X-Axis, dash line: Y-Axis



(h) *Error of XY coordinate velocity*



(i) *Error of s and s_d*
s_d, s: desired and actual position of robot

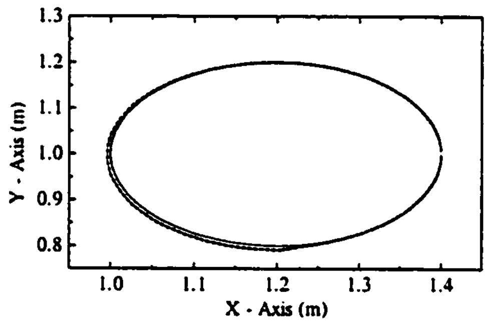


(j) *Driven torques*
Solid line: Joint 1, dash line: Joint 2

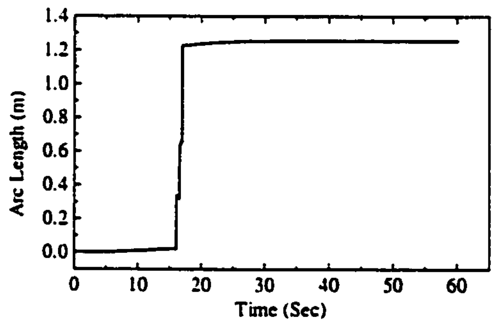
Figure 4.8: *Broken Line Path Tracking*

4.6.3. Case 3: Circle

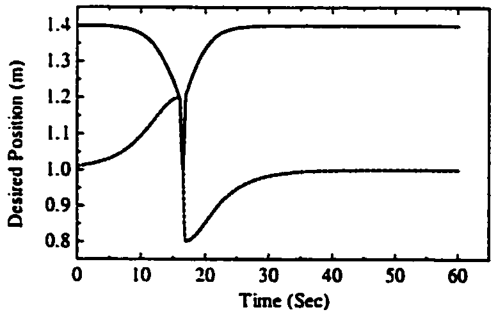
In Case 3, the simulation results of the circle-tracking plan are given (Figure 4.9). The radius of the circle is 0.2m. The initial point is $(1.4\ \ 1.0)$. It is tilted at 90°. $u_m = 0.8 \cdot 1/s^2$ and $a_m = 4.0 m/s^2$. From the simulation results, the maximum position error is less than 0.008m and the maximum velocity error is less than 0.008m/s. Time period when the maximum error occurs is very short.



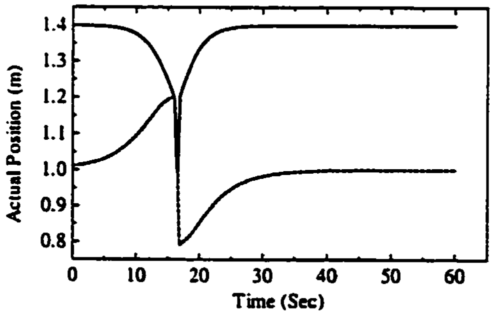
(a) *The actual and desired trajectory Solid line: desired, dash line: actual*



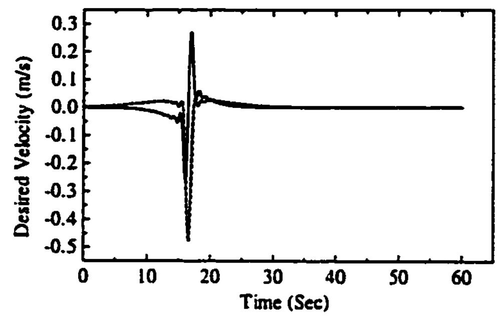
(b) *The distance traveled, s along the given path*



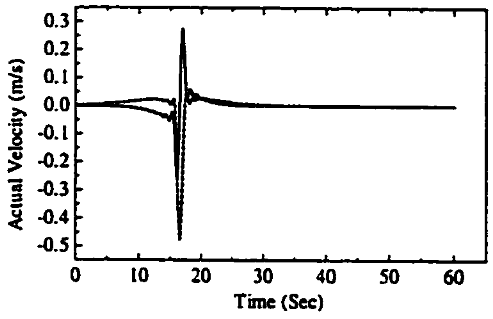
(c) *Desired XY coordinate position Solid line: X-Axis, dash line: Y-Axis*



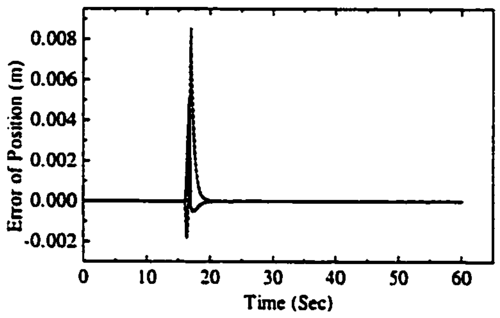
(d) *Actual XY coordinate position*



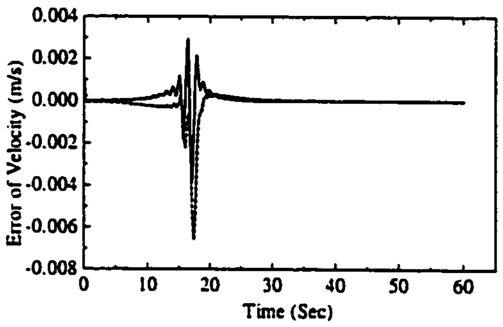
(e) *Desired XY coordinate velocity*
Solid line: X-Axis, dash line: Y-Axis



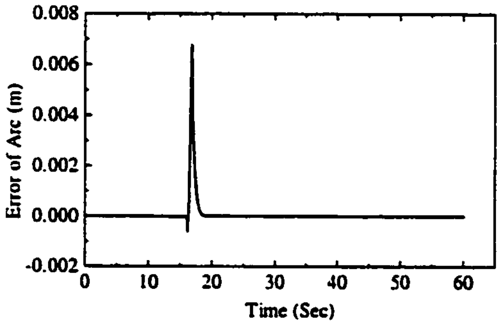
(f) *Actual XY coordinate velocity*



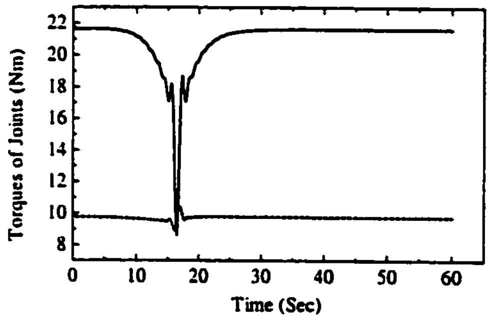
(g) *Error XY coordinate position*
Solid line: X-Axis, dash line: Y-Axis



(h) *Error of XY coordinate velocity*



(i) *Error of s and s_d*
s_d, s: desired and actual position of robot



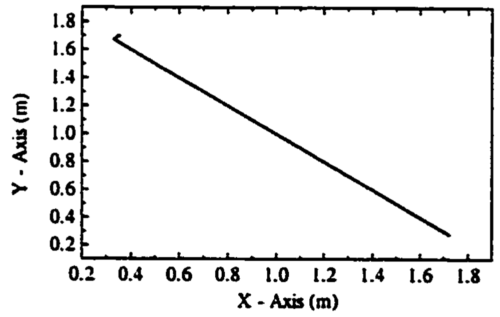
(j) *Driven torques*
Solid line: Joint 1, dash line: Joint 2

Figure 4.9: *Circle Path Tracking*

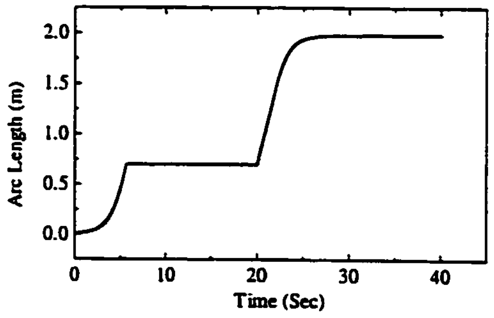
4.6.4. Case 4: Straight Line with an Unexpected Obstacle

In this case, during a straight-line tracking motion, an unexpected obstacle stopped the robot motion. Figure 4.10 shows simulation results. The trajectory planning is the same as Case 1.

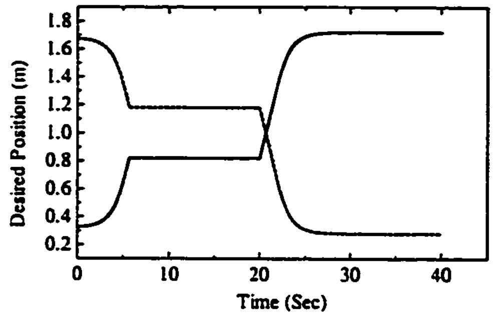
From Figure 4.10(b), when an unexpected obstacle occurs, the path-based motion reference, the arc length *s*, stops increasing and remains the same value. From Figure 4.10(c) and (d), the desired and actual XY coordinate position remain constant. So errors will remain constant. If the time-based plan were implemented, the errors would keep increasing and eventually result in instability. From Figure 4.10(j), during this time, joint torques remains constant. As long as the obstacle can stand this small torque, the robot does not move. From Figure 4.10(g) and (i), errors remain constant. From Figure 4.10(c) and (d), when the obstacle is removed, the robot completed the rest of the planned motion without replanning. This demonstrates that the event-based control law can provide the robot with the ability of handling the unexpected event. So it can be used in the Internet-based teleoperation to overcome unexpected time-delay.

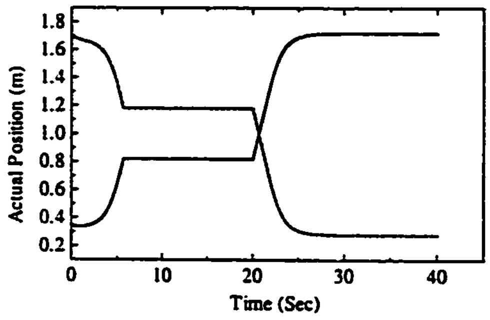


(a) *The actual and desired trajectory Solid line: desired, dash line: actual*



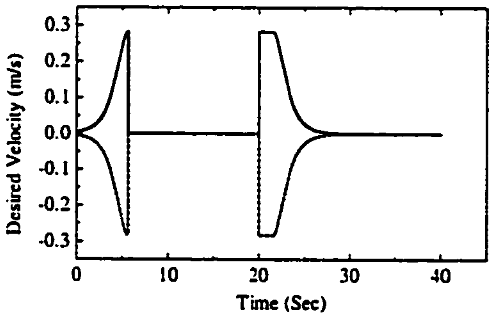
(b) *The distance traveled, s along the given path*

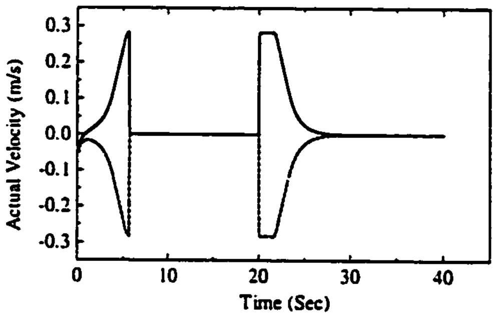




(c) *Desired XY coordinate position*
Solid line: X-Axis, dash line: Y-Axis

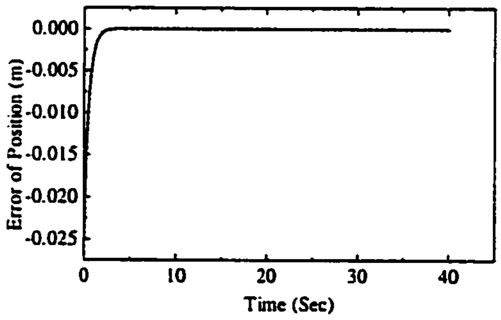
(d) *Actual XY coordinate position*

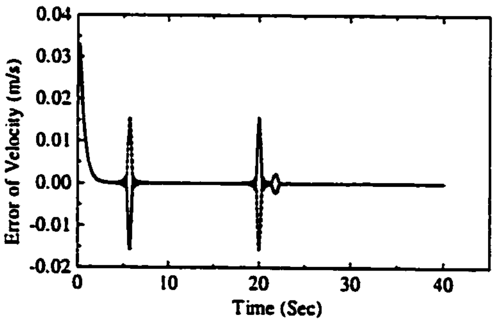




(e) *Desired XY coordinate velocity*
Solid line: X-Axis, dash line: Y-Axis

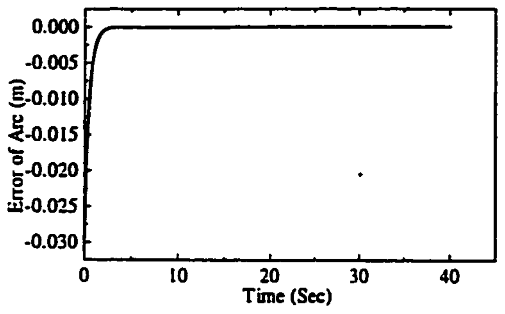
(f) *Actual XY coordinate velocity*



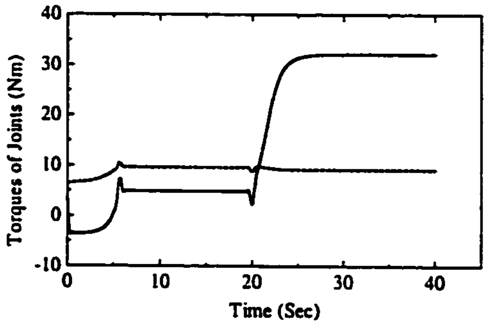


(g) *Error XY coordinate position*
Solid line: X-Axis, dash line: Y-Axis

(h) *Error of XY coordinate velocity*



(i) *Error of s and s_d*
s_d, s: desired and actual position of robot



(j) *Driven torques*
Solid line: Joint 1, dash line: Joint 2

Figure 4.10: *Straight Line Path Tracking with Unexpected Obstacle*

Chapter 5

Conclusions

5.1 Summary

We performed the research on the teleoperation of a robot system from a remote computer over the Internet communication networks.

Our research has demonstrated that telerobotics can be feasible with much less communication bandwidth using advanced techniques to compensate the disadvantages and problems associated with the Internet infrastructure. This mainly relies on the event-based control system concepts. The robot's local semi-autonomous intelligent controller performs a remote network user's commands.

The system developed can be used for teleoperation experiments via the Internet to investigate various issues related to the Internet-based teleoperations. Having experienced certain bandwidth restrictions in our studies, we can propose new classes of applications using telerobotics.

5.2 Future research

Some further work may concentrate on real-time image capture and the development of more sophisticated vision systems to extract the relevant data from the environment and provide it to the operator. Because the ideal of this research area depends on the virtual immersion of a human operator in the remote workcell, haptic and force reflective data could be presented to augment these means.

Advanced control policies are still an active research area that improve promise to overall system performance.

Bibliography

1. R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay", *IEEE Transaction on Automatic Control* 34(5), pp. 494-501.
2. P. G. Backes, K. S. Tso, and G. K. Tharp (1998), "Mars Pathfinder Mission Internet-Based Operations using WITS", *Proceedings of the 1998 IEEE International Conference on Robotics & Automation,* Leuven, Belgium, pp. 284-291.
3. A. K. Bejczy, "Challenges of Human-Robot Communication In Telerobotics", *IEEE International Workshop on Robot and Human Communication*, pp. 1-8.
4. A. K. Bejczy, G. Bekey, R. Taylor, and S. Rovetta (1994), "A research methodology for tele-surgery with time delays", *First International Symposium on Medical Robotics and Computer Assisted Surgery.*
5. K. Brady and T. J. Tarn (1998), "Internet-based remote teleoperation," *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, Leuven, Belgium, pp. 65-70.
6. A. E. Bryson, Jr. and Y. C. Ho (1975), "Applied Optimal Control: Optimization, Estimation and Control", John Wiley & Sons, New York.
7. S. D. Brierley, J. N. Chiasson, E. B. Lee and S. H. Zak, "On stability independent of delay for linear systems", *IEEE Transaction on Automatic Control 27(1),* pp. 253-254.

8. T. M. Chen and R. C. Luo (1997), "Remote Supervisory Control of Autonomous Mobile Robot Via World Wide Web", *ISIE*, pp. ss60-ss64.

9. L. Conway, A. V. Richard, and M. W. Walker (1990), "Teleautonomous systems: Projecting and coordinating intelligent action at a distance", *IEEE Transactions on Robotics and Automation*, 6(2), pp.146-158.

10. M. Culverhouse, C. Walnum, N. Howell and G. Perry, "Using Visual J++", *Que Corporation.*

11. J. Deep and P. Holfelder, "Developing CGI Applications with Perl", *Benchmark Productions, Inc.*

12. P. DePasquale, J. Lewis, and M. Stein, "A Java interface for asserting interactive telerobotic control," *Proceedings of the SPIE - The International Society for Optical Engineering Conference*, Vol. 3206, pp.159-69.

13. E. D. Elliott and R. Eagleson (1997), "Web-Based Tele-Operated Systems Using EAI", *IEEE*, pp. 749-754.

14. M. Gertz, D. Stewart, and P. Khosla (1994), "A human-machine interface for distributed virtual laboratories", *IEEE Robotics and Automation Magazine.*

15. K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley, (1995) "Desktop teleoperation via the World Wide Web," *Proceedings of IEEE International Conference on Robotics and Automation*, Part Vol.1, pp.654-9.

16. K. Goldberg, M. Mascha, S. Gentner, J. Rossman, Rothenberg, C. Sutter, and J. Wiegley (1994), "Beyond the Web: Excavating the real World Via Mosaic", *Second International WWW Conference.*

17. K. Hirai and Y. Satoh, "Stability of systems with variable time delay", *IEEE Transaction on Automatic Control* ac-**25**(3), pp. 552-554.

18. H. Hirukawa, T. Matsui, H. Onda, K. Takase, and Y. Ishiwata, "Prototypes of Teleoperation Systems via a Standard Protocol with a Standard Human Interface".

19. G. Hirzinger, B. Brunner, R. Koeppe, K. Landzettel, and J. Vogel (1997), "Teleoperating space robot – impact for the design of industrial robots", *ISIE*, pp.ss250-ss256.

20. T. T. Ho and H. Zhang (1999), "Internet-Based Tele-Manipulation", *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 1425-1430.

21. V. L. Kharitonov and A.B. Zhabko, "Robust stability of time-delay systems", *IEEE Transactions on Automatic Control* **39**(12), pp. 2388-2397.

22. W. S. Kim, B. Hannaford and A. K. Bejczy, "Force reflection and shared compliant control in operating telemanipulators with time delay", *IEEE Transaction on Automatic Control* **38**(2), pp. 176-185.

23. A. K. Kojima, K. Uchida and E. Shimemura, "Robust stabilization of uncertain time delay systems via combined internal-external approach", *IEEE Transaction on Automatic Control* **38**(2), pp. 373-378.

24. Z. Li, T. J. Tarn, and A. K. Bejczy, "Dynamic workspace analysis of multiple cooperating robot arms", *IEEE Transaction on Robotics and Automation*, Vol.7, No. 5.

25. Z. Li, T. J. Tarn, A. K. Bejczy, and B. K. Ghosh (1989), "Motion space analysis of an object handled by two robot arms" *Proceedings of the 28th IEEE Conference on Decision and Control.*

26. R. Luck, A. Ray, and Y. Halevi, "Observability under recurrent loss of data", *AIAA Journal of guidance, Control and Dynamics 15*, pp. 284-287.

27. O. Michel, P. Saucy, and F. Mondada (1997), "Khep On The Web: An Experimental Demonstrator in Telerobotics and Virtual Reality", *IEEE*, pp. 90-98

28. M. Mitsuishi, Y. Iizuka, H. Watanabe, H. Hashizume, and K. Fujiwara (1998), "Remote operation of a micro-surgical system", *IEEE*, pp.1013-1019.

29. F. Monteiro, P. Rocha, P. Menezes, A. Silva, and J. Dias, "Teleoperating a mobile robot. A solution based on JAVA language," *Proceedings of the IEEE International Symposium on Industrial Electronics*, Part Vol.1, pp. SS263-7.

30. B. Morgan, el al, *Microsoft Visual J++*, Sams.net Publishing.

31. T. Mori and H. Koname, "Stability of time-delay systems", *IEEE Transaction on Automatic Control* 34(4*)*, pp. 460-462.

32. K. Munawar and M. Uchiyama (1998), "Distributed Event-Based Control of Unifunctional Multiple Manipulator System", *Proceedings of the 1998 IEEE International Conference on Robotics & Automation*, pp. 1817-1822.

33. G. Niemeyer and J. E. Slotine, "Stable adaptive teleoperation", *IEEE journal of Oceanic Engineering* **16**(1), pp. 152-162.

34. R. Oboe, and P. Fiorini (1997), "Issues on Internet-based Teleoperation", *Proceedings of the Fifth IFAC Symposium on Robot Control.*

35. T. Sato, J. Ichikawa, M. Mitsuishi, and Y. Hatamura (1994), "A new micro-teleoperation systems empolying a hand-held force feedback pencil", *International conference on Robotics and Automation.*

36. K. Schilling and H. Roth, "Advanced Telematic Methods for the Teleoperations of Robots".

37. T. B. Sheridan (1993), "Space teleoperation through time delay review and prognosis", *IEEE Transactions on Robotics and Automation*, 9(5), pp. 592-606.

38. K. S. Siyan and J. L. Weaver, "Inside Java", *New Riders Publishing.*

39. M. R. Stein, " Printing on the World Wide Web: The Puma Paint Project".

40. T. Suzuki, T. Fujii, H. Asama, K. Yokota, H. Kaetsu, and I. Endo (1998), "A Multi-robot teleoperation system utilizing the Internet", *Advanced Robotics*, Vol. 11, No.8, pp. 781-797.

41. T. Suzuki, T. Fujii, K Yokota, H. Asama, H Kaetsu, and I. Endo (1996), "Teleoperation of Multiple Robots through the Internet", *IEEE International Workshop on Robot and Human Communication*, pp. 84-89.

42. J. Tan, N. Xi, and W. Kang (1999), "Non-time Based Tracking Controller for Mobile Robots", *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, pp.919-924.

43. T. J. Tarn, A. K. Bejczy, C. Guo, and N. Xi, "Intelligent Planning and Control for Telerobotic Operations", pp. 389-396.

44. T. J. Tarn, N. Xi, and A. K. Bejczy (1996), "Path-based Approach to Integrated Planning and Control for Robotics Systems", *Automatica*, Vol. 32, No. 12, pp. 1675-1687.

45. T. J. Tarn, A. K. Bejczy and N. Xi, "Intelligent Motion Planning and Control for Robot Arms", pp. 677-680.

46. T. J. Tarn, N. Xi and A. K. Bejczy (1992), "Motion Planning in Phase Space for Intelligent Robot Arm Control", *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems,* pp. 1507-1514.

47. K. Taylor, and J. Trevelyan, "Australia's telerobot on the Web," *26th International Symposium on Industrial Robots*, pp.39-44.

48. K. Taylor, B. Dalton, and J. Trevelyan (1999), "Web-based telerobotics", *Robotic* vol.17, pp. 49-57.

49. K. Taylor and B. Dalton, " Issues in Internet Telerobotics ", *FSR'97 International Conference on Field and Service Robotics.*

50. K. Taylor and J. Trevelyan (1995), "A Telerobot on the World Wide Web", *National Conference of the Australian Robot Association.*

51. K. Taylor and J. Trevelyan, "Paper Presented at 26th International Symposium On Industrial Robots", http://telerobot.mech.uwa.edu.au/robot/singapor.htm.

52. Y. Wakita, S. Hirai, K. Machida, K. Ogimoto, T. Ikoto, P. Backes, and S. Peter (1996), "Applications of Intelligent Monitoring for Super Long Distance Teleoperation", *IROS Proceedings*, Osaka, Japan.

53. N. Xi and T. J. Tarn (1998), "Planning and Control of Internet-Based Teleoperation", *Part of the SPIE Conference on Telemanipulator and Telepresence Technologies*, pp. 189-195.

54. N. Xi (1993),"Event-Based Planning and Control for Robotic Systems", *Doctoral Dissertation, Washington University.*

Web Site:

55. Ken Goldberg's Telegarden, http://www.usc.edu/dept/garden/

56. Railroad at the University of ULN, Denmark, http://rr-vs.informatik.uni-ulm.de/rr/

57. A web interface for NASA's Sojourner Rover, http://mars.graham.com/wits/

58. Lunar Rover Initiative, http://www.frc.ri.cmu.edu/projects/lri/

59. CyberCut, http://CyberCut.berkeley.edu/

60. Mercury Project, http://www.usc.edu/dept/raiders/

61. Drinking Maiden, http://digimuse.usc.edu/robot/

62. PumaPaint, http://yugo.mme.wilkes.edu/~villanov/

63. Robotic Telescope Observatory site, http://www.eia.brad.ac.uk/rti/

64. Ken Taylor's Telerobot, http://telerobot.mech.uwa.edu.au/

65. Interactive Machine Control Website, http://www.aml.gvsu.edu/~ihcws/process.html

66. 12. NASA Space Telerobotics Program:

http://ranier.oact.hq.nasa.gov/telerobotics_page/realrobots.html

Appendix A

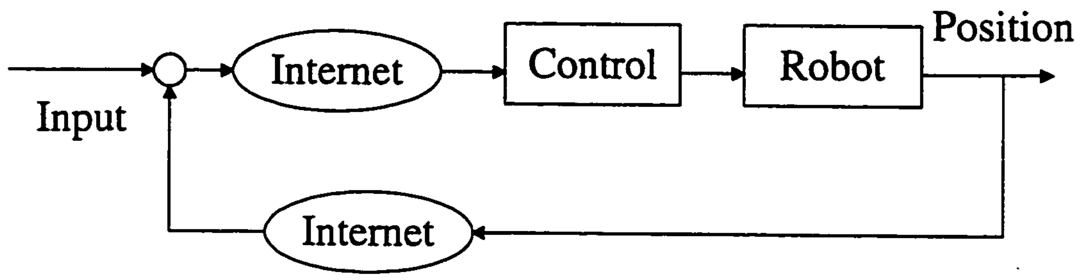


Figure Appendix1: *System Architecture for Image Transfer*

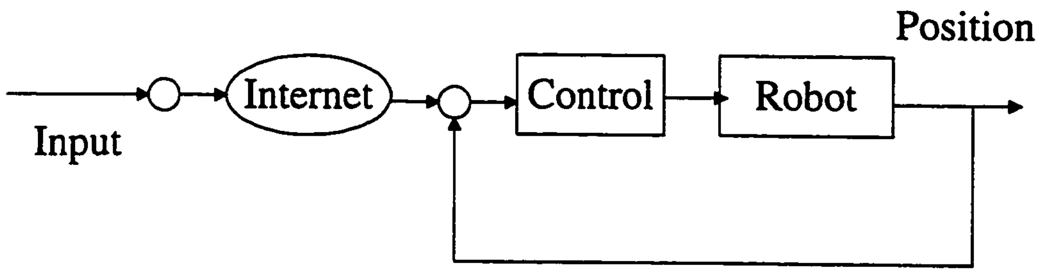


Figure Appendix2: *System Architecture for Control Algorithm*

Appendix B

```
/***********************************************************************
 *                                                                     *
 *  Event-Based Control Simulation of the Robotic Manipulator          *
 *                                                                     *
 ***********************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/***********************************************************************
 *                                                                     *
 *                    Constant Definetion                              *
 *                                                                     *
 ***********************************************************************/

#define GRAVITY  9.8
#define PI 3.1415926

#define M1  2
#define M2  1
#define L1  1
#define L2  1

#define XC  1.2
#define YC  1.0
#define RA  0.2
#define RB  0.15

/*#define AMAX  0.16
#define VMAX  0.4
#define UMAX  0.8
#define WMAX  VMAX*VMAX*/

#define AMAX  0.8
#define VMAX  sqrt (0.8)
#define UMAX  4.0
#define WMAX  0.8

#define S0  0.0
#define SF  PI*(1.5*(RA+RB)-sqrt(RA*RB))
#define STOP  0.7
```

```
#define SP  0.1
#define S1  (S0+AMAX/UMAX)
#define S2  (S0+WMAX/AMAX/2)
#define S3  (S0+AMAX/UMAX+WMAX/AMAX/2)
#define S4  (SF-AMAX/UMAX-WMAX/AMAX/2)
#define S5  (SF-WMAX/AMAX/2)
#define S6  (SF-AMAX/UMAX)

/* #define AMAX  0.3   */
/* #define VMAX  0.2   */
/* #define S0  0    */
/* #define SF  1.98  */
/* #define S1  SF/3  */
/* #define S2  SF*2/3 */

/*******************************************************************
 *                                                                 *
 *              Variable Definetion 1                              *
 *                                                                 *
 * D,H,C        : The design parameter matrices and vectors        *
 * ActD,ActH,ActC: The actual parameter matrices and vectors       *
 * Tore        : The calculated control torque.                    *
 * Y          : The task space (x,y,z) coordinates                 *
 * Yv          : The derivative (velocity) of Y                    *
 * U          : U=Yad+Kv*(Yvd-Yv)+Kp(Ypd-Y)                        *
 *                                                                 *
 *******************************************************************/

double D[2][2],ActD[2][2],InActD[2][2],H[2],ActH[2],C[2],ActC[2],Tore[2];
double Y[2],Yv[2],U[2];
double v, a, v1, a1, theta, s, m, n;

/*******************************************************************
 *                                                                 *
 *              Variable Definetion 2                              *
 *                                                                 *
 * Qext[2][2] : the matrix of joint position and velocity. The first *
 *          coloum represents the position and the second          *
 *          represents the velocity.                               *
 * Kp,Kv     : the gain matrix of position and velocity            *
 * Q[2],Qd[2] : theta,thetadot                                     *
 *                                                                 *
 *******************************************************************/

double Qext[2][2],Q[2],Qd[2];
double Kv[2][2],Kp[2][2];
```

```
/***********************************************************************
 *                                                                     *
 *              Variable Definetion 3                                  *
 *                                                                     *
 * Int_Step: the total integrating steps in one sampling time.         *
 * Cur_Step: the current sampling step.                                *
 * Iend    : the total sampling steps.                                 *
 * Sam_Step: =Iend.                                                    *
 * Tend    : the end time.                                             *
 * SamTime : the sampling time.                                        *
 * dt      : integrating time.                                         *
 *                                                                     *
 ***********************************************************************/

int Int_Step, Cur_Step, Iend, Sam_Step;
double Tend, SamTime, dt;

/***********************************************************************
 *                                                                     *
 *              Variable Definetion 4                                  *
 *                                                                     *
 * Ypd    : the desired position trajectory in task space              *
 * Yvd    : the desired velocity trajectory in task space              *
 * Yad    : the desired acceleration trajectory in task space.         *
 * ErrorP : the error in position                                      *
 * ErrorV : the error in velocity                                      *
 *                                                                     *
 ***********************************************************************/

double Ypd[2],Yvd[2],Yad[2];
double ErrorP[2],ErrorV[2],ErrorS;

/***********************************************************************
 *                                                                     *
 *              Variable Definition 5                                  *
 *                                                                     *
 * Qpd, Qvd, Qad: The desired position, velocity, and acceleration of  *
 *                angle.                                               *
 * Qpe, Qve    : The position and velicity error of angle.             *
 * Kconst      : The gain.                                             *
 * Heq         : The heq parameter matrix.                             *
 * Diag        : The gain.                                             *
 *                                                                     *
 ***********************************************************************/
```

```
double Qpd[2], Qvd[2], Qad[2];
double Qpe[2], Qve[2];
double Z[2], Kconst, Heq[2][2], Diag[2][2];


/************************************************************************
 *                                                                      *
 *              Variable Definition 6                                   *
 *                                                                      *
 * STOP   : The position of obstacle.                                   *
 *                                                                      *
 ************************************************************************/


/************************************************************************
 *                                                                      *
 *              Variable Definetion 7                                   *
 *                                                                      *
 * Time         : the time reference of sampling.                       *
 * FileName1-14 : the name of result data file of Event-Based Control.  *
 * *fp          : file pointer.                                         *
 *                                                                      *
 ************************************************************************/

double Time;
char FileName1[15]="angle1.dat";
char FileName2[15]="xyd1.dat";
char FileName3[15]="xya1.dat";
char FileName4[15]="errp1.dat";
char FileName5[15]="erra1.dat";
char FileName6[15]="trajd1.dat";
char FileName7[15]="traja1.dat";
char FileName8[15]="xydv1.dat";
char FileName9[15]="xyav1.dat";
char FileName10[15]="torq1.dat";
char FileName11[15]="planp1.dat";
char FileName12[15]="planv1.dat";
char FileName13[15]="errs1.dat";
char FileName14[15]="theta.dat";
FILE *fp1, *fp2, *fp3, *fp4, *fp5, *fp6, *fp7, *fp8;
FILE *fp9, *fp10, *fp11, *fp12, *fp13, *fp14;
```

```
/************************************************************************
 *                                                                      *
 *              Subroutines                                             *
 *                                                                      *
 ***********************************************************************/


/****************************************
 * set the initial value of variables   *
 ****************************************/

void Define_Para()
{
 int i,j;

 Tend=60;               /*the end time*/
 SamTime=0.01;              /*sampling time period*/
 dt=0.0001;             /*integrating time*/


 for(i=0;i<2;i++)
  for(j=0;j<2;j++)
  {
   if(i==j)
    {
     Kv[i][i]=500;       /*the gain parameters*/
     Kp[i][i]=800;

    }
   else
   {
    Kv[i][j]=0;
    Kp[i][j]=0;
   }
  }

  Qpd[0]= 0.625047994436+ 0.5295253254941;
  Qpd[1]=-2*0.5295253254941;

  for(i=0;i<2;i++)
  {
   Qvd[i]=0;
   Qad[i]=0;
  }
}
```

```
/******************************************
 *  get the inverse matrix :              *
 *  InMat is the inversion of Mat         *
 ******************************************/

int InvertMat(double Mat[2][2],double InMat[2][2])
{
 int i,j,k,is[2],js[2];
 double maxx,p;

 for(i=0;i<2;i++)
  for(j=0;j<2;j++)
   InMat[i][j]=Mat[i][j];

   for(k=0;k<2;k++)
   {
    maxx=0;
    for(i=k;i<2;i++)
     for(j=k;j<2;j++)
     {
      p=fabs(InMat[i][j]);
      if(p>maxx)
       {
        maxx=p;
        is[k]=i;
        js[k]=j;
       }
     }
    if(maxx=0)return(-1);
     if(is[k]!=k)
      for(j=0;j<2;j++)
       {
        p=InMat[k][j];
        InMat[k][j]=InMat[is[k]][j];
        InMat[is[k]][j]=p;
       }
      if(js[k]!=k)
       for(i=0;i<2;i++)
       {
         p=InMat[i][k];
         InMat[i][k]=InMat[i][js[k]];
         InMat[i][js[k]]=p;
       }
       InMat[k][k]=1/InMat[k][k];
      for(j=0;j<2;j++)
```

```
          if(j!=k)
           InMat[k][j]=InMat[k][j]*InMat[k][k];
          for(i=0;i<2;i++)
           if(i!=k)
            for(j=0;j<2;j++)
             if(j!=k)
              InMat[i][j]=InMat[i][j]-InMat[i][k]*
              InMat[k][j];
          for(i=0;i<2;i++)
           if(i!=k)
            InMat[i][k]=-InMat[i][k]*InMat[k][k];
      }
      for(k=1;k>=0;k--)
      {
       if(js[k]!=k)
        for(j=0;j<2;j++)
        {
          p=InMat[k][j];
          InMat[k][j]=InMat[js[k]][j];
          InMat[js[k]][j]=p;
        }
       if(is[k]!=k)
        for(i=0;i<2;i++)
        {
         p=InMat[i][k];
         InMat[i][k]=InMat[i][is[k]];
         InMat[i][is[k]]=p;
        }
      }
    return(0);
   }


/******************************************
 *  Multiple of Mat[2][2] and Vect[2],        *
 *  result is stored in Mul[2]               *
 ******************************************/

void Multi(double Mat[2][2],double Vect[2],double Mul[2])
{
 int i,j;

 for(i=0;i<2;i++)
  Mul[i]=0;
  for(i=0;i<2;i++)
   for(j=0;j<2;j++)
```

```
      Mul[i]=Mul[i]+Mat[i][j]*Vect[j];
 }


/********************************************
 * Multiple of two vectors into one scalar          *
 ********************************************/

void MultiVect(double Vect1[2], double Vect2[2],double Val)
{
 int i;

 Val=0;
 for (i=0;i<2;i++)
   Val+=Vect1[i]*Vect2[i];
}

/*Add of two Vect : Vect1[2] + Vect2[2] = SumVect[2]*/

void VectAdd(double Vect1[2],double Vect2[2],double SumVect[2])
{
 int i;

 for(i=0;i<2;i++)
   SumVect[i]=Vect1[i]+Vect2[i];
}


/******************************************
 * Subb of Two Vect :                          *
 * Vect1[2] - Vect2[2] = SubbVect[2]          *
 ******************************************/

void VectSubb(double Vect1[2],double Vect2[2],double SubbVect[2])
{
 int i;

 for(i=0;i<2;i++)
   SubbVect[i]=Vect1[i]-Vect2[i];
}
```

```
/***********************************************
 *  multiple two matrix : Mat1 * Mat2 = Mat3            *
 ***********************************************/

void MultiMat(double Mat1[2][2],double Mat2[2][2],double Mat3[2][2])
{
 int i,j,jj;
 double sum;

 for(i=0;i<2;i++)
   for(j=0;j<2;j++)
   {
     sum=0;
     for(jj=0;jj<2;jj++)
       sum+=Mat1[i][jj]*Mat2[jj][j];
     Mat3[i][j]=sum;
   }
}


/************************************************
 * print a matrix on the screen when debugging              *
 ************************************************/

void printmat(double mat[2][2])
{
 int i;

 for(i=0;i<2;i++)
   printf("%10.6f,  %10.6f\n",
 mat[i][0],mat[i][1]);
 printf("\n");
}

/**************************************
 * transpose a matrix Mat into MatT             *
 **************************************/

void Transpose(double Mat[2][2],double MatT[2][2])
{
 int i,j;

 for (i=0;i<2;i++)
   for (j=0;j<2;j++)
     MatT[i][j]=Mat[j][i];
}
```

```
/****************************************
 * A 1x2 vector multiply a 2x2 matrix        *
 ****************************************/

void MultiVect2(double Vect1[2],double Mat[2][2],double Vect2[2])
{
 int i,j;
 for (i=0;i<2;i++)
 {
  Vect2[i]=0;
  for(j=0;j<2;j++)
   Vect2[i]+=Vect1[j]*Mat[j][i];
 }
}


/**********************************************
 * function in soluting the dynamic equation          *
 * to get actual trajectory                           *
 **********************************************/

void Model(double x[2][2],double dxs[2][2])
{
 int i;
 double TmpVect1[2],TmpVect2[2];

 InvertMat(ActD,InActD);

 for(i=0;i<2;i++)
  dxs[i][0]=x[i][1];

 VectAdd(ActH,ActC,TmpVect1);
 VectSubb(Tore,TmpVect1,TmpVect2);
 Multi(InActD,TmpVect2,TmpVect1);
 for(i=0;i<2;i++)
  dxs[i][1]=TmpVect1[i];
}


/************************************************************
 *                                                          *
 *       Get the Parameter Matrix                           *
 *                                                          *
 ************************************************************/
```

```
/******************************************************
 * get the actual D matrix using the actual Q value            *
 ******************************************************/

void GetD(double DD[2][2])
{

 DD[0][0]=M2*L2*L2+2*M2*L1*L2*cos(Q[1])+(M1+M2)*L2*L2;
 DD[0][1]=M2*L2*L2+M2*L1*L2*cos(Q[1]);

 DD[1][0]=M2*L2*L2+M2*L1*L2*cos(Q[1]);
 DD[1][1]=M2*L2*L2;

}


/********************
 * the H matrix          *
 ********************/

void GetH(double HH[2])
{
 HH[0]=-2*M2*L1*L2*sin(Q[1])*Qd[0]*Qd[1]-M2*L1*L2*sin(Q[1])*Qd[1]*Qd[1];
 HH[1]=M2*L1*L2*sin(Q[1])*Qd[0]*Qd[0];

}

/********************
 * the C vector          *
 ********************/

void GetC(double CC[2])
{
 double Ang12;

 Ang12=Q[0]+Q[1];

 CC[0]=GRAVITY*(M2*L2*cos(Ang12)+(M1+M2)*L1*cos(Q[0]));

 CC[1]=GRAVITY*M2*L2*cos(Ang12);

}
```

```
/***********************************
 * get the design or actual         *
 * parameter matrice or vectors     *
 ***********************************/

void GetPara(double DD[2][2],double HH[3],double CC[3])
{

 GetD(DD);
 GetH(HH);
 GetC(CC);
}


/**********************************
 * solve the dynamic equations      *
 * using Runge_Kutta method         *
 **********************************/

void Integrate()
{
 int i,j,ii;
 double xt[2][2],xf[2][2],rk[2][2];

 for(i=0;i<Int_Step;i++)
 {

  GetPara(ActD,ActH,ActC);
  Model(Qext,xf);
  for(j=0;j<2;j++)
  {
   for(ii=0;ii<2;ii++)
   {
    rk[ii][j]=xf[ii][j]*dt;
    xt[ii][j]=Qext[ii][j]+0.5*xf[ii][j]*dt;
   }
  }
  Model(xt,xf);
  for(j=0;j<2;j++)
  {
   for(ii=0;ii<2;ii++)
   {
    rk[ii][j]=rk[ii][j]+2*xf[ii][j]*dt;
    xt[ii][j]=Qext[ii][j]+0.5*xf[ii][j]*dt;
   }
  }
```

```
    Model(xt,xf);
    for(j=0;j<2;j++)
    {
     for(ii=0;ii<2;ii++)
     {
      rk[ii][j]=rk[ii][j]+2*xf[ii][j]*dt;
      xt[ii][j]=Qext[ii][j]+xf[ii][j]*dt;
     }
    }
    Model(xt,xf);
    for(j=0;j<2;j++)
    {
     for(ii=0;ii<2;ii++)
     {
      rk[ii][j]=rk[ii][j]+xf[ii][j]*dt;
      Qext[ii][j]=Qext[ii][j]+rk[ii][j]/6;
     }
    }
    for(j=0;j<2;j++)
    {
     Q[j]=Qext[j][0];
     Qd[j]=Qext[j][1];
    }
  }
}


/**********************************
 * Get Jacobi matrix according      *
 * the actual joint position        *
 **********************************/

void GetJq(double Jq[2][2],double Qp[2])
{
 double Ang12;

 Ang12=Qp[0]+Qp[1];

 Jq[0][0]=-(L1*sin(Qp[0])+L2*sin(Ang12));
 Jq[0][1]=-L2*sin(Ang12);

 Jq[1][0]=L1*cos(Qp[0])+L2*cos(Ang12);
 Jq[1][1]=L2*cos(Ang12);

}
```

```
/**********************************
 * get the actual positions in         *
 * 3 coordinates in task space         *
 **********************************/

void CalY()
{
 double Ang12;
 double Jq[2][2];

 Ang12=Q[0]+Q[1];

 Y[0]=L1*cos(Q[0])+L2*cos(Ang12);
 Y[!]=L1*sin(Q[0])+L2*sin(Ang12);
 GetJq(Jq,Q);
 Multi(Jq,Qd,Yv);
}


/***************************************
 * get the desired position,velocity,        *
 * acceleration in 3 coordinates in          *
 * task space.                               *
 ***************************************/

void GetYpvad()
{
 double X0=1.41;
 double Y0=1.0;

 if (Y[1] >= YC) {
  theta=asin(fabs(Y[1]-YC)/sqrt(Y[0]*Y[0]+Y[1]*Y[1]));
  s=RA*theta;
  Ypd[0]=XC+RA/(sqrt(1+(Y[1]-YC)*(Y[1]-YC)/((Y[0]-XC)*(Y[0]-XC))));
  Ypd[1]=YC+RA/(sqrt(1+(Y[0]-XC)*(Y[0]-XC)/((Y[1]-YC)*(Y[1]-YC))));

  if(Y[0] < XC) {
   theta=PI/2+theta;
   s=RA*theta;
   Ypd[0]=XC-RA/(sqrt(1+(Y[1]-YC)*(Y[1]-YC)/((Y[0]-XC)*(Y[0]-XC))));
   Ypd[1]=YC+RA/(sqrt(1+(Y[0]-XC)*(Y[0]-XC)/((Y[1]-YC)*(Y[1]-YC))));
  }
 }

 if (Y[1] < YC) {
  theta=asin(fabs(Y[1]-YC)/sqrt(Y[0]*Y[0]+Y[1]*Y[1]));
  if(Y[0] < XC) {
   theta=PI+theta;
   Ypd[0]=XC-RA/(sqrt(1+(Y[1]-YC)*(Y[1]-YC)/((Y[0]-XC)*(Y[0]-XC))));
   Ypd[1]=YC-RA/(sqrt(1+(Y[0]-XC)*(Y[0]-XC)/((Y[1]-YC)*(Y[1]-YC))));
  }
  else {
   theta=2*PI-theta;
   Ypd[0]=XC+RA/(sqrt(1+(Y[1]-YC)*(Y[1]-YC)/((Y[0]-XC)*(Y[0]-XC))));
   Ypd[1]=YC-RA/(sqrt(1+(Y[0]-XC)*(Y[0]-XC)/((Y[1]-YC)*(Y[1]-YC))));
  }
  s=RA*theta;
  printf("theta= %f, Time= %f\n", theta, Time);
 }

 m=-(Ypd[1]-YC)/RA;
 n=(Ypd[0]-XC)/RA;

 if (s<=S1) {
  v=sqrt(UMAX*(s-S0)*(s-S0));
  a=UMAX*(s-S0);
 }
 else if (s <= S2) {
  v=sqrt(2*AMAX*(s-S1)+UMAX*(S1-S0)*(S1-S0));
  a=AMAX;
 }
 else if (s <= S3) {
  v=sqrt(-UMAX*(s-S3)*(s-S3)+WMAX);
  a=-UMAX*(s-S3);
 }
 else if (s <= S4) {
  v=VMAX;
  a=0.0;
 }
 else if (s <= S5) {
  v=sqrt(-UMAX*(s-S4)*(s-S4)+WMAX);
  a=-UMAX*(s-S5);
 }
 else if (s <= S6) {
  v=sqrt(-2*AMAX*(s-S6)+UMAX*(S6-SF)*(S6-SF));
  a=-AMAX;
 }
 else {
  v=sqrt(UMAX*(s-SF)*(s-SF));
  a=UMAX*(s-SF);
 }
```

```
/* if (s<=S1) {
   v=sqrt(UMAX*(s-S0)*(s-S0));
   a=UMAX*(s-S0);
  }
  else if (s <= S6) {
   v=sqrt(-2*AMAX/(SF-S0-2*AMAX/UMAX)*(s-(S0+SF)/2)*(s-
(S0+SF)/2)+AMAX*AMAX/UMAX+2*AMAX/(SF-S0-
2*AMAX/UMAX)*(AMAX/UMAX+(S0-SF)/2)*(AMAX/UMAX+(S0-SF)/2));
   a=-2*AMAX/(SF-S0-2*AMAX/UMAX)*(s-(S0+SF)/2);
  }
  else {
   v=sqrt(UMAX*(s-SF)*(s-SF));
   a=UMAX*(s-SF);
  }
*/
  Yvd[0]=m*v;
  Yvd[1]=n*v;

  Yad[0]=m*a;
  Yad[1]=n*a;

  ErrorS=sqrt(Ypd[0]*Ypd[0]+Ypd[1]*Ypd[1])-sqrt(Y[0]*Y[0]+Y[1]*Y[1]);
}


/**************************************
 * Get the Jacobi matrix derivative          *
 **************************************/

void GetJqDot(double JqDot[2][2],double Qp[2],double Qv[2])
{

 JqDot[0][0]=-(L1*cos(Qp[0])*Qv[0]+L2*cos(Qp[0]+Qp[1])*(Qv[0]+Qv[1]));
 JqDot[0][1]=-L2*cos(Qp[0]+Qp[1])*(Qv[0]+Qv[1]);
 JqDot[1][0]=-(L1*sin(Qp[0])*Qv[0]+L2*sin(Qp[0]+Qp[1])*(Qv[0]+Qv[1]));
 JqDot[1][1]=-L2*sin(Qp[0]+Qp[1])*(Qv[0]+Qv[1]);

}


/******************************************************
 * calculate the U vector in the control expression.          *
 ******************************************************/

void GetU()
```

```
{

 double TmpVect1[2],TmpVect2[2],TmpVect3[2];

 {
  CalY();
  GetYpvad();
  VectSubb(Ypd,Y,ErrorP);
  Multi(Kp,ErrorP,TmpVect2);
  VectSubb(Yvd,Yv,ErrorV);
  Multi(Kv,ErrorV,TmpVect1);
  VectAdd(Yad,TmpVect1,TmpVect3);
  VectAdd(TmpVect2,TmpVect3,U);

 }

}


/***************************
 * get the torgue vector            *
 ***************************/

void GetTorque()
{

  double TmpVect1[2],TmpVect2[2],Jqd[2][2];
  double Jq[2][2],InJq[2][2];

  GetJq(Jq,Q);
  GetJqDot(Jqd,Q,Qd);
  InvertMat(Jq,InJq);
  Multi(Jqd,Qd,TmpVect1);
  VectSubb(U,TmpVect1,TmpVect2);
  Multi(InJq,TmpVect2,TmpVect1);
  Multi(D,TmpVect1,TmpVect2);
  VectAdd(TmpVect2,H,TmpVect1);
  VectAdd(TmpVect1,C,Tore);
}



/************************************
 * The Control Procedure for NLF            *
 ************************************/
```

```
void NLFControl()
{
 GetPara(D,H,C);
 GetU();
 GetTorque();
}


/***********************************************
 * calculate the initial position of 3 joints           *
 ***********************************************/

void InitQ()
{
 int i;
 for (i=0;i<3;i++)
   Qext[i][1]=0;

 /*we get these from the inverse kinematics*/

 Qext[0][0]=Q[0]= 0.625047994436+ 0.5295253254941;
 Qext[1][0]=Q[1]=-2* 0.5295253254941;

}


/*****************************************************
 * record the value of run time variables into files.              *
 *****************************************************/

void Record()
{

 fprintf(fp1, "%10.7f, %10.7f, %10.7f\n", Time,Q[0],Q[1]);
 fprintf(fp2, "%10.7f, %10.7f, %10.7f\n", Time,Ypd[0],Ypd[1]);
 fprintf(fp3, "%10.7f, %10.7f, %10.7f\n", Time,Y[0],Y[1]);
 fprintf(fp4, "%10.7f, %10.7f, %10.7f\n", Time,ErrorP[0],ErrorP[1]);
 fprintf(fp5, "%10.7f, %10.7f, %10.7f\n", Time,ErrorV[0],ErrorV[1]);
 fprintf(fp6, "%10.7f, %10.7f\n", Ypd[0],Ypd[1]);
 fprintf(fp7, "%10.7f, %10.7f\n", Y[0],Y[1]);
 fprintf(fp8, "%10.7f, %10.7f, %10.7f\n", Time,Yvd[0],Yvd[1]);
 fprintf(fp9, "%10.7f, %10.7f, %10.7f\n", Time,Yv[0],Yv[1]);
 fprintf(fp10, "%10.7f, %10.7f, %10.7f\n", Time,Tore[0],Tore[1]);
 fprintf(fp11, "%10.7f, %10.7f\n", Time,s);
 fprintf(fp12, "%10.7f, %10.7f\n", s,v);
 fprintf(fp13, "%10.7f, %10.7f\n", Time,ErrorS);
 fprintf(fp14, "%10.7f, %10.7f\n", Time, theta);

}

/**********************************************************************
 *                                                                    *
 *              The Main Function                                     *
 *                                                                    *
 * Function: 1) set all the intial parameters and calculate some      *
 *              parameters off-linely.                                *
 *                                                                    *
 *           2) NLF control voltages are calculated and and applied on the *
 *         dynamic model equation. Results are recorded in "nlf.dat"  *
 *                                                                    *
 **********************************************************************/


int main()
{

 Define_Para();


 Sam_Step=(int)(Tend/SamTime);   /*the total sampling steps*/
 Int_Step=(int)(SamTime/dt);     /*the integral steps in each sampling time*/
 Iend=Sam_Step;


 printf("Simulation in progress, please wait...\n");

 fp1=fopen(FileName1, "w");
 fp2=fopen(FileName2, "w");
 fp3=fopen(FileName3, "w");
 fp4=fopen(FileName4, "w");
 fp5=fopen(FileName5, "w");
 fp6=fopen(FileName6, "w");
 fp7=fopen(FileName7, "w");
 fp8=fopen(FileName8, "w");
 fp9=fopen(FileName9, "w");
 fp10=fopen(FileName10, "w");
 fp11=fopen(FileName11, "w");
 fp12=fopen(FileName12, "w");
 fp13=fopen(FileName13, "w");
 fp14=fopen(FileName14, "w");

 Cur_Step=0;
 InitQ();                /*get the initial value of Q*/
 Time=0;

 do {
  Time=SamTime*Cur_Step;
  NLFControl();
  Record();
  Integrate();
  Cur_Step++;
 }
 while(Cur_Step<=Iend);

 printf("Result data is stored in DAT files\n");

 fclose(fp1);
 fclose(fp2);
 fclose(fp3);
 fclose(fp4);
 fclose(fp5);
 fclose(fp6);
 fclose(fp7);
 fclose(fp8);
 fclose(fp9);
 fclose(fp10);
 fclose(fp11);
 fclose(fp12);
 fclose(fp13);
 fclose(fp14);

 return 0;
}
```

```
import java.awt.*;
import java.applet.*;
import java.io.*;
import java.text.*;
import java.lang.*;
import java.net.*;

import vrml.external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;
import vrml.external.field.*;

class LinkNode extends Applet {
   Node node = null;
   EventInSFRotation input = null;
   EventOutSFRotation output = null;
}

class Finger extends Applet {
   float[] Joint = new float[2];
   LinkNode Link1 = new LinkNode();
   LinkNode Link2 = new LinkNode();
}

class ObjectNode extends Applet {
         float[] position = new float[3];   // VRML only use float values
         float[] rotation = new float[4];
   Node node = null;
   EventInSFVec3f Vecin = null;
   EventOutSFVec3f Vecout = null;
   EventInSFRotation Rotin = null;
   EventOutSFRotation Rotout = null;
}

public class interfacetest extends Applet implements EventOutObserver {
 boolean error = false;

 TextArea textout = null;
 TextArea textin = null;
 Browser browser = null;

 ObjectNode object = new ObjectNode();
 Finger[] finger = new Finger[2];
 Finger Finger1 = new Finger();
 Finger Finger2 = new Finger();
```

```
void buildConstraints( GridBagConstraints gbc,
                int gx, int gy, int gw, int gh, int wx, int wy)
{
 gbc.gridx = gx;
 gbc.gridy = gy;
 gbc.gridwidth = gw;
 gbc.gridheight = gh;
 gbc.weightx = wx;
 gbc.weighty = wy;
}

public void init() {
 GridBagLayout gbl = new GridBagLayout();
 GridBagConstraints gbc = new GridBagConstraints();
 setLayout(gbl);

 buildConstraints(gbc, 0, 0, 1, 1, 20, 20);
 Panel p1 = new Panel();
 p1.add(new Button("Reset"));
 p1.add(new Button("To Point A"));
         gbl.setConstraints(p1, gbc);
 add(p1);

      buildConstraints(gbc, 0, 1, 1, 1, 20, 20);
 Panel p2 = new Panel();
 p2.add(new Button("Straight Line"));
 p2.add(new Button("Broken Line"));
         gbl.setConstraints(p2, gbc);
 add(p2);

      buildConstraints(gbc, 0, 2, 1, 1, 20, 20);
 Panel p3 = new Panel();
 p3.add(new Button("Circle"));
 p3.add(new Button("Straight Line with Obstacle"));
         gbl.setConstraints(p3, gbc);
 add(p3);


 buildConstraints(gbc, 0, 3, 1, 1, 20, 20);
 textin = new TextArea(3, 45);
 textin.setEditable(true);
 gbl.setConstraints(textin, gbc);
 add(textin);

 buildConstraints(gbc, 0, 4, 1, 1, 20, 20);
 textout = new TextArea(5, 45);
 textout.setEditable(false);
 gbl.setConstraints(textout, gbc);
 add(textout);
}

public void callback(EventOut who, double when, Object which){}

public void start() {
 browser = (Browser) vrml.external.Browser.getBrowser(this);
 try {
    object.node = browser.getNode("Objects");
    object.Vecin = (EventInSFVec3f)object.node.getEventIn("translation");
    object.Vecout = (EventOutSFVec3f)object.node.getEventOut("translation");
    object.Rotin = (EventInSFRotation)object.node.getEventIn("rotation");
    object.Rotout = (EventOutSFRotation)object.node.getEventOut("rotation");
    object.Vecout.advise(this, null);
    object.Rotout.advise(this, null);

    finger[0]=Finger1;
    finger[1]=Finger2;
    finger[0].Link1.node = browser.getNode("F1Link1");
    finger[0].Link2.node = browser.getNode("F1Link2");
    finger[1].Link1.node = browser.getNode("F2Link1");
    finger[1].Link2.node = browser.getNode("F2Link2");

     for( int i=0;i<2;i++) {
       finger[i].Link1.input
     = (EventInSFRotation)finger[i].Link1.node.getEventIn("rotation");
       finger[i].Link1.output
     = (EventOutSFRotation)finger[i].Link1.node.getEventOut("rotation");
       finger[i].Link1.output.advise(this, null);
       finger[i].Link2.input
     = (EventInSFRotation)finger[i].Link2.node.getEventIn("rotation");
       finger[i].Link2.output
     = (EventOutSFRotation)finger[i].Link2.node.getEventOut("rotation");
       finger[i].Link2.output.advise(this, null);
     }
 }
 catch (InvalidNodeException ne) {
  add(new TextField("Failed to get node:" + ne));
  error = true;
 }
 catch (InvalidEventInException ee) {
  add(new TextField("Failed to get EventIn:" + ee));
  error = true;
 }
```

```
  catch (InvalidEventOutException ee) {
    add(new TextField("Failed to get EventOut:" + ee));
    error = true;
  }
 }

public void MoveFinger( Finger finger) {
  float[] val = new float[4];
        val[0] = 0;
        val[1] = 1;
        val[2] = 0;
        val[3] = new Float(finger.Joint[0]).floatValue();
        finger.Link1.input.setValue(val);
        val[3] = new Float(finger.Joint[1]).floatValue();
        finger.Link2.input.setValue(val);
 }

public void MoveObject() {
  object.Vecin.setValue( object.position);
  object.Rotin.setValue( object.rotation);
}

public void readdata()
{
  float x[]= new float[7];
        String urls= " ";

  try{
    URL ur= new URL(urls);
  }
  catch(MalformedURLException e)
  {
   textout.appendText(" URL Error "+e);
  }

   try {
    String str, substr;

    InputStreamReader in0= new InputStreamReader(
        new URL(getCodeBase(), "g.sum").openStream());
    BufferedReader in= new BufferedReader(in0);
    DecimalFormat df = new DecimalFormat();

    for(int i=0; i<27; i++)
    {  str = in.readLine();
      if(i>=20)
      {
       substr = str.substring(13,str.length());
       str = substr.trim();
       Number n = df.parse(str);
       x[i-20] = (float) n.doubleValue();
      }
    }
    in.close();
   }
  catch (IOException e) {textout.appendText("IO Error");}
  catch (ParseException e) { }
  for(int i=0;i<7;i++){textout.appendText("x"+i+"="+x[i]+"\n");}

  object.rotation[0]= 0F;
  object.rotation[1]= 0F;
  object.rotation[2]= 1F;
  object.position[2]= 0F;
  finger[0].Joint[1]= x[0];
  finger[0].Joint[0]= x[1]-1.571F;
  finger[1].Joint[0]= x[2]-1.571F;
  finger[1].Joint[1]= x[3];
  object.position[0]= x[4];
  object.position[1]= x[5];
  object.rotation[3]= x[6];

 }

public void Setdata()
{
 object.rotation[0]= 0;
 object.rotation[1]= 0;
 object.rotation[2]= (float)1;
 object.position[2]= (float)0.0;

 object.position[0]= (float)-2.0040;
 object.position[1]= (float)44.2244;
 object.rotation[3]= (float)0.0848;
 finger[0].Joint[1]= (float)0.5506;
 finger[0].Joint[0]= (float)(1.2875-1.571);
 finger[1].Joint[0]= (float)(1.3505-1.571);
 finger[1].Joint[1]= (float) 0.6318;
}

public boolean action (Event event, Object what)
{
 int  i, j, k;
```

```
  if (error) {
        showStatus("Uh Oh...! An error occurs during initialization");
        return true;
  }
  if(event.target instanceof Button) {
   Button button = (Button) event.target;
        if(button.getLabel() == "Reset") {
         for( i=0;i<2;i++)
       for( j=0; j<2; j++) finger[i].Joint[j] = 0;
         for( i=0;i<2;i++) MoveFinger( finger[i]);
         textout.appendText( " Arm reseted !\n");
        }

        if(button.getLabel() == "Circle") {

         textout.appendText( " Follow Circle !\n");
        }
         if(button.getLabel() == "Straight Line") {

         textout.appendText( " Follow Straight Line!\n");
        }
         if(button.getLabel() == "Broken Line") {

         textout.appendText( " Follow Broken Line !\n");
        }
              if(button.getLabel() == "Straight Line with Obstacle") {

         textout.appendText( " Follow Straight Line with Obstacle!\n");
        }


        if(button.getLabel() == "To Point A") {
//    readdata();
    Setdata();
    MoveObject();
    MoveFinger( finger[0] );
    MoveFinger( finger[1] );
    textout.appendText( " To Point A.\n");
        }

  }
  return true;
 }

}     // End of test class
```

```
#VRML V2.0 utf8
#
# hand.wrl

WorldInfo{
      title "Two-link Robotic Manipulator"
      info ["August, 2000"]
}

Viewpoint{
      position 0 0 200
      description "Entry view"
}

Background {
#   skyColor [0.0 0.2 0.7, 0.0 0.5 1.0, 1.0 1.0 1.0]
  skyColor [0.5 0.5 1.0, 0.7 0.7 1.0, 0.9 0.9 1.0]
  skyAngle [1.25 2.0]   #[0.5 1.0]
}

NavigationInfo{
      headlight TRUE
      type "EXAMINE"
}

SpotLight{
      ambientIntensity 1.75
      intensity 3.0
      color 0.3 0 0.5
      direction 1 -1 -1
      location  -45 0 50
}

PROTO Vtext [
      field SFVec3f position 0 0 0
      field MFString symbol " " ]
    { Transform{
        translation IS position
        children Shape{
          appearance Appearance{
            material Material {diffuseColor 0.0 0.0 0.0}
          }
          geometry Text{
            string IS symbol
            fontStyle FontStyle{
```

```
            size 6.5 style "BOLD" family "TYPEWRITER"}
          }
        }
      }
    }

DEF WHOLE Transform{
translation  20 -50 0
rotation 1 0 0 0.15
children[

  DEF HAND Transform{
    children[
      DEF Palm Transform{
        translation -30 -6 0
        children[
          Shape{
            geometry Box {size 25 2.5 15}
            appearance DEF A2 Appearance{
              material Material {diffuseColor 1 0 0}
            }
          }
        ]
      }

      DEF FINGER1 Transform{
        translation -30 0 0
        rotation 0 0 1 -1.57
        children[
          DEF FBase Transform{
            rotation 1 0 0 1.571
            children[
              Shape{
                geometry Cylinder {radius 1.5 height 3}
                appearance DEF A2 Appearance{
                  material Material {diffuseColor 0.6 0.6 0.6}
                }
              }
              Transform{
                translation 3.0 0.0 0.0
                children[
                  Shape{
                    geometry Box {size 5.0 6.0 6.0}
                    appearance USE A2
                  }
                ]
  }
 ]
}

  Transform{
   rotation 1 0 0 -1.571
   children[
    DEF F1Link1 Transform{
     children[
      DEF F1Joint1 CylinderSensor {maxAngle 1.571 minAngle -1.5711}
      Transform{
       children[
        DEF JOINT2 Shape{
         geometry Cylinder {radius 1.5 height 2.5}
         appearance DEF A4 Appearance{
          material Material {diffuseColor 0.2 0.3 0.7}
         }
        }
       ]
      }
      Transform{
       rotation 0.0 0.0 1.0 1.571
       translation -25 0 0
       children[
        Shape{
         geometry Cylinder {radius 1 height 50.0}
         appearance USE A4
        }
       ]
      }
      Transform{
       translation -50 0 0
       scale 1.1 1 1.2
       children [USE JOINT2 ]
      }

      DEF F1Link2 Transform{
       translation -50.0 0.0 0.0
       children[
        DEF F1Joint2 CylinderSensor {maxAngle 3.14 minAngle -1.571}
        Shape{
         geometry Cylinder {radius 0.5 height 2.5}
         appearance DEF A5 Appearance{
          material Material {diffuseColor 0.3 0.7 0.1}
         }
        }
```

```
              Transform{
                rotation 0.0 0.0 1.0 1.571
                translation -20 0 0
                children[
                  Shape{
                    geometry Cylinder {radius 1.0 height 40.0}
                    appearance USE A5
                  }
                ]
              }
              Transform{
                translation -40.0 0.0 0.0
                children[
                  Shape{
                    geometry Sphere {radius 1.0}
                    appearance USE A5
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
  }
  ]
}

DEF FINGER2 Transform{
  translation 30 0 0
  rotation 0 0 1 -1.571
  children[
#   USE FBase

      Transform{
        rotation 1 0 0 1.571
        children[
          DEF F2Link1 Transform{
            children[
              DEF F2Joint1 CylinderSensor {maxAngle 1.571 minAngle -1.571}
              Transform{
                children[
                  DEF JOINT2 Shape{
                    geometry Cylinder {radius 0 height 0}
                    appearance DEF A4 Appearance{
                      material Material {diffuseColor 0.2 0.3 0.7}
      }
    }
  ]
}
Transform{
  rotation 0.0 0.0 1.0 1.571
  translation -25 0 0
  children[
    Shape{
      geometry Cylinder {radius 0 height 0}
      appearance USE A4
    }
  ]
}
Transform{
  translation -50 0 0
  scale 1.1 1 1.2
  children [USE JOINT2 ]
}

DEF F2Link2 Transform{
  translation -50.0 0.0 0.0
  children[
    DEF F2Joint2 CylinderSensor {maxAngle 3.14 minAngle 0}
    Shape{
      geometry Cylinder {radius 0 height 0}
      appearance DEF A5 Appearance{
        material Material {diffuseColor 0.3 0.7 0.1}
      }
    }
    Transform{
      rotation 0.0 0.0 1.0 1.571
      translation -20 0 0
      children[
        Shape{
          geometry Cylinder {radius 0 height 0}
          appearance USE A5
        }
      ]
    }
    Transform{
      translation -40.0 0.0 0.0
      children[
        Shape{
          geometry Sphere {radius 0}
          appearance USE A5
```

```
                    }
                   ]
                  }
                 ]
                }
               ]
              }
             ]
            }
           ]
         }
        ]
       }



        DEF Objects Transform{
         translation 0 55 0
         children[
          DEF Mover PlaneSensor { }
          DEF Rotator CylinderSensor {maxAngle 3.14 minAngle -3.14}
         DEF CUBE Transform{
            children[
             Shape{
              geometry Box {size 0 0 0}
              appearance Appearance{
               material Material {diffuseColor 0 0 1}
              }
             }

          DEF AXIS Transform{
           children[
            DEF BODY Transform{
             translation 0 5 0.5
             children Shape{
              geometry Cylinder{ radius 0height 0}
              appearance DEF COLOR Appearance{
               material Material {diffuseColor 1 0 0}
              }
             }
            }
            DEF ARROW Transform{
             translation 0 11.5 0.5
             children Shape{
              geometry Cone{ bottomRadius 0 height 0}
              appearance Appearance{
                material Material {diffuseColor 0 0 0}
              }
            }
          }
        ]
        }
      Transform{
        rotation 0 0 -1 1.571
        scale 1.2 1.5 1
        children[ USE AXIS ]
      }
      ]
    }
]
}


ROUTE F1Joint1.rotation_changed TO F1Link1.set_rotation
ROUTE F1Joint2.rotation_changed TO F1Link2.set_rotation
ROUTE F2Joint1.rotation_changed TO F2Link1.set_rotation
ROUTE F2Joint2.rotation_changed TO F2Link2.set_rotation
ROUTE Rotator.rotation_changed TO CUBE.set_rotation
ROUTE Mover.translation_changed TO Objects.set_translation
```

```
<html>
<head>
<title>2-link manipulator</title>
</head>
<body bgcolor="#FFFFFF">
<center>
<FONT COLOUR=#FFFF00 SIZE=+3><B><I>
 Two-link Robot Manipulator
</I></B></FONT>
</center>
<hr WIDTH=95%>
<center><IMG SRC="../images/red.jpg" HEIGHT=2 WIDTH=95%
BORDER=1></center>
<DL>
<center>
<embed src="mypath.wrl" Width=400 Height=300 VRML-DASHBOARD="FALSE">
<applet code="test.class" width=250 Height=300 mayscript></applet>
</center>
<hr WIDTH="95%">
<CENTER><FONT SIZE=+2>ART Lab,  University of Alberta</FONT></CENTER>
</body>
</html>
```