

# Building a Telerobotic System on the Internet

Terry Kerr — 11930039

terry.kerr@csse.monash.edu.au

Supervisor: Gordon Lowe

gordon.lowe@csse.monash.edu.au

School of Computer Science and Software Engineering  
Monash University

June 30, 2000

## Abstract

This thesis involves the development of an Internet Telerobotic system using an Adept One Robot located at the Monash University School of Computer Science and Software Engineering. The school wishes to use the telerobotic system as a tool to teach the Graduate Diploma in Robotics by distance education. The robot is located at <http://va12.csse.monash.edu.au/robot/>.

Most telerobotic systems that appear on the Internet have been developed for feasibility studies or for entertainment. All the systems involve some level of autonomy in the remote control to overcome problems associated with control over the Internet caused by low bandwidth and unpredictable transmission delays.

This thesis has involved the development of telerobotic system that allows direct continuous control of the Adept One Robot remotely via the Internet. Direct continuous control without any autonomy is preferred if the system is to be used for teaching purposes.

A PC running the Linux operating system is used as the link between the robot controller and the Internet. Software written in Java runs on the PC, which takes commands from remote control clients, and sends the appropriate commands to the robot controller via a serial line. A Java applet control client has been developed that is similar in functionality to the Adept One Robot manual control pendant. Remote operators can also connect to the server using a Telnet client, which gives programming terminal control of the robot. A camera is mounted above the robot workspace and connected to the PC. The remote operator receives vision feedback of the robot environment using either the Real Networks Realplayer client, or a web browser that supports server-push.

Direct continuous control of the robot is achieved remotely, however any tasks, which are less than trivial are extremely tedious to execute using the Java, control pendant. Telnet terminal control is very successful. Students can write programs, execute them remotely, and get reasonable vision feedback using Realplayer, even across a modem connection to the Internet.

## Contents

<b>1 Acknowledgments</b>	<b>4</b>
<b>2 Introduction</b>	<b>4</b>
<b>3 Some Telerobotic systems</b>	<b>5</b>
3.1 University of Western Australia's Robot on the Web . . . . .	5
3.2 The PumaPaint Project . . . . .	5
3.3 Mercury Project . . . . .	6
3.4 Kehp on the Web . . . . .	7
3.5 Experimental Results with a complex manipulator . . . . .	7
<b>4 Problem Description</b>	<b>9</b>
4.1 The Adept One Robot . . . . .	9
4.2 Controlling the Adept One Robot . . . . .	10
4.2.1 Manual Control Pendant Control . . . . .	10
4.2.2 Terminal Control . . . . .	11
4.3 Requirements of the Telerobotic System . . . . .	11
<b>5 Implementing the Telerobotic System</b>	<b>12</b>
5.1 Overview of the Telerobotic System Architecture . . . . .	12
5.2 Control System . . . . .	12
5.2.1 Overview of the control system architecture . . . . .	12
5.2.2 Control Server software . . . . .	12
5.2.3 Manual Control Pendant client software . . . . .	16
5.3 Vision System . . . . .	17
5.4 Installation of the Software . . . . .	20
<b>6 Performance and Usability</b>	<b>20</b>
<b>7 Conclusion</b>	<b>21</b>
7.1 General Conclusions . . . . .	21
7.2 Limitations and Further Work . . . . .	22
<b>References</b>	<b>25</b>
<b>Appedix</b>	<b>26</b>
<b>A How to use the System</b>	<b>26</b>
A.1 The Server . . . . .	26
A.2 Remote Clients . . . . .	26
<b>B Important System Information</b>	<b>26</b>
B.1 File locations . . . . .	26
B.2 Service Ports . . . . .	27
<b>C Server Software Class Diagram</b>	<b>28</b>
<b>D Pendant Software Class Diagram</b>	<b>29</b>

## List of Figures

1	The Adept One Robot . . . . .	9
2	The Manipulator Work Envelope . . . . .	10
3	An example VALII program . . . . .	11
4	Telerobotic System Architecture . . . . .	13
5	Command Sub-System Architecture . . . . .	14
6	Data flow between pendant and control server . . . . .	15
7	Control Pendant Applet . . . . .	16
8	Real-time Vision System . . . . .	18
9	Server-push Vision System . . . . .	19
10	Telerobtic System Server Classes . . . . .	28
11	Telerobtic System Pendant Classes . . . . .	29

## List of Tables

1	Approximate Vision Feedback Delays . . . . .	20
---	--	----

## 1 Acknowledgments

Firstly I would like to thank my supervisor, Gordon Lowe, for his help and valuable input during the course of this project.

I would also like to thank Rory Edison from Berkeley University for his information on the communication details between the robot controller and Internet server PC for the telerobotic system that he developed. The information he presented helped my understanding and accelerated development of this system.

## 2 Introduction

Telerobotics is the activity of remotely controlling a robot. Internet telerobotics is a specific case of telerobotics where the remote operator communicates with the robot via the Internet. Robots that can be controlled via the Internet have potential uses in many areas including education and training, manufacturing, and entertainment. However, the only public robots that are found on the Internet are there for feasibility studies or for general entertainment purposes. It is not known how many private Internet telerobots are being used by organisations for any practical purposes.

The first telerobots on the Internet started appearing in about 1994. Some of the more popular robots are the PumaPaint robot, which allows remote users to logon and paint artwork on a canvas using the robot, the Mercury project which allows users to dig into a sand pit and find hidden treasure, and the 'Kehp on the Web' robot which allows users to drive a mobile robot around a pre constructed environment.

Most telerobotic systems operate using a closed loop system, which allows direct continuous control of the robot. The operator has a console, which is remotely attached to the robot or robot controller, with vision feedback or force feedback, or both, sent back to the operator closing the loop. There are other forms of robot control as described by Taylor and Dalton [12].

**Direct continuous control.** Remote robot follows inputs from operator.

**Shared continuous control.** Control is at a higher level, ie the robot may vary from its course if it encounters an obstacle.

**Discrete command control.** Implies a higher level of capability in the remote portion of the controller, as it must be able to carry out a command without help.

**Supervisory control.** The remote device operates in a largely autonomous mode, and only interacts with the operator when a problem is encountered.

**Learning control.** The device can learn from its human operator's commands.

Controlling a robot via the Internet introduces several problems if using the traditional closed loop direct continuous control methods. Firstly, there is limited bandwidth for data transfer, especially if the operators are connected via a modem, which will dramatically impact on services such as video feedback. Secondly, the unpredictable variability in propagation delay causes instability in closed loop systems. The delay will degrade the remote operator's intuition and 'feel' for control. These instability problems are generally overcome in Internet telerobotic systems by adopting a discrete command control or higher level of control that incorporates some form of autonomy.

The goal of this project is to develop a system, which allows remote control via the Internet of an Adept One robot located at the School of Computer Science and Software Engineering at Monash University. The system will primarily be used for educational purposes. It will be a tool, which will allow the CSSE School to teach the Graduate Diploma in Robotics by distance education.

The developed system will attempt to overcome all the issues involved with remote control over the Internet. It is initially intended that the system will involve a high level of autonomous control as with

other Internet Telerobotic systems, and for operators to be able to perform a simple interactive task such as playing checkers. However, this would not be so useful as a tool for distance education, and so a lower level direct continuous control method will be attempted if its success can be foreseen in the time frame of the project. An Internet telerobotic system with a direct continuous method of control will lead to a unique telerobotic system in comparison to other systems so far seen on the Internet.

This thesis first briefly describes some other Internet Telerobotic systems and attempts to identify their respective advantages and problems. A description of the Adept One Robot and its controls, which is used in the project, is then given. Following the description of the Adept One Robot, the requirements of the telerobotic system are outlined, and an architectural overview of the system is presented. A comprehensive description of the implementation and problems encountered during implementation follows the architectural overview, then the performance of the system is evaluated, and conclusions are drawn.

### 3 Some Telerobotic systems

Many of the telerobotic systems found on the Internet have been developed purely for research into the feasibility of Internet based telerobotic systems. Most of these systems are available for the public to use, and that is in fact how the researchers collect data on the systems. Several telerobotic systems that have been implemented, which are relevant to this project, are described below. The advantages and problems of each system are identified.

#### 3.1 University of Western Australia's Robot on the Web

A six axis robot [13] is connected to the Internet at <http://telerobot.mech.uwa.edu.au>. The robot workspace consisted of a table with wooden blocks and cylinders placed on it. These could be moved with the robot's gripper. Users submitted requested moves by filling in fields on a HTML form, or by clicking images of the workspace. The operator's browser submitted the form details as a CGI request to the web server, which received the request, and evoked the CGI script to carry out the requested robot move. Once the robot move was complete, new images are taken of the workspace, and a new form with the latest images and robot position was returned via CGI to the user. Due to the nature of the CGI mechanism, a whole HTML page was returned with each request, a portion of which didn't change between requests. Essentially, much of the data sent back was redundant. A more efficient system could be implemented using a Java interface.

Only one person could control the robot at a time, while other users who tried to gain access received an observers page while the robot is still in use. Each user was automatically logged out after an idle time of 3 minutes. The system also had registered users, which had priority, and could gain control over a guest at any time. More than one move could be specified in a given request using a command script language, allowing faster operation for experienced users.

The server was a PC running the Microsoft Windows 3.11 operating system, and Microsoft Winsock to connect to the Internet. Robert Denny's Win-httpd web server was used. All CGI programs were written in C and executed within a DOS shell launched from a batch file. The reliability of the system was barely satisfactory. This was believed to be mainly due to operating system instability, and the awkwardness of the CGI environment.

#### 3.2 The PumaPaint Project

The PumaPaint project [11] was a web robot that allowed users to create original artwork on the World Wide Web. The robot was a PUMA 760 equipped with a parallel-fingered pneumatic gripper. The robot's work environment consisted of an easel holding up a 30" x 50" table top with the canvas mounted on it. Four coloured paint jars were held in fixed positions beneath funnels, and the paintbrushes were held in

funnels mounted on a wooden platform. The robot was capable of picking up one of the four brushes, dipping it in its respective coloured paint jar, and brushing it onto the canvas. There were two cameras mounted to give the remote user an accurate view of the painting. One camera was situated to view the entire canvas, and the other was mounted on the robot to provide a close up view of the brush contacting the canvas.

The robot was controlled remotely by a Java applet. It was believed that Java could produce a superior client interface that was platform independent. The center portion of the Java interface was a virtual canvas. By clicking, holding and dragging the mouse in this area, the user issued commands to the remote robot to apply paint on the real canvas. The blobs drawn on the virtual canvas contained randomly generated gaps and streaks, and the selected colour progressively decreased as the brush stroke continued, to give the user a realistic impression of what was happening remotely. This also aided in reminding the user to replenish the paintbrush. Buttons appeared on the interface, which allowed the user to select the paint colour. The pressure of the brush on the canvas could be set using a slider at the top of the interface. The amount that a paintbrush is dipped into a paint jar when replenished was adjustable from shallow to deep also using a slider on the interface. Both these settings also had the appropriate simulated effect on the virtual canvas when commands were given.

The interface showed the command number, success flag, and current position, but more interestingly, it showed the difference between the last command sent, and the last acknowledgment received. In practice, this field showed that the robot was almost always behind the user, and sometimes, a long way behind. Some of the lag shown by the difference field was caused by communication delay, but much was caused by the lengthy execution time of some of the robot's commands. It generally took much less time for the user to specify commands, than it did for the robot to perform them.

Two separate windows were part of the interface, each showing a camera view of the robot's artwork. Thus in effect, the interface provided both immediate but virtual feedback using the virtual canvas, and delayed but real feedback using the camera views. The command difference field was effectively a measure of the difference between the virtual and real feedback to the user.

It was noted by Stein [11] that there were problems associated with the platform independence feature of the Java programming language. Each development version of the interface had to be thoroughly tested on different computer platforms to verify its robustness. Very often, interfaces would work fine on one platform, but have trouble on others. In almost all cases, a work around was found to the problems.

### **3.3 Mercury Project**

A group [2] at the University of Southern California developed an Internet telerobotic system known as the Mercury Project. Their idea was to make all robot controls available via the standard WWW 'point and click' mouse commands. The robot used was an IBM SR5427 SCARA robot. The robot's work environment consisted of a tub filled with sand. Buried in the sand were ancient artefacts. Mounted on the end effector of the robot manipulator was a small nozzle that could direct air bursts into the sand. A CCD camera was mounted at the end of the robot manipulator.

The remote interface was developed entirely in HTML. It simply consisted of an image as produced by the CCD camera, and a wire frame 2D birds eye view image of the robot underneath an ISMAP. The operator moves the small nozzle at the end of the robot manipulator to any point in the work space by clicking on the ISMAP. The XY coordinates of the click are sent to the server, and the server moves the nozzle to that location, and blows a burst of air into the sand. If the user is lucky, an artefact may be revealed.

The system architecture at the server was surprisingly complex. It consisted of three communicating sub systems. Server A was the HTTP server, which received the XY coordinates from the client's browser. It also controlled access to the server by verifying usernames and passwords entered by the user with Server B, which was a database server running on the same machine. Server A decoded the XY ISMAP

coordinates and sent them via the Internet to Server C running on a different machine, which controls the robot. On Server C, a custom program decoded the XY coordinates into a robot command, and after verifying that the command is legal, moved the robot to the appropriate position. Once the command was completed, Server C captured a frame from the CCD camera, and compressed the image to a GIF of about 17.3Kbytes. It also generated a new schematic view of the robot to be used for the ISMAP at the interface. Both images were then sent back to Server A, which passed them back to the client's browser in response to the initial request.

Server A was a Sun SPARCserver 1000 running the SunOS Release 5.3 operating system, and Server C was a PC running Microsoft Windows. Server A ran almost faultlessly, but Server C frequently stopped responding to requests and had to be rebooted. This was thought to be due to the lack of multitasking, considering that the Server was running the customer decoding applications and the frame grabber and schematic image construction application all at once. Server C also suffered major delays in responding to requests, hence, adding delays to the final response to the user. The size of the compressed images produced by the CCD camera and compression program also caused problems. Although response times of 10 seconds were achieved on the campus LAN, with much of the delay being the time for the actual robot to move, response times increased to up to 60 seconds for users in Europe across 14.4K modem links.

### 3.4 Keph on the Web

Sieglwart and Saucy [10] developed a modular framework for mobile robots controlled via the WWW. Their goal was for the system to easily enable connecting different robots operating in different environments to the Internet. The web interface for the system consisted of five independent modules. Each of them included a server-side program, and a client side Java applet. The five services were a chat service, a video feedback service, a robot guidance service, a virtual representation service, and a login service. The five services were completely independent. The chat service allowed concurrent users to send messages to each other. The guidance services allowed the user to give high level commands to the robot. The virtual representation service gave the user an immediate simulated view of the robot in its environment. The login service controlled access to the robot, allowing only one person control at a time, but multiple people access to the other services. The video feedback service allowed feedback from different video cameras. Users could switch between the different video sources. The video feedback system used custom server-push software rather than server-push over the web (HTTP). A Java server program sent images in either GIF or JPEG format to the Java applet client. The system achieved 10 to 15 medium quality frames per seconds over a LAN.

The interface systems did not work on Microsoft Internet Explorer, only Netscape Navigator. One of the systems implemented using the modular framework used a more advance login service. Users registered in a timetable in order to plan experiments and have the robot available during a certain period of time. This reservation was available through a separate web page. This system also made available an ISDN link directly to its server such that users could dial in, rather than rely on the Internet to connect. This was obviously seen to be necessary due to the large dataflow between client and server for the 5 services, when trying to maintain little delays in the range required for real time robotics.

### 3.5 Experimental Results with a complex manipulator

Leleve et al [3] realised the significant problem of limited bandwidth and unpredictable and sometimes long communication delays associated with the Internet. These technical restraints result in difficulties for the operator of an Internet based telerobotic system to safely control the remote robot. If the operator has to wait 10 seconds before being able to see the result of their action, it is very difficult to perform the task. The lack of bandwidth also prevents the use of high-resolution real-time video of the experimentation site. Their idea to overcome these restraints was to simulate instructions at the operator's client with a virtual

reality model before sending the instructions to the remote robot.

The client acted as a virtual reality man-machine interface. Within the client were a simulation system and a prediction system. The predictor in the client was regularly refreshed by data feedback from the remote robot. The client was able to make simulations and display a prediction of the remote system state before actually receiving real data feedback from it through the Internet. For the simulator and predictor to work properly together, it was necessary for the delay between the simulator receiving operator commands, and the predictor receiving feedback of the results of those commands to be constant. To compensate unpredictable varying round trip times associated with the Internet transmission link, 'Delay Variation Compensators' (DVC) were introduced. They stacked incoming samples as they asynchronously arrived; meanwhile they unstacked these samples at a constant rate, but delayed. The DVC's made it possible for the predictor and simulator to work properly. Network simulations were carried out to determine the exact sampling rate and delay required for the DVC for their particular project.

At the time the paper [3] was published, the system was not fully implemented, and so there were no experimental results available.

## 4 Problem Description

### 4.1 The Adept One Robot

The robot used in the telerobotic system is an Adept One robot. It is an industrial manipulator with 4 degrees of freedom that can have a variety of tools attached to its end-effector. A schematic view of the robot is presented in Figure 1 [1].

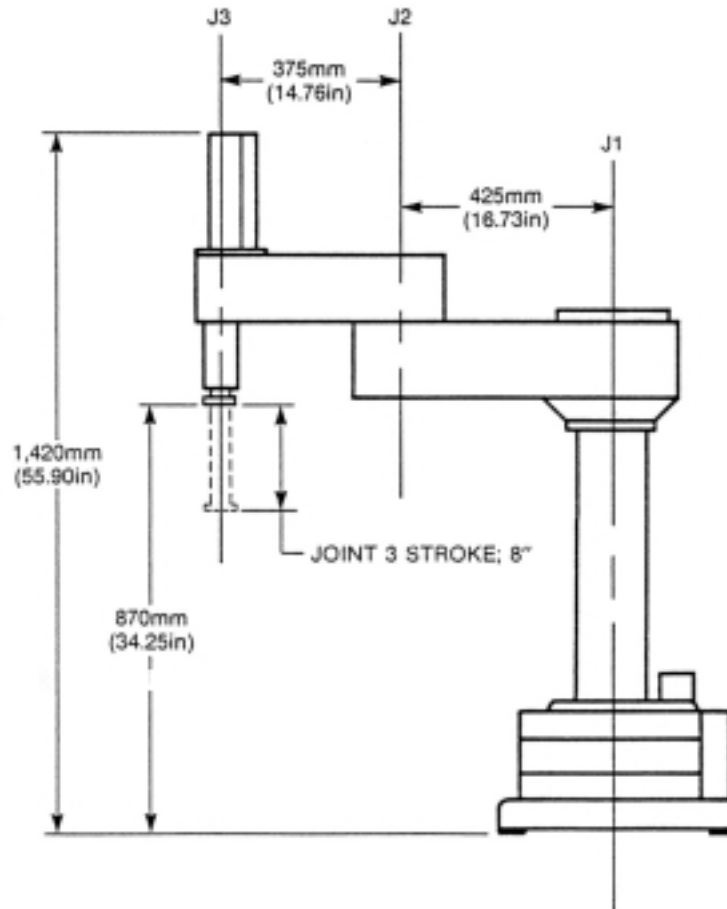


Figure 1: The Adept One Robot

Joint one is a rotation around the robot's vertical mounting column. Joint 2 is a rotation in the middle of the horizontal arm. Joints 1 and 2 together provide for all of the horizontal position of the manipulators end effector. Joint 3 is a linear joint, which extends and retracts vertically at the end of the horizontal arm. Joint 4 is a rotation of the end effector. The Adept One manipulator used in this system has a gripper mounted at the end effector, which can be opened or closed.

The manipulator is connected to and controlled by a computer control system. It provides two methods of control. The first is via a manual control pendant, which allows the user to have full manual control of the manipulator, as well as limited programming capabilities. Using the manual control pendant, the operator can move the manipulator into positions, and store those positions which can be 'played back' at a later time during execution of a program. The manual control pendant can be used to start and stop such application programs. The manual control pendant also provides manipulator position feedback via a LCD panel.

An ASCII terminal is connected via a serial port to the robot controller. This provides the second method to control the manipulator. Using the terminal, a user can write and execute programs written in the Adept VAL II programming language. The programs can instruct the manipulator to do anything, and even seek user interaction in decision making. The robot controller also implements its own file system where programs can be read from or saved to a hard or floppy disk.

The computer system has five serial ports that can be controlled via VAL II programs. These are intended for such purposes as controlling conveyor systems in sync with manipulator movements, or taking feedback from sensors placed in the robot's environment.

## 4.2 Controlling the Adept One Robot

Each of the robot's four joints can be moved individually, or together to move the end-effector to anywhere in the work envelope. The work envelope is shown in Figure 2 [1].

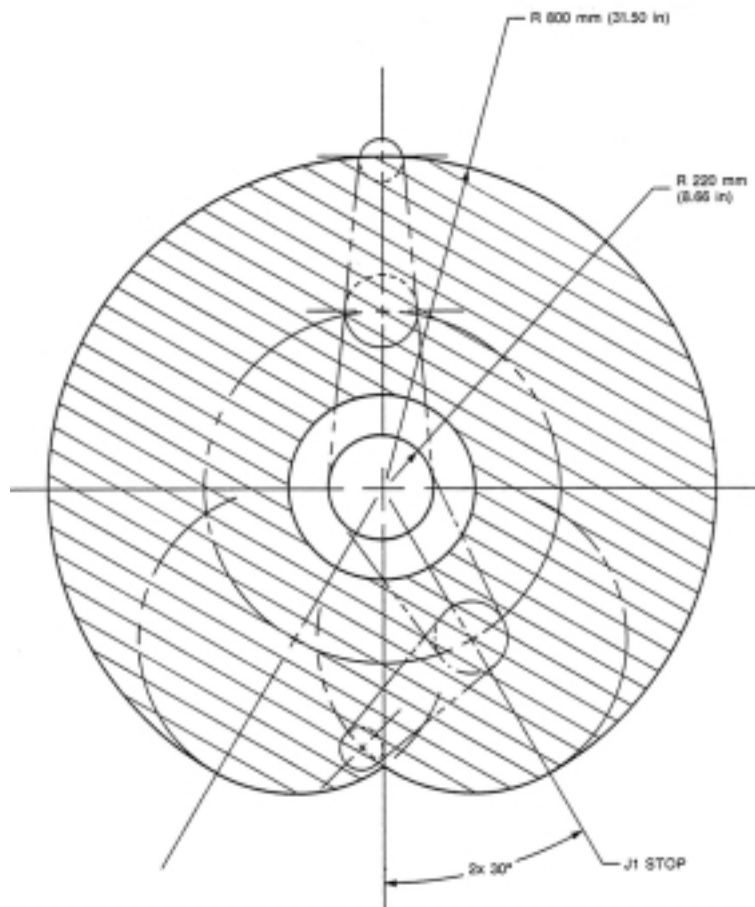


Figure 2: The Manipulator Work Envelope

### 4.2.1 Manual Control Pendant Control

The manual control pendant provides 3 modes of operation significant to this project; world mode, joint mode and tool mode. World mode gives the user the ability to move the end effector to any position specified by world coordinates. Moving the end-effector in world coordinates will result in several of the

manipulator joints moving simultaneously to move the end effector to the correct position. Joint mode gives the user the ability to gain control of any of the 4 specific joints individually. In this case there is no emphasis on the resulting position of the end effector. Tool mode is very similar to World mode, except the coordinate system is not fixed in space. Instead all axis of the coordinate system are relative to the end effector orientation. Of course, each of the modes allows the user to control the end effector tool, which in this case is a gripper, which can be opened or closed.

#### 4.2.2 Terminal Control

The VAL II programming language implements a set of commands that allows the manipulator joints to be controlled individually, or for the end effector to be moved to any specific location in the manipulator work envelope. The language implements several standard flow of control commands such as FOR, IF/ELSE, WHILE. It provides stdin and stdout streams to the user terminal, and input/output control of the system serial ports. Programs are written using a simple editor. An example of a program is shown in Figure 3. This program simply instructs the manipulator to move a stack of blocks.

```

SET start = TRANS(622,155,812,0,180,150)
SET finish = TRANS(530,-340,732,0,180,60)
;setup start pos
SPEED 50
; begin building
OPENI
FOR row = 0 TO 4
    FOR column = 0 TO 2
        SET relstart = SHIFT(start BY column*26,0,-row*20)
        SET relfinish = SHIFT(finish BY 0,-column*26,row*20)
        APPRO relstart,safeheight
        MOVE relstart
        CLOSEI
        DEPARTS safeheight
        APPRO relfinish,safeheight
        MOVE relfinish
        OPENI
        DEPARTS safeheight
    END
END
STOP

```

Figure 3: An example VALII program

### 4.3 Requirements of the Telerobotic System

The initial goal of the project was to give high level autonomous control of the manipulator with remote operators being able to perform a simple interactive task such as playing checkers. During early stages of development it was realised that a system involving direct continuous control, which is more suitable if the system is to be used for distance education, may be possible. The approach taken in the design was changed so that operators have direct continuous control of the robot.

The purpose of the system was to give total control of the manipulator to operators via the Internet. As the primary use of the robot is for teaching, it was necessary to give the remote operators a similar

control interface remotely, as they would normally get at the robot. If the remote interface is sufficiently similar, then students who learn control of the robot remotely will have little problem in their transition to controlling the robot locally.

It was a requirement that a remote operator must have a programming interface like the local terminal, and a manual control interface like the manual control pendant. The system had to provide vision feedback to remote operators with a minimum of delay, even over low bandwidth connections. Remote operators must login to the control server, and only one operator was to be able to login at any given time, preventing conflicting control of the robot. However, it had to be possible for multiple people to view the vision feedback simultaneously even though not logged in or having control of the robot.

Data flow between the robot and the remote client had to be minimised so that satisfactory control of the robot, and vision feedback could be maintained over low bandwidth connections. It was also desirable that multiple vision feedback streams of different quality are available at the server, so the remote user could choose depending on the bandwidth of the Internet connection they are using. The system was to be of modular design. Image feedback and control sub systems were to remain separate to allow for changes, upgrades, and extensions to the software to be easily made.

The system had to be very robust, requiring little or no administration. It had to be designed such that it would never be harmed or crash due to unpredictable time delays through the Internet communication link. The system would remain active for long periods of time with many remote operators logging on and off. Remote operators had to be able to recover from physical robot crashes without any local interaction with the robot. The control clients developed must work on all computer and operating system platforms.

## 5 Implementing the Telerobotic System

### 5.1 Overview of the Telerobotic System Architecture

The architectural design of the telerobotic system is presented in Figure 4 . The link between the robot controller and the Internet is a PC running the Linux operating system. Linux was chosen because of its superior stability over Microsoft Windows operating systems. One of the PC's serial ports is connected to the robot controller ASCII terminal serial port. The PC is connected to the Internet via a network interface. A camera is mounted above the robot work space, and is connected to a Buz video capture device in the PC.

A control server program runs on the PC, which accepts TCP/IP connections from control clients via the Internet. The control server program sends commands to the robot controller, and returns responses from the robot controller to the remote control client. A vision server program also runs on the PC, which accepts TCP/IP connections from vision clients.

### 5.2 Control System

#### 5.2.1 Overview of the control system architecture

The control system consists of a server, and two different clients as shown in Figure 5. One client is a graphical manual control pendant, and the other client is a text programming terminal. The graphical manual control pendant talks to the control server on port 11001, and the programming terminal client connects to the server on port 11000. The server passes on commands to the robot controller via an RS232 serial line.

#### 5.2.2 Control Server software

The server program was written in the Java programming language. Java was chosen due to the simplicity of Internet socket programming. The server takes as parameters on start-up, a serial port number, and

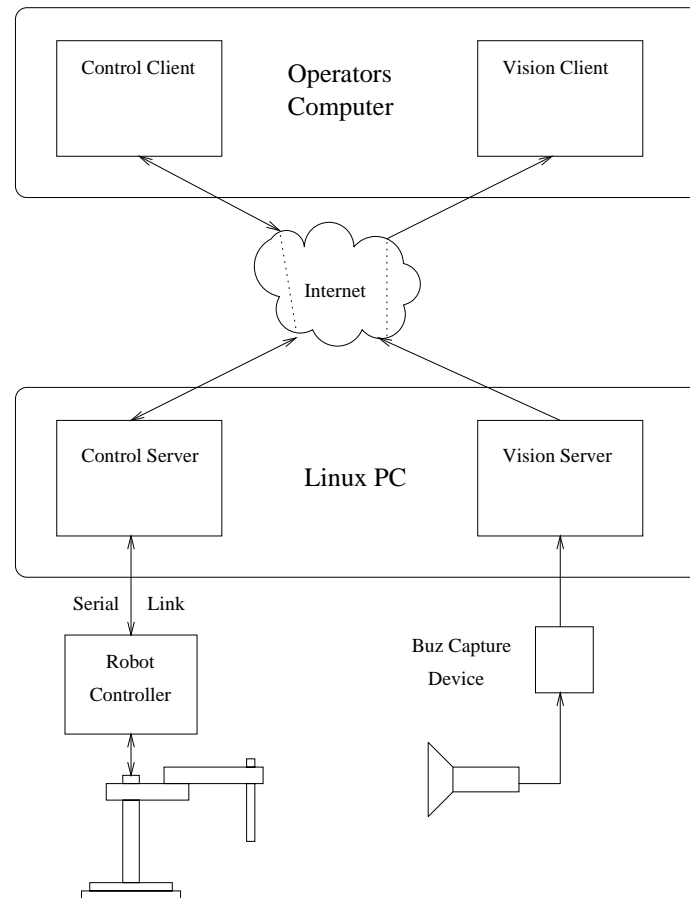


Figure 4: Telerobotic System Architecture

port numbers for manual control pendant and programming terminal connections. The server sets two threads running, one listens on the manual control pendant port, and the other listens on the programming terminal port. A class diagram and class descriptions for the main telerobotic system classes in the server software are presented in appendix C.

Each of the server threads creates a connection object when a connection to the port is established. Each connection object first checks to see if another connection object already exists, ie, if a control client is currently connected to the server. If another object exists, the new connection object sends a 'connection refused' message back to its client, and immediately destroys itself, preventing more than one clients having access to the robot at any one time.

The two threads listening on the different ports create different kinds of connection objects. The thread listening on the terminal port creates a connection object, which uses a communication protocol that is consistent with the TCP/IP application level protocol TELNET [8, 6, 7]; hence, it is effectively a Telnet server. This makes life very easy since any Telnet client can be used as a programming terminal control client for the telerobotic system. The thread listening on the manual control pendant port creates a connection object, which uses a communication protocol that was specifically devised for this system.

Both types of connection objects have the same base functionality. They both require the client to login otherwise the connection is destroyed, and they both enforce a login time limit and a maximum idle time limit before the connection is destroyed. Both these time limits are specified to the server as parameters to the server on start-up.

Each established Telnet connection object simply writes character at a time, input it receives from its

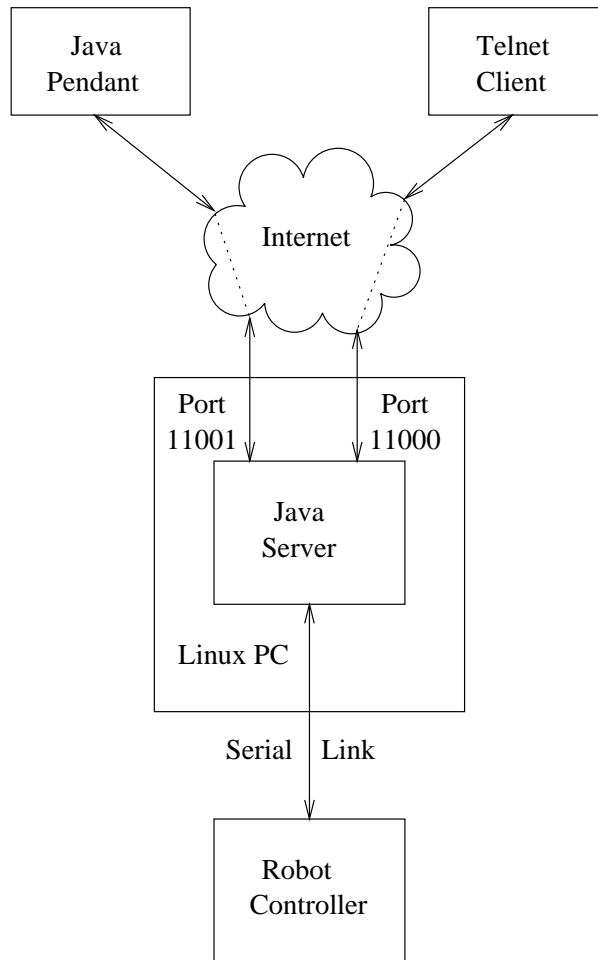


Figure 5: Command Sub-System Architecture

Internet socket to the serial port stream. It also writes character at a time, any input from the serial port stream back to the Internet socket stream. Since the serial port of the Linux PC is connected to the ASCII programming terminal port on the robot controller, the remote Telnet client behaves in an identical manner to the ASCII programming terminal connected locally at the robot controller, except with the added functionality of authentication and maximum login and idle times, and the delay associated with the Internet link.

In a similar fashion to the Telnet connection object, the manual control connection writes input from the Internet socket to the serial stream, and input from the serial stream back to the Internet socket. It does this however line (or command) at a time. It also intercepts data from the serial stream before sending it to the Internet socket, to see if the command from the operator resulted in the robot controller producing an error.

Data flow between the manual control client and the server is shown in Figure 6. If the robot controller produced an error, the appropriate error code is sent back to the client. The connection object then waits for acknowledgment from the manual control pendant client that it has received the error. Due to possible lengthy Internet delay, and the time the manipulator takes to execute some of the commands, many commands may have been sent from the client before an error is detected and sent back to the client. All the commands between the command that produced the error, and the clients acknowledgment of the error received by the server, are ignored.

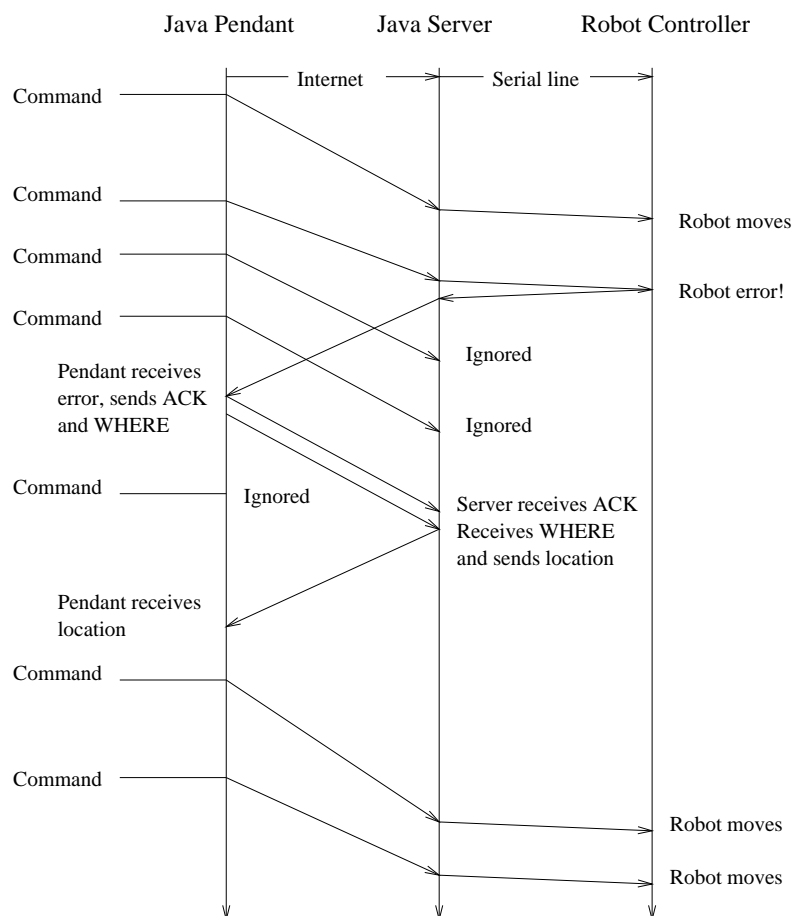


Figure 6: Data flow between pendant and control server

The server appeared to occasionally fault. The problem was thought to be caused by the browsers that were running the control client applets. When the control client applets are stopped and closed, the browsers appear to sometimes keep the Internet socket to the server open. In this case, the server would think that the client is still active when it is not, and would unnecessarily refuse access by a new client.

The problem was overcome by implementing an 'are you there' command (AYT) in the server and client, exactly as used in the Telnet Protocol. When a new connection is established to a client and a connection already exists to another client, the server sends an AYT to the existing client to see whether it is still active. If this doesn't result in a socket error, the client is assumed to still be active, and the new client is refused connection. Otherwise, the old connection is destroyed, and the new connection is given control to the robot. The control client ignores all AYT messages it receives.

However, there still seemed to be a problem. Dead connection objects in the server were not destroyed after clients had disconnected. The threaded connection objects seemed to exist even after being explicitly stopped. This eventually led to the server locking up and refusing any new connections. After some re-engineering of the software, which included explicit calls to the Java garbage collector, the problem seemed to be solved. Connections can be continually made to the server then destroyed and all connection objects are garbage collected as shown by the server terminal.

### 5.2.3 Manual Control Pendant client software

The manual control pendant client is a Java applet, and is intended to be run within a web browser. The applet must be run with username, password, server host and server port number parameters. It has much of the functionality of the Adept manual control pendant. A screen shot of the pendant applet is shown in Figure 7. A class diagram and class description for the main telerobotic system classes in the pendant software are presented in appendix D.



Figure 7: Control Pendant Applet

The single line text box displays manipulator location information and error feedback. Below the display are 3 mode buttons; world state, joint state, and tool state. These place the pendant in the desired mode of control. Below the mode buttons are 7 buttons which allows the operator to select either the axis, or the joint that they which to move. The pendant provides + and - action buttons to move the selected axis or joint, and a slider to determine the step size that is produced by each action button click. There is a logoff button, which closes the applet, and a reset button, which will reset the manipulator after it has suffered a major error resulting in disarming the manipulator.

Pressing either of the action buttons will send a VAL II command to the server. The command and its arguments that the pendant sends are determined from the mode that the pendant is operating in, and the axis/joint that the operator has chosen. The pendant client calculates and maintains the manipulator position. When the operator issues a command, the manipulator position displayed in the text box is updated, then the appropriate command is sent to the robot. The pendant is essentially providing immediate but virtual feedback of the manipulator's new position.

The pendant assumes that every command sent to the server is successful, and that the manipulator's movement is accurate. Data flow between the control applet and the command server is shown in Figure 6. If an error occurs at the robot, the pendant is informed, and sends back an acknowledgment to inform the server that the error has been received. The pendant then immediately sends the VAL II command "WHERE" to the server requesting for the manipulators exact position. The response from the server is used to update the pendants location information. The pendant will not accept any input from the operator after it has been informed of an error, and before it receives manipulator location information from the server.

The work envelope bounds of the Adept One robot are hard coded into the pendant. This means that if the operator makes a movement request that would result in the robot controller returning a 'location out of range' error, the operator is informed immediately and the command cancelled, rather than sending the command, then having the control server response with an error, and the client having to acknowledge the error, and request the manipulators current location, and await that information until accepting any

more user input. The later would result in a lengthy delay, especially over low bandwidth connections.

Due to Java being a platform independent language, it was assumed that the control pendant client would work within all Java Virtual Machines, ie, within all Internet browsers. However, this was not the case. The pendant was initially tested during development using the Linux Java Development Kit appletviewer program. Once completed, it was tested within different browser JVM's, but did not work within any of them. Most of the problems were tracked down and thought to be related to the controlling of threads. Eventually, changes were made and a work around was implemented in most cases that seemed to fix all the problems.

### 5.3 Vision System

Two systems were engineered and tested. Firstly, a system using existing streaming software from Real Nwtworks [9] was implemented. It was initially thought that the software would satisfy all of the vision feedback requirements. Real Networks provide a free data encoder, server, and client, known as Realproducer, Realserver, and Realplayer respectively.

The Real Networks system attempts to provide a mechanism for delivering real-time video and/or audio over the Internet. As a shared datagram network, with its unpredictable delays and guaranteed arrival time of packets, the Internet is not a suitable medium for real-time data transfer. The Real Networks system is based upon the Real-time Streaming Protocol, which attempt to guarantee that most packets will arrive within a fixed, short period of time, providing the ability for continuous playback at the client with correct timing and synchronisation.

The Real-time Streaming Protocol (RTSP) is a client-server application level protocol, which enables controlled delivery of streamed data over an IP network. It is designed to work with three lower level protocols; RTP, RSVP and RTCP. The Resource Reservation Protocol (RSVP) [4] is a control protocol that allows client/server systems to request an end-to-end quality of service. Real-time applications use RSVP to reserve necessary resources at routers along the transmission paths so that the request bandwidth is always available. The Real-time Transport Protocol (RTP) is an IP based protocol for providing support for real-time data. It is designed for multicast of real-time data, but can be used in unicast. RTP is designed to work in conjunction with the auxiliary Real-time Control Protocol (RTCP) to get feedback on quality of data transmission. The RTP periodically sends RTCP packets to convey feedback on quality of data delivery.

A schematic view of the real-time vision system is shown in Figure 8. Realproducer and Realserver are executed on the PC. Realproducer encodes input from the video capture card in real-time into several forms suitable for high or low bandwidth transmission. Realserver broadcasts the streams over the Internet to a maximum of 25 clients (a commercial license for Realserver will remove the 25 client restriction). Computer systems on the Internet with the Realplayer client can then connect to the vision server host, and watch the manipulator move in real time. Each Realplayer client will choose the appropriate encoded stream depending on the bandwidth connection it has to the Internet.

Unless a commercial license for Realproducer is purchased, it only encodes output from the capture device into a format that later version Realplayer clients can decode. However, later version clients were not available for Unix platforms at the time this system was being developed. Due to this lack of availability, it was necessary to attempt to engineer another vision feedback system that would work on a wider variety of client platforms.

The second vision system implemented used HTTP server-push. Basic HTTP is a stateless protocol. A request is made, the response is provided, and the connection from client to server is closed and forgotten. A system could be devised were the client periodically requests an updated image of the manipulator, and each response is written over the previous response. If done quickly enough, the resulting image may seem to be in motion. This however is very inefficient due to the length and unpredictable propagation delay of each request and response from client to server, and is unlikely to be very successful on low bandwidth

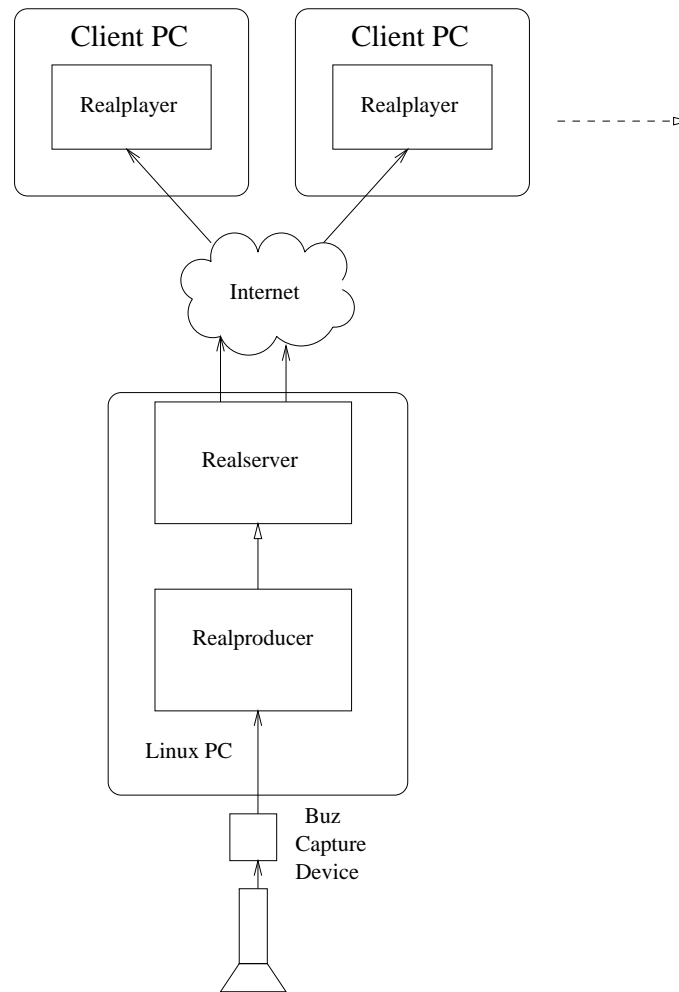


Figure 8: Real-time Vision System

connections.

The HTTP standard includes an experimental response content type: `multipart/x-mixed-replace` [5]. If a client makes a request, and this content type is found in the header of the response, the client will read the data in the response, then keep the connection open and wait for more data. When more data is received, it replaces the previous data with the new data, and so on. This provides a very neat way for the client to make only one request to the vision server, and the vision server periodically sending back updated images of the manipulator. This system is more efficient since the effective propagation delay of getting an updated image is essentially halved, since after the first request, the client doesn't have to further request updated images.

A schematic view of the server-push system is shown in Figure 9. There are three parts to the server-push system; the grabber, image CGI, and client HTML. The grabber is a program written in C, which continuously syncs on the Buz image capture device buffer, and writes the image in JPEG format to a temporary file. A pointer is kept to the temporary file, and each time an image is written to the file, the file pointer is rewound to the beginning of the file, rather than the file begin continuously opened and closed. This method then relies heavily on the Linux operating system disk caching which seems to work very well, as proved by very little hard disk activity.

The image CGI program, also written in C, runs behind an Apache web server. The CGI program takes

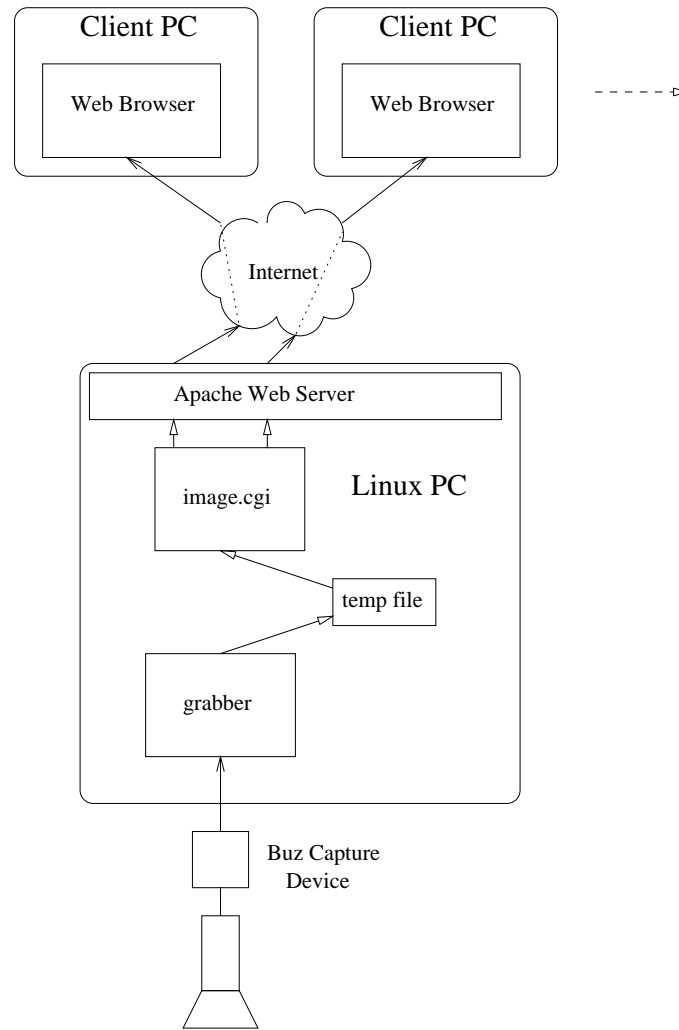


Figure 9: Server-push Vision System

a refresh interval in seconds as a parameter. It responds to a request by first returning a content type header of multipart/x-mixed-replace, then continuously reads in from the temporary file written by the grabber, and returns the JPEG image, with a pause between returning each image as determined by the refresh interval. The client is simply a HTML web page that includes an image with its source being the image CGI program.

The grabber program was compiled with a quality setting in the driver of 15, with a maximum of 100 possible. This was judged to give the optimum quality to image size ratio. The size of each image produced by the grabber on this quality setting was 8752 bytes. This in theory allows 26.3 images per minute across a 28.8Kbps modem link, or an image every 2.3 seconds.

Unfortunately, Microsoft Internet Explorer does not implement the experimental multipart/x-mixed-replace content type, so the server-push system will not work on one of the most common web browsers. It is thought in the future that a Java applet could be written for the client that recognises the experimental content type, and can display the incoming data on any browsers that implement a Java Virtual Machine (JVM).

Shortly before this thesis was published, newer versions of the Realplayer client became available for Unix platforms that are compatible with the free Real Networks server and producer software used in this

project. This allowed remote operators on Unix platforms to take advantage of the real-time vision system.

#### 5.4 Installation of the Software

The PC used in the project is a Pentium III with 128 Megabytes of RAM. The Java control server program, and the server push image grabber program are executed in the background on the PC. The control server is configured to accept Telnet connections on port 11000 and manual control pendant applet connections on port 11001. An Apache web server is run to provide a front for images to be requested from the image CGI program. The Java manual control pendant applet is served by another web server application known as Zope. Zope provides a powerful framework for generating dynamic HTML. The main robot control web site has a form allowing the operator to enter in their username and password to gain control of the manipulator. The form is then submitted to a dynamically generated HTML page within Zope, which dynamically assigns the username and password to parameters within the embedded Java applet.

Another dynamic HTML page is served by Zope to provide the server-push image HTML web page. Below the image on the HTML page is a selection box, which allows the user to select a refresh rate. Upon changing the refresh rate, the form is submitted back to the same HTML page within Zope, which dynamically inserts the refresh rate as a parameter to the image CGI program in the SRC tag of the embedded image.

The Real Networks Realproducer and Realserver also executed on the Linux PC. Realserver accepts connections by default on port 443. The Realplayer client can be downloaded free of charge from the Real Networks web site [9]. Up to 25 Realplayer clients can connect to the vision server simultaneously.

More details on operating the telerobotic system can be found in appendix A. Other important system information is found in appendix B.

## 6 Performance and Usability

The approximate delay for video feedback across various bandwidth transmission links to the Internet for both vision feedback systems is shown in Table 1. The delay is measured from the time the input command is completed at the command client, to the time the robot is seen to move at the vision feedback client.

The theoretical maximum refresh rate is every 1.1 seconds and 2.3 seconds for the 64KBps ISDN and 28.8KBps modem connections respectively. A refresh rate of 3 seconds was found to be optimum by experimentation on the ISDN and modem links. A shorter refresh rate would result in saturation of the connection, dramatically increasing the propagation delay of control commands to the robot, and a longer refresh rate would simply result in longer delay in the vision feedback. The server-push delays in table 1 were measured with a refresh rate of 1 second across the LAN, and 3 seconds across the ISDN and modem links.

System	LAN	64KBps ISDN	28.8kBps Modem
Server-push	1 Sec	5 Sec	12-15 Sec
Real-time	8 Sec	8-10 Sec	8-10 Sec

Table 1: Approximate Vision Feedback Delays

The real-time vision system produced far better quality vision than the server-push system, which makes it most suitable for operators that are watching the execution of a program they have written using the programming terminal. Delay in the vision feedback isn't so important when watching programs execute, but quality and smoothness of the vision are. However, a short delay is more important when operators are controlling the robot using the manual control pendant. The server-push system has a very short delay

over a LAN, which makes it more appropriate for control with the manual control pendant over a LAN, but a longer delay than the real-time system over the slower 64KBps and 28.8KBps connections.

Using Telnet clients as remote programming terminals to the robot seemed to be very successful. Coupled with real-time vision feedback, operators are able to design and test programs remotely and see the results. Programming the robot remotely is no more or less practical than programming locally, except human interaction with the robot's environment is impossible, and depending on the programming task, the operator may have to rely on the work environment being previously prepared. The real-time vision system provides adequate vision feedback for watching the execution of programs. Remote operators using the terminal control did however comment that there was no online help via the Internet for the VAL II programming language. Normally students have access to the Adept VAL II reference within the robotics lab.

Controlling the robot remotely using the Java control pendant is significantly less practical than controlling the robot locally with the control pendant. The Java control pendant doesn't allow the same adaptiveness to a situation, especially without instant vision feedback. All control is incremental and can become very tedious. Any task which normally involves human interaction is obviously impossible, and tasks, which are less than trivial, are very time consuming to perform.

With low quality vision feedback from only one camera, it is very difficult for the remote operator to gain precision. Low quality vision also makes it very difficult for remote operators to recover from their own errors, which will often make it necessary for human intervention at the robot. For it to be possible for specific tasks to be performed remotely, it is necessary for the robot's work space to be manually prepared before hand so that objects are at known locations. If multiple people are to carry out experiments, each operator in turn must leave the work space exactly as it was initially so that the next operator can perform the exercise. It is possible for a remote operator to gain a good understanding of the robot, and carry out simple tasks if the work environment is previously set up for activities.

In general students were very happy with being able to use the system after hours from remote locations such as their homes to compliment the practise they were obtaining in official lab classes. Unfortunately, there were no students that tried the telerobotic system who hadn't previously operated the robot from within the robotics lab, that could evaluate the telerobotic system before this thesis was published.

The system as a whole is not particularly reliable. It was difficult to test how long the system could run unattended since there are many power outages at Monash, which reset the robot controller, and PC. However, even between power outages, the robot controller itself seemed to occasionally lockup (about once every two weeks), rendering the system inoperable. The control server software running on the PC is not able to detect when the robot controller operating system has crashed, hence, its behaviour is undefined, and generally results in the clients connected to the server locking up.

Platform independence of the Java control pendant client still remains a problem. After many attempts to solve problems unique to specific platforms, it still does not work on at least one platform and browser tested. That was Netscape on a Silicon Graphics Indy running the Irix operating system. It does however work on all major browsers and platforms in use today including Netscape under Linux and both Netscape and Microsoft Internet Explorer under Microsoft Windows.

## 7 Conclusion

### 7.1 General Conclusions

The initial goals of developing an Internet telerobotic system to give operators autonomous control of the robot in performing simple interactive tasks were exceeded. The developed system allows direct continuous control of the robot. Students can log in using either a Telnet connection to gain terminal control, or from the web using a Java control pendant. The system successfully limits access to control the robot to

one user at a time, and allows a maximum of one hour of control before automatically disconnecting the operator.

Two vision feedback systems were implemented: a real-time system using software from Real Networks, and a server-push system. The real-time vision system provides significantly higher quality vision feedback than the server-push system, but at the cost of a much longer delay over high bandwidth connections such as a LAN. However, over low bandwidth connections such as a modem connection, the real-time system delay is less than for the server-push system. The real-time vision system is the preferred vision system for both control methods over all types of links except for manual control pendant control over a high speed link such as a LAN. Although having lower quality vision, the server-push system is preferred in this case because it has a very short delay which is important when the remote operator has direct control of the robot.

The terminal control is very successful. Students can write programs and test them remotely, exactly as they would locally at the robot. Students receive high quality vision of the robot using the real-time vision system. The real-time vision system is preferred over high and low bandwidth connections since delay isn't important to operators watching the execution of programs.

The Java control pendant demonstrates the problems and impracticalities with Internet telerobotics. The Java control pendant mimics as close as possible the real control pendant of the Adept One robot, so similar actions are required to command the robot. However, the Java control pendant doesn't give as much 'feel' to control of the robot as the real control pendant does. This is due to the 'point and click' nature of the web interface, as well as the delay in the vision feedback. Even though the server-push vision system produced lower quality vision than the real-time vision system, it is more appropriate over high bandwidth connections when using the manual control pendant due to its shorter delay. However, over low bandwidth connections, the real-time vision system remains the more appropriate vision system.

## 7.2 Limitations and Further Work

In its entirety, the telerobotic system is not particularly reliable. The majority of the problem lies in the instability of the robot controller operating system, and the power failure frequency at the University. Neither was in the scope of this project. A crash in the robot controller operating system will render the system inoperable and unrecoverable either remotely or locally, requiring a reboot. After fixing initial bugs in the Java control server software, it didn't seem to contribute to the instability of the system. On average, the system needed to be rebooted about once every 2 weeks. In order to eliminate the need for human intervention, a watchdog system could be implemented that detects when the robot controller itself has crashed, and reboot it, or at the minimum, send an e-mail to an administrator.

The Java control pendant interface caused many problems during development. It was very difficult to develop the applet such that it was totally platform and browser independent. Eventually it was successfully executed on all major platforms and browsers except for Netscape on the Silicon Graphics platform running the Irix operating system. Due to the platform independency problems encountered, it cannot be guaranteed that the applet will work within new browser releases. Future work could be done to refine the applet and attempt to develop it into a truly platform and browser independent applet.

The real-time vision system implemented requires the remote operator to have the Realplayer client, version 6 or greater. At the time this thesis was published, Realplayer version 7 was available on Windows and most Unix platforms. However the server-push system is the preferred vision system for remote pendant control of the robot over high bandwidth connections. Microsoft's Internet Explorer doesn't support the experimental content type that is required for the server-push system to function; hence, Internet Explorer users cannot utilise the server-push vision system. This is not seen to be a huge problem since the Netscape Navigator browser, which supports server-push, is freely available by download. However, a Java applet could be devised in the future that will run within the Internet Explorer JVM and work as the client in the server-push system. This is not believed to be too hard a task, as the applet only needs to understand the

one HTTP content-type. The Java vision client could then be used within all browsers that implement a JVM.

The complexity of the task an operator can attempt to execute remotely using the Java control pendant is limited. Any task which is less than trivial is extremely tedious to perform. With low quality vision feedback from only one camera, it is very difficult for the remote operator to gain precision. Vision feedback could be improved by adding another camera to the system. However, if the operator receives both camera views simultaneously, two vision data streams need to be transmitted over the operators link to the Internet, which would not be feasible over a standard modem connection. A vision server could be implemented that allows the operator to switch between camera views so that only one vision stream is transmitted over the operators link at a time.

Alternatively, a vision system could be implemented that gives the operator control of the cameras orientation and zoom. This would allow the operator to move the camera and zoom in when extra precision is needed when manoeuvring the robot. Such a system would be complex to implement since the camera would need to be replaced with a more complex camera then connected to the PC via a purpose built hardware interface, and server/client software would need to be implemented to control the camera. It would be a small telerobot system in itself.

Currently the Java manual control pendant gives the remote operator a similar feel for control as the real Adept manual control pendant. The feel for control could be improved by careful enhancement of the Java interface, specifically to make it look and function more like the real control pendant. However, only small improvements would be made in making control less tedious. A different approach could be taken to the Java interface design that moves away from trying to mimic the real Adept manual pendant to an interface more suitable for the Internet. Such an interface could involve the use of a schematic 2D view of the robot's work space which would allow the operator to simply point and click on a position to command the robot to move to the position. This would make larger tasks much easier to perform, but still allows the student operator to gain some understanding of the robot control.

Within the robot work space is a motorized bowl feeder. It is not controlled with the robot controller, but instead manually switched on if it is to be used in experiments. Operators using both the Java manual control pendant and the programming terminal commented that the bowl feeder cannot be operated remotely. A relay could be used to switch the bowl on and off that is connected via a simple hardware interface to one of the auxiliary serial ports on the robot controller. Remote operators using the Telnet terminal control would then have direct access to control the bowl feeder with VAL II commands. A simple addition to the Java manual control pendant would allow operators using the pendant to control the bowl feeder.

A limitation with both methods of remote control of the robot is that the work environment must be previously setup so that exercises can be performed. Setting up the environment normally involves human interaction each time. To help avoid the need for human interaction, several locations in the work space could be predefined in the system. Such a predefined location could be directly above the bowl feeder. Predefined locations such as this would make it easier for remote operators to leave the work environment in the same state as it started off, which will make life easy for the next operator. The predefined locations would be easily accessible from the Telnet programming terminal by using variables, but several simple additions would have to be made to the Java manual control pendant for the operator to be able to directly move the robot to the predefined positions.

Remote operators using the Telnet programming client do not have access to the VAL II programming reference guides as they normally would if operating the robot from within the robotics lab. Before the system can be used for distance education, relevant sections of the Adept VAL II reference guide should be published on the web so that remote operators have access to it.

In general, future work could be done in developing the system to be robot independent. It is anticipated that another robot within the School, the Fanuc, will need to be connected to the Internet to allow remote operation as part of the Schools strategy to offer the Graduate Diploma of Robotics by distance education.

The Fanuc is a 6 axis robot, and is more complex to control. It will be necessary for the vision feedback system to be significantly improved for remote operation of the Fanuc to be successful. It is likely that remote operation of the Fanuc with the Telnet programming control system will work without modification to the control server software. However, small changes to the control server software, and the Java manual control pendant will be needed in order for the Java control pendant to be used for remote operation of the Fanuc.

## References

- [1] Adept Technology Inc. Adept manipulator system: Basic operation, version 1.5, Feb 1985.
- [2] K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley. Desktop teleoperation via the world wide web. In *IEEE International Conference on Robotics and Automation*, May 1995.
- [3] A. Leleve, P. Fraise, P. Dauchez, and F. Pierrot. Teleoperation through the internet: Experimental results with complex manipulator. In *30 th International Symposium on Robotics*, pages 63–70, Tokyo, Japan, October 1999.
- [4] C. Lui. Mutlimedia over ip: Rsvp,rtp,rtcp,rtsp. <http://www.cis.ohio-state.edu/~chu/ipmultimedia>.
- [5] Netscape Communications Corporation. An exploration of dynamic documets. [http://home.netscape.com/assist/net\\_sites/pushpull.html](http://home.netscape.com/assist/net_sites/pushpull.html).
- [6] J. Postal and J. Reynolds. Rfc855: Telnet option specifications. <http://www.cis.ohio-state.edu/htbin/rfc/rfc855.html>.
- [7] J. Postal and J. Reynolds. Rfc857: Telnet echo option. <http://www.cis.ohio-state.edu/htbin/rfc/rfc857.html>.
- [8] J. Postal and J. Reynolds. Rfc854: Telnet protocl specifications. <http://www.cis.ohio-state.edu/htbin/rfc/rfc854.html>, May 1983.
- [9] <http://www.real.com>.
- [10] R. Siegwart and P. Saucy. Interacting mobile robots on the web. In *ICRA'99*, Detroit, May 1999.
- [11] M. R. Stein. Painting on the world wide web: The pumapaint project. <http://yugo.mme.wilkes.edu/~ illano> .
- [12] K. Taylor and B. Dalton. Issues in internet telerobotics. In *International Conference on Field and Service robotics*, Canberra, Australia, December 1997.
- [13] K. Taylor and J. Trevelyan. A telerobot on the world wide web. In *National Conference of the Australian Robot Assocaiation*, Melbourne, Australia, 1995.

## A How to use the System

### A.1 The Server

To start the robot control server software, execute the start up script `/usr/bin/robotserver`. It runs the server application with all the appropriate arguments. Edit the script if the default arguments need to be changed.

To start the real-time vision server, execute the script `/usr/bin/robotvision`. This script will start the Realproducer encoder and the Realserver. Realproducer will popup a window that allows the administrator to start the encoding process and change the default settings.

Alternatively to the real-time vision server, the server-push server can be started by executing `/usr/bin/grabber`.

### A.2 Remote Clients

To connect to the control server with a Telnet client, connect on port 11000. To connect to the control server with the Java control pendant, go to the URL `http://a12.csse.monash.edu.au/robot/` using a Java enabled browser. Doing this will also result in a separate browser window popping up and automatically connection to the server-push vision server.

You can use Realplayer version 6 or greater to connect to the real-time vision server by connecting to the URL `rtsp://a12.csse.monash.edu.au/encoder/output.rm`. To manually connect to the server-push vision server, go to the URL `http://a12.csse.monash.edu.au/robot/image.html` using a web browser that supports server-push.

## B Important System Information

To modify any of the content served by Zope at `http://a12.csse.monash.edu.au/robot/`, connect to the URL `http://a12.csse.monash.edu.au:8080/manage`. Enter the username 'superuser' and the appropriate password. See `http://www.zope.org` for howto's on using and developing within the Zope environment.

### B.1 File locations

All Source code for anonymous ftp `/home/ftp/pub/`

Java Control Server Source and Classes `/usr/robot/`

Control Server connections log file `/usr/robot/connections.log`

Control Server password file `/usr/robot/passwd`

Java Control Server startup script `/usr/bin/robotserver`

Zope home directory `/home/zope/`

Server-push grabber program `/usr/bin/grabber`

Server-push CGI program `/home/httpd/cgi-bin/image.cgi`

Server-push temp files `/home/httpd/cgi-bin/`

Realserver home directory `/usr/realserver/`

Realproducer home directory `/usr/local/rprod/`

## **B.2 Service Ports**

**Zope web server 8080**

**Zope ftp server 8021**

**Apache web server 80**

**Realserver RTSP 554**

**Realserver PNA 7070**

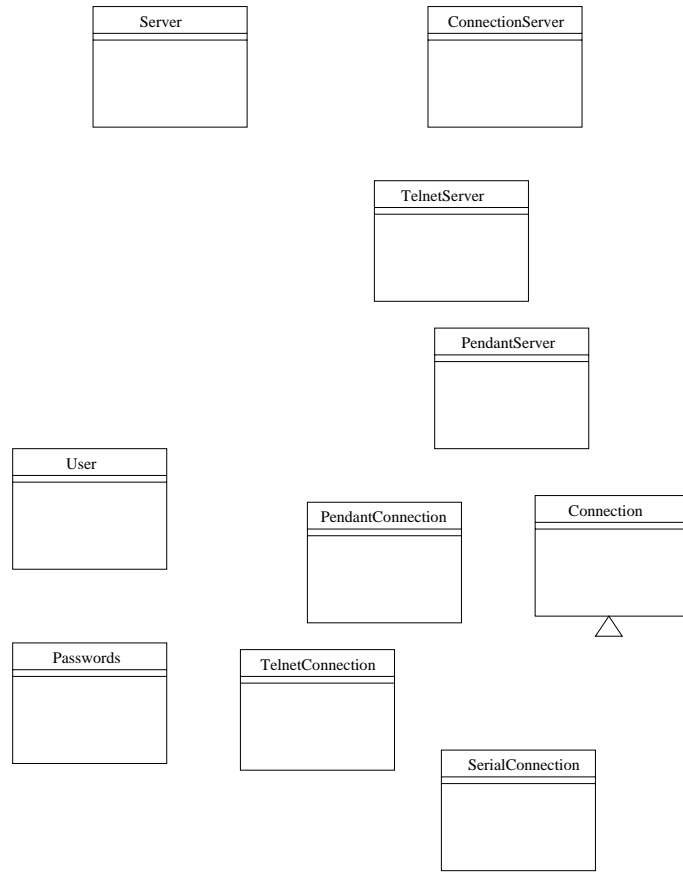
**Realserver HTTP 8090**

**Realserver HTTP admin 10557**

**Telnet terminal robot control 11000**

**Java control pendant robot control 11001**

### C Server Software Class Diagram



## D Pendant Software Class Diagram

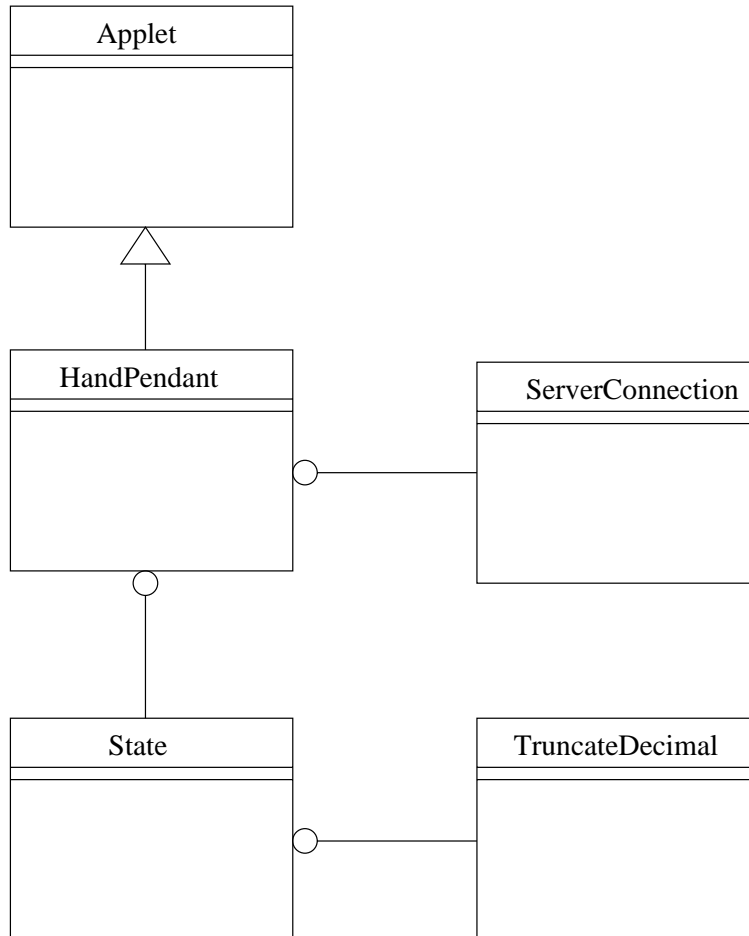


Figure 11: Telerobtic System Pendant Classes

**Applet** Java imbuilt class inherited from to create applet programs.

**HandPendant** Main pendant class. Sets up GUI interface and event listeners.

**ServerConnection** Has an internet socket connection to the server and general methods to talk to the server.

**State** Contains all the functionality for calculating VAL II commands from what is clicked on the GUI.

**TruncateDecimal** Simple class to easily truncated decimal numbers. Used when displaying location information.