# The Application of an XML Communication System to Aid in Prototype Testing of Tele-Operative Devices

Salim Haniff

University of Tampere
School of Information Sciences
Interactive Technology
User Interface Software Development
M.Sc. Thesis
Supervisor: Rami Saarinen
April 2011

University of Tampere
School of information Sciences
Interactive Technology
User Interface Software Development
The Application of an XML Communication System to Aid in Prototype Testing of Tele-Operative Devices
M.Sc. Thesis, 87 pages + 10 appendices pages
April 2011

**Abstract**

In this thesis, an XML communication system was created to provide a description of a user input command. The XML communication system could be incorporated into tele-operative applications to provide a means of controlling tele-operative devices. In addition, assist in prototype testing of new input device interactions. To minimize development time, the XML communication system was designed to be computer language independent and utilize existing Internet Protocol standards for communicating to other computers. An experiment was conducted utilizing the XML communication system for an object following task using 3 input devices (mouse, joystick and face tracking). The results demonstrated that the XML communication system worked in operating the tele-operative camera device and provided sufficient data to analyze the users' interactions with the input devices. The data provided insight on whether or not the devices were suitable for the task.

Table of Contents

# List of Figures

## List of Graphs

## List of Tables



## List of Code Blocks

# 1. Introduction

Software systems are commonly designed to utilize a graphical user interface (GUI) relying primarily on a mouse or keyboard and a display monitor. The system receives commands from the mouse or keyboard and displays the results on the monitor. This is known as the Windows, Icons, Menus and Pointers (WIMP) paradigm (Green and Jacob, 1991), a paradigm that was created in the early 1980s and is still commonly used today (ScienceDaily, 2009). While the WIMP paradigm has been sufficient for the majority of users, users in certain environments or suffering from certain ailments may not be able to fully utilize all the system interactions. For example, users suffering from an ailment may not be able to use their hands or feet to input their commands into the computer system (Steriadis and Constantinou, 2003). Another situation may exist where an instructor is required to provide constant feedback to a student on their performance in a noisy environment, however, the spatial separation between student and instructor may create difficulty in understanding the provided the feedback (Spelmezan, 2009). However, the relatively new advances in computing technology have led to an increase in research and development in the area of post-WIMP interfaces.

The post-WIMP interface paradigm provides system developers the opportunity to utilize other means of receiving input from the user. The system can receive input from the user's gesture, eye-gaze, voice and possibly other input devices equipped with integrated circuit sensors. Allowing the user a choice from a wide array of input devices creates more usability options for the system developers to utilize natural interactions, or direct manipulation of objects controlled by a computer, that could not be done using the WIMP paradigm. For example, interactions in a remote environment may feel more natural to the user if using a head tracker to pan the camera around the environment instead of a series of mouse clicks or keystrokes from the keyboard. The post-WIMP paradigm also reflects using different output devices. (van Dam, 1997)

The post-WIMP paradigm has led to another area of interaction known as 'mutlimodal systems'. Multimodal systems are designed to allow the user to communicate with the computer system utilizing various modalities such as gestures, vision, voice and tactile simultaneously. One early example of a multimodal system, named "Put That

There", was demonstrated by Richard Bolt (1980). In the demonstration, the user pointed to the screen and provided the computer with an instruction, such as "Create a blue square there". The system fused gesture data and voice data from the user to create syntax. After the computer had processed the syntax, it created a blue square in the location the user was pointing to.

The basic principle behind multimodal systems is data fusion and data fission. Data fusion utilizes a mix of input hardware devices to create input or output expressions based on low level component interactions which are then interpreted at a higher abstraction level pertaining to the task domain (Nigay and Coutaz, 1993). In Bolt's demonstration, the basic pointing provided coordinates to the computer and the spoken speech was just an audio signal. The computer processed the audio signal to determine the specific instructions from the user. The vocal instructions combined with the coordinates provided by the finger pointing were fused to create syntax to be evaluated by the computer. The computer evaluated the syntax and applied it to the programmed application domain. In this case, the application was programmed to place objects where the user pointed.

Data fission occurs when a piece of information is decomposed and represented over a single or multiple digital channels (Coutaz et al., 1993). For example, if the computer encounters an error it could communicate that error to the user over various output channels. The computer could display an error icon on the computer screen and it could make an audio sound to attract the user's attention.

Bolt's demonstration of the "Put That There" system is an example of a synergistic multimodal system. A synergistic system combines the use of two or more modalities working in parallel. The data from each of the modalities is combined to create syntax for the computer to evaluate. The alternative to a synergistic system is a concurrent system. A concurrent system treats each modality separately and the data from the modalities are not fused (Nigay and Coutaz, 1993). One example is a user giving a voice command to "empty the recycle bin" while double clicking on a document with a mouse to view the document. A concurrent system may allow system developers to create redundant input in their systems. Such a redundant system would allow users to interact with the system using more than one device. The benefits are if one device is not functioning, then the user could quickly switch to another device and interact with the system. For example, a portable

device may be equipped with speech recognition but if the user is situated in a noisy area the device may fail to recognize their voice due to the noise in the background. The user could then switch to the touch screen interface to interact with the device.

Nevertheless, some input and output hardware devices are proprietary to the system they run on and it may not be feasible to consolidate the various devices onto one complete system. In addition, special hardware and software may be required to run the individual device. It may also be the case that the user and the device are located in remote locations, as in the case of tele-operation of robots. Communication utilizing existing computer network technologies may be able to remedy this dilemma. If a common network messaging system could be created, then communication between such devices may be easier. There have been a series of implementations created and documented that try to create a multimodal system using Integrated Development Environments (IDE) or frameworks. Some examples are W3C's Multimodal Interaction Framework (Larson et al., 2003), CrossWeaver (Sinha and Landay, 2003), ICARE (Bouchet et al, 2004) and ICON (Dragicevic and Fekete, 2004). While each of these does provide many benefits, some are still in the draft stages of trying to outline how they could be implemented or then the implementation requires the developers to conform to specific standards using various computer languages (Lawson et al., 2009; McGee-Lennon et al, 2009). Thus, developers may often be waiting for formalization of the implementation or conforming to strict standards outlined by other organizations.

Analysis of past computer trends has demonstrated that heterogeneous systems with a focus on communication are commonly adopted quicker than those that force restrictions; in addition, keeping the implementation simple has also aided in the adoption of various computing technologies. HyperText Markup Language (HTML) is a perfect example of creating information to be distributed to heterogeneous systems (Dix et al, 2004). Anyone can create a document in HTML and publish the HTML document on a web server. Any computer with access to the Internet can download the HTML document and open the HTML document in their web browser regardless of the operating system they are running or the web browser application they use.

A common trend in internet communication is the use of eXtensible Markup Language (XML), which has a similar structure to HTML, to provide a description of

events or to transfer information from one system to another. The benefits of using XML are that many modern programming languages have libraries that are capable of reading XML structures, XML allows the data to be presented in a textual structured format and extending the structure of the XML data set allows quick accommodation of new changes (Harold and Means, 2004). Previous literature (Cohen, 2001;Dissanaike et al., 2004) has demonstrated successful models in using XML to transmit messages from one computer system to another.

For tele-operated devices, the use of a lightweight XML message system for navigating tele-operated devices may help reduce the computational resources required for such navigation. Tele-operated devices are equipped with motors, sensors and lightweight actuators that allow the operator to have a presence in a remote location and they usually perform multiple tasks which require computation resources (Minsky, 1980). Not only must these devices analyze the environment with the use of sensors and traverse through the environment using the motors, they must interact with the environment with the use of a manipulator arm. Some autonomous robots are equipped with crash avoidance algorithms to prevent collisions with certain obstacles. However, as some tele-operated devices do not possess high computational power as found on desktop computers, it may require additional computation time to analyze and process complex data structures. In this regard, a lightweight XML message system may help relieve the computation resource load.

The purpose of this thesis is to show how existing Internet technologies can be used to create an XML structure and communicate that XML structure to other networked computer systems. The XML-based communication system should allow system developers the flexibility to customize the communication system to meet their needs. This could then be utilized in a multimodal interface environment to enable various technologies to transmit input data from a wide array of input modalities to a wide array of output modalities.

To create the proposed XML-based communication system that would enable the system developers the ability to use their devices over the network, a number of existing user input devices were first analyzed, namely the mouse, the joystick and a gesture based interfaces. The mouse and joystick devices were chosen because computer users are often exposed to these devices during their initial interactions with computer systems, thus, have

quite a long term experience using those devices. Gesture based interfaces was selected to represent a novel interaction technique and for this thesis to demonstrate how flexible the XML-based system was to accommodate input from a gesture based interface. Program source code was then examined to provide some insight on how to extend the devices' functionality to incorporate the proposed communication system. On the basis of that examination, a general explanation of different Internet protocols was drafted to help developers select the appropriate Internet protocols for their application. Equally, the XML data structure was examined to allow developers to utilize it in their computer application. The XML communication system was then used in a pilot experiment to demonstrate how the collected data could be used to evaluate the user's efficiency in using certain devices.

After this introduction, the thesis comprises of six chapters. In chapter 2, the focus is primarily on a selection of input and output devices currently utilized in the Human Computer Interface field. The importance of this chapter is to provide some background knowledge of how certain devices function and interact with the computer. By deconstructing the devices into low level computer interactions, it may be easier for developers to see how to extend the functionality of certain devices. The interactions of the devices with the computer are also essential to understand how the XML based communication system will function, since the XML structure is used to represent the events generated by the devices. In chapter 3, the focus is on deriving the XML communication system needed for the multimodal development environment. The start of chapter 3 covers how computers communicate over the data network using existing Internet protocols. The proposed XML structure is introduced based on previous works from other research papers. The last section of chapter 3 describes how the XML structure and various network scenarios could be used in communicating events in a multimodal environment. The pilot experiment utilizing the communication system derived in chapter 3 is introduced and discussed in chapter 4. Chapter 5 discusses the results of the pilot experiment and considers how further data interpretation could be achieved using the XML communication system. The thesis concludes in chapter 6 by summarising the key findings found in this thesis and proposing possible future areas of research.

# 2. User Interface

The majority of input devices used today to help facilitate interactions between the user and computer systems originated in the mid-60s (Myers, 1998). The traditional computer mouse is one example of an input device created in the early 1960s and demonstrated in 1968 by Engelbart and English (1968). Input devices provide the user with a primary method of data entry into computer systems. The data is then processed by the computer system. Once the computer has processed the data, the computer presents the information in a relevant way to the user.

Despite the global acceptance of the computer mouse as an input device in the vast majority of computer applications, questions regarding the effective usefulness of the traditional computer mouse in certain applications are debatable. For example, a tele-operated video camera may have the functionality of panning, tilting and zooming. If an application was created to control the tele-operated video camera utilizing the traditional 2-D mouse device then only 2 functionalities would be available for the user. The user could hold down the mouse button and move the mouse up and down on the table to tilt the camera. The user could also perform a panning operation by holding the mouse button down and moving the mouse left and right. However, if the user wished to zoom they would have to indicate to the application that they were interested in a zooming operation by selecting different parameters. This would disrupt the seamless operation of the tele-operated video camera.

Buxton (1986) described the notion of device independence as a classification of input devices into generic classifications known as virtual devices. The objective of device independence is to allow one physical device to be interchangeable with other physical devices that are within the same classification. By using virtual devices the emphasis is on finding a suitable device to perform a series of operations rather than forcing a device to perform a series of operations. Buxton emphasized that devices should not be characterized as "mouse" and "joystick" but rather classified as how relevant they are in mapping the users intentions into the application. The new classification could be used to determine the appropriate devices for the right applications. Experimenting utilizing device independence, however, requires additional hardware requirements for the application.

Recent advancements in computer hardware through very-large-scale integration (VLSI) processes as well as the increasing affordability of electronic hardware has allowed

many companies, researchers and hobbyists the possibility of integrating various electronic components into user interfaces. Input devices, such as the mouse, contain various electronic components that the computer uses to capture data from the user. While the mouse is considered to be one of the commonly used input devices, the keyboard can be considered another commonly used input device which is used in WIMP-based interfaces and will be discussed later in this chapter. Despite these advances in hardware, the WIMP-based interface is still prominent in a vast majority of computer applications even though other hardware devices are readily available for system users.

Andries van Dam (1997) points to a new era in user interfaces known as the post-WIMP era. Post-WIMP devices are tailored towards natural human interactions with computer systems. Joystick and other gaming controllers allow application developers to create real-world simulations on the computer. One example from van Dam's paper is the use of the steering wheel and gear shift used in driving simulators. The benefit of using the steering wheel and gear shift controllers was that the user felt a natural experience of driving a car in a computer simulation environment. While it was possible to use the keyboard to create the same interactions, the loss of natural interactions was highly noticeable due to the fact that the turning of the steering wheel would be handled by pressing keys on the keyboard. Keyboard keys perform binary operations where the key is either pressed or not pressed. When playing driving simulators this translates into the steering wheel being turned completely left, completely right or dead centred. This lack of granularity in the degree the steering wheel is turned leads to over-correction in some driving simulators. With the use of the steering wheel and the use of Analog-to-Digital converters however, each degree the steering wheel is turned is translated into a digital signal that the computer can read leading to a natural feeling of the car steering in the simulator.

The post-WIMP era is not confined to manual input; speech interfaces combined with gesturing input can provide many avenues of opportunities. However, implementation of the two interfaces leads to a hard system to tokenize and disambiguate into verbs, noun and modifiers components. Moreover, speech interfaces may introduce some latency into the system, such as time required to convert the user's speech into recognizable commands. In addition, if the error rate is significantly high the user may have to repeat words in different

articulation for the system to understand. An ideal approach to speech interfaces is to limit the use of words for the speech interface devices based on a customized vocabulary set specific to the application. A personalized recognition database could be built based on the user repeating verbal commands repetitively. The system could apply statistical methods to recognize certain waveforms based on the repetition of the user's verbal commands. The accuracy may be improved because the system is trained to recognize various waveforms of the same instruction. The final interface concept mentioned in the post-WIMP paper was the use of gesture based devices. The utilization of non-invasive and accurate sensors could provide spatial and temporal input data to the computer system. These sensors could record head, body and eye movements. From the movements of the body parts gesturing data could be captured and analyzed. Based on that data the computer can provide feedback to the user. In addition, haptic interfaces could be used to send tactile data to a user to notify them of data from the computer system. For example, collision avoidance may be enhanced through the use of vibro-tactile feedback to the user when a tele-operated vehicle is in close proximity of a hazard. Ultrasonic sensors on the vehicle could detect if the hazard is too close to the vehicle and send a signal to the operator's computer. The computer could send a signal to a vibro-tactile which is attached to the body of the tele-operator. The vibro-tactile will then vibrate to alert the tele-operator of the hazard. Various intensities could be applied to provide an indication of how close the vehicle is to the hazard. In the event the user would not able to detect the hazard visually, the vibro-tactile may be able to alert the user instead.

As Andries van Dam pointed out in his paper, input devices should be tailored to allow natural interactions, or direct manipulation in the HCI field, from the user. However, it may not be possible to immediately know the physical hardware limitations, human factors or software logic needed to facilitate these interactions. Despite how well current theories are on interaction, there will always be a need to test designs through implementation and prototyping (Buxton, 1984). In tele-operative environments, other factors need to be measured to ensure that the user can manipulate the device. For example, factors such as latency between the user's input and the feedback could be addressed to avoid confusion. Held's (Held et al., 1966) research concluded that if the latency is longer than 0.3 seconds then the tele-operator may feel that the commands they

are issuing to the device are decoupled and may cause a loss in coordination between vision and hand, especially when the device is being directed through manipulation by the operator. Such a decoupling may impede the accuracy of the manipulation of the device where precision is important. If the system can be built with supervisory controls and indicators of latency between operator and device, the negative impact may be alleviated (Chen et al, 2007). Selecting the appropriate input device for operating a tele-operative device is another factor. A device that forces the user to use repetitive motions may induce certain discomfort from repetitive strain injuries. Other devices may also cause fatigue in muscle groups in the body causing deteriorating performance over an extended period of time. By prototyping the use of the devices in a lab environment, these factors can be addressed and significant financial loss may be avoided before deploying tele-operating devices in the field.

The remainder of this chapter primarily focuses on the various types of input methods in user interfaces, with one section briefly investigating the use of haptics as an output device. The focus on haptics as an output device in this thesis is largely influenced by van Dam's paper (1997) where he emphasizes the need to explore new device interactions. Haptic devices can be used to sense and send information. Barring that in mind, the proposed XML communication system could be flexible to accommodate haptic devices. Thus, the XML structure could be derived by either human input or computer sensors. Each section in this chapter is structured to provide some background and technical information on the device, in addition, implementing the device into the computer system. These sections provide insight on how the system developer can interact with the device. The interaction between the computer and the devices is an important concept to understand because it provides the background in order to create the XML structure in chapter 3. The chapter concludes by describing multimodal interactions and how user interfaces could be created utilizing the benefits of multimodality.

## *2.1. The WIMP Interface*

### 2.1.1. Background

Despite criticisms about the WIMP interface discussed previously, the WIMP interface is still a very important interface to consider in comparison to the other types of user

interfaces. The WIMP interface is still relevant in modern day desktop computing since many applications and operating systems are developed using the components of WIMP. Novice computer users are usually exposed to a WIMP style interface during their initial learning stages of operating a computer. Prior to the WIMP interface, the commonly used interface in the 1970s was the command line interface (Green and Jacob, 1991).

The command line interface (CLI) was the first interactive dialog mechanism between the user and the computer (Dix et al, 2004). The CLI is still used on a large number of computer systems and specialized network equipment. The CLI provides the user direct access to the computer system using a keyboard. Network computer systems, tele-operated devices and network equipment contain a remote shell to provide users a mechanism to interact with those systems. The remote shell provides the user a CLI to interact with the remote system. The user enters commands on the shell prompt line using the keyboard and the computer interprets the command. However, the commands used for command line interfaces can lead to high rates of errors due to inconsistent or inadequate syntax design (Hanson et al, 1984). In addition, to achieve a high level of efficiency using CLI the user must memorize the commands. The user is also required to memorize certain command flags needed to run the command with various options. An experiment was conducted by Neilsen (1993) with novice users performing file manipulation tasks using a graphical user interface (GUI) and the CLI. The experiment concluded that tasks using the GUI resulted in a completion time of 4.8 minutes with an error rate of 0.8 while the same tasks took 5.8 minutes with an error rate of 2.4 using the CLI (Nielsen, 1993). Based on the results of that experiment, a graphical user interface was more efficient for novice users to complete their tasks compared to a command line interface and promoted the use of a mouse and window interface.

The mouse and window interface was the key idea behind the desktop metaphor. The desktop metaphor attempted to model the office environment onto the computer desktop screen, where icons were used to depict various office objects and functions (Smith et al, 1985). Icons are intended to provide a simple means of interacting with the computer system without the use of supplying typed commands on the command prompt. The picture on the icon helps associate the icon to the desired actions the user wishes to conduct. The use of icons is fairly common in many modern day applications even though

the application domain may not be office related. The desktop metaphor helped introduced the concept of direct manipulation to a large audience of computer users.

Ben Shneiderman (1983) describes direct manipulation as a system that provides the user an opportunity to interact with visible objects and concentrate on their work tasks, while the computer's physical transducers become transparent to the user. The key principles of a direct manipulation system are:

- Continuous feedback from the object of interest.
- Physical actions conducted through input devices or labelled buttons are substituted for complex syntax.
- Changes to the object of interest are rapid, incremental, reversible and immediately visible.
- A low learning curve, which enhances adaptation of the system by novice users. Novice users can learn a wide range of commands that are useful for the operation of the system.

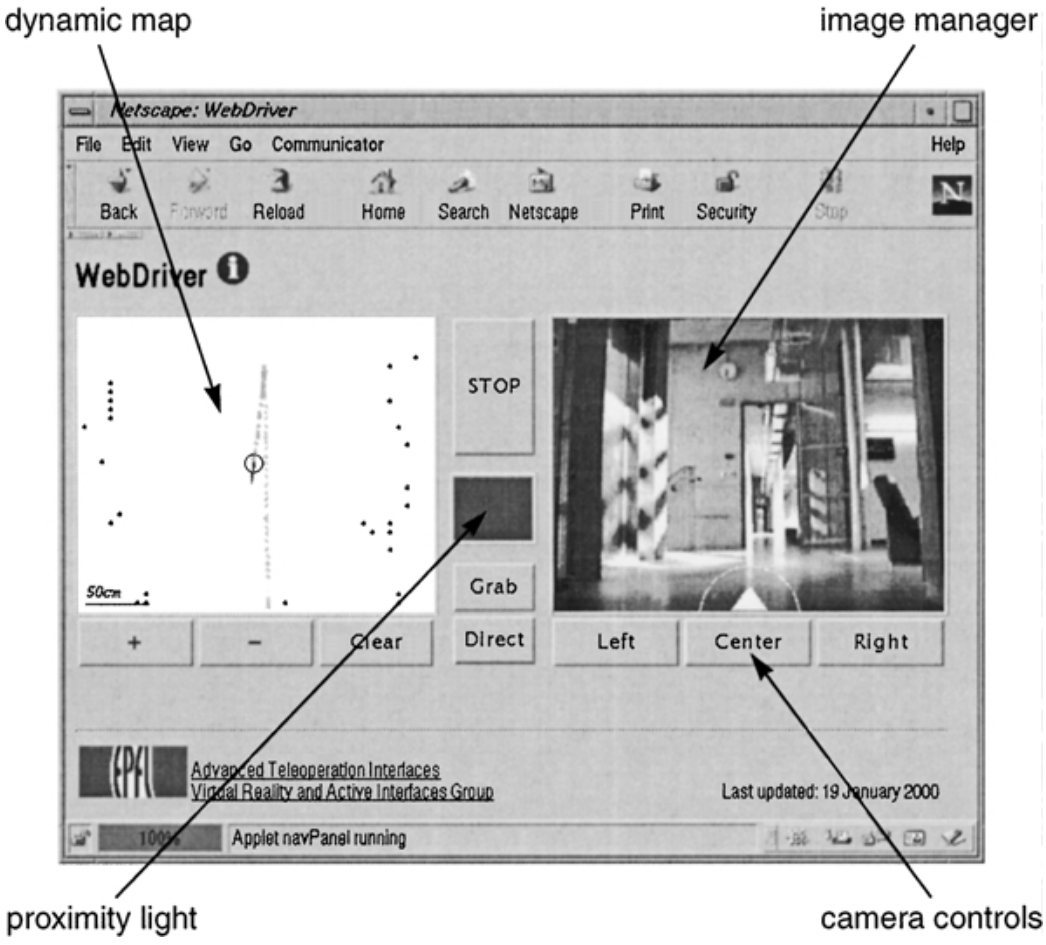



*Figure 2-1User Interface demonstrating the controlling of a tele-operated robot (Fong et al., 2001).*

Figure 2-1 demonstrates how the four principles of direct manipulation were applied in building a user interface to control a tele-operated robot. The “image manager” displays images taken from the robot’s web camera to the user. This provides visual feedback to the user about the remote environment. The labelled actions on the “camera control” buttons help provide the user a method of manipulating the robot’s camera without the need of supplying complex typed syntaxes. The third principle of direct manipulation is achieved by the action of pressing the “Left”, “Center” or “Right” buttons, which sends the action to the robot. The robot interprets the action based on the button that has been pressed and moves the camera. The “image manager” video is updated to reflect the incremental changes. The intuitive labelling of the buttons helps novice users to quickly learn the operations of navigating the robot’s web camera. From this user interface it can be seen how the combination the combination of the desktop metaphor and direct manipulation can be useful. The desktop metaphor and direct manipulation lead to the popularization of WIMP style interfaces (Green and Jacob, 1991).

The Windows, Icons, Menus and Pointers interface, commonly known as the WIMP interface, helped computer systems reach a wide audience base in the early 1980s by designing software to model the physical desktop and office environment into computer systems (van Dam, 1997). The success of the WIMP interface was due to the cohesive flow between the four components (Dix et al, 2004). The advantage of using multiple windows allows tasks to be treated individually. For example, the user interface displayed in Figure 2-1 could be used to help navigate the robot's web-camera. A second window may be opened to view telemetry data from the robot. A third window could display an alarm to alert the user of any unusual activity. By having multiple windows, the main application window is not cluttered by displaying secondary tasks. All windows have the attributes of being resizable, movable and overlapping other windows. Scrollbars on the window's borders allows the user to scroll through the window in the event that the screen resolution is too low to display the full contents of the window. The scroll bar also provides visual indication of the user's progress in the window. All windows include a title bar to provide a brief description of the window. The icon, as described earlier, is the second WIMP element. Icons rely on the user to decipher the image representation so there is a

small learning curve for the user to memorize and associate the images to actions, in addition, the user must coordinate their hand movement with the pointer device movements to click on the icons. Menus, the third element, help categorize actions into associated work functions. An example could be the 'File' menu, located in the menu bar, where users can conduct operations pertaining to the administration of the application's setting, such as 'Open Settings' and 'Save Settings'. The use of menu items could speed up the rate of recalling previously stored settings into the computer application rather than the user manually setting them each time the application is started. The pointer element refers to the use of a pointing device to allow the user to manipulate objects in a window, select other windows or select operations from the menu.

The WIMP interface provides efficiency and speed to users in certain applications (van Dam, 1997). If a user simply wants to interact with the system, for example moving a tele-operated robot through a maze of complex movements, they push the icons associated with the direction of the movement rather than typing in a series commands on the prompt. This method provides novice users a less cumbersome way of interacting with the system rather than repeatedly entering commands on the command prompt. Experienced users also benefitted from the WIMP interfaces by using a combination of keystrokes and short-cuts instead of navigating through various menus.

An immediate limitation of the WIMP interface is the lack of degree-of-freedom the user has to interact with the system. Degree-of-freedom refers to the number of links or joints and the freedom to move around those joints (Pennestri et al., 2005). Basically, a keyboard has a single degree of freedom since the interactions is done by pressing a key down. A mouse has 2 degrees of freedom since it can be moved along an x-axis and a y-axis freely. As van Dam (1997) suggested, computer applications are becoming more complex and WIMP style interactions are a drawback to the advancement of user interfaces. The first drawback can be seen when performing single and multiple tasks. An individual widget can perform a single task relatively well. However, a series of widgets connected together can increase the complexity for the user to perform a goal. For example, printing a document is a simple task. The user can just push the printer icon located on their menu bar to print a document. However, if the user is required to perform a task that involves clicking on multiple buttons then the operation of performing that task

may become complex. The user must memorize the order of buttons to click and they must also move the mouse pointer to those buttons. If the buttons are spatially separated the user will spend most of their time navigating the mouse pointer around the screen and clicking on the buttons. A second drawback is that experienced users are quite often spending time refining the interface rather than using the application. The screen will quite often become too cluttered for users to concentrate on their tasks and expert users will resort to keyboard shortcuts to avoid traversing through a series of menus to find their options. A third drawback of WIMP interactions can be seen in data visualization applications. One difficultly is exploring 3D data using 2D widgets which causes a disassociation between the 3D data and the 2D controls used for traversing the complex data. The final drawback from van Dam's paper about WIMP interactions is that the use of mouse and keyboard for certain users may not provide direct manipulation of certain objects in applications and may cause repetitive stress injuries because the user may have to rely on using additional keystrokes or mouse movements.

### 2.1.2. Implementation

The popularity of the WIMP interfaces has lead to a large number of graphical user interface toolkit APIs available to developers. Those APIs allow developers to create WIMP applications simply and quickly. A toolkit API is a collection of object classes, functions and data structures that could be used in object oriented programming to create graphical user interfaces. The benefits of using object oriented programming allows developers to quickly inherit classes, so developers are not constantly recoding the same functionalities. Qt and Java Swing are examples of frequently used GUI toolkits that are cross-platform. Cross-platform development allows applications to run on a wide range of operating systems and hardware without a significant amount of recoding.

A widget is the basic component in creating a graphical user interface. While the term widget may vary among vendors, the basic fundamental operation of the widget is the same. For example, Java Swing refers to widgets as components and .Net refers to widgets as controls. A widget can be created into an input widget, display widget and container widget. Input widgets are commonly used to receive input from the user. Examples of input widgets are push buttons, sliders, scroll bars and line boxes. Display widgets show

information to the user. This could be in the form of simple text or progress bars. Containers are used to logically group widgets together such as radio buttons or check boxes. Widgets are also capable of sending event based information to other widgets. For example, if a push button widget is pressed then an event could be triggered. The event trigger could then be transmitted to any widgets that are listening for push button events. When the receiving widget has received the event notification it could process the event into something meaningful. In the code shown in code block 2-1, a button labelled 'Quit' could be clicked to send an event to the qApplication object. The qApplication receives the event in a slot function named quit. The quit function processes the event, which in this case causes the application to terminate.

| Sample Qt Code demonstrating Signal and Slots |
| --- |
| …<br>QPushButton *quitButton = new QPushButton(tr("Quit"));<br>connect(quitButton,SIGNAL(clicked()), qApplication , SLOT(quit()));<br>… |

*Code Block 2-1 A simple example illustrating the signal and slot concept used in Qt.*

In Qt, the interactions between the widgets are achieved by a signal and slot method. When a widget is pressed using a mouse pointer the widget sends a signal to another widget's slot. A slot is a mechanism for the widget to listen for signals, in the case above the widget being clicked. If the application designer wishes they could have key presses on the keyboard generate signals, the signals would be sent to the other widget's slot for processing. One example could be the user pressing CTRL-Q to quit the application. Quite similarly other computer programming languages, like Java, use action listeners or event handlers methods for handling widget (or component) interactions. When the widget object is instantiated, the developer can add an action listener function to that object. Once the user interacts with that object, event messages are sent to the action listener function and certain tasks are performed based on the logic contained in the action listener function.

## *2.2. Analog Joystick and Analog Sensors*

### 2.2.1. Background

Joystick controllers are commonly used to control the movement of tele-operated devices (Hainsworth, 2001). Joysticks offer direct manipulation of tele-operative devices easily by pushing the joystick handle in the direction the tele-operative device should travel in. Inside the joystick are analog sensors. Analog sensors are often used to measure various data external to the computer system. For example, analog sensors could be used to measure applied forces, atmospheric pressure, temperature, light intensity and position. Other devices, like some computer mice, contain analog sensors that enable them to read the user's movement when they interact with the devices.

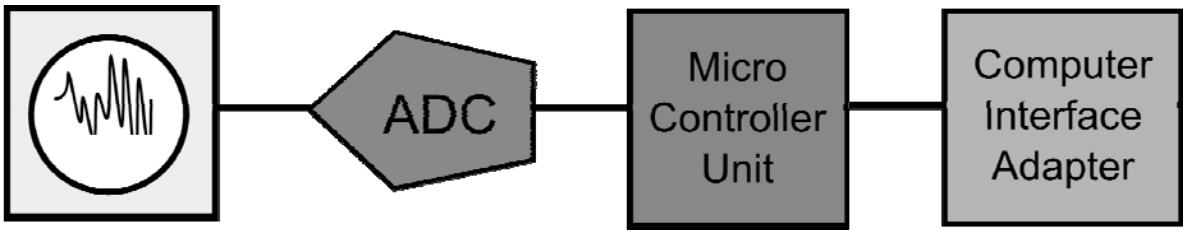



*Figure 2-2 A simplified overview of acquiring a signal and sending it to the computer.*

Figure 2-2 illustrates the basic steps of acquiring information from a sensor and transmitting it to the computer. A sensor may contain 3 leads, one lead for input voltage, second lead for analog voltage out and the third lead for ground. The user provides power to the sensor on the first lead and attaches the second lead to a device to analyze the analog voltage. When a change or movement is detected by the sensor, a change in voltage is sent to the Analog-Digital-Converter. The Analog-Digital-Converter converts the analog signal into a digital signal and sends the digital signal to the microcontroller unit. The microcontroller unit synchronizes communication with the main computer unit and transfers the data from the Analog-Digital-Converter to the main computer unit.

An Analog-Digital-Converter (ADC) allows system developers to capture environmental elements and convert them into digital signals. There are two main tasks the ADC performs, the first task is the sampling the continuous analog signal and the second task is converting the sampled signal into a binary representation. Sampling the signal is performed by using a set time interval that records the analog continuous signal's magnitude. If the ADC is rated with a sampling rate of 1 kHz then the continuous analog

signal's magnitude is sampled 1000 times a second. In the second task the discrete values are converted in binary form. The output resolution defines the accuracy of which the analog signal is being converted into binary form. For example, if the ADC has 3-bits output resolution then only 8 binary output representations of the signal are available but if the ADC has 10-bits output resolution then 1024 binary representations of the signal are available. (Kemna, 2003)

A microcontroller unit is a small computer with limited resources compared to a desktop computer. They are often used for real-time applications such as data acquisitioning or controlling relays, motors and actuators. One of the roles of the microcontroller is to synchronize communication between the ADC and the computer. This may be done through serializing the data stream and sending the data over USB, serial connection or parallelizing the data and sending it over the parallel port. The advances in integrated circuit (IC) data acquisitioning and micro-controllers has made the integration of sensors into computer applications a trivial task. Arduinos and Phidgets, which are affordable microcontroller boards, can be purchased that contain analog inputs, ADCs and USB connections to computers. Joystick controllers also contain on-board micro-controllers to transmit the position of the joystick handle. The task of the programmers is to determine what relevant data is required from the sensors and how to provide informative feedback to the users.

### 2.2.2. Devices

Two common components used to measure physical movement by the user and notify the computer are the rotary potentiometer and the Hall-effect sensor. Both devices could be found in joysticks and some computer mice. The rotary potentiometer could also be used to provide a physical widget into a computer application. When the user turns the knob, the knob can alter functionality in the application. For example, the knob can send a signal to a remote camera to indicate the position the camera should be oriented.

A rotary potentiometer resistor, shown in figure 2-3, is a variable resistor. The potentiometer resistor contains 3 leads. The first lead is used to attach the input direct current (DC) voltage, the last lead is used to attach to ground and the middle lead is attached to the wiper in the potentiometer. The circular housing of the rotary potentiometer contains a wire coil that runs on the circumference of the housing. The wiper

arm is attached to this wire coil. As the electricity runs throughout the potentiometer unit the wiper terminal produces different voltages based on the distance from the positive input terminal to the wiper arm. The closer the wiper arm is to the negative input terminal the higher the voltage is being transmitted through the wiper arm to the wiper terminal. When the distance between the wiper arm and the negative input terminal increases the voltage decreases. This is due to the resistance generated by the wire coil. For computer systems to read the value of the potentiometer the terminal attached to the wiper arm is often connected to the ADC. The microcontroller reads the value from the ADC and passes the value to the computer system. (Nyce, 2004)

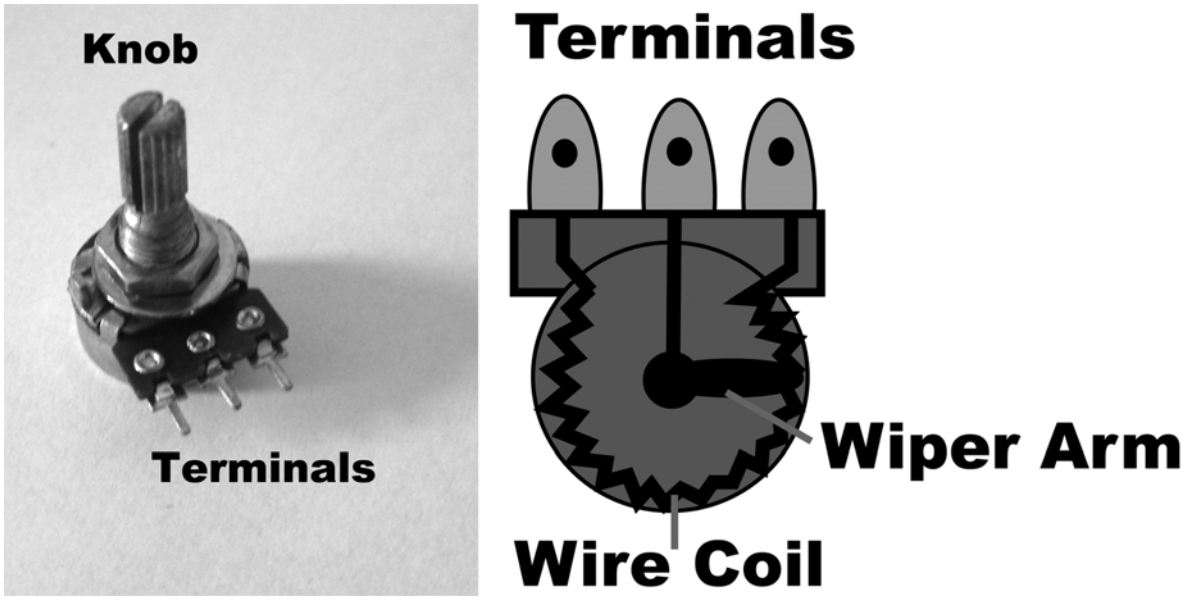
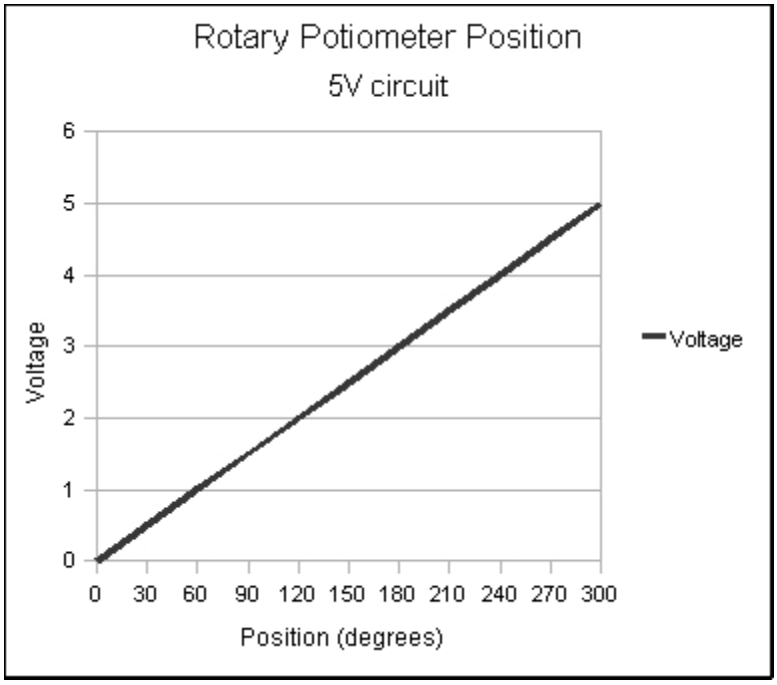



*Figure 2-3 Left image is a physical image of the rotary potentiometer. The right image is a drawing of the internals of the rotary potentiometer.*

A Hall-effect sensor measures the strength and polarity of a magnetic field. A Hall-effect IC sensor contains 3 leads similar to the rotary potentiometer. One lead is for the DC input voltage, the second lead is the Hall voltage output from the IC and the last lead is for ground. Inside the Hall-effect IC sensor is a thin sheet of conductive material known as the Hall element. When a perpendicular magnetic field is applied to the Hall element a voltage is created which is proportional to the magnetic field's strength. The polarity of the voltage produced by the IC on the second lead is dependant upon the magnetic field's polarity travelling through the Hall element. So the voltage may either be positive representing one polarity of the magnet and negative representing the opposite polarity of the magnet. The

Hall-effect IC sensor can be used to determine the position of a permanent magnet based on the strength of the magnetic field. When the south polarity of the magnet is directly over the Hall-effect IC sensor the voltage output from the IC is the highest value. As the magnet moves away from the IC sensor the magnetic field decreases which results in the voltage output from the IC to decrease as well. The output voltage value from the Hall-effect sensor is ratiometric, meaning that the output voltage is proportional to the input voltage applied to the first lead of the IC. (Nyce, 2004)

The benefit of using a potentiometer or magnets and Hall-Effect sensors is that a calibration curve could be created to map the degree of rotation to the voltage output on the wiper terminal (shown in graph 2-1) or the position of the magnet relative to the Hall-Effect IC sensor. When the user manipulates either device a binary value is sent to the computer system by the microcontroller to reflect a change from the external environment. The computer can interpret this change and display the change back to the user through various outputs. The low cost of the potentiometer and Hall-effect sensors make them idea for analog joystick controllers. In Kortum's book (2008), a diagram depicting of an analog 2-axis joystick illustrates the use of two potentiometers. Each potentiometer indicates the position of the stick in each axis. The joystick is attached to a shaft which is attached to the wiper (or knob) of the potentiometer. Calibration software could be used to determine the position of the joysticks relative to the voltages being outputted by the potentiometers. For example, the application can record the voltages of the potentiometers when the joystick is in the top-right, center and bottom-left positions. This calibrates the software to know the limits of each potentiometer and can deduce the location of the joystick based on the calibrated end points. Using the Hall-Effect sensor eliminates the need of 2 potentiometers, where the bottom of the joystick could contain a magnet over Hall-Effect Sensor IC (AustriaMicroSystem, 2010).




*Graph 2-1Graph illustrating the voltage output at various degrees of rotation of the potentiometer.*

### 2.2.3. Implementation

The Simple DirectMedia Layer (SDL, 1998) provides a cross-platform library to develop applications that can utilize the use of the joystick in various applications. The library is designed to work on any operating system and the library is considered to be open source. For the scope of this thesis we will be looking at the joystick API under Linux since it requires fewer lines for implementation. The importance of this section is strictly on how to get the value from the joystick.

The Linux kernel provides a joystick API to allow system developers access the joystick device and read events being emitted from the joystick (Espinosa, 1998). Once the joystick has been attached to the computer a device file is created, usually /dev/input/js0. To read events from the joystick the developer must open a file handler accessing the joystick, example shown in code block 2-2.

| C code for accessing joystick device |
|---|

```
int fd = open (“/dev/input/js0”,O_NONBLOCK);
```

*Code Block 2-2 Example of accessing the joystick port under Linux.*

The O_NONBLOCK option is used in the open function to prevent delays in reading the joystick device. Reading the joystick is done by passing the js_event structure, defined in the joystick.h file, in the C read function. Upon successful return from the read function, the js_event structure will contain a snapshot of the events created by the joystick. The js_event structure contains 4 variables. The first variable is the event timestamp, the second variable is the value, the third variable is the event type and the last variable is axis or button number. The value is an integer from -32767 to 32767 representing the position of the analog joystick on a specific axis. The value could also be represented as -1, 0 or 1 when the digital joystick pad is used. In addition, the values of 0 and 1 are used to represented the states of the buttons, where 1 represents the button is in a pressed state and 0 to represent a released state. To help differentiate what the values mean the event type and number variable in the js_event structure are used. There are 3 event types that occur from the joystick. The 3 event types are button presses, joystick axis movement and initialization. The axis and buttons are uniquely identified to help the developer know which joystick or button was moved or pressed. The unique identifier is an integer value stored in the number variable of the js_event structure. An example listed below illustrates how the joystick events could be read.

C code for reading joystick events

```
int fd = open(“/dev/input/js0”,O_NONBLOCK);
struct js_event jse;

while(1){
      read(fd, jse, sizeof(jse));

      /* Prints out the values contained in the js_struct */
      printf("Event: type %d, time %d, number %d, value %d\n", jse.type, jse.time,
jse.number, jse.value);

     switch(jse.type & ~JS_EVENT_INIT){

       case JS_EVENT_AXIS:
        /* Move the accelerator or brake widget*/
```

```
        if(jse.number == 1){
            if(jse.value < 0)
                jh->setValueGas(-1 * jse.value);
            if(jse.value > 0)
                jh->setValueBrake(jse.value);
         }
         break;

         case JS_EVENT_BUTTON:
         break;
      }
   }
```

*Code Block 2-3 Example of reading joystick values and sending the value to widget slots. Modifications made to code from the joystick API example (Espinosa, 1998).*

In code block 2-3, the read is contained in a while loop that runs onto infinite. The js_event structure is sent to the joystick and the variables are populated. When the program enters the switch block it checks to see which event type has occurred from the joystick. If the event type is a joystick movement then the first case logic block is executed. In this example, the code is determining if the left analog joystick controller was moved upwards or downwards. Once it has determined which joystick has been moved it reads the value of joystick and updates the user interface through the setValueGas or setValueBrake function.

## *2.3. Gestures tracking*

Gesture tracking is the tracking of a certain moving body part. The hand, face or body are quite commonly used body parts to perform gesture tracking. A gesture is just a simple movement made by a certain body part to indicate a certain command.

### 2.3.1. Background

Hand gesture interfaces have been around for decades in the Human-Computer Interaction (HCI) research community. Hand gesture interfaces utilize a combination of hardware and software that aid in capturing and processing information created from hand movements. Recent advancements in gaming consoles have provided game developers the opportunity to incorporate the movement of game controllers into the game environment. The users are not restricted to pressing buttons on a game pad, moving joysticks or steering a wheel.

Instead, by simple movements of the game controller the user can create certain actions in the game depending on those movements. (Prekopcsák et al, 2008)

The origin of gesture based interfaces derived from natural nonverbal communication between people. Instead of a second person to communicate to, a computer takes the place of the second person. Gesture based interfaces try to create a natural and intuitive interaction between the user and computer system rather than the use of a mouse and keyboard. Gesture based interfaces are not limited to just hands however, using computer vision APIs allows video cameras to track various body parts or objects. Using a video camera to track gestures helps make the interface invisible to the user because they are not required to wear any hardware or interact with physical devices. Therefore, any user can approach the system and immediately start interacting with it. (Kortum, 2008)

A brief survey based on published papers in the area of robot navigation using gesture based interfaces emphasized the use of hands and face tracking (Waldherr et al. 1998; Fong et al, 2001; Hasanuzzaman et al, 2004). The benefits of head and hand tracking interfaces were immediately noticeable in navigating the robots. One benefit was that hand gestures were understood universally. For example, a gesturing hand movement to the right usually indicated that the robot or camera should move to the right. The additional use of a second hand allows for 2 independent control input. For example, one hand can motion the robot to move while the other can motion the camera of the robot to pan independent of the robot's movement.

Head gesture interfaces could be created utilizing the Haar-classifier Cascade or by the Camshift algorithm. The Haar-classifier cascade is a supervised machine learning algorithm. A supervised machine learning algorithm is one which requires a user to supply accurate data for the algorithm to build a positive data set. The algorithm relies on the use of Haar-like wavelets to determine the rigid feature of a face, such as eyes, lips and hair lines. When multiple images of faces are supplied to the algorithm it detects the faces by the Haar-like features and stores them. Negative images that do not contain faces are also supplied to the algorithm to learn what images do not contain faces. After the positive and negative images are analyzed, the dataset is now trained to recognize faces. When new images are acquired from the camera and the Haar-classifier is used to scan the images, it uses the trained dataset to detect the face. The Haar classifier can detect other rigid objects

that contain clean straight edges like a car but it has problems with less rigid objects such as tree branches and distinguished shapes like coffee mugs. (Bradski and Kaehler, 2008)

The Haar algorithm described in the previous paragraph requires the scanning of every frame to detect the face. This results in high resource utilization from the computer system. In addition, if the face was tilted or turned sideways the application would lose track of the head position. The Camshift algorithm was develop to track colour information based on continuous image sequences. The tracking of colour images allowed the algorithm to detect the head when it is in various positions. The process for Camshift consists of four steps. The first step is creating a colour histogram based on an image of the face. In the second step, the Camshift algorithm scans the image to locate where a match to the histogram exists. If the location of the face has changed then the Camshift algorithm shifts the centre of gravity to the new location where the face exists, this is the third step. In the last step, the Camshift algorithm calculates the size and rotation of the face. If the face has been moved from the previous location then the internal data structures pertaining to the location of the face are updated. (Hewitt, 2007)

### 2.3.2. Devices

There are two methods for acquiring information for gesture based interfaces. The first method is to use a device with various IC components to measure the orientation and movement of the device. The gaming industry utilizes game controllers with gyroscopes and accelerometers to capture the user's motion. The motion data from these components are transmitted over a Bluetooth connection to the gaming console, where the orientation and movement of the game controllers are mapped into the video game. There are a few demonstrations on the web showing how to connect some game controllers to various computer systems (XI-FI, 2007; CWiid, 2010). The benefit of connecting game controllers to computer systems is that it allows developers to extend the functionality of gaming controllers to other application domains, in particular providing the users an opportunity to use gaming controllers as an input device to manipulate a robotic arm.

The second method for acquiring information on users' gestures is using a video camera. John Underkoffler demonstrated a system that captured his hand gesture movements and manipulated visual data projected onto a screen (Underkoffler, 2010). In the demonstration, IR LED arrays and cameras were used to detect the location of

Underkoffler's gloves. The gloves contained IR reflective material that certain cameras are able to detect. When the gloves were moved the cameras capture the gesture and interpreted it. Based on the gestures, the data displayed on the screen was altered. TrackIR (NaturalPoint, 2005) is a head tracking system based on a camera and IR reflective material used in many video games. With the TrackIR system, the user wears an IR reflective marker located on their head. The camera, which is attached to the computer by USB, records the location of the user's head movements. The user is able to control the in-game camera by moving their head. A “free” alternative can be found on FreeTrack’s website (FreeTrack, 2008). The implementation of FreeTrack is quite similar to TrackIR but home users can develop their own system utilizing their own web camera and acquire the hardware to build the IR system themselves. FreeTrack utilizes a joystick emulation program to create interactions between their head tracker and gaming applications.

Using a video camera can also allow a non-invasive method of capturing gesture based interfaces. Non-invasive methods require algorithms to detect certain features of the human body and track them as the user moves. The non-invasive method also provides low hardware requirements as only a video camera or web camera is used. In addition, any user can step up to the system and immediately begin interacting with it without the need to wear hardware or interact with hardware controller devices.

### 2.3.3. Implementation

To perform a gesture recognition using a video camera three separate steps are required. The first step is isolating the body part or object in the image. This could be done using a computer vision API. The second step is to track the motion of the body part or object and the last step is to classify the gesture movement into a meaningful action. (Kortum, 2008)

OpenCV (Bradski, 2000) is an open source computer vision API that provides software developers various algorithms to help with image processing and gesture recognition. The key advantage of OpenCV are its computational efficiency, focus on real-time data processing and relatively simple to use development framework. These advantages could be used to aid in gesture tracking from a video camera, especially face tracking. OpenCV contains a readily available face detection library set that utilizes the Haar-cascade Classifier and the Camshift tracking. Applications and their source code are

provided by the OpenCV application which could allow developers extend or implement some of the functionality provided by OpenCV into other applications.

Simple example illustrating how OpenCV can be used to detect faces

```
…
CvCapture *webcamera = cvCaptureFromCAM(0);  //This initializes the webcam for capturing images
IplImage *webcamImage = 0;  // This pointer will point to the memory space containing an image
CvMemStorage *storage = 0; // OpenCV memory storage structure pointer
//The line below loads the HaarClassifier training file
CvHaarClassifierCascade *haarClassifier = (CvHaarClassifierCascade*)cvLoad(haarcascade_data.xml);


storage = cvCreateMemStorage(0);

for(;;){
   webcamImage = cvRetrieveFrame(webcamera);  // Grabs image from web camera

   // The line below passes the image from the web camera into a haar detection function.

   CvSeq *haarObjects = cvHaarDetectObjects(webcamImage, haarClassifier, storage, 1.1, 2, CV_HAAR_DO_CANNY_PRUNING, cvSize(40, 40);

   if (haarObjects->total > 1){
      // if there was a face detect then this if statement block gets executed.
   }

   cvClearMemStorage(storage)
}
….
```

*Code Block 2-5 Brief example of how OpenCV is used to detect faces. Code adapted from FaceDetection code (Bradski, 2010).*

Code block 2-5 briefly demonstrates how OpenCV can be used to quickly obtain images from a web camera and detect a face. As the code illustrates, OpenCV provides many functions to help developers create code rapidly to capture images from the web camera and perform processing on them. When the cvHaarDetectObjects function is called the image obtained from the web camera is searched based on the haarClassifier passed to the function. If there are any objects, like the human face, detected then the if block statement gets executed.

## *2.4. Haptics and force feedback*

### 2.4.1. Background

Tele-operated devices may be situated in environments where conditions are unknown to the operator. The operator may have to utilize sensory and visual data provided from the tele-operated device to understand what is occurring in the environment. Sangyoon Lee (Lee et al., 2002) demonstrated a situation where an operator used a joystick to move a robot around an unfamiliar area. When the joystick was moved into a forward position the robot moved forward as well. The robot was programmed with a simple collision avoidance algorithm to prevent it from colliding into obstacles. If the robot was instructed to move and detected an object it turned to avoid the object or did not move at all. The collision situation may not have been a problem for the operator since they may have been able to see that the object is in the path. However, if the object was not visible to the operator because the object was out of the field of vision of the camera, then the operator may become confused as to why the robot was not behaving as instructed. If the joystick was equipped with force feedback technologies then it could indicate to the operator that an obstacle is present by creating force against the joystick movement in the direction of the object.

Haptic interfaces are mechanical devices that support bidirectional input and output of displacement and forces. Haptic interfaces provide the user the ability to explore and manipulate the environment or objects through the use of touch. Exploring of the environment or objects involves extracting physical properties such as shape, surface textures, mass and solidity. Manipulation is how the haptic interface could be used to physically alter the environment or object by the application of force by the tele-operated devices such as a robotic manipulator hand. Haptic interfaces try to mimic the human touch system, in particular the cutaneous and kinesthetic touch system. The cutaneous touch is the sensation of surface features and tactile perception conveyed through the skin. Kinethetic touch system allows humans to interpret spatial distance of limbs relative to the body through muscles and tendons. (National Research Council Staff, 1997)

### 2.4.2. Devices

The Novint Falcon, shown in figure 2-4, is one example of a 3D haptics joystick controller available to consumers. The user can send movement commands to the system by moving the Novint Falcon's grip. The grip is attached to a servo by an arm inside the Novint Falcon. The servo can read the location of the grip and also create a force against the user's movement to create a sense of force feedback. (Novint Technologies Inc., 2008)

*Figure 2-4 A user manipulating the Novint Falcon's grip.*

A servo motor can be used as an actuator to apply force to the human operator. A servo motor is commonly found on radio controlled vehicles. The servo motor is a closed feedback systems. A closed feedback system provides an electric signal indicating the position of the axle in the motor. In an open feedback system, electricity is applied to the motor and the motor axle spins, which is common with DC motors. If the position of the axle is required other electronic components, such as a rotary encoder, and computer programs are required. The benefits of using a servo are it reduces the amount of additional electronic components and the position of the axle is immediately known. If the axle of the servo encounters stress additional power will be applied until the axle reaches its programmed rotation point. (McComb and Predko, 2006)

Inside the servo motor are a series of gears to control the output speed to the axle, a controller board and a potentiometer. A 3 wire cable is attached to the controller board. The 3 wire cable provides positive voltage, ground and a signal to the servo to indicate where the axle should be positioned. The signal transmitted to the servo is in the form of a

pulse width modulation (PWM). A PWM signal is a fixed pulse sent to the controller board, usually 20 ms. Within the 20 ms, an active high voltage will be sent. The width of the active high voltage with respect to time indicates the position to move the axle. For example, if the active high voltage is 1ms then the axle will be at 0 degrees. If the active high voltage is 1.5 ms then the axle will be at 90 degrees and if the active high voltage lasts 2 ms then the axle will be at 180 degrees. The potentiometer combined with the servo controller board determines if the pulse signal and the axle position are in the correct position. If the axle is not in the correct position then power is either increased or decreased so move the axle into the correct position. (McComb and Predko, 2006)

The benefit of using servos for haptics interfaces is that it could provide a fixed amount of force that could be applied to the controller device. In a virtual environment, the user can move the haptic cursor around. If the haptic cursor collides with an object a signal could be sent to the servo to push the joystick in the opposite direction of the user's movement. The amount of force could be determined by the amount of rotation placed on the servo's axle.

### 2.4.3. Implementation

The following scenario illustrates how a haptic device can be created using an Arduino microcontroller board connected to a computer, a servo motor, a sliding potentiometer and a piece of wire, refer to figure 2-5. In this scenario, the sliding potentiometer will indicate the direction and magnitude to pan a web camera. When the potentiometer is positioned in the mid point the web camera does not move, as shown in figure 2-6. As the potentiometer moves to the right, the Arduino reads the voltage value from the potentiometer and sends the data to the computer. The computer reads the value from the potentiometer and sets the acceleration rate of the web camera motor connected to the web camera to pan to the right. As the web camera motor reaches a pre-assigned end point, for example 180 degrees rotation, the servo motor pushes against the metal wired attached from the servo to the potentiometer slider, as demonstrated in figure 2-7. The servo generates a force that the user could feel and realize that the camera is to reaching the end-point.

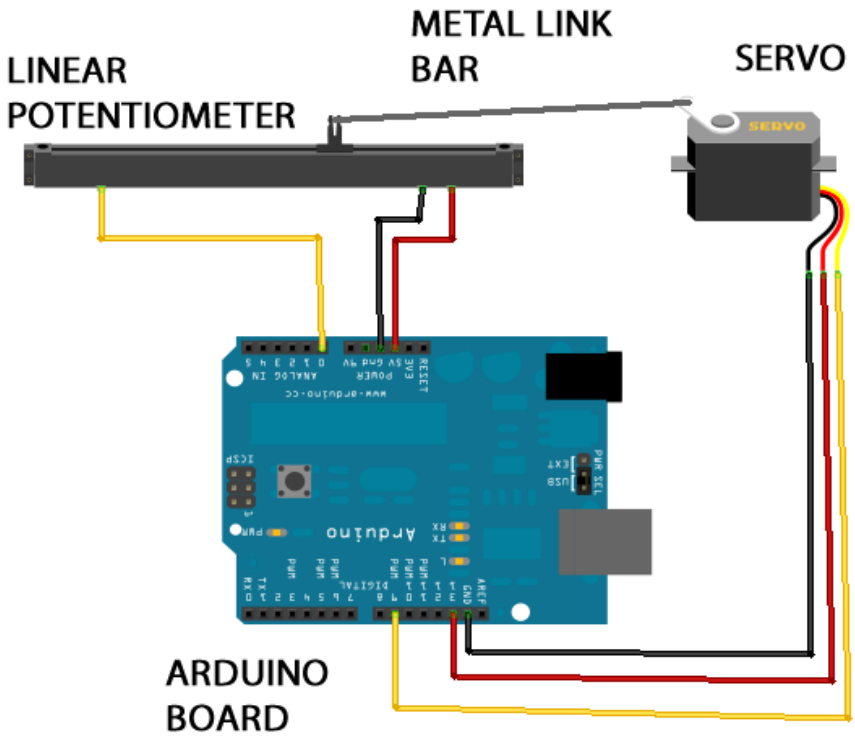



*Figure 2-5 A sketch of the force feedback potentiometer.*

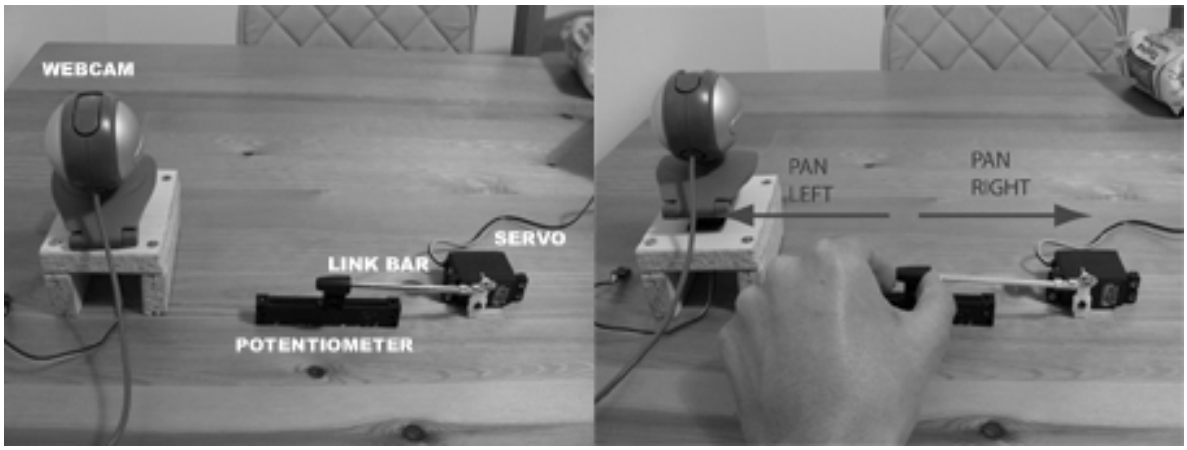



*Figure 2-6 The image on the left is the basic setup of the force feedback system. The image on the right illustrates the user moving the potentiometer left and right to pan the camera.*

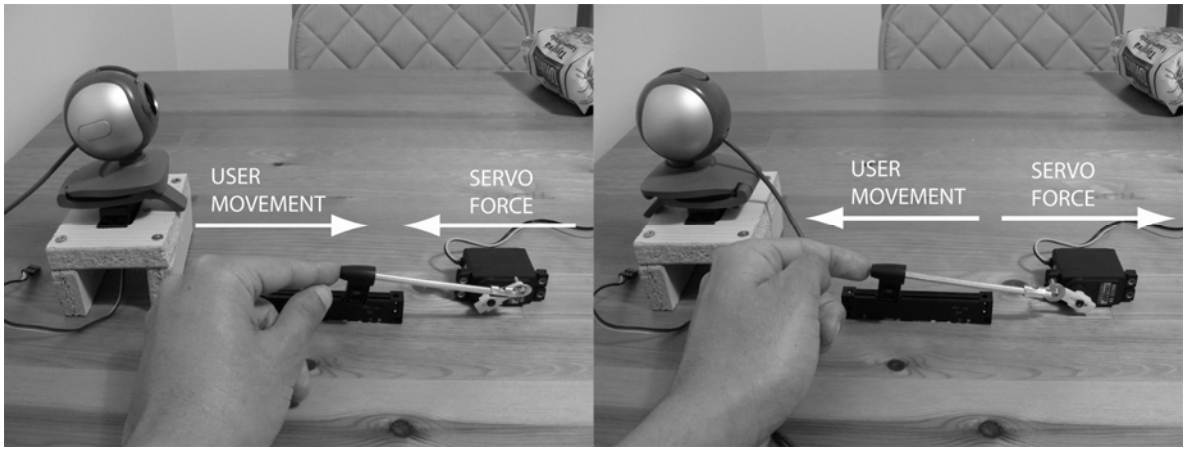



*Figure 2-7 The image on the left demonstrates the servo motor applying force to the potentiometer to indicate to the user that the camera has rotated to 180 degrees. The image on the right demonstrates the servo motor applying force to the potentiometer to indicate to the user that the camera has rotates to 0 degrees.*

The code implementation for this scenario requires 2 steps. First the Ardiuno microcontroller requires programming to read the values from the potentiometer and to turn on the servo motor and rotate to axle on the servo motor to create force, shown in code block 2-4. The second part is to create a computer application to read the values of the potentiometer from the Arduino board and process the value, shown in code block 2-5. In addition, the computer application must send values to the Arduino board to indicate whether force needs to be applied. Most haptics devices, such as the Novint Falcon, provide libraries to abstract the programming of the microcontroller device from the developers. The developers usually focus on the computer application and use various APIs and function calls to get input or send output to the device.

Arduino code used to control the servo

```
#include <Servo.h>

Servo myServo;

void disengageServo(){
  digitalWrite(13,LOW);
}

void fullRight(){
  digitalWrite(13,HIGH);
  myServo.write(180);
}
```

```
void fullLeft(){
  digitalWrite(13,HIGH);
  myServo.write(1);
}

void setup(){
  myServo.attach(9);
  Serial.begin(9600);
}

void loop(){
  if (Serial.available()){
    valueFromComputer = Serial.read();

    /* After reading the BYTE value from USB convert to INT and
    append using multiple of 10 to move decimal*/
    if(valueFromComputer >= '0' && valueFromComputer <= '9'){
      servoPosition = servoPosition * 10 (valueFromComputer - '0');
    }
    else if(valueFromComputer == 'X'){
     disengageServo();
    }
    else if(valueFromComputer == 'W'){
      myServo.write(servoPosition);
      servoPosition = 0;
    }
    else if(valueFromComputer == 'R'){
      fullRight();
    }
    else if(valueFromComputer == 'L'){
      fullLeft();
    }
    valueFromResistor = analogRead(0);
    Serial.println(valueFromResistor);
  }
}
```

*Code Block 2-6 The code used in the Arduino to control the servo.*

Code block 2-6 shows the program that is programmed onto the Arduino's microcontroller. The Arduino is instructed to read the potentiometer from Analog port 0 and send the value to the computer using the Serial.println() function. The Arduino also reads data from the computer using the Serial.read() function. The servo in this example is an RC servo which does not contain a clutch mechanism, therefore, when there is no force

that needs to be applied to the servo there is no power sent to the servo. If force is needed then the computer instructs the servo to energize and move the servo arm to a certain degree.

Interfacing Linux application to Arduino

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <sys/ioctl.h>

int main{

   int fd;
   int baudrate;
   char buf[256], serialport[256];
   int rc;
   struct termios toptions;

   speed_t brate = B9600; /* Sets the baudrate to 9600 to communicate to the Arduino/*

   fd = open("/dev/ttyUSB0", O_RDWR | O_NDELAY); /* Open USB connection to
Arduino*/
   cfsetispeed(&toptions, brate);
   cfsetospeed(&topions, brate); /* Sets incoming/outgoing baudrate in the struct. */
   tcsetattr(fd, TCSANOW, &toptions);
   …
   Some events and graphics to get input from user.
   …
}

int valueFromArduino(){
   char arduinoValue;
   int convertedValue;

   read(fd, &arduinoValue, sizeof(arduinoValue)); /*Read data from Arduino*/
   convertedValue = atoi(arduinoValue);
  return convertedValue;
}

void setServoPosition(int servoPosition){
```

```
  write(fd,servoPosition.toAscii(),sizeof(servoPosition));
}

void initiateRightSideStart(){
  write(fd,'R',sizeof(char)); /*Sends the command to start the force from the right side*/
}

void initiateLeftSideStart(){
  write(fd,'L',sizeof(char)); /*Sends the command to start the force from the left side of
the potentiometer /*
}
```

*Code Block 2-7 Code used to interface with Arduino. Modifications made on code created by Tod E. Kurt (Kurt, 2006).*

Code block 2-7 briefly illustrates how the computer communicates with the Arduino microcontroller board. The computer opens the device file of the Arduino microcontroller board which is attached to the USB port. The application reads the value from the Arduino microcontroller board using the valueFromArduino function and processes the data. Once the data is processed and the application determines that force is needed it sends the data to the Arduino microcontroller board and the process keeps repeating.

## *2.5. Multimodal User Interfaces*

In this chapter various user interfaces that are used in the tele-operation of devices were identified. However, many of the user interfaces utilized one particular hardware component to interact with the system. In a published study, it was determined that offering multiple input devices significantly increased the functionality of the robot across a group of users with different levels of expertise (Fong et al., 2001). The systems in this chapter primarily relied on the user's vision to determine what the device was doing, with the exception of the haptic interface. The computer generated graphics based on telemetry data or video from a camera. However, the graphical data may not provide sufficient situational awareness information to the user when their visual sensory channel becomes overloaded from other visual information. In another study it was proven that additional sensory feedback or multimodal interface design greatly improved the recognition of events requiring attention compared to systems that offered one output modality (Kaber et al., 2006).

Multimodal systems may enable system developers to create a natural interaction with the computer system. By providing the user a multiple array of inputs into the computer system, computer applications can become more robust due to redundant input channels. If one input channel becomes blocked the computer system can receive input from other channels. From the user's point-of-view, if they cannot use one input device because of physical limitations they can use other input devices offered by the system. (Reeves et al, 2004)

Multimodal user interface systems used in tele-operative devices primarily focused on the input and output devices but not the communication between the components in a networked environment. In two published papers in the area of tele-operated robots using multimodality, they described novel uses of new input devices (Fong et al., 2001) or providing the user with multiple feedback across various sensory channels (Kaber et al., 2006). The use of multiple input devices to perform a single goal proved to be very efficient. By providing multiple input modalities users with different level of expertise were able to interact with the robot using various remote tools (Fong et al., 2001). Kaber's paper (Kaber et al., 2006) provided substantial evidence if the user is provided with multi-sensory feedback or multimodal interface design that provided greater situational awareness then the user was able to interpret events better. In contrast, a design that utilizes one modality often makes it difficult for the user to become fully aware of the situation. Quite often a visual display of information is not sufficient as the user of the system may experience visual sensory channel overload if presented with too much information at once.

Designing multimodal user interfaces is not a simple task. As Leah M. Reeves' group (2004) described in their paper on the design considerations for multimodal interfaces, there are many factors to consider. Although the best efforts to address all factors may have been considered in the planning phase, some overlooked items may present themselves in the testing phase. One approach to developing multimodal user interfaces is to use a rapid prototyping approach used in software engineering.

Software prototyping provides numerous advantages in designing multimodal user interfaces. One primary advantage offered by software prototyping is that it serves as a tool for experimenting with new and novel concepts and interactions when creating software systems (Hekmatpour, 1987). Software systems designed using prototyping methodologies

are not constrained by predetermined system requirements from early requirement elicitation stages. Instead, a series of intervals are created to test and validate a small subset of features offered by the system by the prospective users. The active involvement of the users helps to create a user-centric design, as opposed to a system-centric design which forces the users to adopt the interface. Any problems encounter by the users can be quickly determined by testing and resolved in the next interval cycle. Once the problem has been resolved new features are introduced to the interval cycle. The cycle of testing, verifying and incorporation of new functionality continues until the software system has fully evolved. This type of prototyping methodology is known as the incremental prototyping.

Based on the successful result of multimodality devices in both studies (Fong et al, 2001; Kaber et al., 2006) the question remains if the same sort of study could be recreated with a distributed multimodal messaging system. The new system would allow future devices to be easily added to the system for usability testing without restructuring existing architecture using the incremental prototyping methodology.

The focus of the next chapter is to create a system that would enable developers to quickly implement new devices, evaluate the usefulness of the device and determine if the device is achieving the requirements of the system. In addition, provide developers the flexibility to use existing established Internet communication standards and not be burdened by learning new APIs and frameworks.

# 3. Communications System

Some user interfaces described in the previous chapter are typically used to allow the users to directly interact with the computer system located in close proximity to the user. However, the same user interfaces could be modified to interact with other computer systems or networked enabled devices located in a remote location. By making modifications to the existing computer application the devices can communicate over the Internet. The application shown in Figure 2-1 illustrates one example of how the user interface can be used to tele-operate a robot using the Internet.

Describing the events generated by the input devices into a XML structure and transmitting that XML structure over the Internet provides many benefits. Participants can easily connect to the network and participate in experiments using their own applications

and hardware, since the proposed communication system is utilizing current Internet standards. However, they must ensure that data exchanges are in the same XML structure for other systems to understand their data. Participants can create filters to select which modalities they would like to receive data from. Participants can fuse the XML data together to form their own syntax for their specific computer applications. Participants can globally collaborate on a single task, for example navigating a robot around a region of interest, and evaluate the information collectively.

The purpose of this chapter is to explain how the proposed distributed multimodal communication system works. This chapter has been broken down into 3 sections to further elaborate on the specific aspects of the system. The first section describes how computer systems communicate over the Internet. For the scope of this thesis the networking aspects will be described through a software development approach rather than going into full details about the technical specifications. The second section proposes a XML structure that could be used to describe the events generated by an input device. This XML structure could be broadcasted over the Internet to any computer systems or devices that are programmed to listen for the XML structure. Once the other computer system or device receives the XML structure, the contents of the XML message are analyzed and processed by the computer or the device. The third section summarizes and justifies the reason certain network protocols should be implemented in the distributed multimodal communication system.

## *3.1. Network Protocol*

Most technologies that are used to help one device communicate to another device located in a remote location follow the Open System Interconnection Reference Model, known as OSI Model for short. The OSI Model is divided into 7 different layers, shown in figure 3-1. Each layer has a detailed specification of how developers can implement its use in network related applications. The first layer in the OSI model is known as the 'Physical Layer' and mostly specifies various protocols and standards used to physically connect computers to networks, such as transmission mediums and signals. The first layer is usually out of the scope in terms of software application development in some areas of computing. For the scope of this thesis, the third and fourth layers are examined to provide

some insight of how the proposed distributed multimodal communication will work. The third layer titled the 'Network Layer' describes how data can be sent to the other networked devices by various network routing protocols and the fourth layer titled the 'Transport Layer' describes how the data connection is handled by the networked devices.

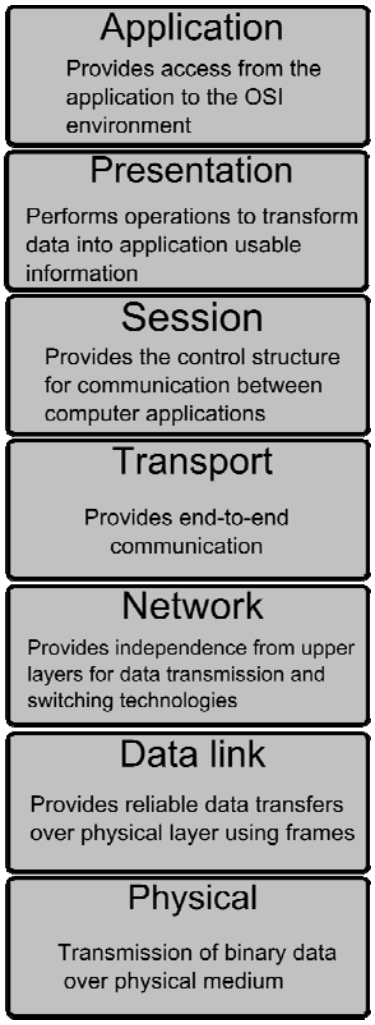



*Figure 3-1 An illustration of the 7 OSI layers where the Physical layer is the first layer and the Application layer is the seventh layer. (Stallings, 2002)*

### 3.1.1. The Network Layer

Computers and other devices attached to the network contain two attributes that are used to address other computers. These attributes are unique to each computer to avoid sending data to the wrong computer. The first attribute is the Media Access Control address (MAC Address) which is a sequence of 12 hexadecimal numbers with a colon or dash between every second hexadecimal to make it readable to humans, for example 00:11:22:33:44:ab .

The second attribute, which is also unique to each computer on a network, is the Internet Protocol (IP) address. The IP address follows the same functionality as the MAC address as a mechanism to address computers uniquely on the Internet, however, there are a few differences. The first difference is that an IP address is a series of 4 octets with a decimal separating each octet which creates a 32-bit address, for example 192.168.1.1. Another difference is that IP addresses are considered global addresses since they could be used to address computers outside of the local area network, whereas MAC addresses are used to communicate with computer within the local area network. (Stallings, 2001)

The IP address is a key element in designing the multimodal messaging system because it allows a mechanism of either sending data packets from one computer to another computer, one computer to all computers on the local area network, or one computer to any computer connect to the Internet. To communicate from one computer to another computer most application programming interfaces (API) simply require the destination computer's IP address and port number. Once the application runs, a connection is established from the source computer to the destination computer. This form of network communication is known as unicast. However, the ideal purpose of this multimodal communication system is to design a one-to-many data broadcast. Using a one-to-many implementation will allow the networked computer hosting the interactive device to broadcast the information generated from the device to all the computers on the local area network or on the Internet. Utilizing this implementation is rather simple as network broadcasting has been built into IP addressing. Broadcasting IP addresses are reserved addresses that are only used for broadcasting data to the local area network. Broadcasting a data packet to the local area network is a similar process as connecting to a single destination computer. The IP address 255.255.255.255 is substituted in place of the destination computer's IP address (Stevens, 1993). By using 255.255.255.255 in the network APIs the computer containing the modality is now able to broadcast data to all the computers on the local area network, this is known as network broadcast. Instances may occur where the computer containing the interactive device needs to communicate with many computers located on the Internet. To communicate with other networks outside of the local area network multicast could be used. Multicast is a reserved set of IP addresses, which range from 224.0.0.1 to 239.255.255.255, that allows data to be broadcasted from one networked computer to any

computer capable of accessing the multicast IP addresses (Stevens, 1993). The concept is similar to broadcast radio where the listener can tune to a certain broadcast frequency to hear the radio station broadcast. Anyone that can receive that broadcast frequency can listen to the radio broadcast. The only restriction is that the router must support multicast routing or the packets will not be transmitted to other networks.

### 3.1.2. The Transport Layer

The focus of the previous section was primarily on how IP addresses could be used to address other computers for network communications. While it may seem that developers of the multimodal applications have a trivial task of placing an IP address into the API network functions, the intention of the previous section was to provide some insight on how to use IP addresses to send packets to various computers. In this section, the focus will shift on how the applications will transport the packets from one computer to another computer or to many computers. This is primarily decided upon the programmer of the multimodal applications.

The role of the Transport Layer is to facilitate how the data packets are transmitted from one computer to another. The Transport Layer provides protocols to determine how the data packets flow, any error correction steps required and the sequence of packet delivery. The Transport Layer contains 2 commonly used protocols which are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

The Transmission Control Protocol provides a reliable, stream-oriented, connection-oriented transport protocol (Stevens, 1993). It is essentially used for transferring a continuous stream of data, for example a large file using the File Transport Protocol (FTP), where the data integrity must be protected. TCP is common in unicast communication because it establishes a link between the 2 computers. Within the established link additional data is transmitted which helps to control the flow of data. When the communication link is established by the 2 computers the data is sent at a synchronized speed between the computers instead of fluctuating speeds. This helps to avoid the receiving computer's buffers from overflowing in the event the computer cannot allocate resources quickly to the network card. TCP also checks if a packet is lost during transmission and if there are errors in the received data packet. The 2 main protocols used

in maintaining link speed and error corrections are the stop-and-wait protocol and the sliding window protocol (Stevens, 1993).

The stop-and-wait protocol works by the source computer sending a data packet to the destination computer. Once the destination computer receives the data packet, it checks the packet for errors. If there are no errors in the packet then it sends an acknowledgment packet to the source computer indicating it has successfully received the data packet error free. There are 2 scenarios that are present in causing a packet to be lost or corrupted. The first scenario is if the source computer sends a data packet and the destination computer never receives the data packet. The second scenario is if the destination computer receives the data packet and sends an acknowledgment packet but the acknowledgment packet is lost or corrupted. To resolve both scenarios a timer is added to the source computer. Once the data packet is sent from the source computer, a timer is started on the source computer. If the timer expires before the source computer receives the acknowledgement packet then the source computer retransmits the data packet. If the source computer receives the acknowledgment packet from the destination computer before the timer expires then the source computer transmit the next data packet in the sequence and the timer resets and starts. The stop-and-wait protocol is very inefficient when speed and bandwidth utilization are being considered since for every data packet sent an acknowledgment packet is required. The inefficiencies in the stop-and-wait protocol were resolved with the sliding window protocol. (Stevens, 1993)

The sliding window protocol is similar to the stop-and-wait but handles bandwidth utilization more efficiently and provides data to the destination computer quicker. A window is a set number of packets that the source computer is sending to the destination computer. The window contains a start packet and an end packet which is defined by the window size. For example, if the window size is 7 then the start packet would be packet number 1 and the end packet would be packet number 7. The source computer transmits all the packets in the window to the destination computer. If all packets are received by the destination computer then a acknowledge packet is sent to the source computer indicating all 7 packets in the window were received. The source computer will slide the window so that the start packet will be 8 and the end packet will be 14. The process repeats until the entire file or data stream has been transferred from the source computer to the destination

computer. If a packet is lost or corrupted during transfer from the source computer to the destination computer then the destination computer will send an acknowledge packet containing the last successfully received packet in the window. For example, if the source computer sends all 7 packets and the fourth packet is lost or corrupted then the destination computer will send an acknowledgment packet indicating packet 3 was successfully received. The source computer will then slide the starting packet window to packet 4 and retransmit the packets within the new window (packets 4 to 7) until an acknowledgement packet of 7 is received. In the event that the acknowledgment packet is lost or corrupted, the source computer has a timer of each packet in the window sent. If the timer expires and an acknowledgement packet is not received all the packets in the window are retransmitted. (Stevens, 1993)

The User Datagram Protocol (UDP) is the second commonly used protocol within the Transport Layer. Unlike the Transmission Control Protocol, UDP is considered an unreliable service. There is no data flow control, no error control and no sequence ordering. Each datagram data packet sent is treated as an individual packet when transmitted from the source computer to the destination computer. However, the low data overhead for processing the packets provides faster information transmissions. In real-time environments, the data from sensors, various computer systems and network components are constantly being transmitted at certain time intervals. If one of the data packets fails to be delivered to the destination computer another data packet will be transmitted shortly. Applications that require broadcasting data packets to the entire network can use UDP, since a connection to every computer may require too many resources and time using TCP. Real-time applications that provide redundant data or real-time transmission, such as video or audio, must be connection-less oriented to prevent a significant amount of latency in the data stream to the destination computers. (Stevens, 1993)

## *3.2. XML Structure*

The eXtensible Markup Language, or commonly known as XML, was derived from the Standard Generalized Markup Language (SGML). XML helps create standardized structured data in textual format. The format is quite similar to the Hypertext Markup Language (HTML), which is used to describe the layout of web documents. However,

XML can be applied to any application domain other than presentation layouts. Information specific to an application domain can be represented in XML by creating a document to represent that information or extending on previous XML documents. Since XML is a text based structure any text editor or text-oriented tools could be used to create and modify the contents of XML. XML is not a proprietary technology, therefore it is free for anyone to use, modify and distribute. Due to those facts, XML has been very successful in the open source community. (Treese and Stewart, 2002)

Analysis of previously published research papers and other projects related to the area of distributed communication over computer networks indicates a trend leaning towards the usage of XML messaging (Gudgin et al, 2007; Larson et al, 2003; Menezes, 2008; Saint-Andre, 2004). The key benefit outlined by Harold and Means (2004) illustrated that XML can be used to textual describe information and/or events. By simplifying the input devices to determine basic events, encoding those events into XML is a trivial task. For example, if the user presses the left arrow key on the keyboard then the XML message could define an event describing a left arrow key press on a keyboard. Another example could be if the user moves their hand to the left then the XML message could define an event describing a gesture movement to the left. The basic idea is to describe the current input event and broadcast the XML structure on the network and then have the output devices interpret how to utilize the XML structure.

The Extensible Messaging and Presence Protocol (XMPP) and Simple Object Access Protocol (SOAP) are 2 examples of Internet messaging protocols that utilize XML. Under the OSI framework both protocols are classified under the Application Layer (OSI Layer 7) since the transport of the packets and the networking of the packets are handled by the Transport Layer and the Network layer described in sections 3.1.1. and 3.1.2. XMPP and SOAP are both open technologies. Open technologies allow any individual or organizations to review, implement or alter these protocols to incorporate these technologies into their systems. Another benefit of being an open technology allows XMPP and SOAP to be implemented in a heterogeneous environment. This allows the organization to have the flexibility to use any operating system and applications rather than structuring their business around proprietary solutions. By having this flexibility, organizations can create new interfaces or modify pre-existing interfaces into the system

rather than waiting for the vendor of the system to create additional add-ons. There is also the potential chance that the vendor does not want to implement a requested feature but the organization could implement the features if the system is built on open technologies.

XMPP is an open protocol designed for near-real-time messaging, presence and request-response services using client-server architecture. In a client-server architecture, a centralized computer known as the server facilitates communication between clients. If client A wishes to transmit data packets to client B, both client A and client B are required to establish a connection to the server first. Client A will transmit the data packet to the server and then the server will transmit the data packet to client B. This communication between the clients and servers are conducted through TCP in XMPP applications. An example of the communication process is shown in code block 3-1. (Saint-Andre, 2004)

Example of a XMPP chat

```
C: <?xml version='1.0'?>
     <stream:stream
        to='somejabberserver.com'
        xmlns='jabber:client'
        xmlns:stream='http://somejabberserver.org/streams'
        version='1.0'>
S: <?xml version='1.0'?>
     <stream:stream
        from=' somejabberserver.com'
        id='id1'
        xmlns='jabber:client'
        xmlns:stream='http:// somejabberserver.org/streams'
        version='1.0'>
...  encryption, authentication, and resource binding ...
C:   <message from='foo@ somejabberserver.com'
            to='bar@ somejabberserver.net'
            xml:lang='en'>
C:     <body>Hello, how are you?</body>
C:   </message>
S:   <message from='bar@ somejabberserver.net'
            to='foo@ somejabberserver.com'
            xml:lang='en'>
S:     <body>Not bad how are you?</body>
S:   </message>
C: </stream:stream>
S: </stream:stream>
```

*Code Block 3-1 An XMPP example of 2 chat clients communicating through the server. "C" is the client and "S" is the server. (Saint-Andre, 2004)*

SOAP is slightly similar to XMPP, in that it is also built as an open protocol. SOAP is a request-response service based on client-server architecture. SOAP works by sending XML formatted data structures over Hypertext Transfer Protocol (HTTP) to a web server. The web server parses the XML structure for data, interprets the data and provides a response to the data in a XML structure to the client. An example of a SOAP communication between a client and a server is shown in code block 3-2. (Arora and Kishore, 2002)

Example of a client querying a server for information using SOAP

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”>
        <soap:Body xmlns:addSum=”urn:http://somesoapserver.com/addSum”>
                <addSum:getSum>
                <iValue1 xsi:type=”int”>1</iValue1>
                <iValue2 xsi:type=”int”>1</iValue2>
                </AddSum:getSum>
        </soap:Body>
</SOAP-ENV:Envelope>

<SOAP-ENV:Envelope xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”>
        <soap:Body xmlns:addSum=”urn:http:// somesoapserver.com/addSum”>
                <addSum:getSumResponse>
                <return>2</return>
                </addSum:getSumResponse>
        </soap:Body>
</SOAP-ENV:Envelope>
```

*Code Block 3-2 An example of a client querying a server for information. The first SOAP-Envelope represents the user’s request and the second SOAP-Envelope represents the server’s response. (Arora and Kishore, 2002)*

The XMPP and SOAP examples shown above were a simple demonstration of XML data communication in a heterogeneous environment between various client computers. Other examples of applications utilizing XMPP are internet messaging, group chat, gaming, systems control, geolocation, cloud computing, data syndication, Voice over IP and Identity services (Saint-Andre et al, 2009). SOAP based applications are prominent in web based applications that handle user authentication, database queries and store-and-forward queue based mechanisms such as sending users email confirmation of a completed task (Cohen, 2001). When XMPP and SOAP data structures were analyzed it seemed that they were not particularly addressing the communications between multimodal messaging between devices. Compared to SOAP, XMPP does provide some flexibility in multimodal

communication since it has been used in systems control based applications. However, documents outlining how to implement XMPP for tele-operating devices are scarce. The World Wide Web Consortium, commonly referred to as W3C, has released a series of documents in an attempt to standardized multimodal communication in a XML structure (W3C, 2009). However, as Menezes (2008) pointed out, W3C documents their standards on how the components could communicate but does not describe how they interact or connect with the web browser in the examples from the W3C implementations. Additionally, the W3C standards seem to be focus on particular use cases making it difficult to abstract the standards to apply to other situations. Menezes (2008) proposed a protocol that could address the gaps of other multimodal protocols and still leave room for flexibility to allow future additions to the system. Key design goals in Menezes' Multimodal Middleware Protocol were a lightweight client, language and platform independence, network independence, publishing events and extensibility. Lightweight clients were essential because many embedded devices and limited resource devices, such as some mobile phones, do not contain computational power like desktop computers. Keeping the computation requirements low allows many devices to participate in a mutlimodal environment. Language and platform independence are critical for a heterogeneous environment because many devices do not share the same development libraries or binary data formats, thus, the protocol must be open to allow every device to participate. Network independence is another key goal because network devices are constantly being improved and made more efficient. Having an extensible protocol allows other developers to improve or add to existing components within the protocol.

Menezes' Multimodal Middleware Protocol describes numerous data types and possible network communication methods between components. The protocol also mentions the use of a centralized multimodal hub as a special component to link various components using a virtual bus. The multimodal hub is responsible for communication between the components and also contains all the computational complexity of notification of events to the components. The multimodal hub allows the components to be thinner in terms of computational resources and simplifies the software development of each component.

One interesting aspect from the Menezes' Multimodal Middleware Protocol is the Event Specification Format (ESF) XML structure, shown in code block 3-3. The ESF XML structure has been created to exchange event information by defining what the event is, what modality initiated the event and a unique identifier to refer to each event. The ESF XML structure is important because it is a step towards a standardized messaging system, like XMPP and SOAP, for a multimodal system.

Example of the Event Specification Format XML Structure

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE esf SYSTEM "mmp.dtd">
<esf>
  <event id="1" description="Cursor Interaction">
    <field name="DeltaX" type="Long"/>
    <field name="DeltaY" type="Long"/>
    <field name="Button" type="Long"/>
    <field name="Pressed" type="Long"/>
  </event>
  <event id="2" description="Voice Command">
    <field name="RecognizedSentence" type="LongString"/>
    <field name="SentenceScore" type="LongString"/>
    <field name="FrameIndex" type="Long"/>
  </event>
</esf>
```

*Code Block 3-3 An example of Menezes' Event Specification XML Structure (Menezes, 2008).*

While there were other documents that had their own data structure for multimodal communication, Menezes pointed out that in multimodal communication networks the emphasis must be on lightweight communication with many opportunities to create independencies. The Event Specification Format XML structure provided by the Multimodal Middleware Protocol is one data structure a developer could use to communicate events created by a modality. The Modified Event Specification Format XML structure defined below was conceived and modified from works done by Heistracher et al (2006) and Menezes (2008).

Modified Event Specification Format XML Structure

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE mesf>
<mesf>
```

```
<event id="0001" sequence="1">
 <device>
  <type>Joystick</type>
  <value>UP</value>
  <auxvalue>1554</auxvalue>
 </device>
</event>
</mesf>
```

*Code Block 3-4 An example of the Modified Event Specification Formal XML structure being propose in this thesis to facilitate multimodal communication.*

The Modified Event Specification Format (MESF) has been designed to quickly capture all the information for one event. While other implementations treat events as a single entity on the network, the MESF takes into consideration other multimodal experiments taking place on the same network. The event tag contains 2 attributes. The first attribute is the Event Identification (id) number. This attribute is used to identify which event (or experiment) this XML packet is participating in. For example, on the network there may be 2 different experiments running. The id will help identify which XML packet is participating in the different experiments. The sequence attribute is used to define the order of the event's occurrence. The device tags nested inside the event tags is used to quickly describe the device's event. The type tag is used to indicate what type of device is emitting the event. The value tag indicates a value that the device is emitting. The value tag can be textual, numeric or alphanumeric. The auxvalue tag is used to provide further information about the value emitted by the device. In this example a joystick modality has emitted an UP value and the auxvalue represents the integer value of the joystick.

## *3.3. Proposed System*

For the purpose of this thesis, the goal is to create a distributed messaging system that would allow any computers or devices participate in a "multimodal environment" utilizing minimal resources. A multimodal environment could be a laboratory setup with various computers containing different types of user interface devices. Those devices may be confined to a specific computer due to resources needed to run the user interface, such as special hardware. The researchers may want to analyze how the combination of different user interface devices may help a user in a certain task. Various test setups could be

arranged quickly to allow the researchers to test their theories of certain interface interactions. The results of the test can then be analyzed and more testing could continue until a valid conclusion could be made. This is similar to the interval prototyping software methodology. To facilitate the communication between the computers in the multimodal environment, the network is used as a data bus by broadcasting the MESF XML structure to all the computers.

Depending on the organizational network layout, one network solution may not be compatible and changes may be needed to provide an effective solution. For example, organizations may have their networks designed for certain functionalities that may not integrate well with certain applications. By providing a system that provides flexibility and independence, the developers are not forced into confined development environments. The OSI model has been developed to standardize the data communication between different networking technologies. Developers creating network applications could utilize various network APIs from any software vendor without requiring knowledge of lower level network communication protocols, such as those found in OSI layers 1, 2 and 3.

The use of multicast would be an ideal solution for distributed messaging since one data packet could be sent to all computers and devices participating in the multimodal environment whether they are local or geographically separated. Multicast has numerous advantages with efficient bandwidth utilization. When multicast was compared to TCP, it was evident that the use of multicast provided efficient use of bandwidth (Legout, 2001). TCP is stream-oriented that uses the sliding window protocol for creating a reliable data transfers. If numerous computers want to participate in data exchange to a central server design, such as XMPP, then the client computers must connect to the server. If the data packet is 80 kilobits and the client wants to send that data to 10 other clients than the total bandwidth for one data packet transfer is 800 kilobits. As more clients are added to the system the total bandwidth goes up by 80 kilobits per client. Combined with other client data packets the network will become congested. Multicast is based on datagram-packets and not stream-oriented. Therefore, one packet of 80 kilobits distributed to 10 clients leads to a total bandwidth utilization of 80 kilobits. If more clients are added the total bandwidth utilization does not change.

However, not all routers are multicast enabled thus this is not a solution for many organizations. In addition, constant mutlicast broadcasting may degrade the network performance of other tasks taking place on the network. For example, all the computers on the network will receive the broadcast data and other data on the network card. The computer must then evaluate the data to determine if any applications on the computer requires that data. Essentially, this could fill the buffer space on the computer's network card with unneeded data and resulting in wasted computational resources. Instead, routing of the packets could be utilized so the computers that want to participate could communicate directly to each other while not affecting others. A possible solution for distant collaboration could be the use of a web server as described in the SOAP architecture or a special server to handle XMPP clients. Modifications could be made to the web server to accommodate multimodal XML event messaging but this requires additional hardware setup and if the server goes offline then the entire multimodal messaging system no longer functions.

Another solution to remedy the network issue is to connect all the computers and devices to a local area network. The computers and devices could communicate using the User-Datagram Protocol. By broadcasting packets to the broadcasting IP address, they are communicating to each device in the same mechanism as multicast. The same bandwidth utilization is achieved as explained in multicast. However, computers and devices outside the local area network will not be able to participate. Either way, using existing networking standards provides the developer many opportunities to create a network for transmitting multimodal data.

By combining the flexibility of using any networking method of transmitting the MESF XML data over the network, researchers and developers can use various tools to help them collect data on experiments. Data loggers could be written to listen on the network to capture these MESF XML packets for real-time analysis or store those packets in computer files for later analysis. Further analysis can provide insight into what has occurred during each experiment. For example, if the experiment was testing out a new user interface for operating tele-operated devices, then test cases could be created and the values could be plotted on a graph to discover trends. In a housing automation environment, a statistical trend analysis could be used based on numerous temperature and

lighting sensors. Using the same MESF XML packet, the sensors could broadcast information based on their readings to the network. A server can collect all this information and forecast predictions based on the user's behaviour. The system can be combined with heating and cooling system to efficiently control the climate to the user based on their interactions with the system. For example, if there is too much light in the room coming from the sun and the user adjusts the temperature for cooler air, then the system could record those events. If the situation keeps repeating where the light sensors records a lot of light combined with a drop of temperature then the process could be slowly automated to avoid the user constantly controlling the temperature themselves. The MESF XML packet was designed to be flexible enough to accommodate any events generated by sensors.

# 4. Use-Case Scenario

The purpose of this chapter is to apply the distributed multimodal messaging system that was described in chapter 3. A scenario was constructed that allowed volunteers to participate in an experiment. The MESF XML messaging system was used to record the volunteers' interactions with the devices. After the experiment was complete, the MESF XML messages were used to demonstrate how results can be analyzed based on the experiments.

## *4.1. Multimodal Tele-Operation*

The use of tele-operated devices has allowed humans to explore environments that may present physiological complications to the human body. Environments that may be complicated for humans to explore are outer space, deep underwater, biological hazardous areas and war torn areas. Tele-operated devices, such as mobile robots, have provided humans the opportunity to remotely survey areas while the operator remains in a safe location. The tele-operated robots are usually equipped with various sensory electronic components that capture the environment around the robot and send the data back to the tele-operators, examples are the recent two Mars rovers.

In this experiment, volunteers controlled a remotely located motorized web camera. A video of a red ball was projected through a video projector onto a screen in a remote

room containing the motorized web camera. The video from the web camera was sent over the local area network to the volunteer's workstation where they observed the video from the motorized web camera. The projected red ball moved around the projector screen and the volunteers attempted to keep the red ball centered on their video screen by moving the motorized web camera. There were a total of 6 different paths for the volunteers to follow which ranged from simple linear paths to paths involving a combination of pans and tilts. Moving the motorized web camera was accomplished through the use of the three input devices.

The selected input devices for this experiment were a WIMP-interface, joystick controller and a face tracker. The selected output device was only a video taken from the motorized web camera and transmitted to the volunteer's workstation computer monitor. The motorized web camera consisted of 2 servo motors on a custom made rig to allow one servo motor to pan the camera and the other servo to tilt the camera. The servos were attached to an Arduino Duemilanove 328. The Arduino was programmed to receive commands from the USB connection and manipulate the servos based on those commands. The computer containing the input devices broadcasted the Modified Event Specification Format (MESF) XML packet to the LAN. The computer, which the Arduino is attached to, listened to the LAN for these MESF XML packets and processed them. Once the packet had been processed the information pertaining to the positioning of the servos was transmitted from the computer to the Arduino. Both computers used in this experiment were running the Linux operating system.

## *4.2. Method*

### 4.2.1. Participants

Volunteers for the experiment were selected based on the order of earliest availability to participate in the experiment. There was an attempt to diversify the volunteers based on their academic backgrounds and experiences with multiple computer interfaces. This was to ensure that an even representation of the population could be represented. Prior to the start of the experiment volunteers received an assessment survey to complete. Based on the assessment survey the following results were determined. There were a total of 8

volunteers in this experiment. 4 participants had no formal computer science training in the area of Human Computing Interfaces (HCI), 2 out of the 4 had brief exposure to devices where the mouse and keyboard was not the main interaction device and the other 2 had an extensive amount of experience with various game controller devices. The 4 other participants had computer science backgrounds with knowledge in the area of HCI. Out of these 4, 3 had extensive experiences with various device interfaces and only one had moderate level of experience with various interfaces.

There was no consent forms used in this experiment and all data collected from the survey only contained the volunteer's identification number. The test coordinator kept the association between identification number and volunteer in memory. There were no recorded documents that could link the results to the volunteers. The data generated from the computer were stored on password protected servers. The web camera used in the face tracking interface application did not store any images onto the computer.

### 4.2.2. Interfaces

The WIMP interface was designed using Nokia's Qt library. Using the Qt library allowed this interface to be cross-platform, meaning that it would be able to run on any operating system. Qt's API provided functions to create the MESF XML data structure and the QUdpSocket class to transmit the XML packet to the broadcast IP address. Although the menu aspect of the interface was not implemented, the basic characteristics of a WIMP interface were still present. The volunteer was shown a window containing four arrow buttons and a button labelled "center", shown below in figure 4-2. The icons provided the mechanism to move the motors in four directions and the pointer, in this case the mouse, was used to allow the volunteer to select their choice they wish to make. When the volunteer clicked on one of the buttons a MESF XML packet was created to describe the action for the motorized web camera to conduct. The MESF XML packet was then broadcasted to the local area network.

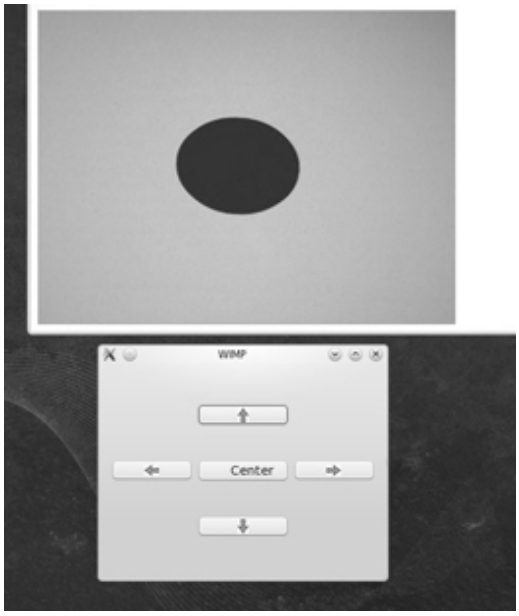



*Figure 4-1 Screen shot taken of WIMP-based controller.*

The joystick interface, shown in figure 4-3, was designed using Linux libraries to access the joystick device and Nokia's Qt libraries to create the MESF XML packet and UDP network broadcasting. By using the Linux libraries the joystick interface was not cross-platform, meaning that it would not run on other operating systems. This represented a modality with confinements specific to the operating system. However, by using the MESF XML packet the joystick interface would still be able to communicate with other devices or computers on the LAN. Unlike the WIMP interface, there was no visual display to the volunteer of the status of the joystick controller. The volunteer's desktop only contained one window which was the video display from the motorized web camera. The volunteer was able to control the motorized web camera by moving the left analog joystick up or down to control the tilt. The right analog joystick could be moved left or right to control the pan of the motorized web camera. When the joystick interface application started, a thread was created to read information from the joystick. The thread prevented the joystick interface application from creating unsynchronised pauses between the movement of the joystick and the transmission of MESF XML packets. Previous tests showed that by not using the thread there was a significant delay between the movement of the joystick and the transmission of MESF XML packets. This lead to a latency which caused the motorized web camera to move unsynchronised to the joystick movement. The

latency would cause the volunteer to over compensate the direction of the motorized web camera while trying to keep the red ball centred on their video screen. A sampling rate function was added after the pilot testing phase to the joystick interface application which helped control the rate of information collected from the joystick. A high sampling rate created a very sensitive feeling with the joystick. The high sampling rate meant that the joystick device was polled very frequently which created a burst of MESF XML packets being sent to the motorized web camera. This caused an uncontrollable experience forcing the pilot user to constantly over compensate their movement. The lower sampling rate created a long latency between the movement of the joystick and the movement of the web camera, since the time for the MESF XML packet to be created was also delayed. The value that was selected for the sampling rate was based on trial and error testing.


*Figure 4-2 A photo of the joystick controller that was used in the experiment.*

The face tracking interface, shown in figure 4-4, was created using Nokia's Qt and OpenCV. Qt was used to display the video obtained from the webcam located on the volunteer's workstation, create the XML data structure, provide the networking API and thread classes. OpenCV was used to capture the images from the web camera located on the volunteer's workstation and perform image analysis to determine where their face was.

Based on the location of the volunteer's face the motorized camera would move in a certain direction. There were two windows on the volunteer's desktop in this implementation. The first window was the video from the motorized web camera and the second window displayed the video from the volunteer's web camera, which contained a video of their face. In the center of the window was an outline of a turquoise coloured box. This box illustrated a position where if the face was detected inside this turquoise outline the motorized camera will not move. If the face was captured outside this box then the motorized web camera would move. When a face is present on the volunteer's web camera a blue box is drawn around the face. The upper left point of the blue box was processed to determine where to move the motorized web camera. If the motorized web camera needed to move then the new servo motors positions were passed to a function to create the MESF XML packet. Qt's network API algorithms were used to broadcast that MESF XML packet to the LAN. Preliminary testing, which were conducted prior to the experiment, showed that the face tracking interface application was failing to correctly capture the face of the volunteer 100% of the time. This led to an uncontrollable motorized web camera because the face tracking portion of the application needed to restart the face capturing processes once it has lost track of the volunteer's face. Upon detection of the volunteer's face, from the face tracking application, a new set of coordinates were created and mapped to move the motorized web camera. This created a sporadic increase of unwanted MESF XML packets being sent to the motorized web camera, which resulted in the motorized camera moving in wrong or undesirable directions. Preliminary tests also showed that it would be quite difficult for volunteer's to move around the lab environment, while keeping their face in focus of the web camera and observe both windows at the same time. An improved solution was created to salvage this interface. A photo identification card was use in place of the volunteer's face. The test showed that there was a significant improvement between the volunteer's action with the photo identification card and the movement of the motorized web camera. Changes to the algorithm were made to model the movement of the motorized web camera following a tripod metaphor. When the face on the photo identification card moved below the turquoise outline the motorized camera tilted upward, moving the card above the turquoise outline moved the motorized downward camera, moving the card left

of the turquoise outline moved the motorized right camera and moving the card right of the turquoise outline moved the motorized left camera.

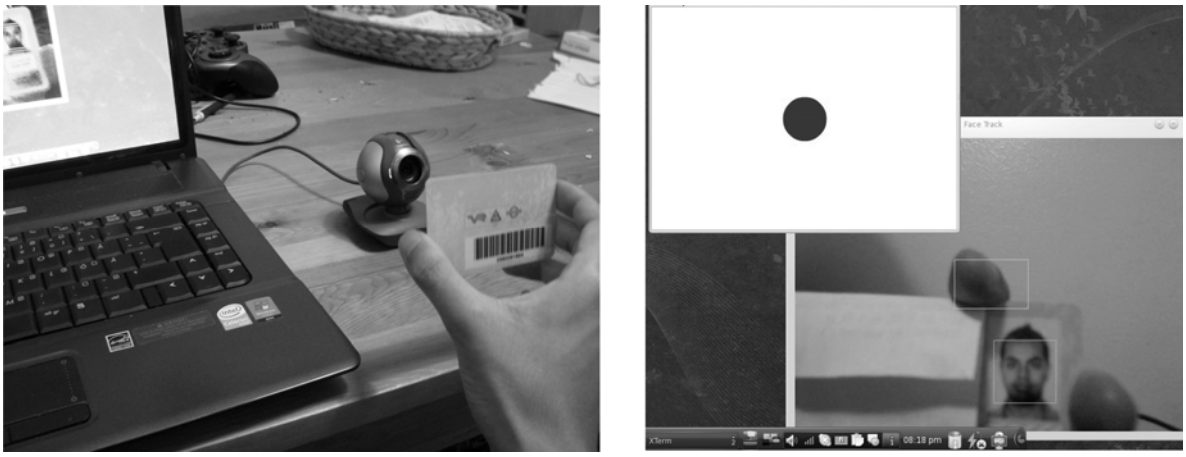


*Figure 4-3 The photo on the left illustrates the user holding the identification card near the web camera. The photo on the right shows the computer's desktop. The window on the top left is displaying the video from the tele-operated camera. The window in the bottom right is showing the application used to track the face.*

The video of the motorized web camera to the volunteer's workstation was created by running an application called xawtv. From the volunteer's desktop, a SSH connection was created to the computer containing the motorized web camera. The X display was forwarded back to the volunteer's workstation. Other methods were tested but there was a significant amount of latency with the video from the motorized web camera to the volunteer's workstation. The latency would have led to over or under compensation of the motorized camera's movement by the volunteer.

### 4.2.3. Design

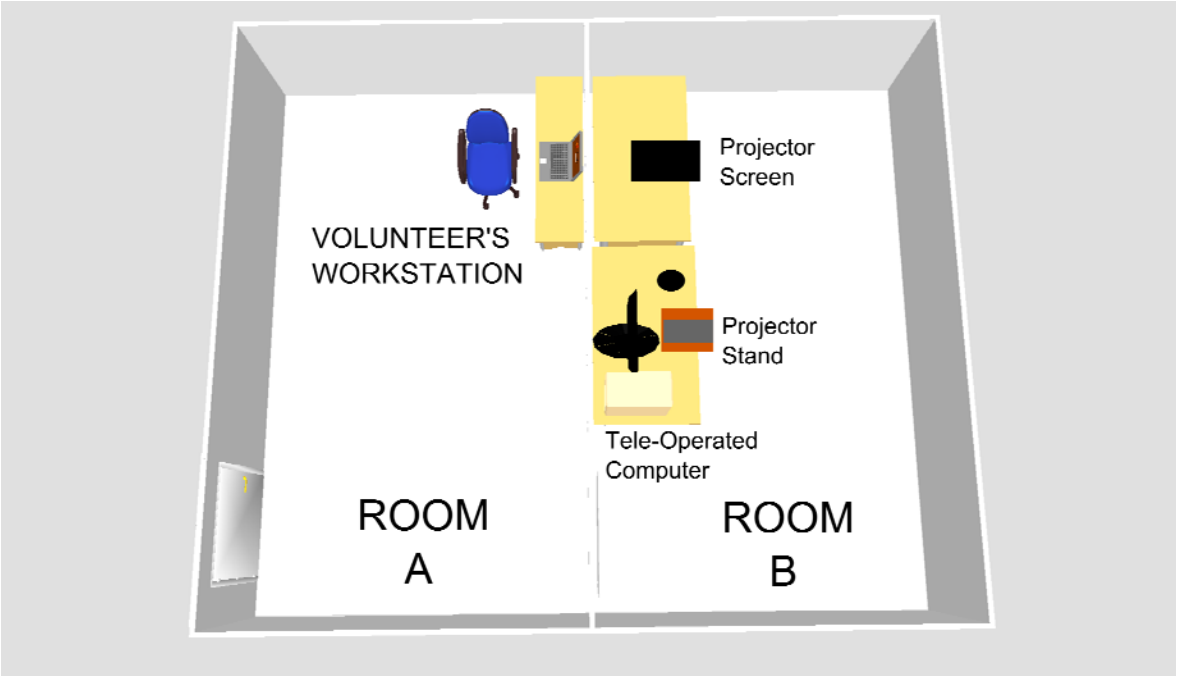



*Figure 4-4 A simple sketch of the experiment setup.*

Two rooms were used in this experiment, shown in figure 4-x. The first room (Room A) was used for the volunteers to be seated at a computer workstation with the 3 devices attached to the computer. The second room (Room B) contained the motorized webcam attached to a computer. There was a hole in the wall so the computers could be attached to the same network switch so they could communicate on the local area network. All the computers participating in this experiment were assigned IP addresses that belonged to the local area network provided to the lab via DHCP. In this case the IP address range was from 153.1.60.1 to 153.1.60.255. All the devices communicated using the proposed MESF XML packet from chapter 3 of this thesis. In addition, all devices related communication utilized UDP packets and broadcasted to the network's broadcasting IP address, 255.255.255.255. When the computer in Room B captured and processed the MESF XML packet, the data was written to a log file and the servo commands were issued to the motorized web camera.

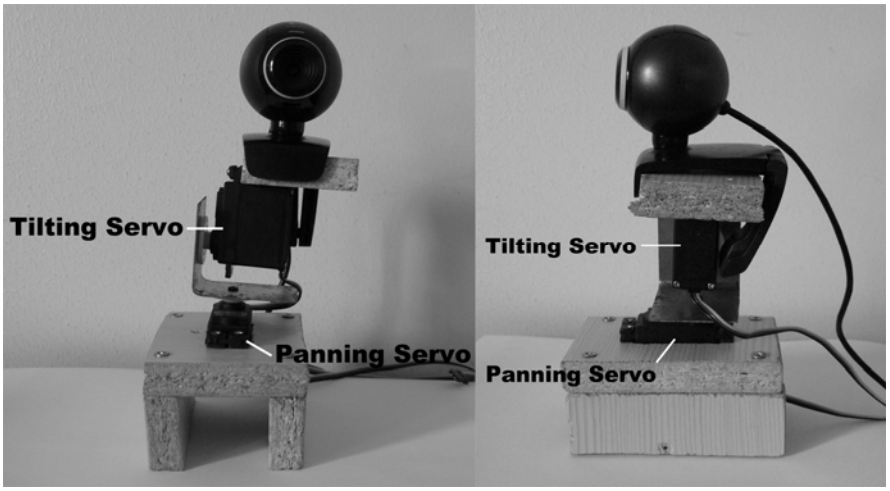



*Figure 4-5 A photo of the motorize web-camera setup. Note: the slating was a result of a transportation accident. When the experiment was carried out there was no tilting of the apparatus.*

The servos on the custom built motorized camera rig, shown in figure 4-5, were attached to the Arduino Duemilanove 328. The Arduino was connected by USB cable to the computer. The webcam attached to the motorized camera rig was also attached to the computer via USB cable. A cardboard box with white paper attached was placed 27 cm in front of the motorized web camera. The cardboard box was used as a projector screen. A projector was placed on an elevated wooden platform behind the motorized web camera. The distance between the lens of the projector and the projector screen was 85 cm. The projector was angled so that the image was 21 cm above the table surface. The projector image dimension was 52 cm x 36 cm. A sketch of the setup can be seen in figure 4-6. The video for the projector was provided by the secondary video port on the graphics card located on the computer in Room B. A calibration was made by placing the motorized web camera in the middle of the projected image. When Arduino controller board was activated the servos were forced into 90 degree positioning. The image was checked to see if the red ball was located in the center of the video display.

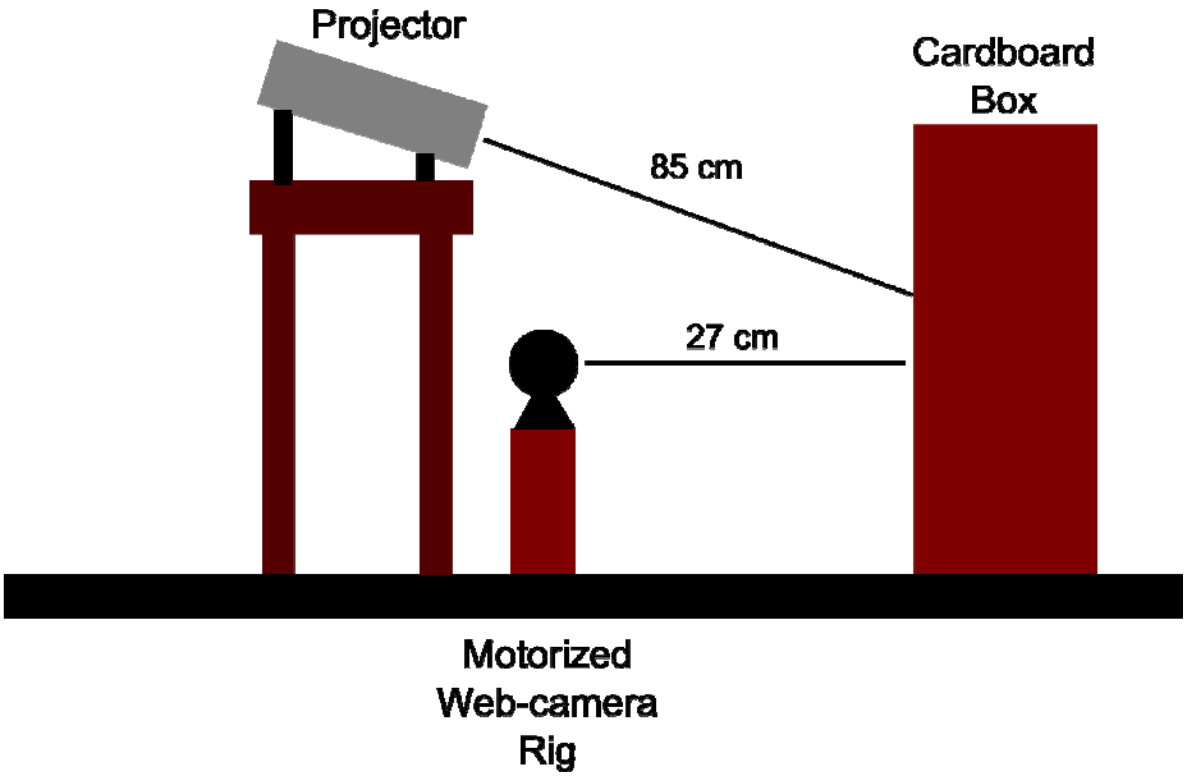



*Figure 4-6 A sketch of the layout of the projector, motorized camera rig and the cardboard box used as a projector screen.*

There were a total of 6 different paths created for this experiment. In every path the red ball started in the middle of the screen, travelled through a path and ended in the middle of the screen. The total time for the red ball to make its journey was 40 seconds. The 6 paths were designed in Flash using the motion tween feature.

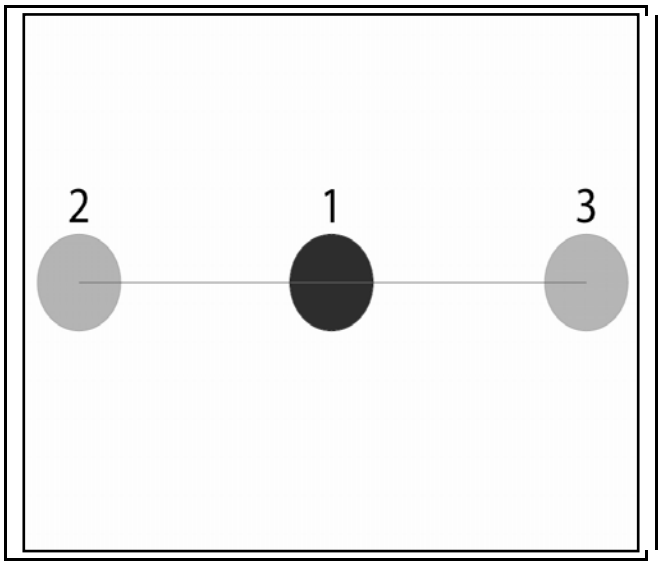



*Figure 4-7 An illustration of Path 1*

Path 1, shown in figure 4-7, was the simplest path the volunteers followed in the experiment. The red ball simply moved from the center (point 1) to the left (point 2), all the way to the right side (point 3) and back to the center of the screen (point 1).

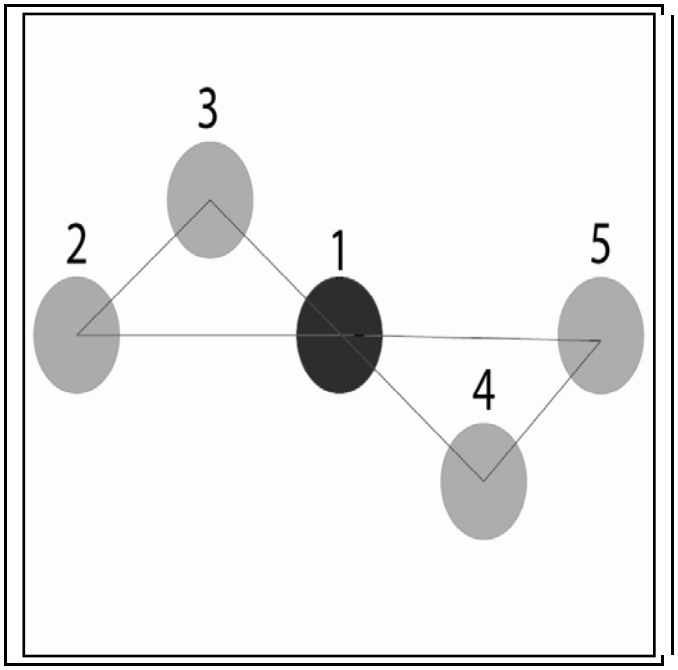



*Figure 4-8 An illustration of path 2*

Path 2, shown in figure 4-8, is similar to the first path with the addition of the volunteers using the tilt functionality of the motorized web camera. The red ball first moved from the center (point 1) to the far left (point 2). From point 2, the ball traveled in the up-right direction to point 3. Once the red ball arrived at point 3 it proceeded to point 4 travelling in a down-right direction. When the red ball arrived at point 4 it proceeded to point 5 and then traveled left back to the center (point 1).

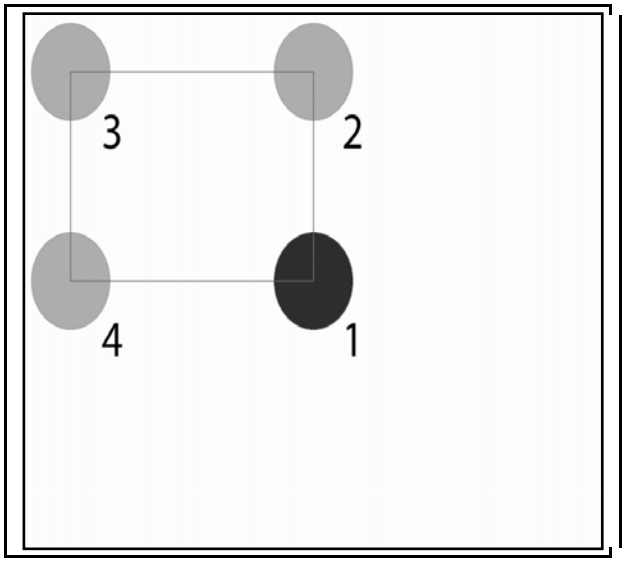



*Figure 4-9 An illustration of Path 3*

In path 3, shown in figure 4-9, the red ball moved in a box formation on the upper left portion of the projector screen.

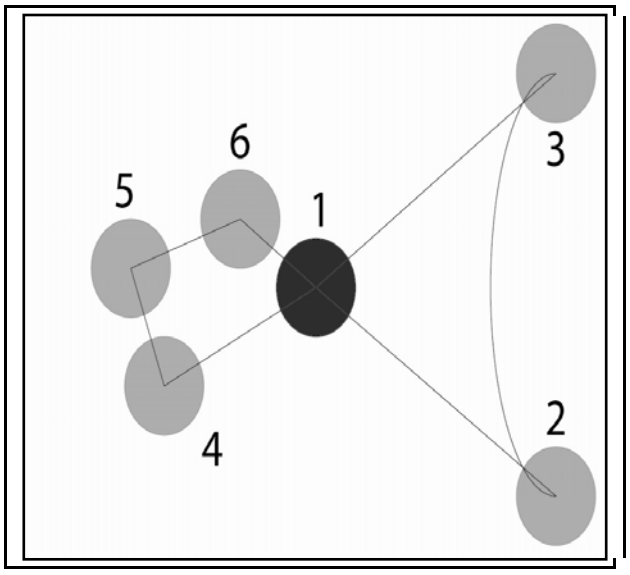



*Figure 4-10 An iillustration of Path 4*

In path 4 (figure 4-10), the introduction of curve and angled paths were introduced to the volunteers to follow.

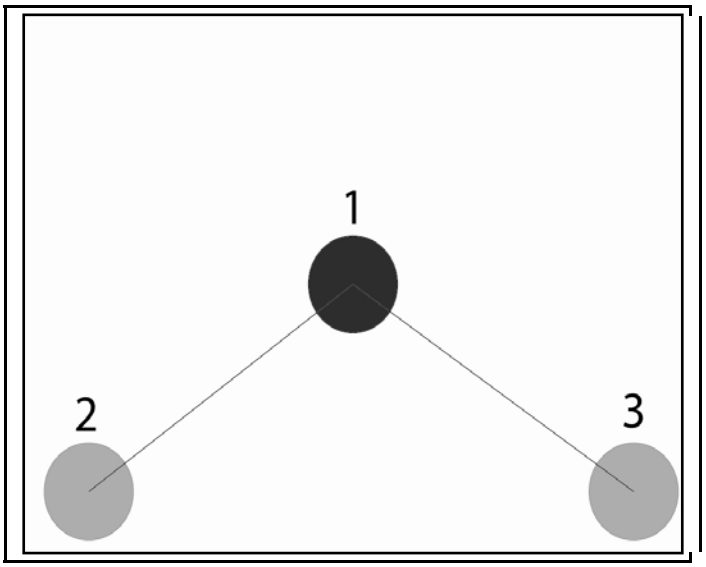



*Figure 4-11 Illustration of Path 5*

Path 5 (figure 4-11) emphasised the use of diagonal movements. The volunteers followed the red ball from the first point to the second point. The red ball then moved back to the middle of the screen and then travelled down right to point 3 and returned to the center.

Path 6, shown in figure 4-12, was designed to be very difficult for the volunteers to follow. The paths were fairly random with 2 exceptions. The goal was to generate noisy data to compare the results against easier paths, like Path 1, to detect any noticeable trends.

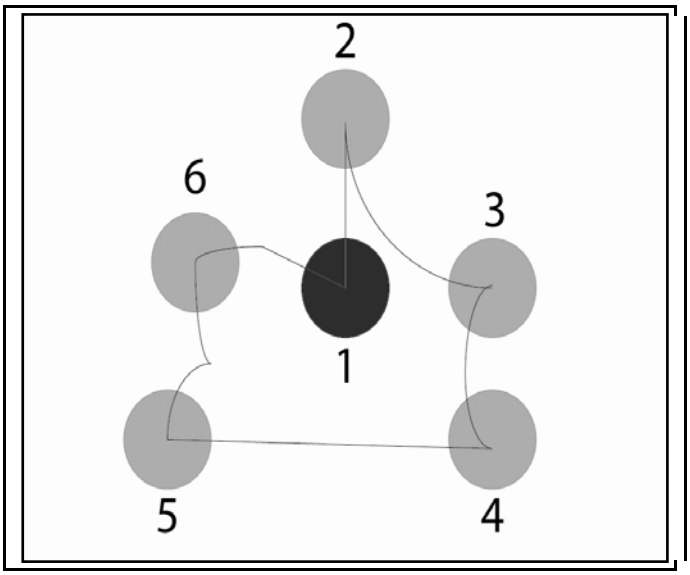



*Figure 4-12 An illustration of Path 6*

### 4.2.4. Procedure

The volunteers were handed an instruction sheet outlining what the experiment entailed and how to use each input device. Once the volunteers completed reading the sheet they filled out an assumption ranking sheet indicating which device they thought would be the easiest to the hardest. They placed the number 1 beside the easiest input device, 2 for the moderate level of difficulty and 3 for the most difficult.

Each experiment was done individually. The order in which the volunteers used the devices was randomly picked by a simple computer application and the order of the paths was also randomly picked by the computer application. When the volunteer was seated in front of the device they were given time to learn how to use the device. Learning was conducted by allowing the volunteer's to freely use the device to navigate the motorized web camera. A stationary red ball was projected onto the centre of the projector screen to provide a point of reference for the volunteers. Once they were comfortable in the operation of the device they alerted the test coordinator to start the experiment. Each path took 40 seconds from the start of the red ball movement to the final destination of the red ball. When one path was completed they were given a brief break averaging 30 seconds. When they wanted to proceed to the next path they alerted the test coordinator who started the

next path. This was repeated until all 6 paths were completed. After the completion of 6 paths they moved onto the next device and the process was repeated until all 3 devices were used for all 6 paths. The total time to learn the device and complete all 6 paths with the 3 input devices was on average 30 minutes.

When the experiment was completed the volunteers filled out a questionnaire. The questionnaire was composed of 6 statements presented in a Likert scale using a +/- 4 scale to determine 'likability' for each input device. These statements were derived and modified from Davis' (1989) publication. The 6 statements were:

1. Learning to operate the input device was easy for me.
2. I found it easy to get the input device to do what I wanted it to do.
3. My interaction with the input device was clear and understandable.
4. I found the input device to be flexible to interact with.
5. It was easy for me to become skilful at using the input device.
6. I found the input device easy to use.

To conclude the experiment the volunteers were asked again to rank the order of devices in terms of difficulty to see if their opinions have changed from their initial impressions. A brief conversation with the volunteers was also conducted to record any other thoughts and opinions about the experiment.

## 4.3. Results

From the pre-experiment survey of the volunteers, which were recorded into table 4-1, there was a wide consensus that the mouse device would be the easiest device to use in this experiment to follow the red ball. When the volunteers were asked for their reasoning, the majority responded that they were quite familiar with the operation of the mouse and icon clicking because they had a significant amount of experience using GUIs with the mouse. The face tracking interface was ranked the hardest amongst the majority because they felt that the instructions made the operation of the device a little intimidating and they did not have quite a sufficient amount of experience using face tracking interfaces prior to this experiment.

| | Easy (1) | Medium (2) | Hard (3) |
|---|---|---|---|
| User0001 | Mouse | Joystick | Gesture |
| User0002 | Mouse | Joystick | Gesture |
| User0003 | Mouse | Gesture | Joystick |
| User0004 | Mouse | Joystick | Gesture |
| User0005 | Joystick | Mouse | Gesture |
| User0006 | Mouse | Joystick | Gesture |
| User0007 | Joystick | Mouse | Gesture |
| User0008 | Joystick | Gesture | Mouse |

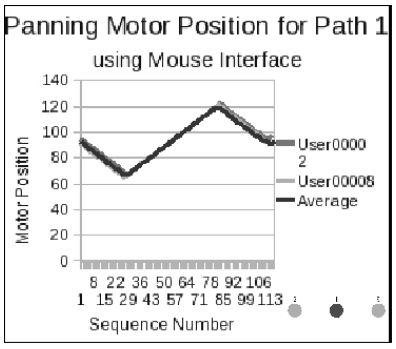
*Table 4-1 Pre-experiment assumptions of level of difficulty to operate the device.*

The data that was acquired by capturing the MESF XML packets was stored on the computer with the motorized web camera attached. The data was converted into a comma separated values (CSV) format to allow easier post-processing. UNIX shell scripts were used to parse the CSV motor tilt commands and pan commands into separate files. The data from the separate files were imported into a spreadsheet where individual spreadsheets were created based on the criteria of path number and tilting or panning of the motors. In those individual spreadsheets an average path was created based on the position of the motor at a sequence number of all the volunteers. A standard deviation for each motor position at a sequence was used to determine if there was a significant amount of deviation between the volunteers, see table 4-2 for a sample set. When the average was plotted on a line graph, the general shape of the graph provided quick insight as to whether the general usage of an input device was easy for the volunteers to use.

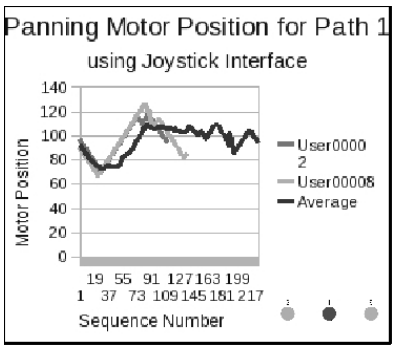
| WIMP | User00001 | User00002 | User00003 | User00004 | User00005 | User00006 | User00007 | User00008 | Average | Standard Deviation |
|---|---|---|---|---|---|---|---|---|---|---|
| Path1 (pan) | 101 | 95 | 89 | 92 | 92 | 90 | 91 | 91 | 92.63 | 3.81 |
| | 100 | 94 | 88 | 91 | 91 | 89 | 90 | 90 | 91.63 | 3.81 |
| | 99 | 93 | 87 | 90 | 90 | 88 | 89 | 89 | 90.63 | 3.81 |

*Table 4-2 A sample of the spreadsheet containing data from mouse interactions on Path 1*

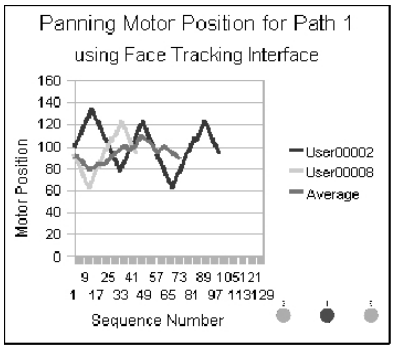
Path 1 was design to be the simplest path for the volunteers to follow. As shown in Graph 4-1a, the shape of the graphed line did not show any signs of over correction from




*Graph 4-1a Graph illustrating the panning motor positions for Path 1 using the Mouse Interface. (Refer to Graph A-1 in the Appendix for an enlarged graph)*




*Graph 4-1b Graph illustrating the panning motor positions for Path 1 using the Joystick Interface. (Refer to Graph A-3 in the Appendix for an enlarged graph)*

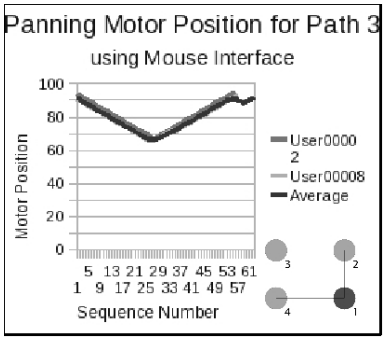



*Graph 4-1c Graph illustrating the panning motor positions for Path 1 using the Face Tracking Interface. (Refer to Graph A-5 in the Appendix for an enlarged graph)*

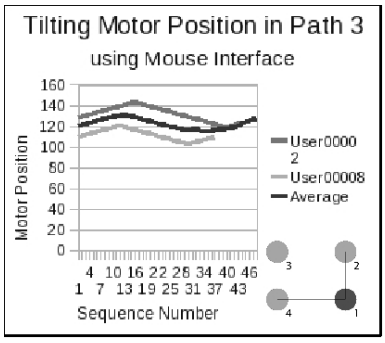
the volunteers. From Graph 4-1a, the pan motor value starts roughly at 90 degrees. As the motor pans to the left the value decreases until the red ball has completed its journey to the left of the screen, 70 degrees on the Y-Axis. When the red ball proceeded to the right of the screen it passed the centre, 90 on the Y-Axis. The red ball continued its journey until it has arrived to the far right side of the projector screen, which is illustrated by the value of the pan motor increasing. As the ball returned to its final destination, the value of the motor position decreased to 90 marking the starting point.

Graph 4-1a clearly shows that on average the volunteers were able to pan the motorized camera to follow the red ball using the mouse interface. Analysis of the joystick interface graph for path 1 panning, shown in Graph 4-1b, indicated that on average the volunteers were experiencing difficulties with following the red ball. Upon closer inspection it became evident that the user with a lot of experience with gaming interfaces seemed to produce results similar to the mouse interface graph. This could indicate that the user was comfortable in using the joystick device to follow the red ball. Whereas, the users with moderate experience with gaming devices experienced more difficultly as they were over compensating the direction of the motorized web camera. The over compensation could also be caused by the sensitivity of the joystick controller. The results of the face tracking interface for panning of path 1, shown in Graph 4-1c, demonstrated that the volunteers were experiencing difficulties operating the interface. On average, volunteers had troubles remembering the tripod metaphor. Another problem identified during the

experiment was the face recognition portion of the application failed to recognize the face on the photo identification card when the card was tilted. The volunteers were not aware unless they checked the face tracking interface application to verify the software was still capturing the face.



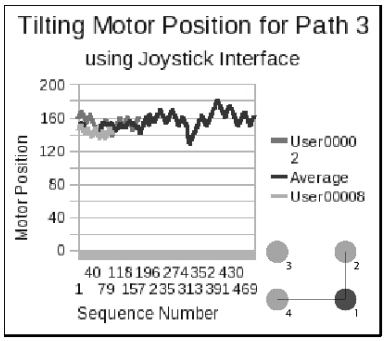

*Graph 4- 2a Graph illustrating the panning motor positions for Path 3 using Mouse Interface. (Refer to Graph A-13 in the Appendix for an enlarged graph)*



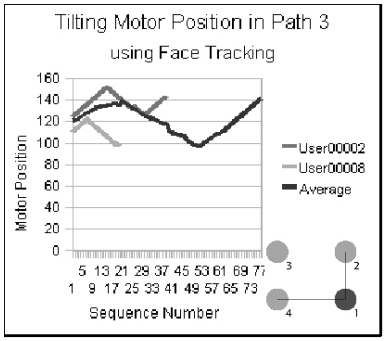

*Graph 4-2b Graph illustrating the tilting motor positions for Path 3 using Mouse Interface. (Refer to Graph A-14 in the Appendix for an enlarged graph)*

The results from the graphs for path 3 indicated that the volunteers were able to use the mouse interface for panning and tilting the camera to follow the path of the red ball, refer to graphs 4-2a and 4-2b. Graph 4-2a illustrated that the volunteers started panning the camera from 90 degrees on the servo. As the red ball moved from the centre of the projector screen to the left the value decreases until the red ball has reached its third point. Unlike the graphs from path 1 the red ball returns back to the middle of the projector screen and does not proceed to the far right. Graph 4-2b illustrates the tilting motor positions of the motorized web camera. The starting position for each volunteer seemed to be different. This may have been the result of the weight of the web-camera forcing the tilt servo out of alignment. However, Graph 4-2b illustrates as the motorized web camera tilts upwards the value increases and as the camera tilts downwards the value decreases. Graph 4-2b also illustrates that the volunteers did not over compensate the movement of the motorized web camera.

The joystick device produced similar results as path 1, with respect to panning. Volunteers that had a significant amount of time using game controllers showed less over correction of operating the motorized web camera.

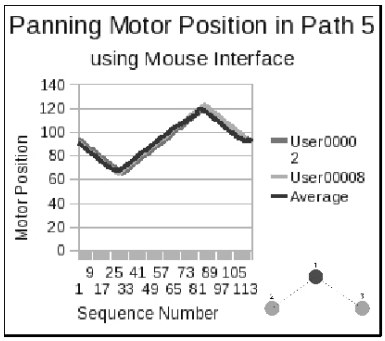



*Graph 4-2c Graph illustrating the tilting motor positions for Path 3 using Joystick Interface. (Refer to Graph A-16 in the Appendix for an enlarged graph)*




*Graph 4-2d Graph illustrating the tilting motor positions for Path 3 using Joystick Interface. (Refer to Graph A-18 in the Appendix for an enlarged graph)*

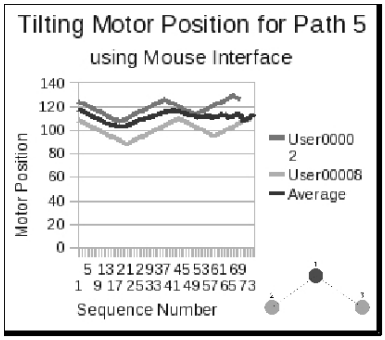
However, when the results of the tilting operations were analyzed all volunteers were demonstrating over correction of the motorized web-camera, shown in Graph 4-2c. The face tracking interface produced the same general panning results from path 1. The volunteers were able to pan the motorized web camera but with some difficulties. The tilting operations created problems for many of the volunteers. Some volunteers lost track of the red ball and were not able to catch up to the red ball to continue on with the path. In Graph 4-2d, User0008 was following the path of the red ball but towards the end User0008's web camera lost track of the photo identification card resulting in the face tracking application failing to send commands to the motorized web camera. A successful result would have been the end point of the camera's tilt position being the same as the camera's starting point tilt position. However, the off-centred weight of the web-camera on the tilt servo could be an interfering factor.



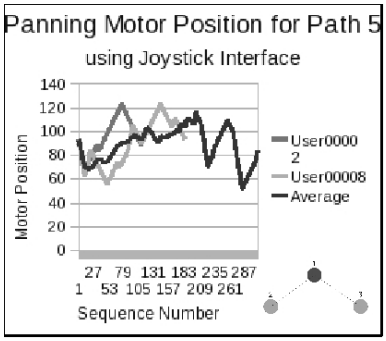

*Graph 4-3a Graph illustrating the panning motor positions for Path 5 using Mouse Interface. (Refer to Graph A-25 in the Appendix for an enlarged graph)*

Path 5 was designed to test the volunteers' skills in moving the motorized web-camera in diagonal directions using all three devices. This test required the use of both the pan and tilt operations, unlike in path 1 and 3 where the operations of the pan and tilt were mutually exclusive. The results were the same as the other paths.

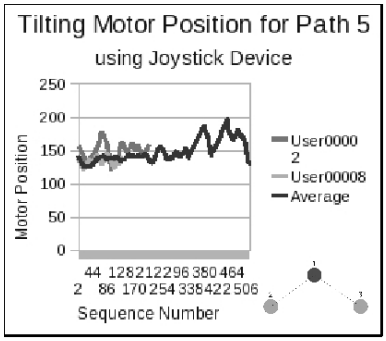
The volunteers exhibited strong mastery of the mouse interface. Results from Graph 4-3a, demonstrated that the volunteers were successful in panning the motorized web-camera from centre to left, left to right and right to centre




*Graph 4-3b Graph illustrating the tilting motor positions for Path 5 using Mouse Interface. (Refer to Graph A-26 in the Appendix for an enlarged graph)*




*Graph 4-3c Graph illustrating the panning motor positions for Path 5 using Joystick Interface. (Refer to Graph A-27 in the Appendix for an enlarged graph)*



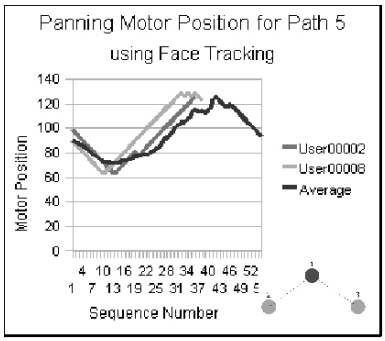

*Graph 4-3d Graph illustrating the tilting motor positions for Path 5 using Joystick Interface. (Refer to Graph A-28 in the Appendix for an enlarged graph)*

again. The Graph 4-3a should resemble Graph 4-1a since the volunteers were panning the motorized in the same directions. The Graph 4-3b illustrates two v-shape curves side-by-side each other since the path essentially proceeded downwards and back to centre twice. Again, the downward force of the web camera on the servo caused the skewed data set for the average curve.

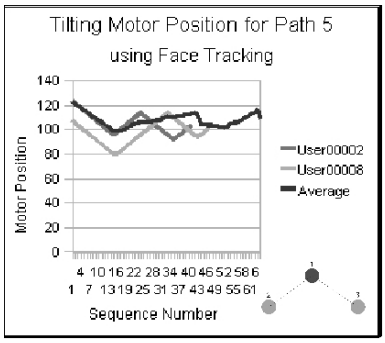
The joystick still remained a troublesome device for many of the volunteers to utilize. The task of following a diagonal path forced many of the volunteers to over correct the direction of the motorized web-camera. This was strongly evident when the graphs 4-3a and 4-3c were compared. The graphs illustrating the data from the joystick showed jittery lines throughout the path whereas the lines were smoother with the mouse interface. This was another indicator that the mouse was easier to use over the joystick.

Despite the pre-assumptions of the volunteers claiming that the face tracking interface would be the most difficult to use, results show that they were mastering the use of the face tracking interface better than the joystick. When the motor panning curve on Graph 4-3e was compared to graph 4-3a, they were slightly similar and did not show the jittery curve displayed in Graph 4-3c. The only problem that was apparent was the face tracking application losing track of the photo id card. The tilting operation still demonstrated problems for the volunteers when Graph 4-3f was compared to Graph 4-3b.

The results for paths 2, 4 and 6 indicated the following trends seen in the other paths. The volunteers were able to show a strong mastery using the mouse interface. The



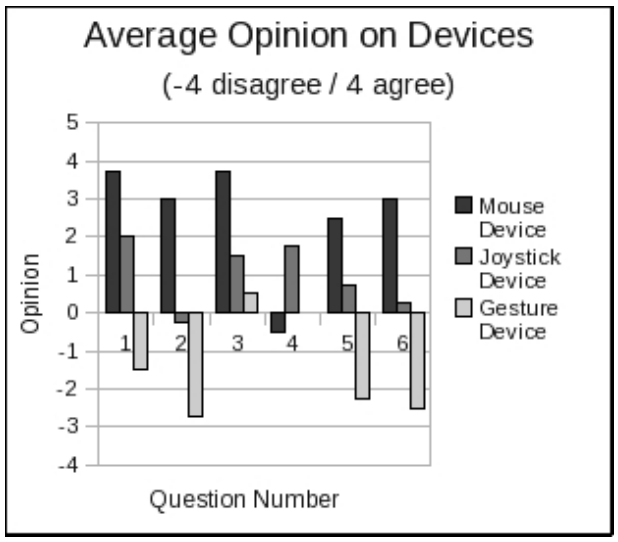

*Graph 4-3e Graph illustrating the panning motor positions for Path 5 using Face Tracking Interface. (Refer to Graph A-29 in the Appendix for an enlarged graph)*




*Graph 4-3f Graph illustrating the tilting motor positions for Path 5 using Face Tracking Interface. (Refer to Graph A-30 in the Appendix for an enlarged graph)*

joystick interface presented a lot of problems with over-correction. The over-correction issue became highly noticeable when pan and tilt directions were combined. The face tracking interface demonstrated that it could be a useful interface if the volunteers had more experience using it.

The results from the survey are shown in Graph 4-4. The graph was constructed by averaging the volunteers' response to each question pertaining to the devices and then plotted on the bar graph. From the graph the volunteers clearly favoured the mouse device over the other 2. This was not quite alarming since all the volunteers have had extensive experiences using a mouse and clicking on icons. The interesting element to note was the negative response of the mouse for question 4. All the volunteers had indicated that if the mouse application included diagonal icons to click on then the mouse device would have been a lot more flexible to use.




*Graph 4-4 – Displaying the results from the survey on each device.*

1. Learning to operate the input device was easy for me.
2. I found it easy to get the input device to do what I wanted it to do.
3. My interaction with the input device was clear and understandable.
4. I found the input device to be flexible to interact with.
5. It was easy for me to become skilful at using the input device.
6. I found the input device easy to use.

The joystick was the second favoured input device. Although the joystick device drew a lot of criticism, as one of the volunteers indicated that the size constraint made the handling of the motorized web camera more difficult because only minute movements were required to move the camera. Moving the joystick in an abrupt manner caused the motors to pan or tilt the camera over the targeted path. This created a highly sensitive movement resulting in over-correction of the motorized web camera. If the motorized camera was situated in an environment, such as a sports field, and tracked a target at a great distance moving quickly then the sensitivity might be beneficial. The sensitivity and over-correction lead to a low score in question 2 of the survey.

The face tracking was the least favoured interface in this experiment. Many volunteers found the tripod metaphor complicated to remember during the experiment and there was a delay between the volunteer's movement and the movement of the camera. The delay confused the volunteers as they weren't certain if the system was working or if they were doing something incorrectly. A couple of the volunteers claimed discomfort in holding the identification card in front of the web-camera for a lengthy period of time. The result of these two factors resulted in low ratings in questions 1 and 2 of the survey. The frustration from the volunteers was apparent in questions 5 and 6 of the survey where the volunteers felt they would not be able to become skilful at the face tracking interface since it was not an easy interface to use.

| | Easy | Medium | Hard |
|---|---|---|---|
| User0001 (Before) | Mouse | Joystick | Gesture |
| User0001 (After) | Mouse | Joystick | Gesture |
| User0002 (Before) | Mouse | Joystick | Gesture |
| User0002 (After) | Mouse | Joystick | Gesture |
| User0003 (Before) | Mouse | Gesture | Joystick |
| User0003 (After) | Mouse | Joystick | Gesture |
| User0004 (Before) | Mouse | Joystick | Gesture |
| User0004 (After) | Mouse | Joystick | Gesture |
| User0005 (Before) | Joystick | Mouse | Gesture |
| User0005 (After) | Mouse | Joystick | Gesture |
| User0006 (Before) | Mouse | Joystick | Gesture |
| User0006 (After) | Mouse | Joystick | Gesture |
| User0007 (Before) | Joystick | Mouse | Gesture |
| User0007 (After) | Mouse | Joystick | Gesture |
| User0008 (Before) | Joystick | Gesture | Mouse |
| User0008 (After) | Mouse | Joystick | Gesture |

*Table 4-3 Before and after results of the volunteers' assumption on the level of difficulty in operating the devices.*

Table 4-3 reflects the volunteers' assumption before and after the experiments on the use of different input devices. While opinions differ before the experiments amongst the volunteers, once the experiment was finished they all agreed that the mouse was the easiest device to use. The joystick was the second easiest device and the gesture interface, or alternatively known as the face tracking interface in this experiment, was very difficult to operate.

# 5. Discussion

The pilot experiment in chapter 4 helped illustrate how the MESF XML system could be used in the evaluation of various user input interfaces. The results from the experiment demonstrated that the WIMP-style (or mouse in this experiment) user interface was unanimously favoured by the volunteers. Observations from the graphs showed that the volunteers had fewer difficulties in maintaining track of the red ball's path, when the path was in a horizontal or vertical line. The volunteers criticized the WIMP-style interface when the path was diagonal since it required additional effort clicking on 2 buttons to achieve the desired movement. The results from the WIMP interface were as expected because WIMP interactions are very common in computer user interfaces. All the

volunteers had a prolonged exposure to WIMP interfaces and the discrete movements made navigating the motorized web-camera easier compared to the non-discrete movements using the joystick and the face tracker.

The results for the joystick interface were unexpected. Joystick devices have been used by computer systems for significant period of time. Joysticks were used on computers to play games shortly after computers were available to the average consumer so they are not a novel interaction device. Even volunteers that had an extensive gaming experience demonstrated problems using the joystick interface to control the motorized web camera. The difficulty of moving the motorized web-camera was found in the implementation of the computer program to read the joystick and transmit the XML MESF packet. When the analog joysticks were moved they sent a XML MESF packet to the motorized web-camera to move 1 degree in the direction requested by the volunteer. However, the implementation of the code did not measure the magnitude of the analog joystick movement. So if the volunteer moved the joystick to the far left or slightly left, the program treated the joystick command as a general left movement. This created confusion because as the magnitude of the joystick's movement decreased the computer program was still sending a 1 degree move to the left. This was causing the volunteers to over correct the position of the motorized web-camera. The results displayed on the graph were not necessarily negative, since it helped demonstrate how plotting data using the XML MESF could be used to interpret volunteer's difficulties in operating a device. From these results we can alter the computer program and compare future results to the results from this experiment. Ideally, the graphs should start looking similar to the graphs from the WIMP-interactions. Unfortunately, due to not being able to collect a proper control data and plotting the graph, there was no other way to determine accuracy of each volunteers' graph.

The face tracking interface produced results that were as expected. The face tracking interface was not a common interface used by the volunteers, while some did have a little experience with gesture-based interfaces the implementation in this experiment was new to them. One interesting observation was despite the unfavourable opinions on the usefulness of the face tracking interface, some volunteers seemed to have performed well on simple paths compared to the joystick paths. This was due to the implementation of the joystick application and the lack of magnitude measurements. In the face tracking

application the motorized web camera was moving in small increments, whereas the joystick was sending a burst of movements to the motorized web camera. If the experiment was ran solely on the qualitative aspect, using just the responses from the survey, then the face tracking interface may have been ruled out as a possible interface to consider in future applications pertaining to tele-operation of a motorized device. However, the quantitative results from the motor positions allowed a comparison between multiple devices.

The pilot experiment conducted in this thesis did not have any control data to illustrate the proper path the red ball would have travelled. While the path could be calculated based on the pixel position of the projected object on the screen and compared to the angle of the servos, that data would have been inaccurate because the video feed presented to the volunteers contained optical distortions caused by the lens of the web-camera and the position of the web-camera relative to the projector screen. For example, although path 1 was a straight horizontal line many of the volunteers were observed sending tilting commands to keep the red ball in the center of the video screen. The use of averaging the motor position helped in creating the line graph to observe the general path taken by the volunteers. Basic paths such as horizontal and vertical paths create linear paths which were easily observed on the graph. The graphs do not show whether the volunteers were trailing the path of the red ball or directly centred on the red ball, therefore, the accuracy of following red ball centred on the volunteer's video window cannot be determined. By having the MESF XML packets broadcasted to the computer network, an application was created to capture these packets and write them to a data file. This allowed post-analysis of the interface interactions of the volunteers. If desired, an application could be created to rebroadcast the contents of the data file back to the network. The benefit of rebroadcasting would allow researchers to see first-hand the results of the volunteer's interactions. For example, if the researcher sat in the same position as the volunteers they could observe what the outcomes of the volunteers' interactions were without having to view video time recorded sessions. The researchers can see if the volunteers did have the red ball centred on the video screen. Modifications would have to be made to the MESF packet structure to accommodate a timing feature. Since the communication system is utilizing XML adding the concept of time is trivial as XML is a simple text structure. A

time-stamp tag could be added to the MESF XML structure to indicate when the packet was sent or when the packet was received.

In terms of multimodality, Philip Kortum (2008) indicated that gesture based devices are not precise compared to the mouse and keyboard, especially in CAD. However, the use of gesture movements helps the operator create 3D contours easier by mimicking natural real-world interactions. For example, working with 3D clay the user can simply pinch an area to pull the 3D clay rather than clicking points on the clay with a mouse and clicking other widgets to manipulate the clay. In the pilot experiment, it was evident that the mouse demonstrated relative ease in moving the motorized web-camera. The joystick and head tracking interface were less precise but they provide quicker movements.

# 6. Conclusion

New techniques and devices are becoming available to researchers and consumers faster than before. The growth of new interfaces can be seen in the gaming community. However, many of these devices were conceptualized in the mid-60s and early 70s. The dominance of the mouse and keyboard is still widely used in the general population of computer users despite its downfall. System developers must think carefully if adding new interfaces to application will create positive experiences or frustrate the user. To evaluate the usefulness of a device a framework needs to be used that allows developers and researchers to quickly evaluate the device and not be burdened with learning new APIs and other frameworks. The purpose of this thesis was to use existing data communication protocols and easily configurable data structures to issue events created by various interfaces. By allowing developers to use their existing programming language of choice and adding network and XML functionality to their applications this would enable them the opportunity to develop and prototype new interactions faster. The developers do not necessarily need to learn a new API set or figure out the mechanics of new frameworks.

The Modified Event Specification Format was created based on work from the OpenInterface project. While other XML and frameworks were analyzed the OpenInterface XML structure was easily understandable and easily configured for the pilot experiment in chapter 4. The MESF XML combined with existing Internet protocol was

used to demonstrate how the proposed system in the thesis could be used by other researchers. The results, in chapter 5, helped demonstrate how the XML packets could be captured and used to evaluate the effectiveness of the devices in a certain tasks.

The use of the MESF XML system was not utilized in a multimodal environment, however, an experiment could be devised to test multimodality of various interfaces easily based on the devices used in the pilot experiment. An experiment could be created where the volunteers must manoeuvre the motorized web-camera to certain points on the projected screen. The volunteers could be tested on the use of mouse/joystick or mouse/face tracker combination or any other devices using other input channels. The evaluation could be based on their mean time to complete the tasks. The data provided from this experiment could give an indicator which interface provides better coarse movement control to the volunteer, since the mouse provides the best fine movement control based on the results from the pilot experiment.

**Future Work**

Brain-computer interface are now becoming affordable options to incorporate into computer applications. Tan Le (2010) demonstrated a device that could be placed on the head of a person that can read the person's brainwaves. The usefulness of the brainwave interface could be compared to other interfaces conducting the same task. A baseline dataset could be created using an interface that is normally used in performing the task. The task could be repeated using the brainwave interface device and the results could be compared, similar to the experiment in chapter 4 where the three devices were compared.

Other areas of work that could be investigated based on the MESF messaging system could be the application to real world environments. One in particular is controlling a tele-operated robot using a WI-FI network. Another area to investigate the MESF messaging system is performing more prototype testing using the MESF XML communication system to derive more meaning XML tags and attributes. The benefit would be other researchers can get an understanding of which tags and attributes will be beneficial to their experiments. This may help create a formal XML specification based on the MESF XML structure.

# References

austriamicrosystems AG "AS5011 - Hall-sensor based contactless human interface device", http://www.austriamicrosystems.com/eng/Products/Magnetic-Encoders/EasyPoint-Joystick-Encoder/AS5011, http://www.austriamicrosystems.com/eng/Products/Magnetic-Encoders/EasyPoint-Joystick-Encoder/AS5011 Accessed on June 28, 2010., 2010.

Alan Dix, Janet E. Finlay, G. D. A. R. B. *Human-Computer Interaction 3rd Edition*, Pearson Education Limited, 2004.

Bajracharya, M., Maimone, M. W. and Helmick, D. "Autonomy for Mars Rovers: Past, Present, and Future," *Computer* (41), 2008, pp. 44--50.

Bolt, R. A. "“Put-that-there”: Voice and gesture at the graphics interface," *SIGGRAPH Comput. Graph.* (14), 1980, pp. 262--270.

Bouchet, J., Nigay, L. and Ganille, T. "ICARE software components for rapidly developing multimodal interfaces" 'Proceedings of the 6th international conference on Multimodal interfaces', ACM, New York, NY, USA, 2004, pp. 251--258.

Bradski, G. "The OpenCV Library," *Dr. Dobb's Journal of Software Tools* (), 2000.

Bradski, G. and Kaehler, A. *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly, Cambridge, MA, 2008.

Buxton, B. "There's More to Interaction than Meets the Eye: Some Issues in Manual Input," *User Centered System Design:New Perspectives on Human-Computer Interaction.* (), 1986, pp. 319-337.

Coutaz, J., Nigay, L. and Salber, D. "The MSM Framework: A Design Space for Multi-Sensori-Motor Systems" 'Selected papers from the Third International Conference on Human-Computer Interaction', Springer-Verlag, London, UK, 1993, pp. 231--241.

CWiid "CWiid", http://abstrakraft.org/cwiid/, http://abstrakraft.org/cwiid/, 2010.

van Dam, A. "Post-WIMP user interfaces," *Commun. ACM* (40:2), 1997, pp. 63--67.

Davis, F. D. "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology.," *MIS Quarterly* (13:3), 1989, pp. 318+.

Dragicevic, P. and Fekete, J.-D. "Support for input adaptability in the ICON toolkit"'Proceedings of the 6th international conference on Multimodal interfaces', ACM, New York, NY, USA, 2004, pp. 212--219.

Engelbart, D. C. and English, W. K. "A research center for augmenting human intellect"'AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint

computer conference, part I', ACM, New York, NY, USA, 1968, pp. 395--410.

Espinosa, R. H. "Joystick API Documentation", Online, Accessed on 11.08.2010, 1998.

Fong, T., Thorpe, C. and Baur, C. "Advanced Interfaces for Vehicle Teleoperation: Collaborative Control, Sensor Fusion Displays, and Remote Driving Tools," *Autonomous Robots* (11), 2001, pp. 77--85.

FreeTrack "FreeTrack", http://www.free-track.net/english/, 2008.

Geetanjali Arora, S. K. w. N. *Building Web Services with XML*, Premier Press, Incorporated, 2002.

Gordon McComb, M. P. *Robot Builder's Bonanza*, McGraw-Hill, 2006.

Green, M. and Jacob, R. "SIGGRAPH '90 Workshop report: software architectures and metaphors for non-WIMP user interfaces," *SIGGRAPH Comput. Graph.* (25:3), 1991, pp. 229--235.

Hainsworth, D. W. "Teleoperation User Interfaces for Mining Robotics," *Auton. Robots* (11), 2001, pp. 19--28.

Hanson, S. J., Kraut, R. E. and Farber, J. M. "Interface design and multivariate analysis of UNIX command use," *ACM Trans. Inf. Syst.* (2:1), 1984, pp. 42--57.

Harold, E. R. and Means, W. S. , Laurent, S. S., (eds.) *XML in a Nutshell*, O' Reilly Media, Inc., 2004.

Hasanuzzaman, M., Ampornaramveth, V., Zhang, T., Bhuiyan, M., Shirai, Y. and Ueno, H. "Real-time Vision-based Gesture Recognition for Human Robot Interaction" 'Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on', 2004, pp. 413 -418.

Heistracher, T., Widmann, R., Stadlmann, B. and Zellinger, J. "An event-based communication environment for multi-modal traffic information systems" 'Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE', 2006, pp. 1214 -1219.

Hekmatpour, S. "Experience with evolutionary prototyping in a large software project," *SIGSOFT Softw. Eng. Notes* (12), 1987, pp. 38--41.

Hewitt, R. "Seeing With OpenCV: Follow That Face!," *Servo Magazine* (), 2007.

Inc., N. T. *Haptic Device Abstraction Layer Programmer's Guide*, Novint Technologies Inc., 2008.

James A. Larson, T.V. Raman, D. R. "W3C Multimodal Interaction Framework", W3C, Online (WWW), 2003.

Kaber, D. B., Wright, M. C. and Sheik-Nainar, M. A. "Investigation of multi-modal interface features for adaptive automation of a human-robot system," *International Journal of Man-Machine Studies* (64:6), 2006, pp. 527-540.

Kemna, Armin Hosticka, B. J. *Modular Low-Power, High-Speed Cmos Analog-to-Digital Converter*, Kluwer Academic Publishers, 2003.

Kortum, P. , Kortum, P., (eds.) *HCI beyond the GUI: design for haptic, speech, olfactory and other non-traditional interfaces*, Morgan Kaufmann publications, 30 Corporate Drive, Suite 400, Burlington, MA 01803, 2008.

Kurt, T. E. "Arduino-serial: C code to talk to Arduino", http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino/, 2006.

Lawson, J.-Y. L., Al-Akkad, A.-A., Vanderdonckt, J. and Macq, B. "An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components" 'EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems', ACM, New York, NY, USA, 2009, pp. 245--254.

Le, T. "Tan Le: A headset that reads your brainwaves", http://www.ted.com/talks/tan_le_a_headset_that_reads_your_brainwaves.html, 2010.

Lee, S., Sukhatme, G., Kim, G. and Park, C.-M. "Haptic control of a mobile robot: a user study" 'Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on', 2002, pp.  2867 - 2874 vol.3.

Legout, A., Nonnenmacher, J. and Biersack, E. W. "Bandwidth-allocation policies for unicast and multicast flows," *IEEE/ACM Trans. Netw.* (9), 2001, pp. 464--478.

Martin Gudgin, Marc Hadley. "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", W3C, Online (WWW), 2007.

McGee-Lennon, M. R., Ramsay, A., McGookin, D. and Gray, P. "User evaluation of OIDE: a rapid prototyping platform for multimodal interaction" 'EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems', ACM, New York, NY, USA, 2009, pp. 237--242.

Menezes, N. "Specification of proposed extensions", OpenInterface, Online, 2008.

Myers, B. A. "A brief history of human-computer interaction technology," *interactions* (5:2), 1998, pp. 44--54.

National Research Council Staff. *More Than Screen Deep: Toward Every-Citizen Interfaces to the Nation's Information Infrastructure*, National Academies Press, 1997.

NaturalPoint "TrackIR:: head tracking view control immersion for flight racing and action simulator", http://www.naturalpoint.com/trackir/, 2005.

Nielsen, J. *Usability Engineering*, Morgan Kaufmann, 1993.

Nigay, L. and Coutaz, J. "A design space for multimodal systems: concurrent processing and data fusion" 'Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems', ACM, New York, NY, USA, 1993, pp. 172--178.

Nyce, D. S. *Linear Position Sensors : Theory and Application*, John Wiley & Sons, Incorporated, 2004.

Pennestri, E., Cavacece, M. and Vita, L. "ON THE COMPUTATION OF DEGREES-OF-FREEDOM:A DIDACTIC PERSPECTIVE"'2005 ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference', Long Beach, California, USA, 2005.

Prekopcsák, Z., Halácsy, P. and Gáspár-Papanek, C. "Design and development of an everyday hand gesture interface" 'MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services', ACM, New York, NY, USA, 2008, pp. 479--480.

Reeves, L. M., Lai, J., Larson, J. A., Oviatt, S., Balaji, T. S., Buisine, S., Collings, P., Cohen, P., Kraal, B., Martin, J.-C., McTear, M., Raman, T., Stanney, K. M., Su, H. and Wang, Q. Y. "Guidelines for multimodal user interface design," *Commun. ACM* (47:1), 2004, pp. 57--59.

Results, I. "Human-Computer Interaction: Beyond – Way Beyond – WIMP Interfaces," *ScienceDaily* (), 2009.

Saint-Andre, P. "Extensible Messaging and Presence Protocol (XMPP): Core", Jabber Software Foundation, Online (WWW), 2004.

Shneiderman, B. "Direct Manipulation: A Step Beyond Programming Languages," *Computer* (16:8), 1983, pp. 57 -69.

Sinha, A. K. and Landay, J. A. "Capturing user tests in a multimodal, multidevice informal prototyping tool" 'Proceedings of the 5th international conference on Multimodal interfaces', ACM, New York, NY, USA, 2003, pp. 117--124.

Smith, D. C., Ludolph, F. E. and Irby, C. H. "The desktop metaphor as an approach to user interface design (panel discussion)"'ACM '85: Proceedings of the 1985 ACM annual conference on The range of computing : mid-80's perspective', ACM, New York, NY, USA, Chairman-Johnson, Jeff A., 1985, pp. 548--549.

Spelmezan, D., Jacobs, M., Hilgers, A. and Borchers, J. "Tactile motion instructions for

physical activities" 'CHI '09: Proceedings of the 27th international conference on Human factors in computing systems', ACM, New York, NY, USA, 2009, pp. 2243--2252.

Stallings, W. *Wireless Communication and Networking*, Prentice-Hill Canada, Inc., 2001.

Steriadis, C. E. and Constantinou, P. "Designing human-computer interfaces for quadriplegic people," *ACM Trans. Comput.-Hum. Interact.* (10:2), 2003, pp. 87--118.

Stevens, W. R. *TCP/IP illustrated (vol. 1): the protocols*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

Treese, G. W. and Stewart, L. C. *Designing Systems for Internet Commerce*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

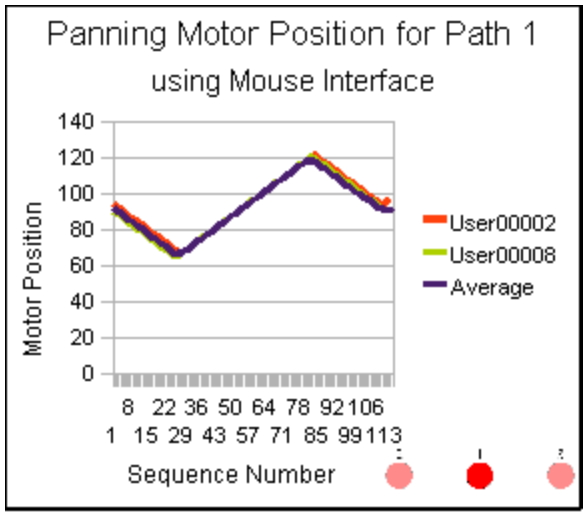
Underkoffler, J. "John Underkoffler points to the future of UI", http://www.ted.com/talks/john_underkoffler_drive_3d_data_with_a_gesture.html, Accessed online http://www.ted.com/talks/john_underkoffler_drive_3d_data_with_a_gesture.html, 2010.

Waldherr, S., Thrun, S. and Romero, R. "A neural-network based approach for recognition of pose and motion gestures on a mobile robot" 'Neural Networks, 1998. Proceedings. Vth Brazilian Symposium on', 1998, pp. 79 -84.

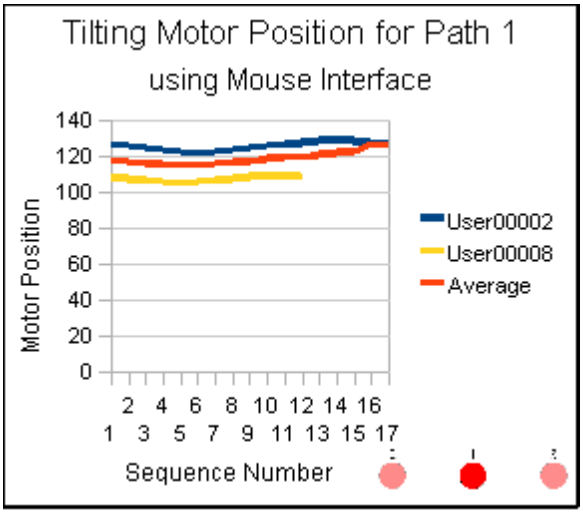
XI-FI "Wii Remote X11 input for Linux", http://nintendo-scene.com/800, http://nintendo-scene.com/800, 2007.

# Appendix A

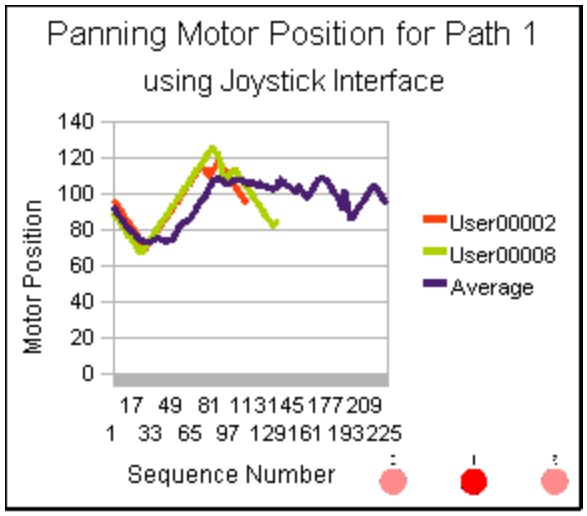
In this section, the results generated from each path utilizing all the input devices are presented. The graphs on the left hand side are the panning motor positions and the graphs on the right hand side are the tilting motor positions.



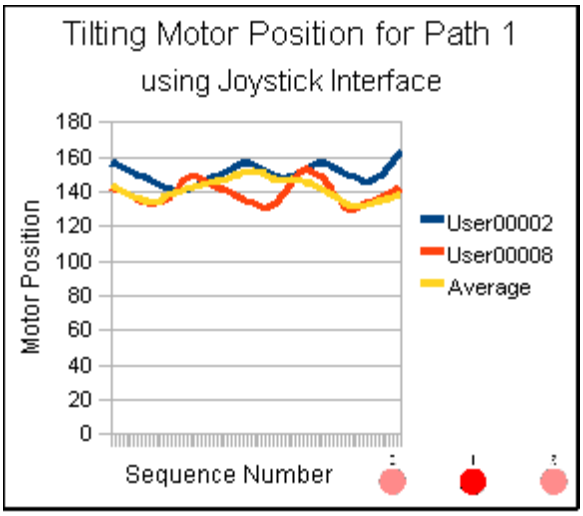

*Graph A-1 Graph illustrating the panning motor positions for Path 1 using the Mouse Interface.* ***(This is Graph 1-a from section 4.2.5.)***



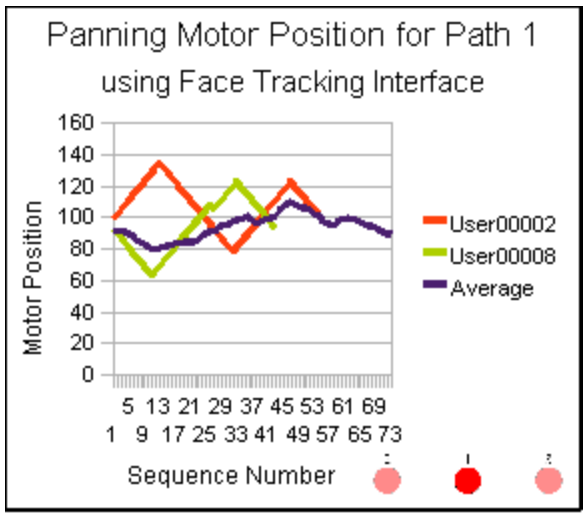

*Graph A-2 Graph illustrating the tilting motor positions for Path 1 using the Mouse Interface.*



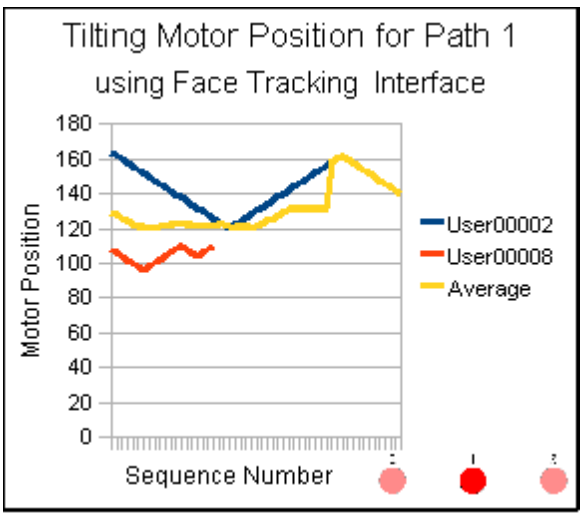

*Graph A-3 Graph illustrating the panning motor positions for Path 1 using the Joystick Interface.* ***(This is Graph 1-b from section 4.2.5.)***



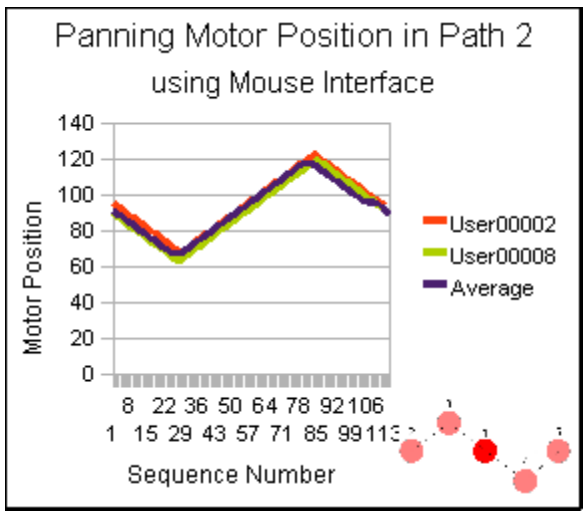

*Graph A-4 Graph illustrating the tilting motor positions for Path 1 using the Joystick Interface.*



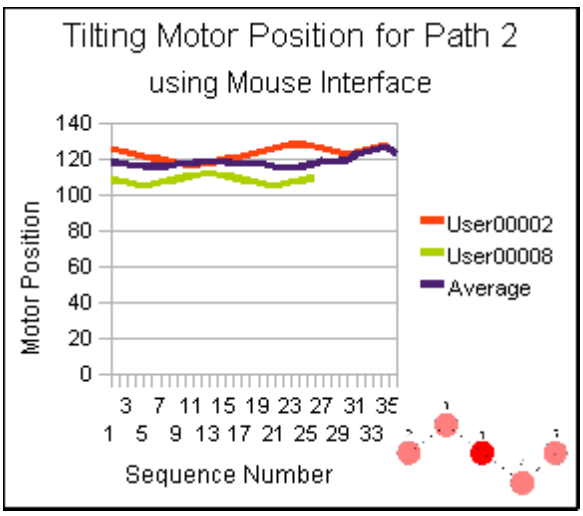

*Graph A-5 Graph illustrating the panning motor positions for Path 1 using the Face Tracking Interface.* ***(This is Graph 1-c from section 4.2.5.)***



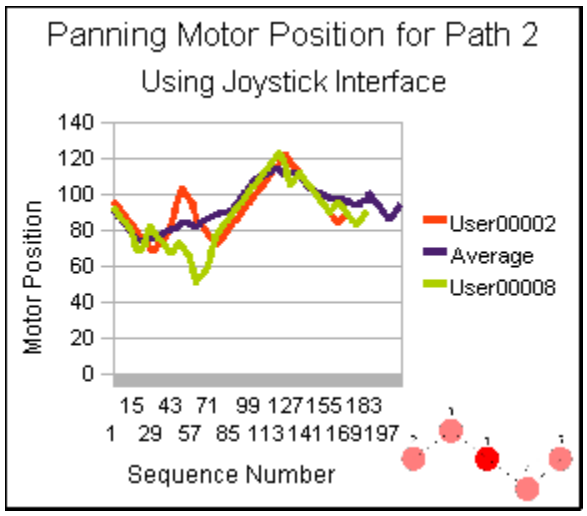

*Graph A-6 Graph illustrating the panning motor positions for Path 1 using the Face Tracking Interface.*



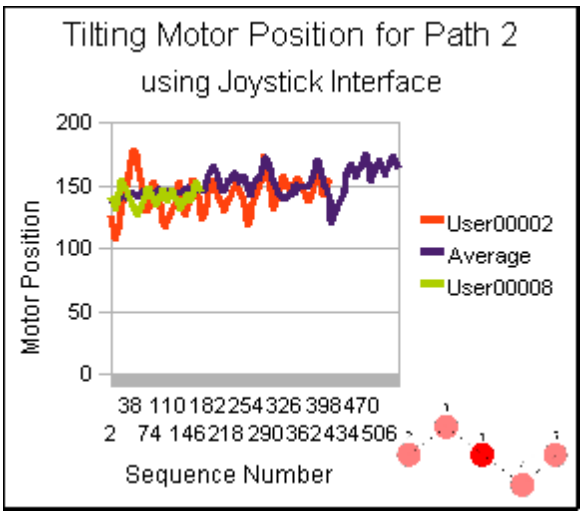

*Graph A-7 Graph illustrating the panning motor positions for Path 2 using the Mouse Interface.*



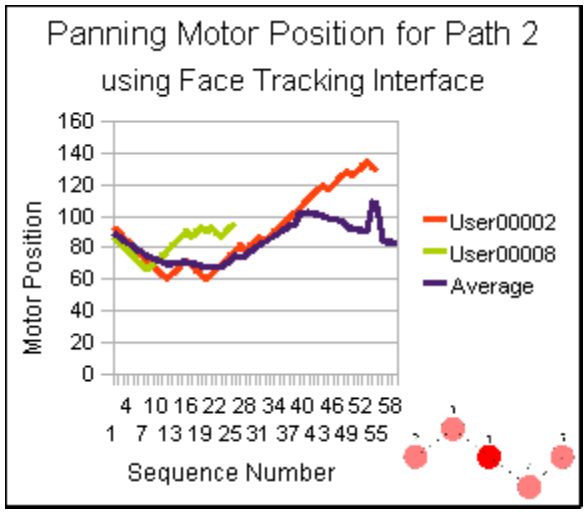

*Graph A-8 Graph illustrating the tilting motor positions for Path 2 using the Mouse Interface.*



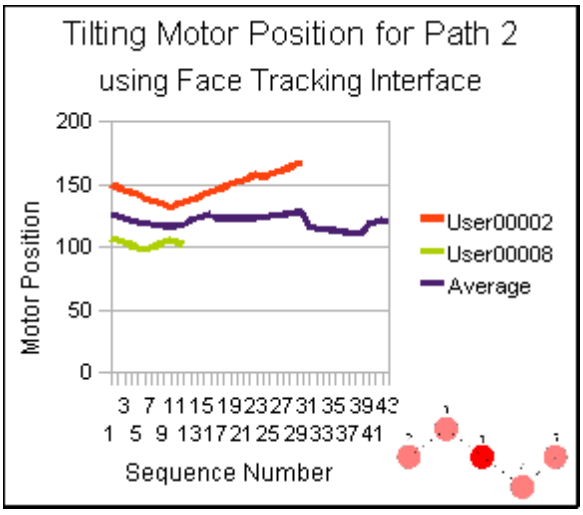

*Graph A-9 Graph illustrating the panning motor positions for Path 2 using the Joystick Interface.*



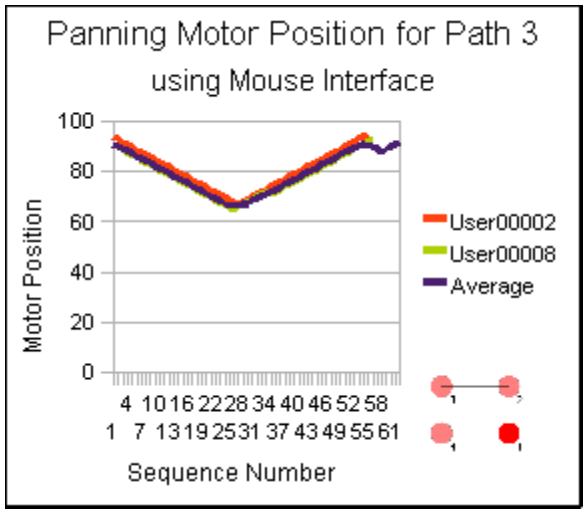

*Graph A-10 Graph illustrating the tilting motor positions for Path 2 using the Joystick Interface.*



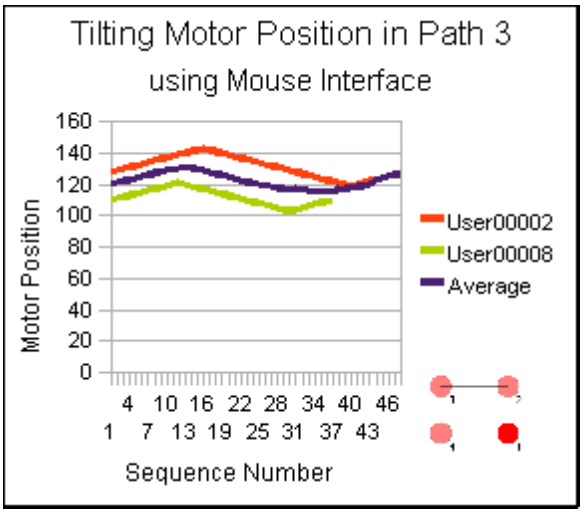

*Graph A-11 Graph illustrating the panning motor positions for Path 2 using the Face Tracking Interface.*



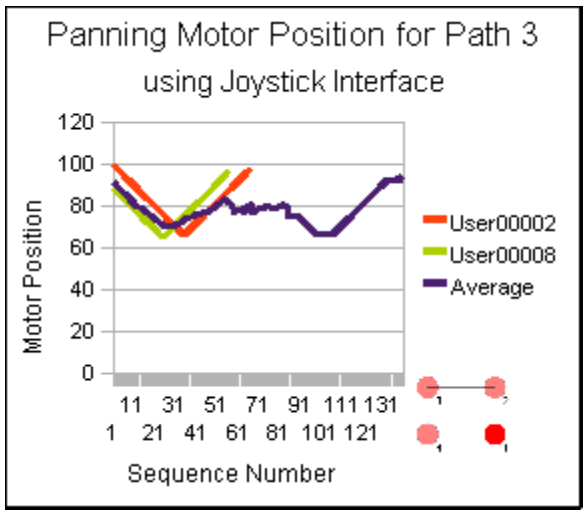

*Graph A-12 Graph illustrating the tilting motor positions for Path 2 using the Face Tracking Interface.*



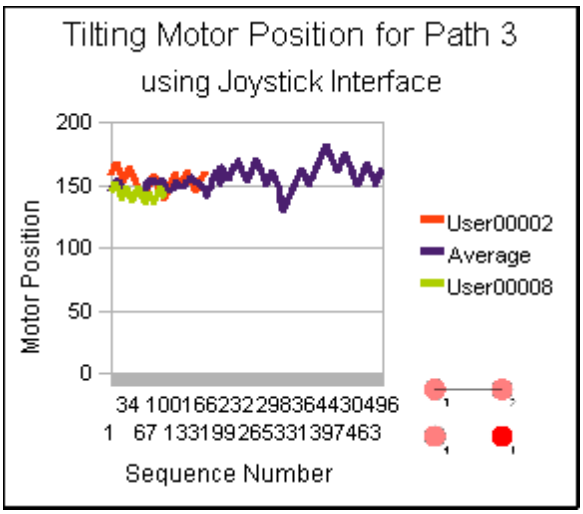

*Graph A-13 Graph illustrating the panning motor positions for Path 3 using the Mouse Interface.* ***(This is Graph 2-a from section 4.2.5.)***



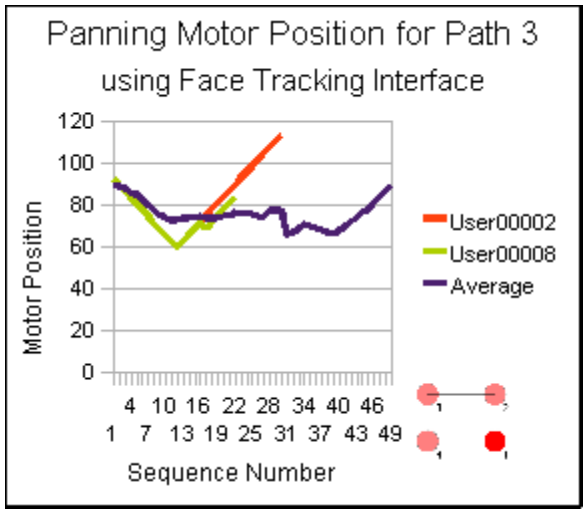

*Graph A-14 Graph illustrating the tilting motor positions for Path 3 using the Mouse Interface.* ***(This is Graph 2-b from section 4.2.5.)***



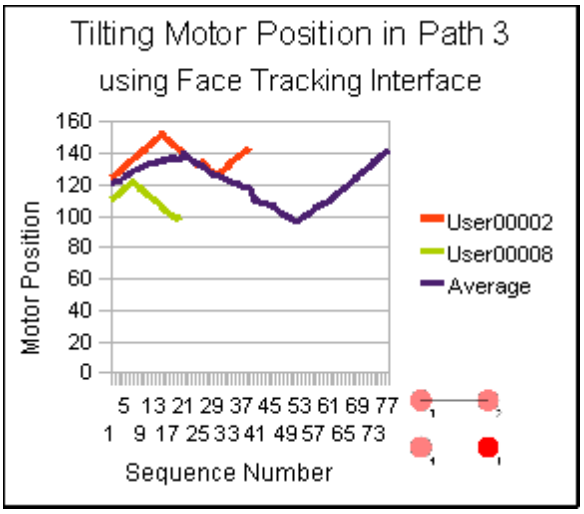

*Graph A-15 Graph illustrating the panning motor positions for Path 3 using the Joystick Interface.*



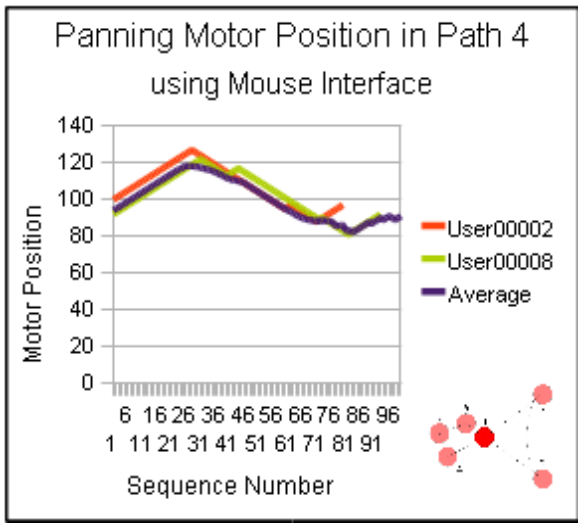

*Graph A-16 Graph illustrating the tilting motor positions for Path 3 using the Joystick Interface.* ***(This is Graph 2-c from section 4.2.5.)***



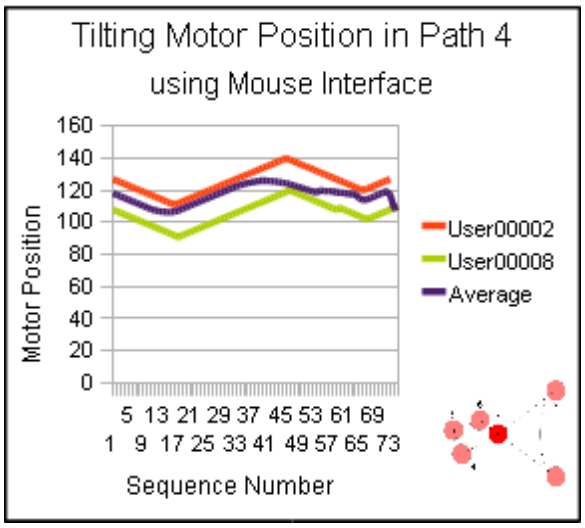

*Graph A-17 Graph illustrating the panning motor positions for Path 3 using the Face Tracking Interface.*



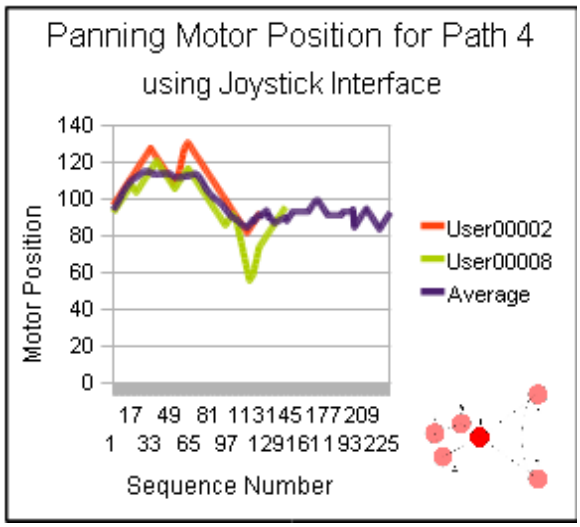

*Graph A-18 Graph illustrating the tilting motor positions for Path 3 using the Face Tracking Interface.* ***(This is Graph 2-d from section 4.2.5.)***



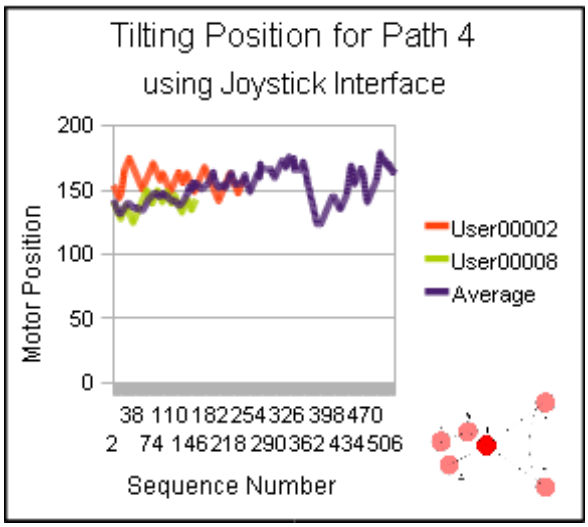

*Graph A-19 Graph illustrating the panning motor positions for Path 4 using the Mouse Interface.*



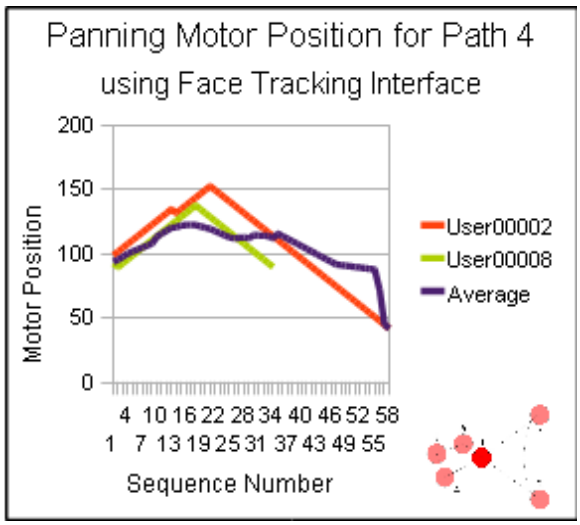

*Graph A-20 Graph illustrating the tilting motor positions for Path 4 using the Mouse Interface.*



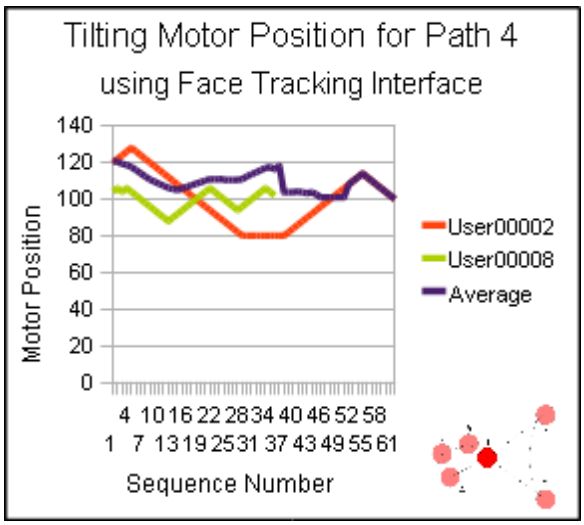

*Graph A-21 Graph illustrating the panning motor positions for Path 4 using the Joystick Interface.*



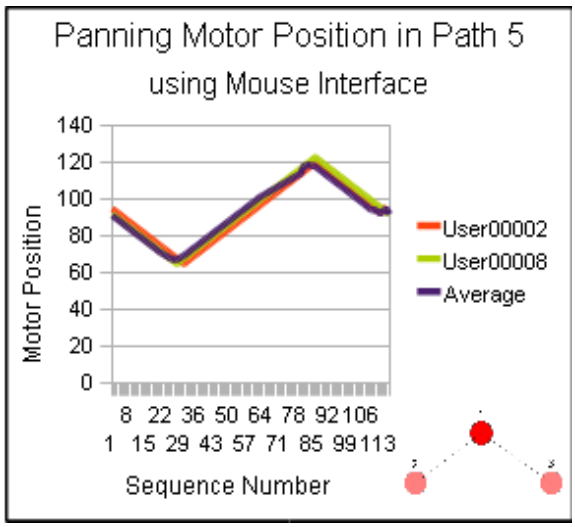

*Graph A-22 Graph illustrating the tilting motor positions for Path 4 using the Joystick Interface.*



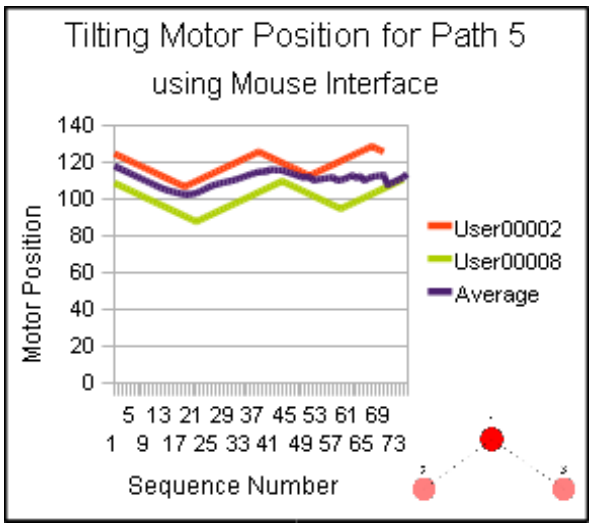

*Graph A-23 Graph illustrating the panning motor positions for Path 4 using the Face Tracking Interface.*



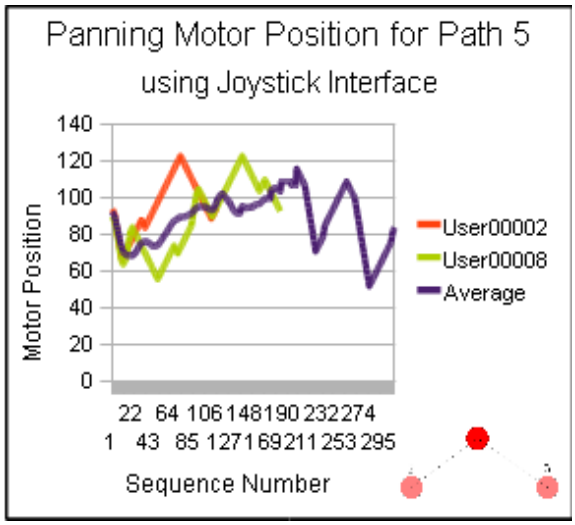

*Graph A-24 Graph illustrating the tilting motor positions for Path 4 using the Face Tracking Interface.*



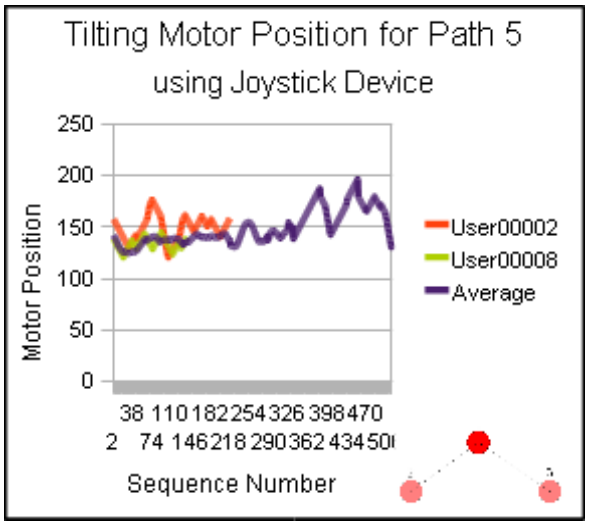

*Graph A-25 Graph illustrating the panning motor positions for Path 5 using the Mouse Interface.* ***(This is Graph 3-a from section 4.2.5.)***



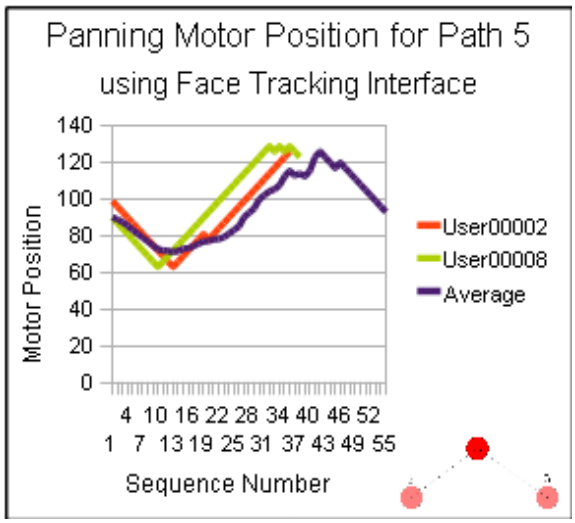

*Graph A-26 Graph illustrating the tilting motor positions for Path 5 using the Mouse Interface.* ***(This is Graph 3-b from section 4.2.5.)***



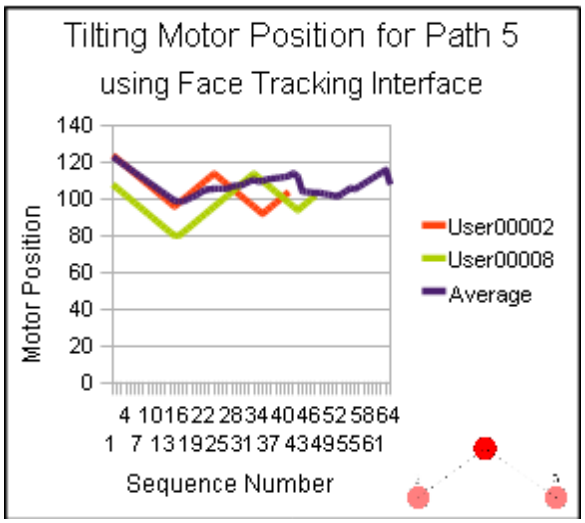

*Graph A-27 Graph illustrating the panning motor positions for Path 5 using the Joystick Interface.* ***(This is Graph 3-c from section 4.2.5.)***



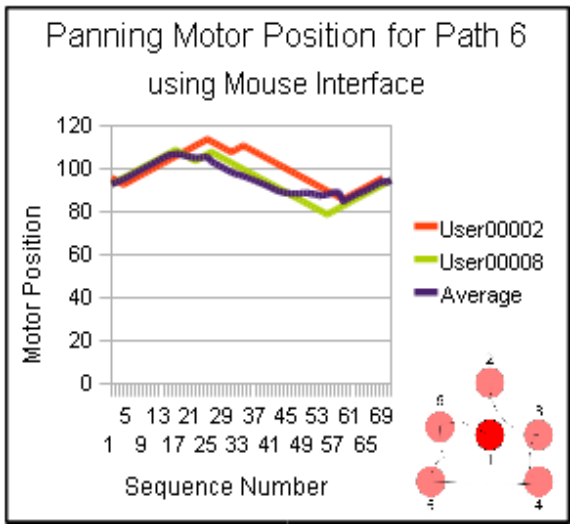

*Graph A-28 Graph illustrating the tilting motor positions for Path 5 using the Joystick Interface.* ***(This is Graph 3-d from section 4.2.5.)***



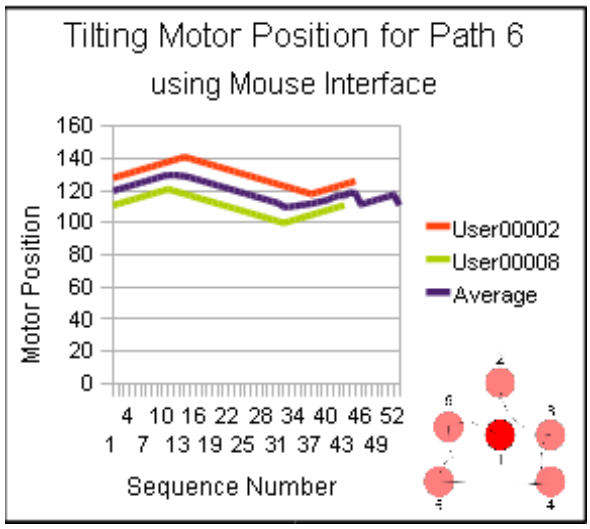

*Graph A-29 Graph illustrating the panning motor positions for Path 5 using the Face Tracking Interface.* ***(This is Graph 3-e from section 4.2.5.)***



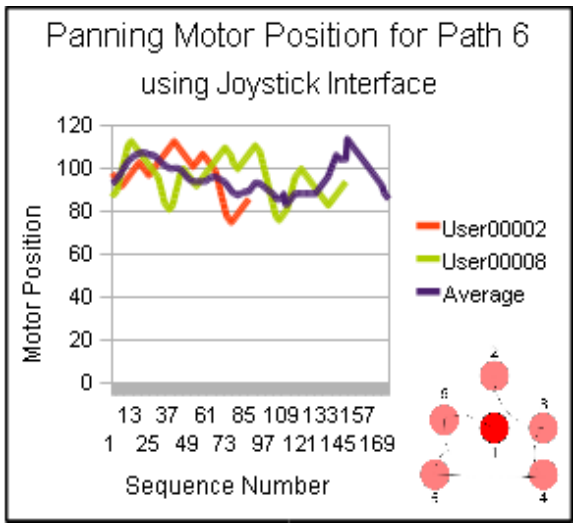

*Graph A-30 Graph illustrating the panning motor positions for Path 5 using the Mouse Interface.* ***(This is Graph 3-f from section 4.2.5.)***



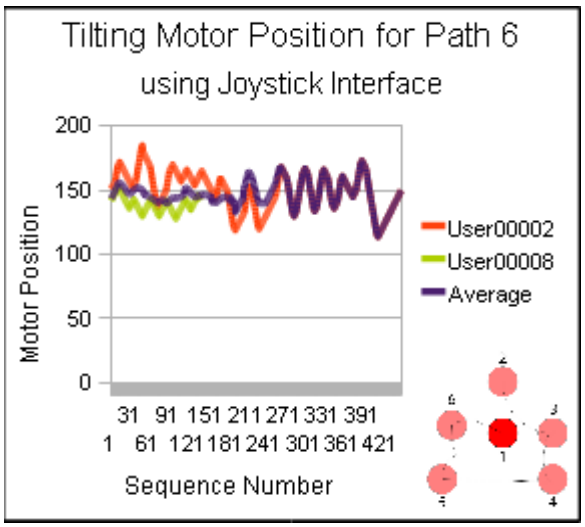

*Graph A-31 Graph illustrating the panning motor positions for Path 6 using the Mouse Interface.*



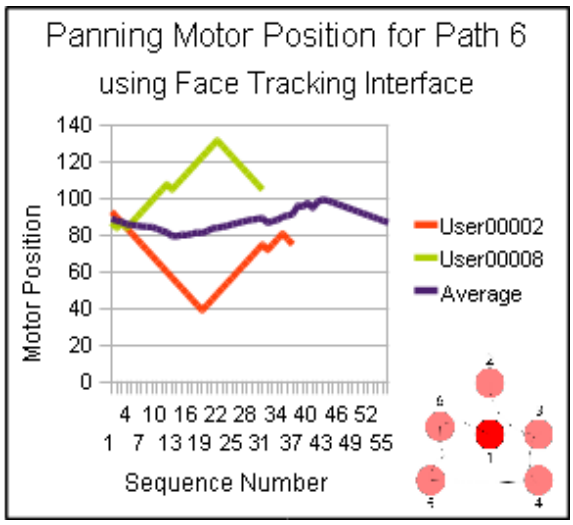

*Graph A-32 Graph illustrating the tilting motor positions for Path 6 using the Mouse Interface.*



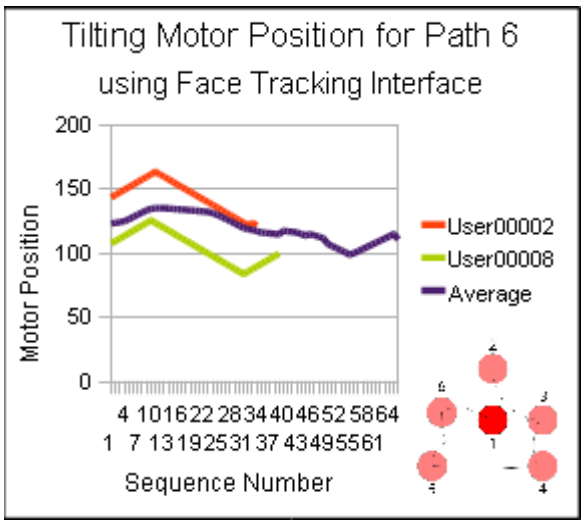

*Graph A-33 Graph illustrating the panning motor positions for Path 6 using the Joystick Interface.*




*Graph A-34 Graph illustrating the tilting motor positions for Path 6 using the Joystick Interface.*




*Graph A-35 Graph illustrating the panning motor positions for Path 6 using the Face Tracking Interface.*




*Graph A-36 Graph illustrating the tilting motor positions for Path 6 using the Face Tracking Interface.*