ANIMATRONIC PUPPETRY

Lee Hearn

Matthew Ramage

B. Eng. Mechatronic Engineering Project Report

School of Mechanical Engineering

Curtin University of Technology

2005

31 October, 2005

The Head
School of Mechanical
Engineering Curtin University of
Technology Kent Street
BENTLEY WA 6102

Dear Sir

We submit this report entitled “Animatronic Puppetry”, based on Project 491/492, undertaken by us as part-requirement for the degree of B.Eng. in Mechatronic Engineering.

Yours faithfully

Matthew Ramage

Lee Hearn

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

In modern animation, a tool that is frequently made use of is that of animatronics. This field of special effects is concerned with making all number of otherwise inanimate puppets move. This involves all number of movement methods, including real time movement controlled by puppeteers, stepping repeatable movements for stop frame animation, as well as autonomous movement, where a puppets eyes may want to watch a specific object, while the remainder of the puppet is under operation via an alternate method.

This type of animation is still in its early stages, with the technology to perform these tasks only becoming available in the last twenty years. As a result, this provides an area in which major advancements can still be made, and many avenues of research are still available. With animatronics being embraced by a range of people, ranging from studio animators to students to home hobbyists, there is a large demand for robust and fully featured control systems for these puppets.

This interest has led to the creation of education courses in the field of animatronics, and there creation. At Curtin University, students enrolled in Art and Design units are able to elect an animatronic design elective, to create a puppet that is capable of being automated. In this course, the puppets built are automated using hobby servo control motors. This is because of the low cost, robust abilities, variety and availability of hobby servo motors that can be bought from1 most local hobby shops. To take full advantage of the puppets, the creation of a complete intelligent, interactive control system was offered as a final year project for students in the field of Mechatronic Engineering. On of the puppets can be seen below in Figure 1.1. Because of this collaboration with arts students studying Animatronic Puppetry 292, and engineering students working in Mechatronic Project 492, the name ‘x92 Puppet Controller’ was adapted.

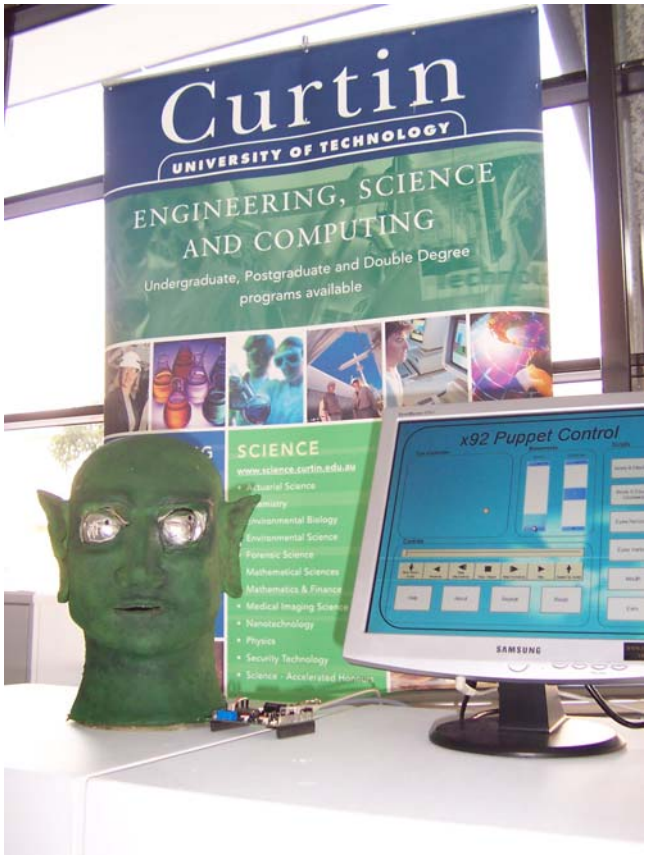


Figure 1.1 - Green Goblin Puppet Featured In the Smarter Than Smoking Animation Expo (2 – 17 July 2005), shown with a touch screen interface. (Puppet built by Magnus)

This document describes the overall control system designed to operate these particular puppets, however, since the actuators for the puppets are high versatile, there are extended applications for the controller that can be considered. The development of the controller for these puppets required detailed implementation in a large number of areas, with each bringing a new challenge and added degrees of complexity. Each aspect of the controller provides a standardized interface to its surrounding elements, and in doing so, groups each section into separable testable elements. This benefits development through the constraint of errors and eases diagnosis by simply analysing element interfaces. In addition to this, the final system that is developed can be upgraded piecewise, saving complete redevelopment, and providing the ability to improve specific elements of the controller without modification to the remaining sections. Finally, should any element of the controller fail, or not meet its requirements, it can be

replaced, without having to modify the remainder of the system. A block diagram of the controller is shown in Figure 1.2.

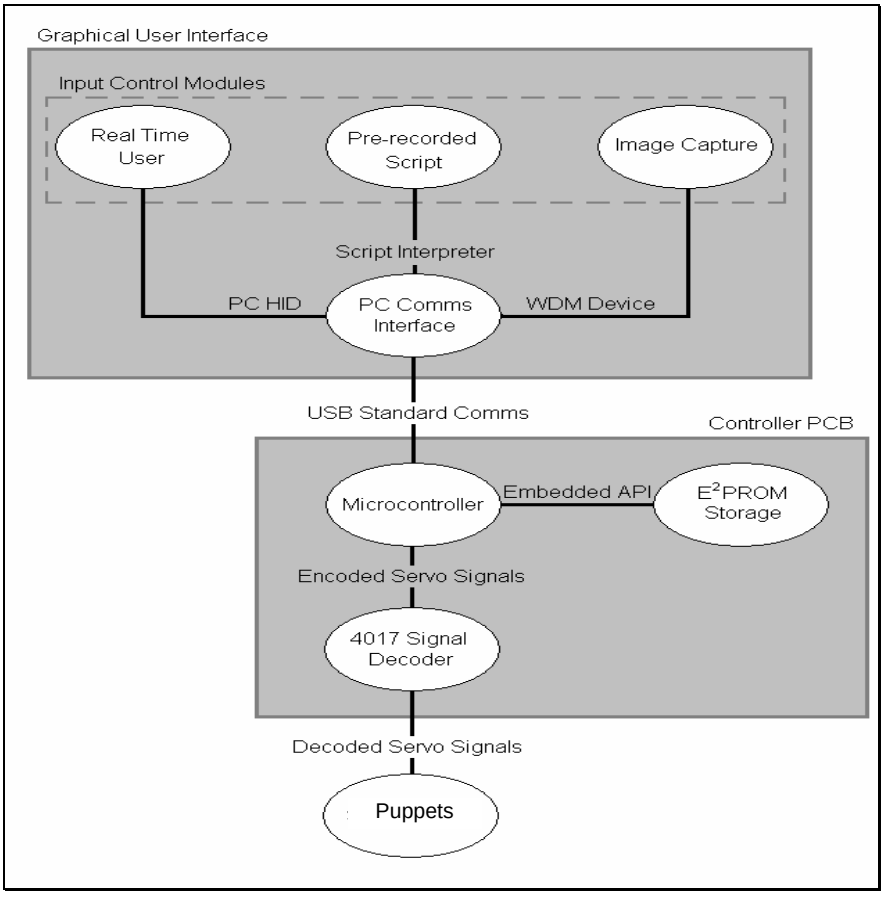


Figure 1.2 – Controller Functional Block Diagram

To make the controller as accurate and expandable, the use of a microcontroller as the main control element was a simple decision, however, by selecting this form of implementation the final controller will require a number of supporting systems. To obtain the maximum benefit from this system, robust firmware was required that was capable of operating a heavily interrupt based environment, with a large number of time critical requests. To achieve maximum results without adding to greater degree of complexity to the system, the C programming language was used to design and implement this firmware, since it is a low level language, yet able to implement communications, signal encoding and onboard static data storage in a far simpler fashion than would otherwise have been possible. The firmware methods for signal encoding are found in chapter 2, while information regarding the firmware's USB communications and static data storage used for configuration and identification can be

found in chapter 4. Because USB was chosen as the communications standard, the controller had to be made fully USB compliant, allowing excellent ease of use once developed, but requiring a very in depth knowledge of the USB standard for successful implementation. With suitable firmware on the microcontroller, a custom circuit board was designed to run not only the microcontroller and USB communications components, but also the subsystems in chapter 3 including the required overtone crystal circuitry, and the 4017 decade counters for the servo signal decoding.

Once the controller is designed, an interface is required for users to be able to interact with it. Again great care was taken to select the most suitable programming language given the constraints, and so Microsoft's Visual Basic 6 was used to create a graphical user interface. The first task of the GUI is the implementation of the communications, again requiring USB compliance, this time achieved using the Application Programmer Interface (API) calls to facilitate enumeration (retrieving all necessary device information autonomously). The specific API calls used, along with the enumeration method are detailed in chapter 4.

With the controller and its communications interface complete, three independent methods of control were all implemented for the controller, to provide the user with several ways of manipulating there puppet. These methods are:

- Direct – Provides real time manipulation of servos by a user
- Scripted – Runs series of repeatable movements from preconstructed scripts.
- Intelligent Vision Control – Where an incoming image is analysed to provide an autonomous input for the controller

All of these control methods are incorporated into a single GUI, allowing every input to only move servo's within user configured limits. Direct control is far simpler than the other methods, requiring no configuration or instruction to use. Scripted control allows users to create scripts using a click and place timeline interface to specify positions in time for each of the servos. These scripts can then be saved, loaded, and played

concurrently, with users having the option of organising the order and number of times each script is played. Intelligent vision control is the most advanced control method implemented. Using a video capture device such as a web cam, the software can locate and track an object of a specific colour. The location of the detected object is used to control two servo motors, enabling the animatronic device to interact with its surrounding environment. These three control methods, direct, scripted and intelligent are covered in detail in chapters 6, 7 and 8 respectively.

The ability of the system to control large numbers of puppets is beneficial since this allows easy synchronisation, and provides a single effective control interface. This scalability is in part achieved through the modular approach, with the human interface element of the system required to support multiple puppet controls. This allows the human interface to take input for large numbers of puppets and in turn relay this input to a number of control elements, with each individual control element not having to know if any others are even connected.

The completed controller provides people who are not technically minded a simple interface with servo motors for any application, although the intended user interface is for use primarily with puppet armatures. The standard communications interface means that users will have no specialized hardware other than the controller's box, and access to a suitable PC. The availability of this type of control system opens the door for many people that would otherwise choose not to become involved in this field to be able to design and experiment in ways that were not possible before. Examples of these are robotics enthusiasts, artists wishing to become involved in the field of animatronics and students, since it is fast to learn, and versatile enough to support a large variety of applications.

This document outlines the most significant aspects of the creation of this controller. Firstly, the major design issues, and physical aspects that had to be catered for are discussed, in particular those facets that would effect the method of implementation chosen. With this information, the general physical architecture of the system is

outlined, as well as the supporting subsystems and proposed system overview, highlighting aspects from position input to signal generation. This shows the split in the system between the software based control methods, and the physical embedded system that would be required to generate the control signals. Noting that this split should be bridged as seamlessly as possible, the next major aspect given is the communications system used between these two elements. With the communications known, the methods used for controlling the hardware were created, including a basic graphical user interface. This user interface contained the features required to control the hardware itself, but not to obtain any input, and so from here each of the three methods of control developed are given. The first covered was the simplest direct control method, used solely for real time adjustment of positions, followed by scripted control, capable of adjusting the same inputs in an accurate repeatable fashion according to a fixed script. Finally, the intelligent vision control method is given, capable of adjusting values for the position autonomously based on the analysis of an incoming video stream.

2 FEATURES AND IMPROVEMENTS

The features of the new controller must firstly be backwards compatible with the previous control method, and provide this same functionality to the end user as easily as possible, and so in order to attain the best result, the merits of the existing controller were taken note of, so that both good and bad features can be adopted and avoided respectively. The servo controller being replaced is an analogue electronic control system consisting of a dual 555 timer configuration for control of each servo. The system uses a single 555 timer connected as a multivibrator to trigger additional 555 timers that deliver one output pulse per trigger at a width determined by a charging resistance, set using a potentiometer. More information on this setup, including schematics and simulated outputs can be found in Appendix A – Legacy Controller.

This system provides direct analogue control, with input from a local operator but provides no capability for repeatable outputs or autonomous control systems. The adaptation of this particular controller to fulfil the project goals would require an interface capable of generating a variable resistance or variable charging potential. The implementation of an accurate variable resistance would require the purchase of expensive digitally controlled potentiometers, while the generation of a variable charging potential would be possible via a digital to analogue converter, or DAC, but to control a number of servo motors in this manner would require excessive numbers of digital input/output pins. The additional downside to generating control signals for DAC input's to the existing controller is the need to provide a suitable digital input, which again would require more components, and an added degree of complexity to the system.

The puppets that must be controlled have been pre-constructed with actuated armatures already in place. The actuators placed in these armatures are hobby servo's, with inputs from the legacy control board.

2.1 WHY OPEN LOOP CONTROL WAS ACCEPTABLE

As a result of examining the previous controller, the lack of feedback is noted as an area that may require attention. The previous controller, while having a secondary closed loop system built into the servos, provides ultimately an open loop control system, as shown in Figure 2.1. The option of adding accurate feedback of the servo position to the initial controller was considered via two potential implementations, labelled F_2 in Figure 2.1:

- absolute position feedback via a positioning potentiometer,
- and an incremental positioning system via a rotary encoder/decoder system.

It was then considered if the implementation of the secondary feedback was necessary given the existence and accuracy of the already implemented system within the servo motor.

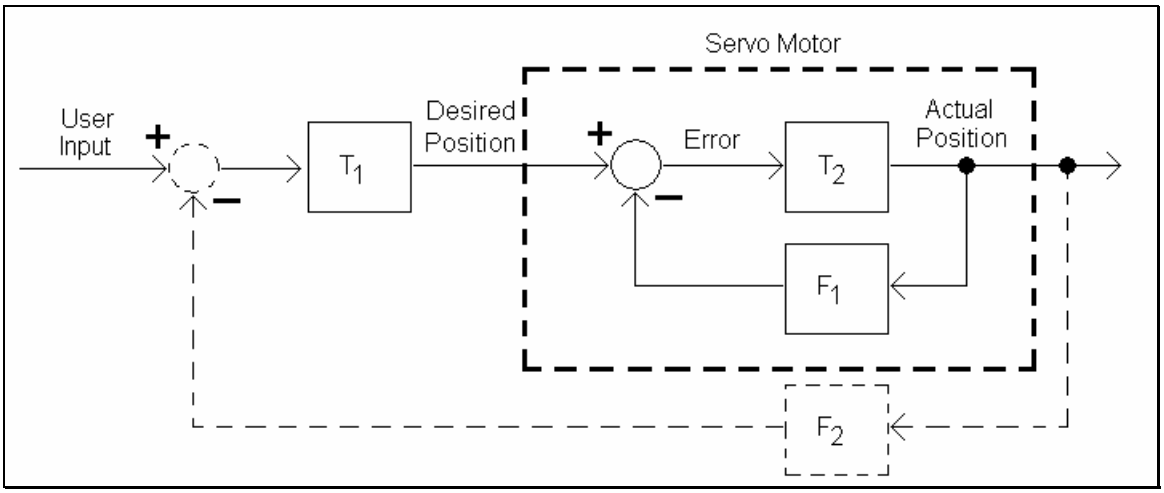


Figure 2.1 – Control System Implementation

The absolute positioning system using the potentiometer required the placement of a position sensing potentiometer on the shaft of the servo motor, and the respective resistance of the potentiometer constantly updated via an analogue to digital converter, or ADC. These potentiometers would have to be calibrated and maintained to ensure that accurate position feedback was being obtained, as well as the necessity of possible recalibration should the characteristics of the potentiometer change. The need for the controller to remain as simple and cost effective as possible for an end user, coupled

with the prohibitive cost of accurate positioning potentiometers, provided too large an obstacle to consider this method of closed loop control.

The alternate method of control was inserting an encoder/decoder system onto the shaft of the servo so that at any single time an absolute position could be read from the decoder IC. This position read using this method, while being absolute, is obtained using incremental readings from the rotary encoded shaft. This meant that at any given time, an absolute shaft position could not be read directly. Because of this the accuracy of the absolute position read is reliant on the encoder knowing the initial position of the shaft, as well as assuming the servo was accurate to set the initial position. This, together with the necessity for a clean and stable environment for accurate operation, as well as the cost of implementation, provided negligible benefit, and so was not implemented.

After eliminating the possibilities for implementing a closed loop system, the inbuilt position control system in the servo was considered for its standalone viability. This position control system operated on the same principals as the position control potentiometer. Within the servo, a small potentiometer was attached to the output shaft of the servo. This provides an internal servo control board an accurate measurement of the servo's position, allowing it to auto-correct its position, moving to that prescribed by the input. This method of feedback provides an absolute position control for the servo, with all error compensated for because the reference is relative to the housing of the servo. This provides an accurate output provided the servo is operated within its load and electrical boundaries.

Ultimately, because a feedback control system with accurate calibration and no additional electronics is implemented effectively in the servo motors, the addition of the control feedback F_2 was not considered a significant benefit to the project or its objectives. The use of this existing control system as a standalone system provided sufficient functionality for use, and allowed extra time to be dedicated to other elements of the controller that would be more beneficial.

2.2 SERVOS AND CONTROL SIGNALS

With the position control system within the servo functioning well enough to avoid another feedback, the servo control signals are the sole method of controlling the servo positions in the puppet armatures. By analysing the previous controller to deduce the control method used, the control signal was determined that is able to provide a suitable replacement system. The servo control circuits are sent a pulse width modulated signal, with a 50Hz frequency, and a pulse width of between 1000µs and 2000µs. A pulse width of 1500µs places the servo in neutral position, defined as the position at which equal travel is possible in both clockwise and counter clockwise directions. This leads to the 1000µs and 2000µs pulses making it turn to its clockwise and counter clockwise extremities respectively, as shown in Figure 2.2 [4].

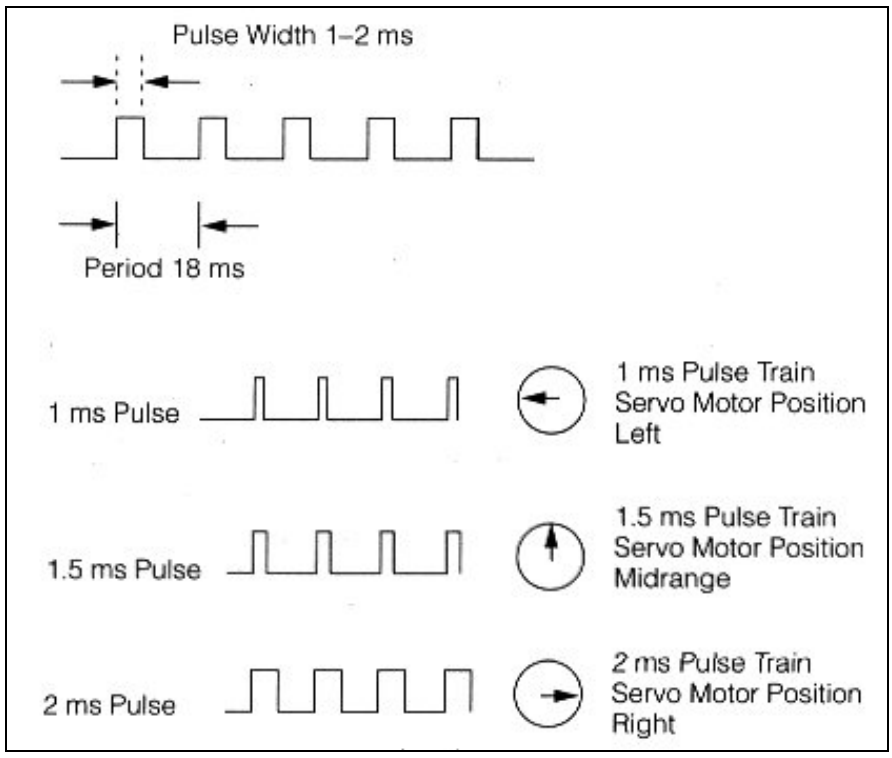


Figure 2.2 – Servo Input and effect on 180º motor

This is known as positive pulse width modulation control, and has been adopted by the hobby industry as an assumed standard, though no formal specification exists. With all major hobby servo motor and controller manufacturers using this as their control standard, the need to remain backwards compatible is essential, and so this standard is

maintained. It is also noted that this supports the control method seen in use by the legacy control system, discussed in Appendix A – Legacy Controller.

It is using this type of servo control, and maintaining the functionality of the legacy controller that the internal architecture of the new controller was designed. This determined the major contributions a new control system would make available, as well as allowing elegant solutions to what was previously impossible problems, such as repeatable movements.

Using these control methods for the servo motors, and allowing the servo motors to operate using their own closed loop control system, the method of implementation for the controller was then assessed. This requires the selected method of implementation to be capable of sustaining the servo control methods discussed in this chapter accurately and for extended periods of time. In addition, it would be desirable to select a method of implementation that would be capable of expansion beyond the current requirements, in areas such as possible feedback, or modified control structure.

3 THE NEW CONTROLLER

After identifying the requirements for the output stage of the new controller, a method of signal generation for the output stage, as well as abilities and methods of manipulation of these output signals provides the requirements of the new controller. To increase the ability to upgrade and modify the controller methods of control, the control schemes are not implemented in hardware on the controller, but instead left as part of the software. This however adds to the need for a robust communications interface allowing those methods of control to be implemented on a PC, and then applied to the servos via the controller. The control board receives a specific output from a PC, interprets the data received and directly generates outputs to the motors as required. In this fashion, the control board can simply be seen as a module to extend the functionality of the PC, providing it with a user friendly interface capable of actuating the servos. This can be seen in Figure 1.2.

While the controller must have the ability to drive the four puppet servos concurrently, additional features have been included to provide the user with added functionality from the advanced interface capabilities. To broaden the applications of the controller, additional servo outputs have been implemented, as well as three ports that can be configured for digital I/O, onboard static memory and variable control encoded servo outputs, whereby each of the three servo output modules can be reconfigured for high speed PWM, instead of generating a nine servo encoded output.

The implementation of the control board has been as modular as possible, with each element performing a dedicated task. This allows for simplified upgrades in the future, as well as enhanced diagnosis of component failure and reduction of replacement costs. In addition, all connection to the board has been done using industry standard connectors where possible, and using PCB mount terminal blocks for non-standardized interfaces.

3.1 ADVANTAGES OF A MICROCONTROLLER

With the requirements known, there are several major benefits of using a microcontroller instead of using discrete components to interpret the inputs and generate the outputs. The first major benefit is that because the control positions are generated using a PC, these positions are most readily represented using data, as opposed to the analogue signals in discrete control systems. This however gives rise to the need for a method of interpreting digital positions, and transmitting large amounts of data over a small number of wires. The microcontroller, being a digital device, is capable of intelligent communications, and through serial communications, the number of wires, and final complexity of the system is decreased dramatically.

The second major reason that a microcontroller was chosen, is that it can provide far more accurate output that could be generated in any other fashion, and with a far greater certainty when under production with far less calibration. Methods using discreet components are often inaccurate, or require constant calibration, providing degrading accuracy over time for reasons such as large tolerances on critical components, and interpretation from converters such as digital to analogue converters (DAC's), which have a resistor divider chain and tap decoder circuit. Using a microcontroller, the output signals are accurate to within 62.5 nanoseconds ± 100ppm (0.01%). This is because of the 16-bit free running PCA counter that is clocked from the 32MHz crystal, and is accurate to within the same tolerances as the 3rd overtone crystal clocking the system.

To facilitate both the highly accurate timing required for the servo control signals, as well as the communications interface, particularly the PC communications interface discussed in Chapter 4, the Atmel AT89C5131A-L [5] microcontroller is chosen. This controller has a large number of independently operable on-chip communications interfaces, with each interface being implemented via dedicated onboard circuits with specific interrupts generated for service requests, reducing the CPU overhead. In addition, the controller has a 5 bank Programmable Counter Array (PCA) for generation

of pulse trains with very high speed and accuracy capable of driving larger numbers of servo motors than would be available with typical microcontrollers, and again minimizing the amount of CPU interaction required.

The ability to operate these communications and high speed output interfaces with accurate timing requires that the microcontroller service each interface as quickly as possible, and remain engaged in any one service for the lowest possible number of clock cycles. The use of the AT89C5131's interrupt registers allows each interface to be serviced only when required, and using the onboard interrupt priority registers, can be configured so that interrupt requests with the most critical timing have a higher precedence. The controller firmware is optimized for the more frequent and higher priority interrupts, causing the system to execute these operations in shorter times, so as to avoid blocking other interrupts for extended periods of time.

3.2 SIGNAL DECODING SYSTEM

The microcontroller's Programmable Counter Array (PCA) is capable of operating each of its five modules different modes, including input capture, high speed output and PWM modes, however, for the purposes of multiple pulse generation, high speed output mode was chosen. In this mode, each module has a 16 bit capture/compare register that, when equal to the free running counter, toggles the pin between high and low states. In addition to toggling the logic state of the pin, an interrupt can be generated, in this case to find the next value to load into the capture compare register.

Because all PCA modules share a single interrupt vector, the firmware's interrupt routine first determines the trigger of the interrupt. If it is caused by a compare with a module register, that module is then progressed by either loading a fixed increment in the capture register, to generate a pulse of a certain width, or by setting a time respective to the output position, causing the capture to occur after a varying time equal to the servo position. This method will progress through the servo offset times giving a series

of short pulses whose rising edges are separated by the required servo position pulse width. This is seen as the 'Encoded Signal' in Figure 3.1.

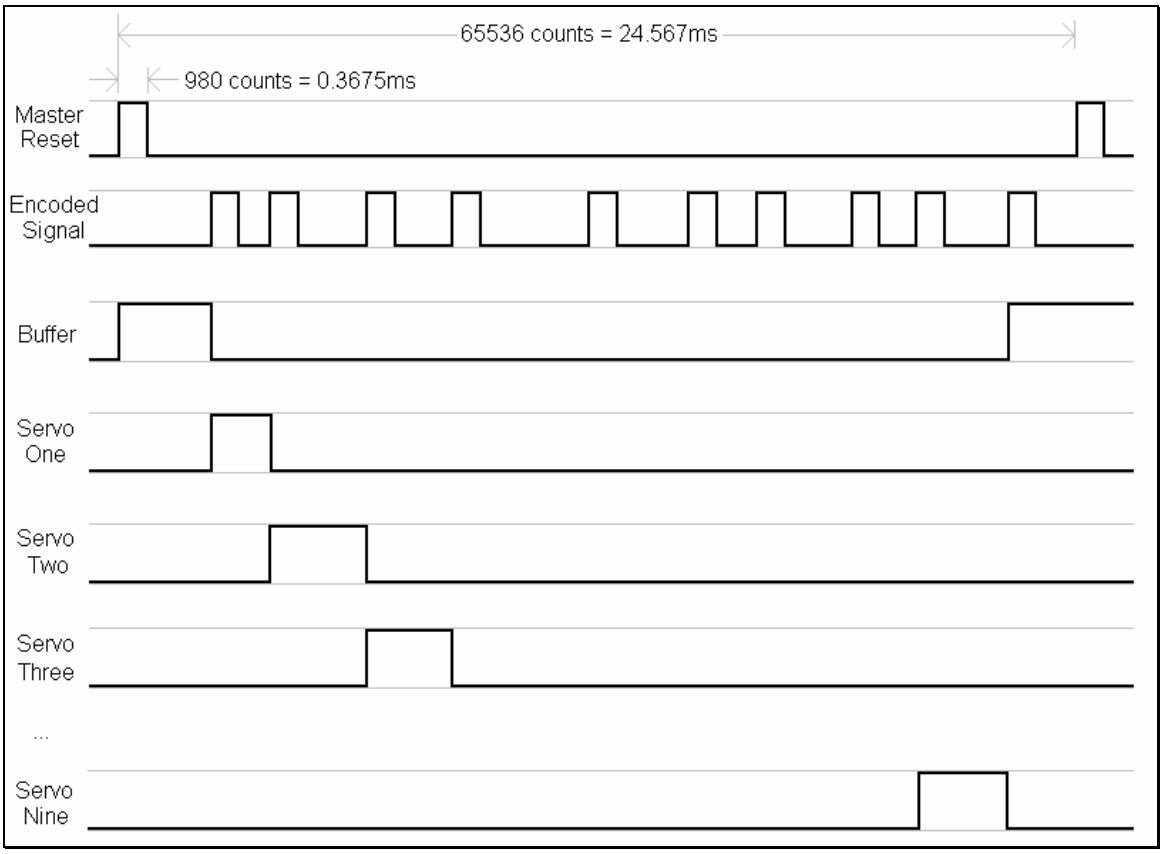


Figure 3.1 – Timing Diagram for Signal Decoding

These pulses are used as the trigger for a 4017 decimal decoder chip [11], which simply advances a single logic high through ten pins at each rising edge. This gives the effect of causing ten separate pulses at the 10 decoder outputs, each with the pulse width equal to the time between pulses from the microcontroller. With the first pin used as a time buffer to allow for varying servo times, a total of 9 servo motors can be connected to each of the decoder chips. These servo signals are labelled 'Servo One' through 'Servo Nine', with the signal for first decoder output labelled 'Buffer', seen above in Figure 3.1.

To avoid creep in the decoder, and to ensure that the microcontroller toggle is always accurate, a second interrupt is generated each time the 16 bit PCA timer overflows. This interrupt sets the logic states of the PCA output pins, and also sets the logic state of a separate pin, used as a reset for the counters. This causes the counters to be reset every time the PCA timer overflows, ensuring that any creep would be eliminated. The signal for the reset line is also shown in Figure 3.1, and is labelled 'Master Reset'.

3.3 SYSTEM OSCILLATOR

The highly accurate timing required in the processor, as well as the desire to minimise code execution times, a high speed crystal module was required for the onboard oscillator. The onboard oscillator is parallel resonant, and typically used in a Pierce configuration, shown in Figure 10.3. This is capable of generating an accurate clock signal with a crystal module up to 32MHz, with anything greater requiring an external 48MHz oscillator must be used, however these are far more expensive, and so were not used.

With a 32MHz crystal module, two types of crystal cut are available, with each greatly affecting the circuit design. The first of these is a fundamental crystal, in which the crystal oscillates at its fundamental frequency, and the second is an overtone crystal, in which the crystal must be forced to operate at a higher order harmonic of its natural frequency. Fundamental crystals are often more expensive since the accuracy required in cutting the smaller crystal is far greater, although they do have a higher pullability, or variance in output when the load capacitance changes. In the controller, the load capacitance is able to be kept stable, with the accuracy of the crystal a far larger concern, and so an overtone crystal is selected.

In addition to this, there are several other factors that encourage the use of an overtone crystal. The availability of fundamental 32MHz crystals is very low, since they are hard to manufacture because of the high accuracy cutting required. Fundamental crystals of this frequency require BT cut crystals, instead of the AT cut that may be used in

overtone crystals, increasing the cutting complexity and reducing the stability of the crystal. The microcontroller also has a parallel oscillator, which prohibits the use of series resonant crystals, and so a standard AT cut, parallel resonant 32MHz crystal module is used configured as a Pierce oscillator with an LC tank for tuning the crystal to run at the desired overtone.

To implement an overtone crystal, the circuit connected to the oscillator input and output on the chip must be designed with a suitable frequency trap to inhibit the crystal oscillating at its fundamental frequency. The addition of the LC overtone tank causes the crystal to oscillate at its overtone because of the two modes of resonance that it introduces. This is discussed in more detail in section 10.3, Appendix C – Overtone Crystal.

3.4 INFORMATION STORAGE USING ONBOARD E²PROM

The microcontroller has 1024 bytes of onboard electronically erasable programmable read only memory (E²PROM). This provides the option of storing data on the controller such as configuration data required when the controller is powered down or disconnected. This memory space is written in pages using the 128 byte column latch, allowing each write to program between 1 and 128 bytes.

The memory space is accessed using direct memory access in C, allowing the exact memory locations for each byte to be dictated, while still maintaining the ability to program a large number of bytes in a single write. While reading and writing to the E²PROM however, a delay is incurred by the chips slow access times for the column latch. To accommodate this, when reading and writing to this area of memory, all non-essential interrupts are disabled, with only the interrupts required for communications and the watchdog timer remaining active. All other interrupts are re-enabled once the required information has been read from the chip.

With the ability to store information on the controller, a standard report was created to allow the information to be readily changed and retrieved. This in turn allows users to go on to configure the device specifically for the hardware that is attached to it. More information regarding the configuration and its applications is given later in section 4.3.4.

Through understanding of the new design approach, and by knowing both the advantages and disadvantages of using a microcontroller capable of allowing all of the above to be fulfilled, aspects of supporting infrastructure were formed. Microcontrollers are able to transfer data quickly and efficiently, as well as possessing highly accurate timing, and so a method of interaction with the microcontroller was required that would allow equivalent accuracy, while still possessing the ability to effect large numbers of changes in parallel. It is with these requirements in mind that the use of a standard PC, with its ability to interface with people, was chosen to collect the input data, be it directly from a person or computer generated. The only problem with selecting this as the means of data collection was that it would then have to relay this data to the microcontroller. For this translation of information between devices, the most elegant method of communications was sought.

4 COMPUTER – CONTROLLER INTERFACE

Because the control system is intended to improve ease of use over the previous controller as well as extending the available functionality, the communications interface between the control board and the controlling PC had to demonstrate its ability to facilitate a large number of requirements, the most important of which are were:

- Ease of use
- Distance
- Speed
- Reliability
- Accuracy
- Compatibility / Portability
- Maintainability
- Versatility

The controller interface is the vital link providing the user with a method of interaction with the microprocessor, and so when entering the selection process, some of the above attributes were considered for a number of major communications interfaces used in digital communications. The results for each are given in Table 4.1.

It became clear that because distance is required as well a high level of compatibility, serial communications was essential. This also allowed a lower pin count on the selected microprocessor, but the protocol would have to be capable of supporting adequate speed and communications structures. In addition to this, the support available for the particular interfaces, the availability of components, ease of testing facilities and previous applications also effected the decision. It is from this information that the final decision as to which interface to use used was decided.

Interface	Format	Number Of Devices (maximum)	Length (maximum, feet)	Speed (maximum, bits/sec.)	Typical Use
USB	asynchronous serial	127	16 (or up to 96 with 5 hubs)	1.5M, 12M, 480M	Mouse, keyboard, disk drive, modem, audio
RS-232 (EIA/TIA- 232)	asynchronous serial	2	50-100	20k (115k with some hardware)	Modem, mouse, instrumentation
RS-485 (TIA/EIA- 485)	asynchronous serial	32 unit loads (up to 256 devices with some hardware)	4000	10M	Data acquisition and control systems
IrDA	asynchronous serial infrared	2	6	115k	Printers, hand-held computers
Microwire	synchronous serial	8	10	2M	Microcontroller communications
SPI	synchronous serial	8	10	2.1M	Microcontroller communications
I^2C	synchronous serial	40	18	3.4M	Microcontroller communications

Interface	Format	Number Of Devices (maximum)	Length (maximum, feet)	Speed (maximum, bits/sec.)	Typical Use
IEEE-1394 (FireWire)	Serial	64	15	400M (increasing to 3.2G with IEEE-1394b)	Video, mass storage
IEEE-488 (GPIB)	Parallel	15	60	8M	Instrumentation
Ethernet	Serial	1024	1600	10M/100M/1G	Networked PC
MIDI	Serial	2 (more with flow-through mode)	50	31.5k	Music, show control
Parallel Printer Port	serial current loop	2 (8 with daisy-chain support)	10-30	8M	Printers, scanners, disk drives
PCI / PCI-X	Parallel	unlimited	onboard	1G, 8G	I/O cards, control interfaces, graphics cards
PCI-E (1/2/4/8/16/32x)	Parallel	unlimited	onboard	2G, 4G, 8G, 16G, 32G, 64G	I/O cards, graphics cards

Table 4.1 – Communications Standards Comparison

4.1 ADVANTAGES OF USB

Because of the nature of the control, the most significant characteristic was the speed, not only of raw data transfers, but because serial communications are used and the device can share a common bus with a number of other devices, the maximum latency of the data transmissions for each device on that bus. The versatility of USB, with its large number of concurrent devices and the ease of implementing large numbers of controllers simultaneously were aspects of interest. USB had a clear speed advantage over other asynchronous serial communications like RS-232 and IrDA, as well as a distinct benefit with its compatibility because of its connectivity to PC's, and not simply other specialised devices, as in the case of Microwire, SPI and I^2C. In addition to this, USB did not have accuracy and latency issues, found with long haul protocols like ethernet and RS-485, or complicated installations as with PCI and PCI-E interfaces. Because the USB protocol was found to have significant benefits in each of the criteria listed at the beginning of this chapter, it was chosen as the communications protocol for the given application. The most critical elements where USB was superior are detailed below, with focus on the benefit in the particular application.

4.1.1 COMPATIBILITY AND MAINTAINABILITY

To ensure the highest levels of compatibility, USB has significant backing and support with the copyright on the USB 2.0 standard assigned jointly to seven major computer and software manufacturers: Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC and Philips. These companies have jointly agreed that the specification will be freely available to anybody, and have created the USB implementers forum to distribute implementation information, as well as assess any devices that wish to display the USB logo for compliance.

USB specifies both a physical standard, including port specifications and cable characteristics, as well as a logical standard that ensures that provided devices comply with the standards minimum requirements, they will be enumerable by a compliant host,

and typically operable under a standard operating system that is USB enabled, such as Microsoft Windows 98 or later. This high level of standardisation ensures the controller firmware and relevant software should be easy to maintain, with detailed specifications available freely to investigate for possible modifications. By designing the controller to conform to this standard, its ongoing support and compatibility can be assured because of the large numbers of devices designed to use it, and the companies who support it.

4.1.2 EASE OF USE

A very desirable feature for the controller was to provide a simple to use interface for end users. With no need to configure communications interfaces or provide settings such as port addresses or IRQ numbers, installation of devices becomes literally plug-and-play. This is attributed to the in depth protocol that is required for all USB peripheral's. The sheer volume of enumeration data compared to other standards does increase development time, however once complete a practically invisible and highly robust interface was available for the controller to operate over.

Another major benefit to USB is its inline power supply. Since the controller does not require large amounts of current when simply testing, programming or observing output signals, it is able to draw its power directly from the USB power rails, and does not require any additional power source to be provided. This can reduce the number of cables, as well as the need to transport large amounts of hardware if it is not necessary.

In addition to this, USB devices are hot-swappable, and so can be added and removed from the system at any time, able to remain stable regardless of valid user interventions. The designed controller is capable of identifying and detaching as the specification requires, with enumeration being performed upon connection to a valid USB hub. As well as this, removal from the USB port does not corrupt the device firmware, and the device can maintain output signals if power is still provided via an alternate supply.

Because USB connectors are typically provided externally, the controller is capable of being connected without the removal of the computers enclosure, such as is found on a large number of older control systems. This means that the average home user can connect it to their computer with a minimum of effort, and still obtain all of the same functionality.

In addition to the standard connection and enumeration procedures, the controller is also capable of intelligent recognition by an application. This means that when a controller is attached or removed, it is automatically added to the device taxonomy, and is able to not only be recognised as a controller, but also reveal information regarding the version of the device and its capabilities. This is a level of recognition and manipulation that is not found in the majority of other interfaces.

4.1.3 SPEED, RELIABILITY AND ACCURACY

The various transfer types in USB made it a clear leader for data transmission. With the critical timing of the data being sent to the controller, and high frequency of communications, latencies and regularity of the transmission became extremely critical. Because of the nature of the data, a large number of smaller asynchronous transactions are required to take place, instead of the larger less frequent transfers used in long haul protocols able to transfer data over distances greater than five meters. The controller is designed compliant with USB 2.0's full speed transfer specification, and uses two of the transfer types available under USB. The first is control transfers, which must be supported by every USB device for enumeration, and the second is interrupt transfers. The controller uses interrupt transfers for the pipes containing data used to manipulate the actuators and signals with maximum latencies. With this transfer type, the control is capable of scheduling a 64 byte transfer each frame, with a frame length of one millisecond. This equates to a total transfer rate of 64kBps at full speed with a latency of 1ms when operating at capacity. An overhead of 13 bytes per transaction is incurred, taking the maximum efficiency on each transfer to approximately 80% if the full 64 byte report is utilized. Each full speed USB bus can support a maximum of nineteen 64

byte transactions with no more than one transaction per pipe. This transfer rate is however far greater than that required by the controller, and so a single IN and a single OUT transaction are performed every 20ms. This ensures that the bus has enough available transmission for multiple devices.

Because the interface is able to be catered for by an integrated serial interface engine within the microcontroller, it is a very reliable system, and with the signal transfer protocols in place under USB, all data transmitted and received can be assumed correct. This provides virtually unparalleled levels of reliability and accuracy in the transmissions. Additionally, because this accuracy reduces communications overhead and the high transfer speed is ample for the current controller, the option of upgrading the device to run according to the USB 2.0 High Speed standard would allow far greater data transmission rates, since this allows devices to schedule three 1024 byte packets in each 125µs microframe, taking the transfer speed from 64kBps to 24.567MBps, or more than 380 times the data. For the current purposes however, this was not necessary, with the full speed USB data transmissions providing sufficient data transfer rates with excellent accuracy.

4.1.4 LOW COST AND TIME OF IMPLEMENTATION

Typically mutually exclusive goals, the implementation of USB in the controller has improved both attributes when compared to alternate standards. The controller is able to be produced using standard components that have large numbers of applications, and so highly specialised equipment is not necessary, causing the price of the equipment to be reduced. In addition to this, because of USB's universal nature, a number of free development and testing tools are made available, or are available at small cost, reducing both the extended development time for USB peripherals over that of less complex interfaces, and providing a lower chance of incompatibility or device failure. Because of this the controller was able to be designed for cheaper production than otherwise possible, with lower tolerances and greater confidence of its abilities.

4.2 USB DEVICE TYPE – HUMAN INTERFACE DEVICE

Now that the features of the communications protocol have been defined, the controller's device type must be defined to facilitate communications via the operating system. Once the controller's characteristics are enumerated by the host, the operating system must load a driver that is capable of acting as an interface between applications written to use that device and the hardware communications from the USB host.

When developing the control's interface, the ability to use a precompiled driver became a significant decision, since there are strong arguments made both in favour of and against the use of these utilities. Ultimately, the abilities of the precompiled and well established Human Interface Device (HID) class [2] driver was found to not diminish the abilities of the controller, still allowing full use of all of the beneficial features of the USB protocol outlined in Section 4.1, and so was selected as a suitable driver for the firmware interface. The largest consideration for this is that the driver provides a standard hardware to software interface [9] via the windows Application Programmers Interface (API), to enhance compatibility.

4.2.1 COMPATIBILITY / PORTABILITY

The major advantage of using a standard device type interface is that the device is then compatible with all computers that comply with the same standard. The fact that the driver is distributed by the OS manufacturer means that updates to the driver, including updated enumeration methods, and improvements to the standard drivers to reduce machine overhead and improve stability will also have a flowthrough effect to the operation of the controller. A high level or portability for USB HID devices ensures that most people will be able to use the device without complications, and with fewer setup issues.

While the application software is currently only designed to operate on a machine with a Microsoft operating system, the ability to generate applications that run on other

operating systems requires only application code for that operating system, with application calls being implemented through the relevant HID driver for that particular operating system.

4.2.2 REDUCED DEVELOPMENT TIME AND OVERHEADS

Using the well established HID class as a framework to build the interface characteristics of the controller meant that the development process for the controllers firmware, as well as applications to use the controller, were significantly shorter than if they had to be created in addition to a custom device driver. The controller's firmware was aided by the HID framework because it provided a clear set of transfer methods for which the controller had to comply, with detail provided about how these methods should be implemented.

While the guidelines for the firmware communications were useful, the benefit to the creation of the application to drive the interface was far greater. This is because under the Microsoft Windows operating system, the API provided was used to perform all data transfer with the controller. This means that a great deal of low level calls, often requiring extensive knowledge and research into how Windows communicates with devices of this type were not required, but rather were replaced by simpler, more robust calls that provide a less error prone method of communication. More about the API calls used can be found under chapter 4.4.

Finally, to ensure maximum compatibility with the USB specification, the free testing utilities provided by the USB implementer's forum were used on the controller to find any area's where compliance was not met. The utilities that were used were the HID descriptor tool, used to aid in the creation of the report descriptors used in the firmware, discussed in section 4.3, and the USB Command Verifier. This utility is capable of running a USB device through all required states for compliance, and providing a compliance report, highlighting any problems in its output report. The use of these utilities is outlined in section 4.5.

4.2.3 RESTRICTIONS ARISING THROUGH CONFORMITY

The disadvantage of selecting a HID generic device driver was that the controller's abilities had to be modified to conform to the HID specification. In the original HID 1.0 specification, devices are required only to have one interrupt IN endpoint as well as the standard control endpoint 0, restricting OUT transactions to control transfers only. Shortly after this was extended under the HID 1.1 specification with devices having interrupt OUT endpoints capable of accepting transfers on either the control pipe, or using the endpoint assigned to any interrupt OUT pipe. This meant that all HID compliant devices should be capable of operating without the use of any interrupt OUT transfers to maintain backwards compatibility. Obtaining accurate timing using this standard becomes very hard. Because of this, while the device is operable under a HID 1.0 compliant driver, the HID driver for Microsoft Windows are all updatable to a HID 1.1 compliant driver, allowing the use of the devices interrupt OUT endpoint.

In addition to this issue with backwards compatibility, the HID specification does also restrict the transfer types available. Under the HID compliant specification, devices cannot use bulk or isochronous transfers. This is not of significant consequence to the current requirements of the controller, since all of the required abilities are better suited to the two available transfer types. This does have some effect on possible future development, but should a custom device driver be written, it should be used to replace the generic HID class driver, and support both older and newer controllers.

4.3 USB FIRMWARE REQUIREMENTS

Having chosen USB as the communications interface the device would have to conform to all the guidelines for USB devices set out in the USB 2.0 specification [1] to remain compatible. In addition to this, the device would have to conform to the requirements of the USB HID specification [2]. Because of these stringent requirements, the development language of C++ was chosen. This code is responsible for all of the client

side USB communications between the host PC and the controller, as well as generation of applicable clocking signals. The clocking signal is implemented using the external 32MHz crystal oscillator, and the internal Digital Phase Lock Loop (DPLL) which acts as a frequency multiplier, to generate the 48MHz ± 0.5% clock signal required to run the USB SIE. The communications firmware is loosely based on the structure provided by Atmel, demonstrating the implementation of USB enumeration. This code was tested then rewritten to provide a more robust interface for the device. Each major section of the code is explained with reasoning for methods of coding choice.

4.3.1 COMMUNICATIONS METHOD

The example code given by Atmel is based on a polling method for USB communications, with the code executing a simple check of the SIE status flags to determine if any data was required to be read in or out of the endpoint buffers. This method was found to run well only when minimal code was implemented in the main program loop, and so was quickly abandoned for a more elegant approach using the interrupt registers, with flags that can be triggered each time the serial interface engine requires attention. Using this method, the code with non-critical timing could be implemented in the main runtime loop without interference to the communications.

To enable the USB interrupt service routines (ISR's), three levels of interrupt enable registers had to be configured. The enable all interrupt flag, the enable USB interrupt flag, and an enable endpoint interrupt flag for each of the subsequent endpoints that were implemented. These flags trigger an interrupt at the predetermined USB interrupt vector. Since only a single interrupt vector is triggered when the SIE requires attention, the subsequent status flags for each of the endpoint is checked, and the endpoint that requires attention is serviced. This involves checking the status of that endpoint and performing the required read, write, stall or other command with that endpoints FIFO buffer.

4.3.2 ENUMERATION / AUTHENTICATION

Before communications could begin, the standard enumeration procedure that exists for all USB devices is undertaken. This happens when the controller is first attached to the system bus, or when the device is reconfigured as a result of detaching from the host or during the hosts maintenance routines. The enumeration process consists of a series of standard requests from the host to the controllers' endpoint 0. These requests retrieve a number of descriptors detailing the device and its abilities and requirements, and have the ability to change the device state to one of six possible states defined in the USB 2.0 specification. These states are attached, powered, default, addressed, configured and suspended. These states identify what capabilities of the controller are functional at any given time, with the descriptors giving a full description of the controller and its capabilities in these states. The types of descriptors that were required to be implemented on the controller are:

- USB standard descriptors
- Device Descriptor
- Configuration Descriptor
- Interface Descriptor
- Endpoint Descriptors
- String Descriptors
- HID class descriptors
- HID Descriptor
- Report Descriptors

The firmware on the controller, upon receiving a request for a standard descriptor retrieves the descriptor, and sends it on the control endpoint, breaking up descriptors that are larger than the endpoint 0 FIFO buffer, and including a zero length packet (ZLP) after the completion of a descriptor that is an even multiple of the control endpoints buffer size.

Once the required descriptors are transferred, the device interfaces are enabled and the host and controller can begin standard communications. At this time the controller is in the configured state and can begin standard operations. It is now that the host schedules the interrupt transactions requested by the device with the required endpoints, satisfying the maximum latency requirements of the device. If the current USB host does not have enough idle time to schedule the transfers on that bus, it will abort communications with the device.

The descriptors with interpretations are included in Appendix B – Descriptors. While this configuration process is arduous, and the construction of the many descriptors requires a large amount of very specific data, once complete, USB delivers a robust communications system that will outperform the others available in this situation.

4.3.3 DATA TRANSMISSIONS

Once enumeration is complete, the controller and the host have created two interrupt transfer pipes, one for IN transactions and another for OUT transactions. The IN pipe has a 255ms maximum latency for communications to reduce the data transmissions required, while the OUT pipe is configured to have a 16ms maximum latency. The `bInterval` value in the endpoint descriptor determines the maximum time that can elapse before a transfer is requested. In addition to this delay, and because the controller is HID compliant, it is capable of setting an idle rate for the pipe. This idle rate is the maximum time interval that may elapse before a report must be sent, instead of simply replying with a negative acknowledge (NAK) packet. For HID's, the polling interval of the USB host may be far smaller than the time difference between varying user inputs, and so if the data in interrupt IN reports has not changed since the previous report request, the device simply returns a NAK, indicating that the data has not changed. On enumeration the idle rate for the controllers' pipes is set to 0, meaning that they will only send a new report if the data has changed or a new report is required. This way the controller and the data bus are both able to remain idle should the report data remain constant. Using these two timing settings, `bInterval` and idle, the data

on the bus is reduced to a minimum, and so accurate timing for a larger number of devices can be assured.

4.3.4 ACCESSING E²PROM FUNCTIONALITY

To enable access to the onboard E²PROM the device firmware using the available data transmissions, a 1000 byte feature report was declared in the HID report descriptor. Feature reports were defined in the HID specification to allow a device specific method for enabling and disabling features in devices, or reporting on the status of features that the device is using.

These reports are used in the controller as the dedicated method for maintaining device configuration information. If a feature report is request by the host, then the contents of the controllers E²PROM is dynamically loaded and sent in successive 32 byte packets over the control endpoint, with the final packet being only 8 bytes. If a feature report is sent, then as each data packet is received and programmed into successive memory locations.

This method of access reduces optimises the system by removing the need for a RAM space capable of buffering the entire transmission. Since the size of the data is greater than the RAM on the controller, it is impossible to implement a data buffer without moving to a less efficient memory model, which greatly increases both code size and execution times. To program the E²PROM however, requires a large number of writes, and so should not be performed too many times. This should not necessarily be a problem since the controller should only be reconfigured when a new puppet is connected to the servo outputs, and at this time, the controller needs only to be reprogrammed a single time to edit all of the information on it.

4.4 USB SOFTWARE REQUIREMENTS

With the firmware complete, the software must also fulfil a number of requirements to detect, connect and communicate with a controller. In order for applications to communicate with the controller, it detects when and devices are attached or removed from the system, and accesses the newly updated set of devices currently attached to the computer to identify each, then in turn enumerates each to check if it is a valid controller. Once a controller is detected, the program creates a handle to the HID device driver, enabling it to communicate with the controller, letting all hardware specific communication requirements remain under the control of the driver, shielding the application from having to communicate differently with every different device.

4.4.1 DETECTING A CONTROLLER

To do this under Windows, the operating system for which the application interface was generated Microsoft has provided specific API functions to facilitate enumeration. The API is written using C++, and so to call it using Visual Basic 6, all of the desired functions had to be redeclared with equivalent declarations in the native language. Once the redeclarations were provided, the application could access these standard API calls to communicate with controllers.

The first step to detecting any device in a dynamic environment was to determine when devices were added and removed from the system to ensure that at this time the program can immediately deal with the addition or removal, avoiding the possibility of communication attempts with devices that are no longer connected. This is achieved by diverting all calls to the programs message handler, and watching for messages alerting the program that the devices attached to the system have changed, known as the `WM_DEVICECHANGE` message.

Once the system is aware that a change has occurred, or the program is simply being started for the first time, it must search for all valid controllers. Valid HID devices are

then enumerated by retrieving the globally unique identifier (GUID) that identifies all HID devices on the system, known as the HID GUID. Then, using this GUID, the API function `SetupDiGetClassDevs` returns the handle to a handle, `DeviceInfoList`, which is for a structure containing all of the information about all installed HID devices on the system. This structure is then read in parts using the `SetupDiEnumDeviceInterfaces` API call, with the index for the device that is currently being enumerated, and receives a `SP_DEVICE_INTERFACE_DATA` structure that contains the enumeration information. This in turn then allows the program to retrieve a pathname to the device that is being checked using `SetupDiGetDeviceInterfaceDetail` with the data already obtained.

From here, the application can test if the device is currently installed as a controller, by checking the pathname against that of the other controllers installed on the system, and if a current controller has the same pathname, the device is currently already connected. If the pathname of an attached device does not exist, then the program opens a handle for communications with the device using `CreateFile`, and can then use the HID specific API calls to determine if the device is a controller. The `HidD_GetAttributes` function returns a `HIDD_ATTRUBUTES` structure that contains `VendorID` and a `ProductID` field. If these fields are consistent with those for a controller, then the device can be added to the array of installed controllers and communicated with, using `ReadFile` and `WriteFile`. If these are not appropriate, then the application closes the handle for communications with the device, and proceeds to the next device in the `DeviceInfoList` structure until all devices have been checked, and all valid controllers are enumerated.

4.4.2 COMMUNICATIONS WITH THE CONTROLLER

Once controllers are connected and ready for communications, the application uses the standard API calls `ReadFile` and `WriteFile` to communicate with them, however, these calls are blocking calls, halting program execution until the controller is capable of responding successfully. This is not a problem with `WriteFile` since while under

standard operation the controller is always able to accept an incoming report, and so does not halt the program. Using `ReadFile` is harder since the controller may not have transmitted a report since its previous transaction, since the data in the report could have changed. This results in the application failing to respond, and stalling the system. This is a highly undesirable effect, and so instead of simply calling `ReadFile`, an asynchronous API call is implemented using `ReadFileEx`, which instead sets the flag of an `EventObject` structure signalling the completion of the read operation. This is achieved by calling `ReadFileEx` with a valid `EventObject` handle created using the `CreateEvent` function, and then calling `WaitForSingleObject` to check if the program successfully read from the object. This could be completed differently with a program language that is capable of running multiple threads, however, this is not implemented at this stage.

The data sent and received using these functions is simply a string of bytes that needs to be interpreted. A report structure has to be determined that can operate on each of the fields appropriately. A structure has been set up for each report to ensure that the data in the reports can be accurately written and read in both directions.

4.4.2.1 INPUT REPORT

This is the report that is transmitted from the controller to the host, and currently is used only in the implementation of 8 bits of digital input, and a byte for a rotary encoder. This equates to only two bytes, however, to allow for expansion, a further 3 bytes of data have been included as reserved. The first byte of the report is the value of read from the input port, Port 2, and the 2nd is for the encoder. The last four reserved bytes are indiscriminate and cannot be set.

Byte(s)	Use
1	Bitwise equivalent to input Port 2 at time of communication
2	Arbitrary value used for rotary encoder
3-5	Reserved

Table 4.2 – Input Report Byte Configuration

4.4.2.2 OUTPUT REPORT

This report is transmitted from the host to the controller, and contains the values for the positions of each of the servo output signals, as well as a configuration bit that can define the mode of operation for the PCA modules one through three. Bytes one through twenty seven correspond to a position for servo outputs one through twenty seven respectively. These outputs are able to range between 0 and 255, providing a complete range of motion for the connected servos. This is shown in Table 4.3. As with the input report, reserved bytes are included in the structure, in this case, there are two. Bytes 28, 32 are reserved, and do not effect the operation of the controller, however, in order to maintain future functionality, applications should maintain these bytes as a 0.

Byte(s)	Use
1-27	Byte wise equivalent to the output position of each of the 27 servos. Bytes 1, 10 and 19 also used as PCA module PWM duty cycle control.
28	Reserved.
29	Used to select PCA module control type. Refer to Table 4.4 below.
30	Bitwise equivalent to Port 0 output vales.
31	Bitwise equivalent to Port 3 output values.
32	Reserved

Table 4.3 – Output Report Byte Configuration

The lower three bits of byte 28 are treated as boolean mode bits, with module 0 being set by the least significant bit, module 1 by the second bit, and module 2 by the third bit,

shown in Table 4.4. Setting these bits will cause the modules to operate in high speed output mode, controlling up to nine servos per module, while clearing these bits will cause the module to revert to PWM mode, and operate a single PWM output. In PWM mode the byte corresponding to the lowest servo position for a specific module, either 1, 10 or 19, is used to set the duty cycle of the PWM output signal.

Bit(s)	Use
1	Used to set module 0 control type. Set for servo control, clear for PWM.
2	Used to set module 1 control type. Set for servo control, clear for PWM.
3	Used to set module 2 control type. Set for servo control, clear for PWM.
4-8	Reserved.

Table 4.4 – Output Report Byte 29 Bit Configuration

4.4.2.3 FEATURE REPORT

The feature report is used to program the E²PROM as discussed in section 4.3.4. This report is defined for both IN and OUT directions, and contains equivalent information. The direction of transfer simply dictates whether the controller is relaying information or reprogramming itself.

In the feature report, the first byte is the length of the control descriptor, which is comprised of a number of smaller pieces of information. Bytes two and three are the firmware version number, with the first byte representing the version, and the second representing the release. The next 20 bytes are used by the application to store a password to prevent people from programming the controller using the program that are not authorised to do so. This is followed by a 60 byte controller name, and a 40 byte icon name, to identify the controller and load an icon of it. After this there is one byte identifying the number of servo's that the controller has descriptors for. This is shown in Table 4.5 below, including the area reserved for storage of individual servo descriptors.

Byte(s)	Use
1	Controller descriptor length in bytes
2	First ordinal of firmware version – Version Number
3	Second ordinal of firmware version – Release Number
4 – 43	ASCII string - Filename for controllers icon
44 – 103	ASCII string – Name of the controller
104 - 123	ASCII string – Controller access password storage
124	Number of servo descriptors contained
125 – 125+6n	Storage for servo 'n' descriptors. Refer to Table 4.6. $0 < n < 27$
126+6n – 1000	Reserved for future implementation.

Table 4.5 – Feature Report Format including Control Descriptor Format

The servo descriptors are each six bytes in length, with bytes one through five representing upper bound, lower bound, idle position, movement type and movement index respectively, and the sixth being a reserved byte. These descriptors are stored consecutively directly following the control descriptor, and are currently limited to a maximum of 27 by the control software. The layout is given in Table 4.6.

Byte(s)	Use
1	Upper boundary servo position
2	Lower boundary servo position
3	Servo idle position
4	Movement Type ID
5	Movement Index
6	Reserved

Table 4.6 – Feature Report – Servo Descriptor Format

Because the number of descriptors is currently limited, the remaining information in the feature report is reserved for future use, providing up to a guaranteed 714 bytes of free storage that could be implemented to store extra information at a later date.

4.4.3 COMMUNICATIONS INTERFACE MODULE

To provide a simpler method of access to these functions, the USB communications software was included in a module, capable of detecting and utilising each controller attached to the system. A second module was also included to declare each of the required API declarations used in this process, and was written using the guidelines provided in the book USB Complete. This module contains the declarations of the API calls that are required to complete successful communications in Visual Basic 6.

The second module performs all tasks specific to communications, and is able to provide a highly simplified user interface for any programmer to access the controller. This module effectively shields an application programmer from having to understand any of the API calls that are necessary to enumerate the controller. The module has several functions that return all the information that is required to communicate with multiple controllers.

4.4.3.1 DETECT

The detect function is the basis of the module. This function is capable of testing every HID currently installed on the system, adding and removing them from a managed array, and then providing an array of type long variables, with the number of elements in the array representing the number of controllers attached, and the value of each element representing the index of that controller in the managed array.

The reason for using an array as opposed to simply returning a number of controllers, is that if multiple controllers are connected, and one other than that with the highest index is removed, the index of the remaining controllers should remain unchanged. Using this

means that communications with controllers is done relative to its array value, and not the array index. Should the function encounter an error while calling any of the API calls, or should it encounter an internal error, it has a predefined set of errors that are given, as well as exiting immediately, allowing another programmer to retrieve the last API call error, should that be desired.

4.4.3.2 READDEVICE

This function uses the asynchronous read method to attempt to read a report from a controller. The controller is told what index is intended to be read, and the appropriate data is updated in the input array. This provides a simple method of retrieving data from a device without having the added stress of implementing custom routines to read each device using asynchronous timing routines.

To avoid complication, each controller is provided with its own input array, so that while the maximum amount of data that is stored is increased, the number of reads and writes to that data is minimized. This is also significant because since the controllers reports are small in comparison to the amount of RAM on modern computers, these structures are able to be stored and dynamically modified at high speeds with minimal write cycles. This function is currently used only for testing the status of the 8 bit digital input, however can be modified to have larger effects in the future.

4.4.3.3 WRITEDEVICE

`WriteDevice` is used to set the positions of the servo motors, as well as edit the output types of the PCA modules. As with `ReadDevice`, each controller has a dedicated output report, that is maintained by the application. To write data to a controller, the values of a specific controllers output array are changed, and then `WriteDevice` is called with a controller index. This then in turn calls `WriteFile` with the appropriate controller handle, output report length and pointer to the report buffer, transmitting the required information to the controller.

4.4.3.4 READDEVICEFEATURE AND WRITEDEVICEFEATURE

In order to obtain the information in the controllers E²PROM and when required reprogram the controller, the application must be able to specify the type of transfer that is used when transmitting data. The only Microsoft operating system that this has been implemented under is Microsoft Windows XP, with expectations newer releases will maintain backwards compatibility. This is one of the reasons that Windows XP is a prerequisite for running the controller successfully.

Now, using these functions, the application is able to both obtain and reprogram the first 1000 bytes of the controllers E²PROM by simply calling the desired function with the appropriate controller index, and providing an adequate data buffer for the information to be either sent from or written into. This highly simplified process hides the complexity that is typically required when programming memory of this nature.

4.4.3.5 DISCONNECTDEVICE

This function is used solely to remove a controller from being enumerated by the application should a user wish to disable communications between the application and the controller. This function is only necessary when the application is exited before the controllers attached to the system have been disconnected. This is typically implemented in an exit routine, but is provided as an additional option to users in the application to enable removal of unwanted controllers from the system.

4.5 TESTING CONDITIONS AND REQUIREMENTS

To test the capabilities of the system, the overall goals should be assessed, primarily the ability of the controller to be attached, enumerated and then accessed by the application. The abilities of the hardware and software are however mutually independent, and the success of each not relevant to the capabilities of the other.

4.5.1 FIRMWARE TESTING

The USB communications aspect of the controller is required to be capable of a complete enumeration with a compliant host controller. There are several aspects to this installation that should be checked. The first of these is the ability for the target operating systems to enumerate all devices. Several operating systems from Windows 98se onward, including Windows 2000, Windows XP, and Windows ME should all be capable of enumerating the controller, and adding it into the device taxonomy with nothing more than attachment by the user. The only exception to this is if the Windows generic HID class drivers were not installed with the operating system, which is the default option only for Windows 98se. Updated drivers are however freely available form Microsoft.

In addition to this, the controller should be able to satisfy testing when inspected using the compliance tools provided from the USB implementers forum. The first of these is the HID report descriptor tool that is used in the creation of report descriptors. The controllers report descriptor should not contain any invalid options in its descriptor, and should pass testing when checked using this utility.

The final test that the controller should satisfy is testing with the USB Command Verifier. This tool, capable of placing a device in every possible state that it should support, will test all standard USB communications which should be available for any generic HID device. In addition to this, it can also test a device for HID compliance, ensuring that it will be capable of compliant communications via compliant hosts with generic drivers. The output reports from this program are included in

Appendix D – Firmware Testing Results.

4.5.2 SOFTWARE TESTING

The application must be able to communicate with the controller when installed under the required operating system. When a valid controller is connected and detected successfully by the host computer, the application should be capable of locating that controller in the computers taxonomy, and creating a communications handle to it. This communications handle must be bidirectional, and capable of successfully sending and retrieving only the specific reports that are intended for the controller, including the interrupt in, interrupt out, control, setup and feature reports.

The applications E²PROM reading and writing should only be attempted under operating systems that support the API commands `Hid_SetFeature` and `Hid_GetFeature`, which includes Windows XP and greater operating systems. To avoid incorrect operation, the application should, when executed from a non-compliant operating system, fail to load, and inform the user that the application failed to load because their operating system does not meet specification, so that the problem can be corrected.

The application should be capable of detecting when devices have been added and removed from the system, and at these times check whether to add or remove controllers from the application managed array. Also, should a controller become unresponsive, an automated disconnect after a certain number of failed transfers should be executed to ensure that the system can maintain stability.

Using this approach to data communications, the microcontroller is able to respond quickly and accurately to the input form a person, while having the ability to relay information back to the user. This data communication standard posses the ability to allow the controller use for many years to come, and through the implementation of this

current technology, the abilities of the new controller are enhanced to a point that would otherwise not be possible. However while this does provide the PC with an excellent method of communication with the controller, the PC must still be capable of both sending and retrieving the required information from the user. This is solved using the graphical nature of current Windows based operating systems, utilised by a single program able to interpret the users' inputs. This single program is the graphical user interface (GUI) and provides the end user with a method of manipulating the outputs of a controller using the input methods provided on the computer. The purpose of the GUI is to allow the user a simple method of accessing the features on the controller using the USB communications interface as simply as possible.

5 GRAPHICAL USER INTERFACE

The Graphical User Interface (GUI) is the PC based interface through which the user is able to interact with these controllers. This interface consists of a number of graphically responsive controls that allow users to visually interpret the state of a given controller, and to adjust all of the features of the controller as simply as possible.

5.1 INTERACTION WITH MULTIPLE CONTROLLERS

The GUI provides an effective means by which a user can connect and interact with up to 10 concurrent controllers. This enhances the ability of a single host by eliminating the need for a separate host for every controller. This type of control causes the system to become both more robust and less hardware intensive. The option of communicating with more than one controller leads to a modular approach towards performing all operations with these controllers.

5.1.1 DATA STRUCTURES FOR MULTIPLE CONTROLLERS

To facilitate a modular approach to communications in code, data structures were used to include all the necessary data for any given controller. Each controller upon enumeration has its configuration read from its E²PROM and the data structure is filled allowing a simple array of these structures to represent each of the controllers attached to the system. This leads to the development of standard communications and interaction operations to be performed using the information from a single data structure.

This approach allows controllers to be handled individually, without having to write specific routines to operate on each. This approach also allows the application to reduce

the complexity of users switching between controllers, by simply modifying the index of the array element on which it is operating.

5.1.2 SIMPLICITY OF NAVIGATION

To maximize the GUI's ease of use, the ability to switch between connected controllers and perform all required actions was achieved through a tab-style interface. In this situation, the index of the selected tab allows the application to determine which array element that it is going to operate on. This type of interface, shown in Figure 5.1 and Figure 5.2 as 'Controller Selection Tabs' is a familiar windows construct for end users.

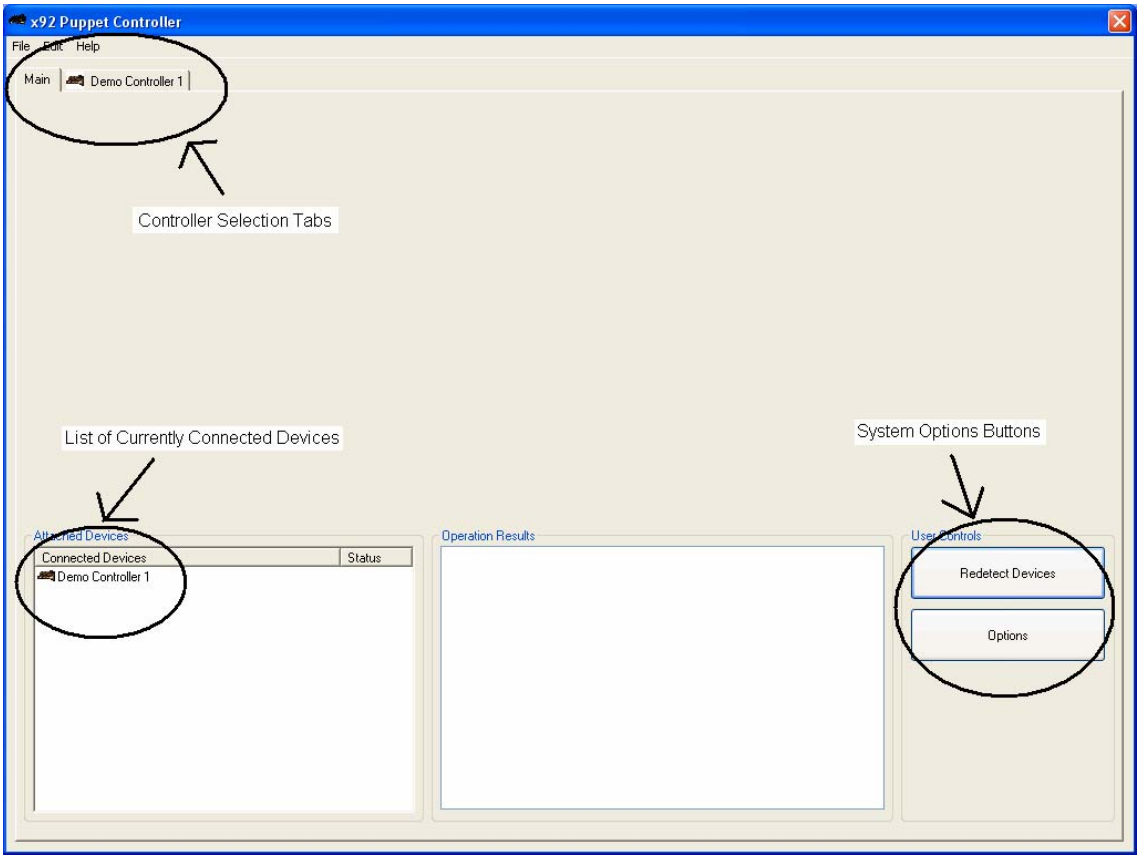


Figure 5.1 – Main Window Breakdown

To facilitate easy navigation, and a simple display for attached controllers a single main screen was implemented as seen in Figure 5.1, allowing users to view all of the controllers connected to the system. This screen also allows the user to manually

redetect all controllers that are connected to the computer should the computer not detect a controller connection or disconnection automatically. If the manual redetection fails, then the user can assume the controller is not operating properly.

When at the main window, users can simply click the 'Controller Selection Tabs' indicated to move to a controller interaction screen that simply displays the information for the selected controller. This reduces the number of items that must be loaded to display the status of all of the attached controllers, and so reduces the overhead of the application when running. This window provides access to the various control methods discussed in following chapters using the 'Control Method Selection' option buttons, direct control using the horizontal sliders in the 'Direct Control Sliders and Information Frame', as well as adjusting the output control type with the 'Module Control Type Selection' check boxes and operating the digital I/O with the 'Digital Inputs and Outputs' buttons.

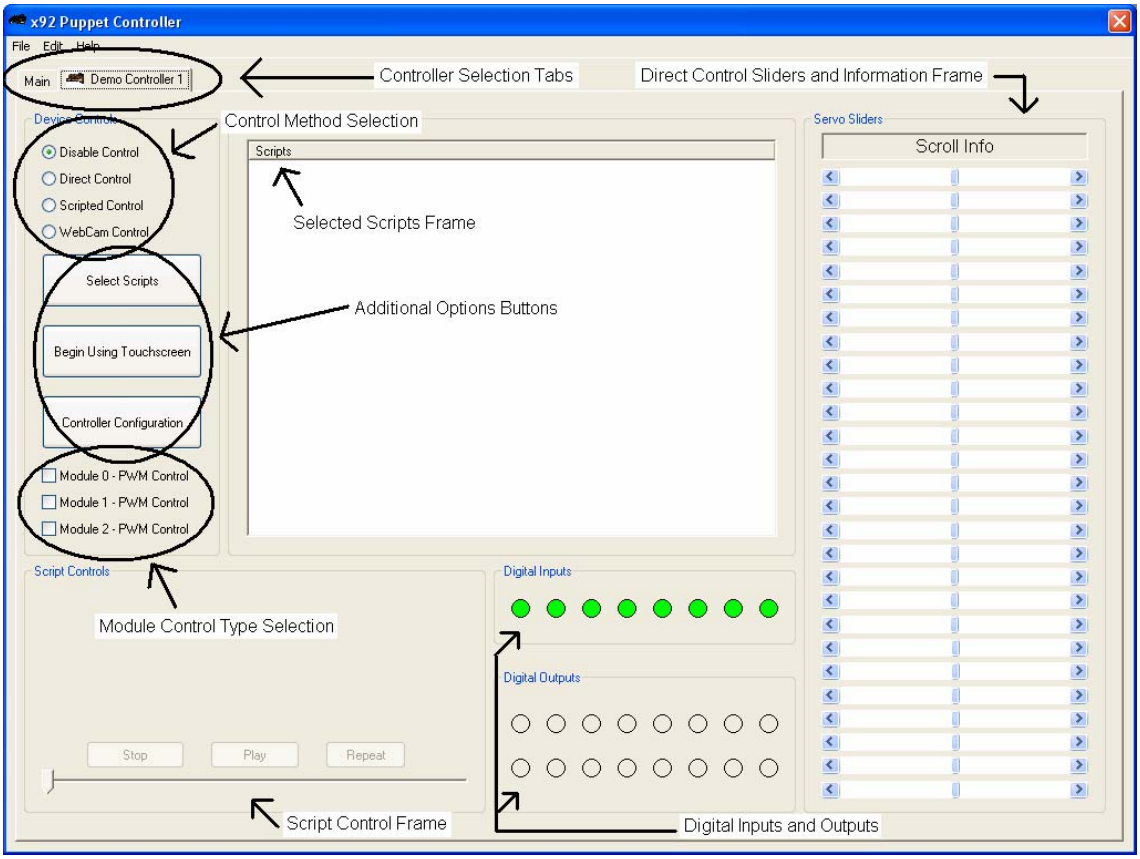


Figure 5.2 – Controller Window Breakdown

From the main screen, users are also capable of accessing the 'Options' dialogue shown below in Figure 5.3, allowing adjustment of system options effecting all controllers. Currently, this dialogue is used only to allow mapping of scripts and selection of icon and script storage folders.

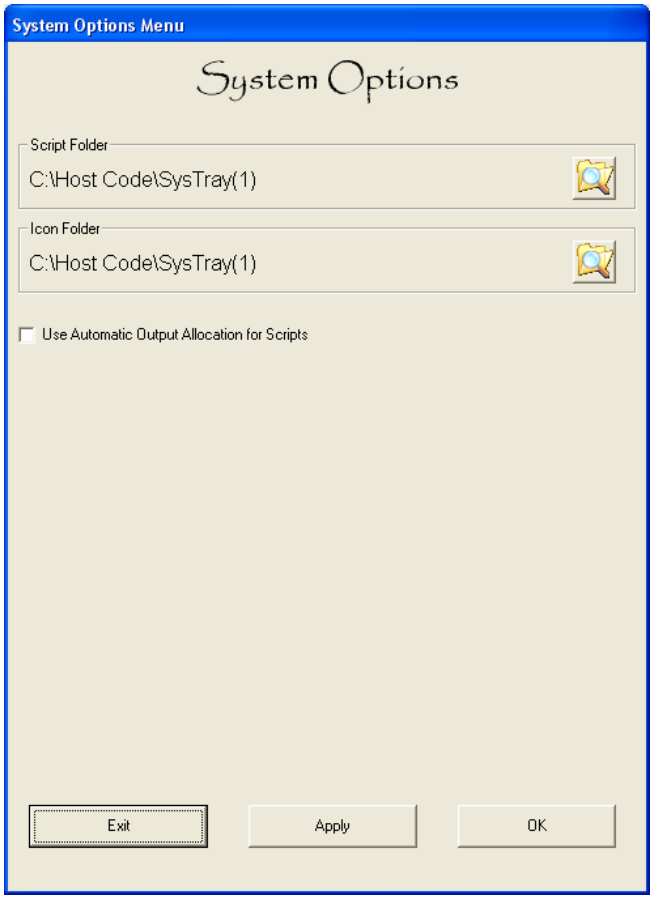


Figure 5.3 – System Options Dialogue

These screens collectively provide the backbone of the interaction system, so that users are able to locate, access and use controllers. While only the lowest level control is available from these windows, operations remain simple and logical, so that beginners are able to navigate easily and without being overwhelmed. More complex operations, such as programming settings for controller E²PROM interface are kept to separate windows.

5.2 INTERFACE TO REPROGRAM CONTROLLER E²PROM

To maximize the benefit from the onboard E²PROM for each controller, a data structure as discussed in section 5.1.1, was designed to include the information required to identify a controller and configure its outputs. One aim of the GUI was to create a method for viewing and editing the controller information in a simple, safe, error free fashion, while still maintaining a robust interface able to relay a large amount of information quickly. The resulting dialogue is shown in Figure 5.4, where the information specific to the controller is included at the top of the window, including the controller firmware version, name, icon and number of connected servos. Beneath this is the information regarding each of the connected servos. The number of small boxes containing servo information dynamically increases and decreases to match the number of servos connected to the controller as indicated in the controller information.

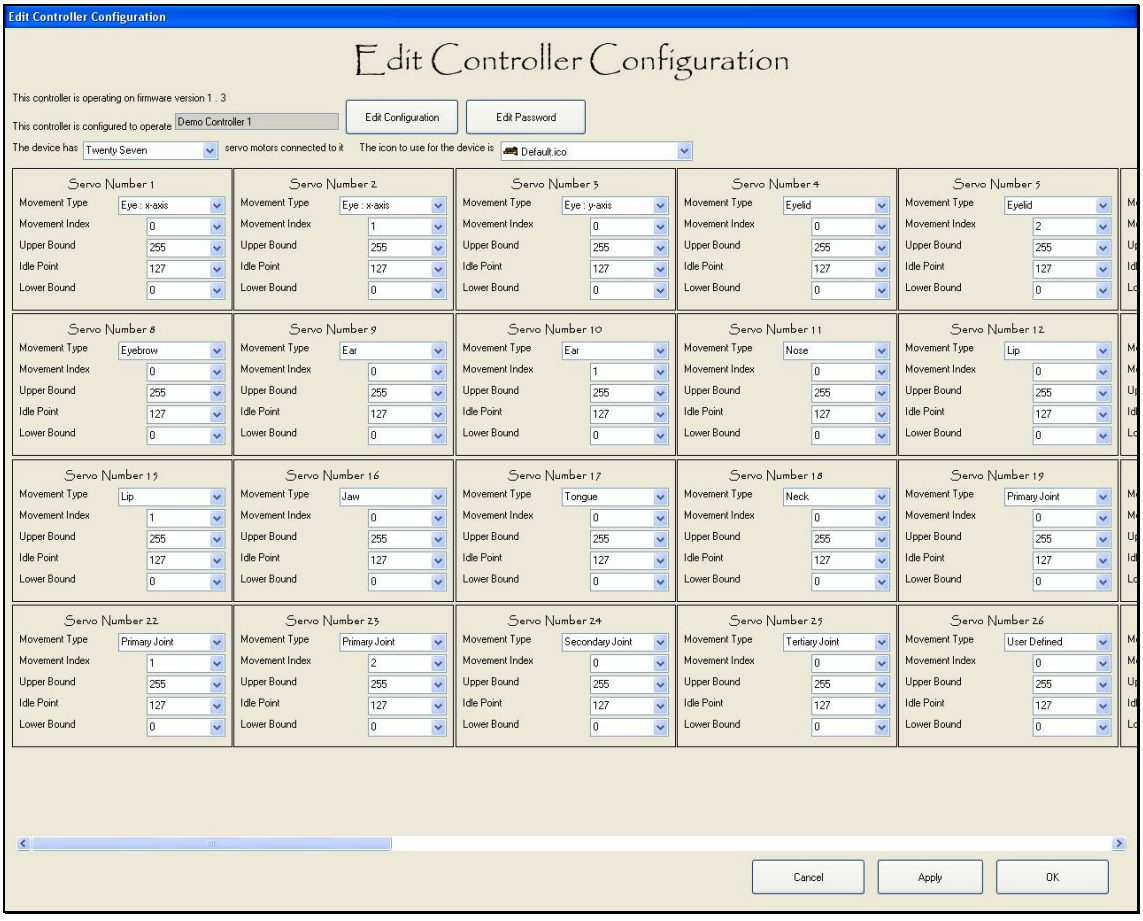


Figure 5.4 – Controller E²PROM Configuration Dialogue

In addition to this, where possible, the user is given combo boxes to choose a setting, as opposed to allowing input from the keyboard. This increases simplicity, since the user is able to simply choose an input, and only valid inputs are given. In instances such as the controller name, a text box is provided, since any keyboard input is valid.

5.2.1 CONTROLLER SPECIFIC PASSWORD

To increase security, a password is stored in the controller, so that anyone wishing to modify the contents of the controllers E²PROM must first have the password to reprogram it. This ensures that only valid users are able to program a controller, and that controller values in the configuration screen are not inadvertently changed. If the 'Edit Configuration' button is clicked in the 'Controller Configuration' dialogue, the user is presented with the option to input the required password. This is shown in Figure 5.5.

Figure 5.5 – Password Input Dialogue

As well as simply having a password specific to each controller, the password can be changed at any time using a simple challenge-response system where a user must first validate themselves by entering the current password, and then enter a new password

identically in to concurrent fields. If both of these requirements are fulfilled then the controller is reprogrammed with the new password. This involves reprogramming the entire contents of the E²PROM, and so repeatedly changing the password is not recommended. In the instance of a lost or forgotten password, overrides are available, but it is more likely that for general use, a separate program will be written with a further challenge response system to avoid people performing illegal overrides.

Figure 5.6 – Password Change Dialogue

These options provide a small level of security for people that wish to use there controller in a public place, so as to avoid either tampering, or confusion as to ownership. To provide further security, the password is not stored as plain text within the E²PROM. This is to prevent people from being able to retrieve the user's password, which they may also use for other, more significant, systems such as online banking. Before writing the password to E²PROM, a one way hash algorithm is first applied. The hash returns a fixed length string that contains a unique 'fingerprint' or signature of the password, which is then stored. When the user is requested to enter their password before changing configurations, it is also hashed then compared with the stored hash. The algorithm used in this software is the MD5 hash, the operation of which is outside

the scope of this thesis. More information on the hashing process can be found in the Network Working Group's Request for Comments (RFC) 1321 [22].

5.2.2 MOVEMENT TESTING INTERFACE

After successfully entering the password, a user may edit the information regarding the controller. This includes setting the upper and lower boundaries, idle point and movement information. When setting the upper and lower boundaries, as well as the idle point of any servo, the user is able to right-click the title of the servo and is presented with a single slider control that is capable of moving the selected servo through the controller's maximum range of motion. This window, illustrated in Figure 5.7, is useful when a user wishes to view specific position values or locate a specific boundary condition. By simply dragging the slide to a desired position and clicking the relevant "set" button, the appropriate boundary is stored ready for programming.

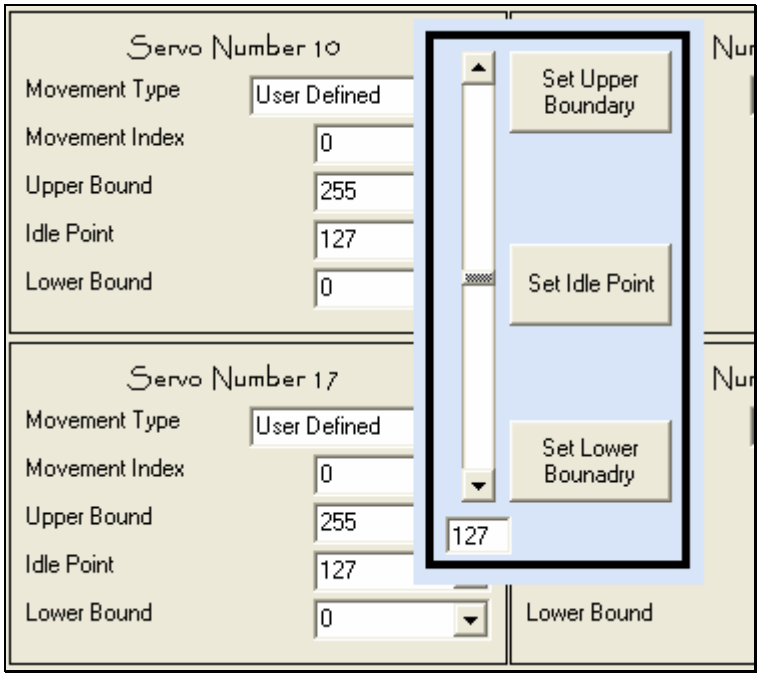


Figure 5.7 – Servo Movement Test Dialogue

This slider is automatically removed from view when the mouse cursor is moved outside its boundary, and can again be displayed for any available servo by simply right

clicking the relevant title. This feature however does not need to be used to program the servo boundaries, with combo boxes being provided for each element of the servos position information.

5.2.3 Detection of Invalid Settings

To avoid unexpected or erroneous outputs, the servo setting must range from having the lower bound at the lowest desired value, the upper bound and the highest desired value and the idle point between these two bounds. This is a reasonable assumption, and if incorrect settings are requested, the application displays a message box alerting the user to the problem, and reverts to the previous setting.

5.3 Use of INI files to Store GUI Configuration

To allow correct operation of the GUI, a number of variables must be stored and loaded between executions of the application. These include paths for icon and script locations, as well as personal preferences for mapping scripts. More settings are available for video control, but are not covered in the standard options dialogue.

To store these settings, standard initialisation (INI) files are used. These files are capable of providing a key, variable name and value for each of the required settings. These files provide a robust interface that is easily expandable and able to cater for all variables that should be displayed.

5.4 Structure For Generic Control Method

Since the GUI provides a number of control methods, each of these has a certain number of aspects that are essential to correct control, and are the responsibility of the individual control module. To ensure that each module is capable of accessing the

controller effectively it must be able to obtain the values from the user, accurately scale them and relay them correctly to the controller that is required.

5.4.1 PROVIDING AN EFFECTIVE CONTROL INTERFACE

Control interfaces allow the user to both easily identify what actions must be taken on their part to effect change in the controller, with this action delivering a logical result. This type of control requires that each control system ensure that it is capable of accurately interpreting whatever method it retrieves its input from, and is capable of providing the user with some method of display to know that the input has been acknowledged and interpreted correctly.

To ensure that each interface functions as accurately as possible, users are be able to easily move the desired output from one point to another, including positioning near or at the extremities. For greater accuracy, where possible, numeric values can be displayed to ensure that the same position can be returned to at a later time. This can be seen in Figure 5.2, where positioning is performed via sliders, and the numeric position of the current slider could be displayed in the 'Scroll Info' window.

Each control interface has to remain stable should a user wish to exit it at any stage of execution, and removes the need for the user to undergo complex switching practices. This is highlighted by the fact that the user is able to select the control interface simply by way of radio buttons on the controller interaction screen.

5.4.2 STANDARDIZATION OF CONTROL

One of the major goals for generic control methods towards providing an effective control interface was to provide safeguards against users inadvertently requesting controllers move a puppet beyond its limits. To provide these safeguards, the information stored about each servo motor is used to provide both limits and a scaled

linear interpolation to each of the outputs. This allows direct control sliders to appear to move unrestrained, with the control interface outputting scaled values in their place.

While this was desirable, a greater benefit is when standardisation was implemented with scripted or automated control. Under this scenario, a user generated script, or automated action is able to be calculated from an input, and mapped to any output, so that scripts that are written for certain puppets, are not restricted because of that puppets servo boundaries, but rather is able to be scaled and mapped to equivalent servos on another puppet. This ability to make scripted routines transferable greatly reduces the overhead when generating scripts, and increases the portability and functionality of scripts, while reducing the script generation complexity. This is seen when using the script generation section of the application, discussed in section 7.1.

5.4.3 RESTRICTING INVALID/UNWANTED USER OPERATIONS

With each control method there is the possibility for a user to attempt to perform a prohibited operation, such as attempting to move a controller outside its boundary position, or entering values that are not able to be interpreted effectively. These types of operation are eliminated through the control method, with each ensuring that it will filter all of the provide inputs to ensure a valid output. This is facilitated using scale functions that are able to test value bounds and provide linear interpolation between inputs and outputs. Using these function ensures that all values taken by any control method will remain within the scope of the desired output.

The remaining features of the control interface allow users to configure, test and secure controllers and the application to allow safe, efficient use of multiple controllers. This now provides the basis for the development of any number of effective control methods for effecting change in the controllers' outputs. The control methods designed from this point need only manipulate sections of code capable of setting the required output data.

With this section of the GUI complete, users are able to maintain and access each connected controller, and so all that remains is to create specific control methods for

obtaining position data in time. Three such control methods were implemented, and integrated into the existing GUI, and in the following chapters each of these is discussed along with the major applications for each. While each of the implemented control methods does provide different functionality, and require different methods of operation, each conforms to the generic attributes required of a suitable control method that were outlined in section 5.4. This ensures that each of these control methods are capable of performing as necessary.

6 DIRECT CONTROL

The direct control module is designed to provide highly accurate servo control, with each servo having its position set by user interaction in the GUI. This method of control provides users with the ability to test the outputs of many servos without any complex procedures. All inputs are restricted for each output by allowing the user to only vary the inputs by the allowed amount. This is useful for testing, interactive displays, or any situation where the required positions are not known beforehand.

6.1 DIRECT CONTROL METHOD

Designed to be the first point of interaction for new users of the controller and application, this method had to maintain as much functionality as possible while still maintaining a clear, easily operable appearance, not getting bogged down in options and settings for control. A first impression to the product must be clean, intuitive and reliable, giving the user a sense of control and accomplishment when executed correctly. It is with the desire to provide these things, that the most rudimentary control method was created.

To provide the base data collection function in direct control, the application takes the values of a number of servo position sliders and sends them to the controller. The boundaries of the sliders are set to correspond with those of the servo that it controls. This prohibits a user from being able to manipulate the servo beyond the preconfigured boundary values, which are initially set to the limits of the controller ability. This method of control is provided on the controller interaction screen and is readily available at any time to test the manipulation of the controller within boundaries.

While very powerful, the presentability and desire for ongoing use of this type of control, labelled 'Direct Control and Sliders Information Frame' in Figure 5.2, is limited. Feedback provided to the user is not excessive, and there is little ability to make an interactive control method without interfacing other forms of hardware. This however does not detract from the intention of the method, since all its requirements, including intuitive control and a base control method that is very stable and requires very little in the way of processing power or configuration.

The screen displays the servo number, textual movement type and numeric position between 0 and 255 to the user, but cannot provide information or movement for more than one servo position at a time, since only a single slider is selected at any instant in time. As a result, this method of control can be rather arduous on systems requiring large numbers of servo movements, and so specific interactive controller interfaces can be designed to run dedicated controllers more effectively. One example of this is included with the touch screen interface.

6.2 TOUCH SCREEN INTERFACE

The current touch screen interface is designed specifically for presentation purposes, and gives the user the ability to move up to four servo's via a click and drag interface. This particular touch screen was designed for a puppet type head, and so provides the user with a two axis region in which two servos can be manipulated by dragging a pointer from top to bottom and left to right. In addition to this, two additional large vertical sliders have been provided for any other servo outputs that may be required. In display scenarios, the two axis pointer was used to control the eyes of a puppet, while the sliders controlled a mouth and eyebrows respectively via a touch screen, so that no mouse or keyboard were required. The final screen is shown in Figure 6.1.

This type of control is simply an extension of the standard direct control interface, and can be designed for application with any controller, providing simplified, well featured control methods for use by controllers with specific requirements. The ability to create

these control environments is currently restricted solely to programmers, requiring complete recompilation of the application, and a working knowledge of the code that must be written. This may prove to be too hard for ongoing development, and should large numbers of specialized control situations be required, an in application method for creating these interfaces may be generated.

Overall, the direct control methods and the subset of touch screen control is a working model on which to base more complex control methods. While not requiring a large amount of processing power, this is the most robust of the control methods, allowing the controller to be fully functional on low power PC's still maintaining compatibility with the Windows XP operating system. This however is not intended for advanced users or those wanting a more full featured experience, and so more advanced control methods are included.

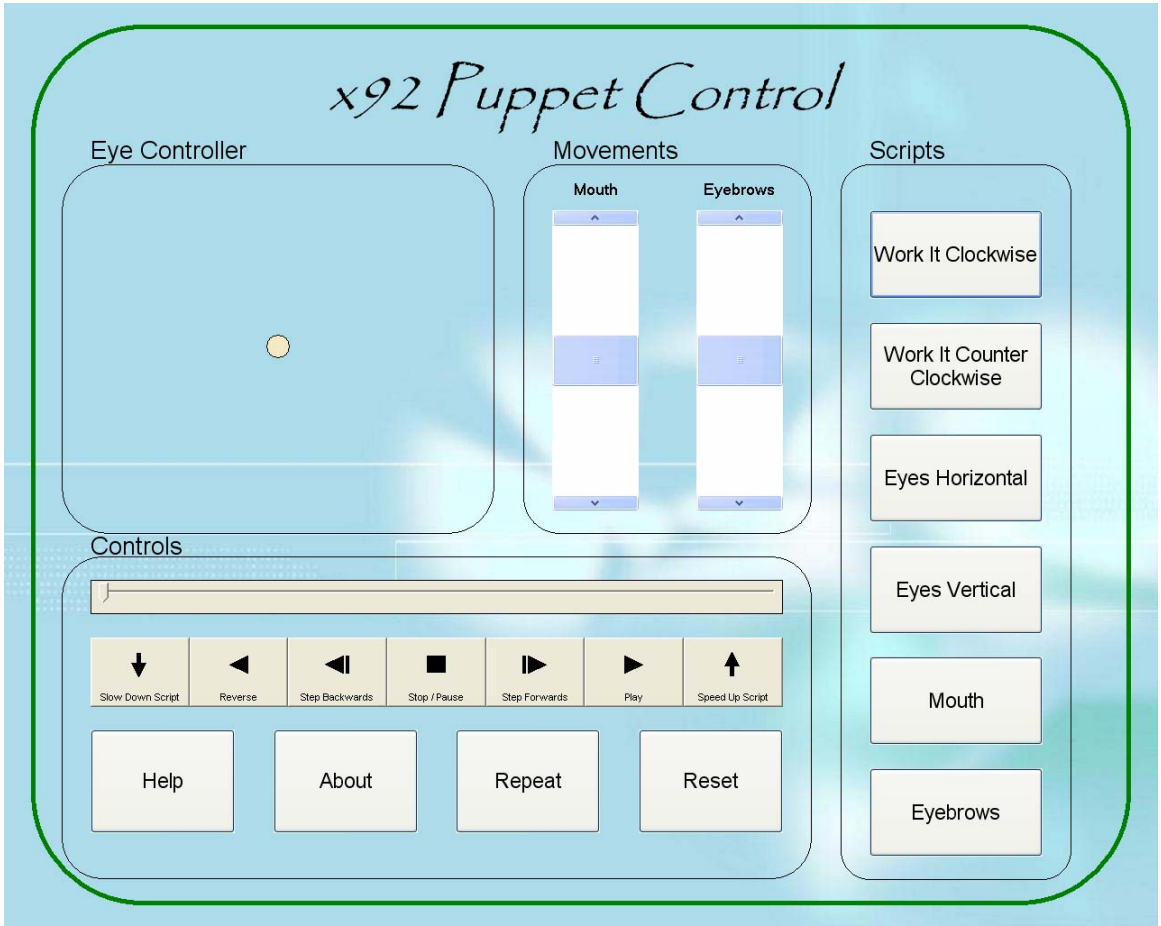


Figure 6.1 – Touch Screen Control Interface

This touch screen interface was used when the puppet was demonstrated in several presentations. These included an animation exhibition in Perth, as well as during the Open Day for Curtin University. This demonstrates only a small amount of the overall functionality of the controller, although the touch screen was demonstrated in a simple enough fashion that people were able to understand, and interact with the puppet without instruction. This indicated that the concept of the direct control method was successful in its attempt to remain as ser friendly as possible, and provide a logical input response that could be attempted by anyone.

As well as testing the ease of use of this controller, the exhibitions allowed several other aspects of the controller to be put to the test, including stability, reliability and particularly operation restrictions, with the control algorithm having to apply suitable boundaries on servo outputs to keep the servos in the puppet within operating tolerances.

After creating the direct control module, the next logical step in control algorithms was to create a module that would allow users to pre-program there inputs. This still requires input from a user, but once set, is capable of performing a predefined sequence of outputs without any user interaction. This type of control allows users to create a complete routine, and then when required accurately reproduce the same output at the touch of a button. The implementation of this is built on a similar principal to that of direct control, except that when running, inputs are generated by an additional program function.

7 SCRIPTED CONTROL

The idea of scripted control is to cause the controller to move a number of connected servos simultaneously in a predetermined pattern. While this concept was relatively simple, providing accurate positioning and timing for a puppet with up to 27 concurrent channels required a structured approach to be taken. Scripts must have a rigid structure, and all information from the scripts must be formatted and presented correctly upon generation so that interpretation is less complex. To increase compatibility and simplicity, comma separate value (CSV) files were used to store the script data. This type of control is able to be used for displays with either no or restricted interactivity, as well as performing tasks that are known and can be programmed and repeated large numbers of times.

7.1 CREATING AND EDITING SCRIPTS

The software provides an easy and intuitive way to create new scripts and edit existing scripts. As creating a new script is equivalent to editing a blank script, the same interface is used for each. The interface displays the motion profiles for multiple servos, each in a different colour, allowing the various movements within a script to be easily synchronised. The user can choose which servos are displayed at any time, reducing clutter and allowing a sub set of servos within the script to be edited separately.

The motion profiles for the servos are shown as a position versus time graph, with the servo motor that is currently being edited having visible markers at each of the defined key frames. This allows the user to quickly see which servo is being edited, and where the key frames are positioned. The positions of the servos at any point in time are calculated using linear interpolation between the key frames to either side.

When creating a new script, the key frames do not have to be created in chronological order, but can be inserted at any point within the position versus time graph. This allows the user to go back and adjust portions of the motion profile that they are not completely satisfied with. The servo position can be modified for an existing key frame by clicking a new location close, with respect to time, to the existing key frame. To assist the user, the cursor changes to a vertical set of arrows, indicating that they can click to modify the position of the existing key frame. Key frames can also be deleted by 'right clicking' the unwanted point, removing it from the motion profile.

Once the user is satisfied with the script they have been editing or just created, they can save it to a file, allowing it to be used within any sequence for any animatronic puppet as desired. Figure 7.1 below shows an example of the script editing interface.

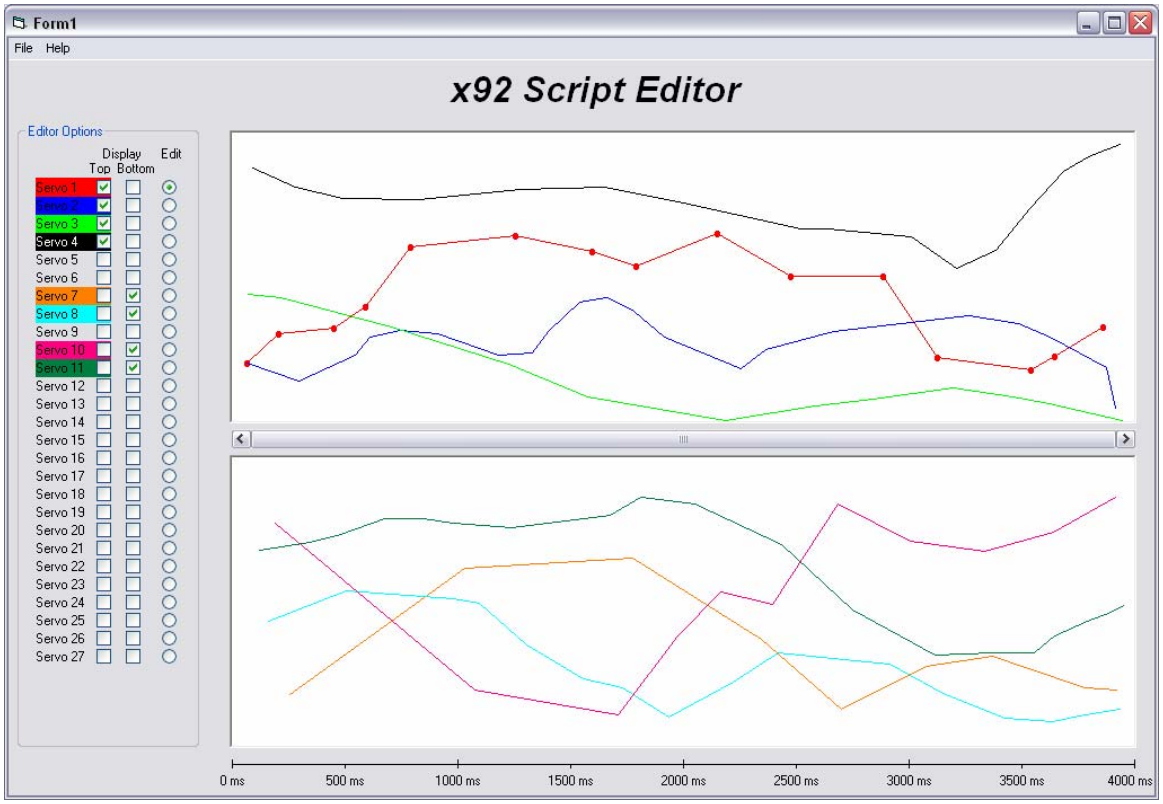


Figure 7.1 - Script Editing Interface

As can be seen in the above screenshot, the user has two main areas for displaying the scripts' motion profiles. The user can choose which servos they would like to display, and which area to display them in. This allows for grouping sets of related servos, while still ensuring the display area does not become over crowded. The colour of each servo is also displayed, allowing easy recognition of the motion profiles of the various servo motors.

7.2 STORAGE OF SCRIPTS

Scripts, once created, are stored using a specific structure that is interpreted by the application when re-loading them. The data contained within each script is grouped in two methods. Firstly, all data has line delimited, so that as the script is read, each line feed with carriage return character will advance the method of interpretation of data on that line. After a specific line is loaded, the data contained within that line is comma separated, using the ',' character to separate numeric variables on that line. This allows a number of related numeric variables to be grouped on a single line, for instance a script servo reference number, a script time, and a position. In addition to this, the script interpreter allows for script comments, added by simply providing a '%' symbol as the first character on the line. This will cause the script interpreter to discard that line, and simply continue interpreting the data on the following line. A standard layout of scripts is given in Table 7.1.

Section	N°. of lines	Relevant Data
1	1	Number of servos used in the script – `num_servos`
2	1	Single line comment about script
3	`num_servos`	Servo mapping information, comma separated values: `script_servo_reference`, `movement_type`, `movement_index`
4	variant	Servo position information, comma separated values `script_servo_reference`, `time_from_start`, `position` (0 - 255)

Table 7.1 – Script Information and Layout

Using this format, and with the ability to add comments within the script, both users and the application is able to readily access the information contained within the script. The files stored with this type of information are currently given the extension x92, however, additional methods of script interpreting can be implemented and given alternate extensions, since the ability to select alternate file types is provided, as discussed in section 7.4.1.

7.3 LOADING OF SCRIPTS

Scripts are loaded within the GUI, and mapped to a selected controller. The concept of scripting is simply that each servo attached to the system would have a specific position at a specific time, and so each controller structure has subsequent servo structures, with each servo structure possessing all the required information about the individual servo, as well a time and position array, with each pair of points in these arrays representing a position for the servo to be at, and a time for it to be there. A single function was created that could be passed a script filename and a controller structure, and from that is would append the position and time arrays for each servo with those dictated by the script.

7.3.1 AUTOMATIC SERVO MAPPING VIA MOVEMENT ID AND MOVEMENT INDEX

The GUI is capable of interpreting a script, and mapping outputs and inputs with matching `movement_type` and `movement_index`. This means that when loading scripts for selected puppets, the user is not even made aware that mapping the inputs and outputs is necessary.

The added benefit of this is that scripts written for one controller can be transferred easily to another controller with similar features, with all boundaries being set accurately, and each respective input controlling the expected output. This ability to channel the correct information to the correct servo is crucial to providing a method of control that is simplistic and powerful. This functionality was made possible because of the implementation of the E²PROM onboard storage on the controller, and so it is using this information, combined with the script reference information stored in the x92 script structure, whose sections are outlined in Table 7.1. The script mapping is applied with each servo in the script representing a distinct input, with a unique movement ID and movement index, read from the script mapping data (section 3). Each of the time and position data pairs contained in the script body (section 4) are loaded into servos attached to the selected controller with identical movement ID's and movement indexes. This is shown in Figure 7.2.

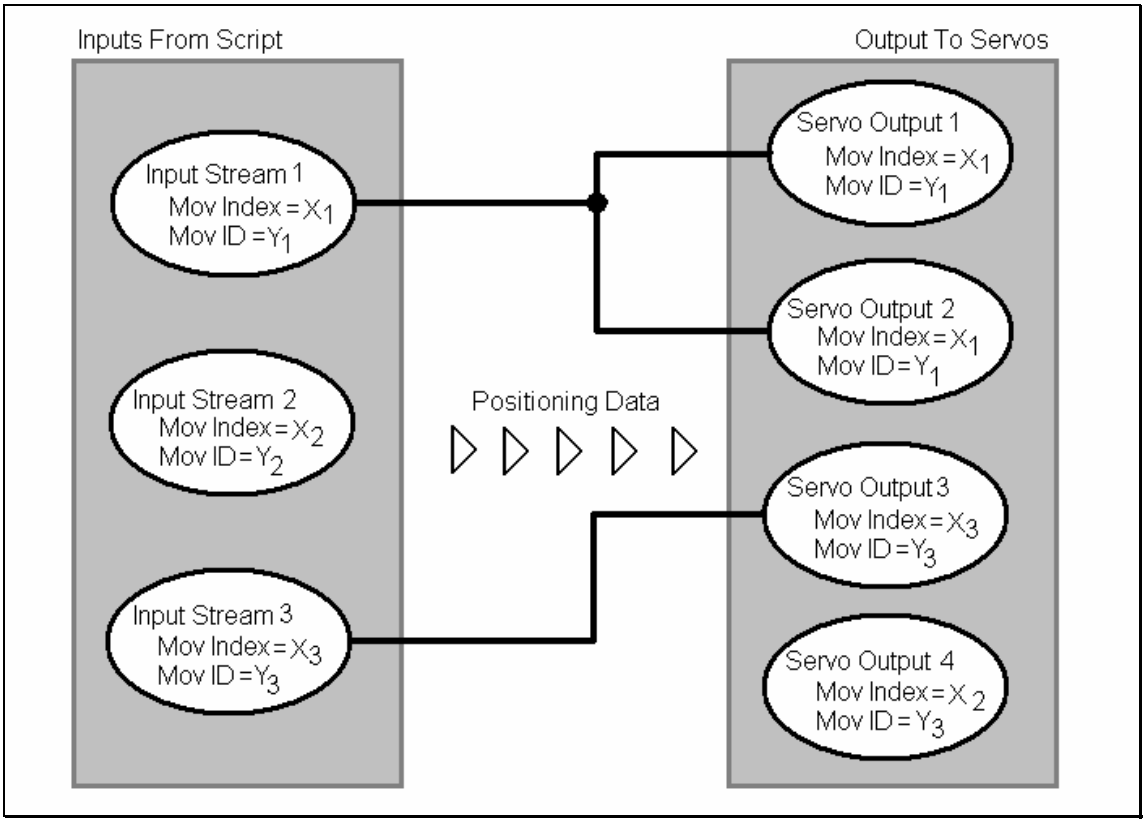


Figure 7.2 – Distinct Script Mapping

It can also be seen that any given input will not map to an output with a different movement ID or movement index, allowing the largest possible number of mapping variations, however, if more than a single servo attached to the controller has identical movement ID's and movement indexes, a script with the same information will map to all associated outputs. This could be useful when correlating movements across a number of similar attributes within a puppet, or matching movements between multiple puppets connected to a single controller.

7.3.2 HANDLING ERRORS IN STORED SCRIPTS

With the ability to create and store scripts within the program, the likelihood of encountering errors is reduced. There is however the possibility that scripts are generated by hand, and so several checks are put in place to ensure that when interpreting a script, values that are clearly incorrect are filtered and the user notified to

correct the problem, as well as the filtering of blank lines and other anomalies in script files.

In order to provide a more robust script, the option of adding commented lines to the script is included. All lines beginning with a % symbol are used to identify them as comments. This allows users who are generating scripts by hand, or editing scripts that were made within the application to add comments allowing them to record information about the function of each section of the script.

7.4 AMALGAMATION OF SCRIPTS

To increase the versatility of the scripting module, the amalgamation of scripts was implemented through simple selection criteria. This leaves the user free to generate a number of small scripts, each having a small effect, and then selecting to run these scripts concurrently, providing a succinct progression with the appearance of a single, more advanced script. This is a similar concept to motion studies, where complex actions are taken and broken down into there simplest movements, and each of these movements can be analysed and perfected. This could be useful in both generation complex actions, or in script generation, where a number of people can each work on a part of a controllers output, and then amalgamate these movements to cause a larger complex output.

7.4.1 SELECTION OF STORED SCRIPTS

To select scripts, the files in the script directory, set in the system options covered in section 5.3, are loaded into a FileBox, and then filtered for the desired file types. The types of files able to be loaded are predetermined, with each representing a different type of input. Currently, the only file type used is line delimited CSV files, and are recorded using the x92 script structure, and so the x92 file suffix is used. File types are selected using the combo box labelled ‘Current File Type’ in Figure 7.3.

The desired scripts are then selected in the FileBox and added to the selected scripts box, showing not only the scripts that are selected, but also the order of the scripts progression through time, should more than one be selected. Should users wish to perform the same action a number of times, it is also permissible to include the same script multiple times, with each having the same set of actions, but performed at different times. The selected scripts and their order are shown in the list labelled 'Chosen Scripts' in Figure 7.3.

7.4.2 ORDERING OF SELECTED SCRIPTS

Scripts added to the list are added after all of the scripts already chosen. Once added, scripts can be moved up and down the list with the selected buttons, and can be removed from any point in the list should they no longer be required. Upon acceptance of a desired set of scripts, they are loaded into the controller structure under each of the servo structures from top to bottom. This means that the script at the top of the selected script box will be the first to execute, and the controller will proceed downward through the actions of each of the available scripts until it reaches the bottom script.

This progression allows users to view what actions will be taken and in what order, so that upon compilation of multiple scripts, a logical output is achieved. In addition, should a change in the scripts be required, the user can return to the script selection screen and the previous selection of scripts will be automatically loaded into the selected script box, allowing for simple editing.

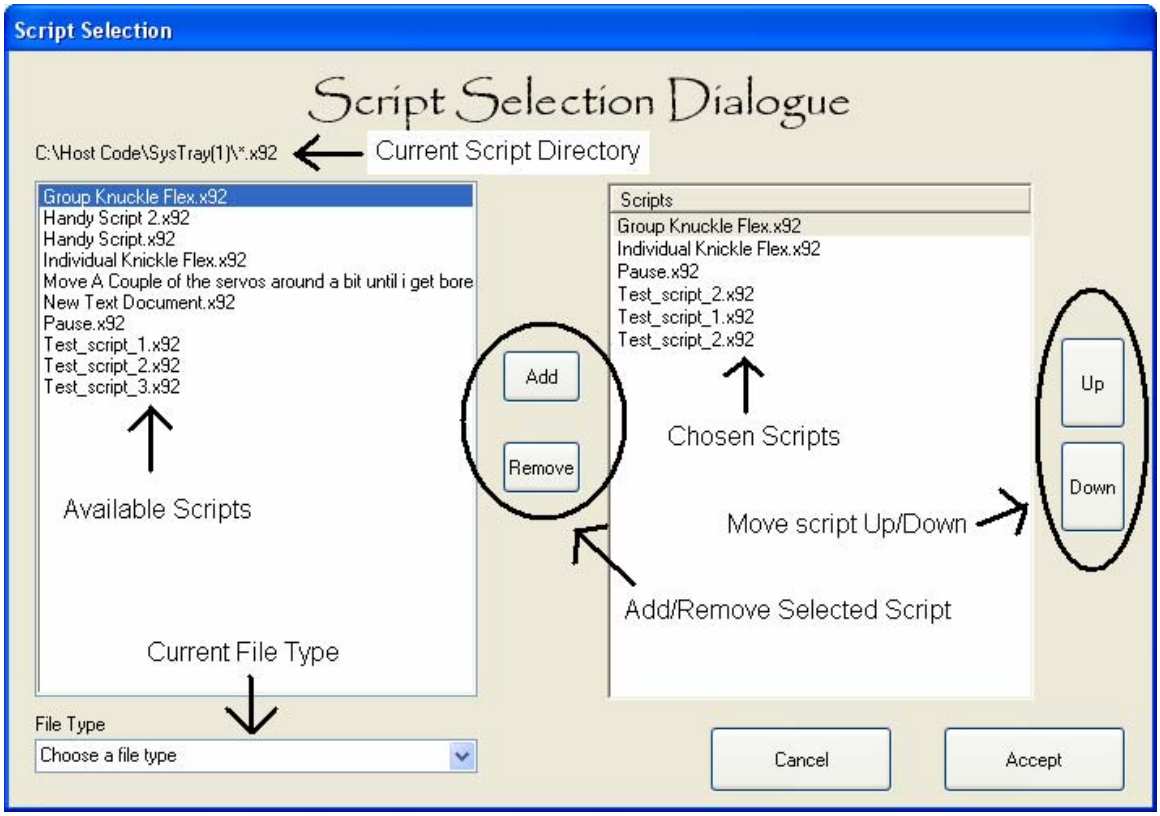


Figure 7.3 – Script Selection Dialogue

7.5 TESTING CONDITIONS AND REQUIREMENTS

Scripts generated using the script generation functionality of the application should be able to map to any controller with the script input channels having the same movement type and movement index being mapped to all servos having the same information. Scripts must append each previous script in the script selection box, and not overwrite its information, with the completion time of the previous script becoming the starting time of the next. In this fashion the controller will proceed through the script values loaded into the control structure as though each in turn are part of the same set of actions, while maintaining accurate timing.

While loading scripts the software filters lines commented with a %, blank lines, invalid or non-numeric information and is capable of interpreting any modular number of input

channels defined by the script, as well as viewing the script comment and its mapping information.

Both the direct and scripted control modules created require specific user input to define the movement of any given output. While this provides a robust interface for known movements, or movement with a user present to modify the output, it is desirable to have a control interface that once configured, is capable of interpreting a given input and responding accordingly. Since one of the more versatile senses in terms of feedback and response is sight, a streaming video feedback control system is implemented to provide the controller with a completely autonomous method of response. This control method, once configured, completely removes the user from the control system, and is capable of responding to the preconfigured stimulus to deliver autonomous responses. This method of control, while being more computer intensive than the other control methods, delivers a third, highly versatile method of control capable of greatly increasing the abilities of the controller.

8 INTELLIGENT VISION CONTROL

The intelligent vision control system is another method that can be used to control the animatronic puppets. Through the use of a web camera, the puppet can be made to respond to objects that the camera sees. This system allows a high degree of interactivity between the animatronic and its surrounding environment.

This method of control requires a video capture device to be connected to the host computer, allowing the visual stimulus to be processed within the software. The location of an object detected within the video frame is calculated and used to determine the position of two servo motors. These motors would generally be used to control the movement of a puppet's eyes in the x and y axis, but can be used for any purpose the puppet designer sees fit.

For the purposes of this project, an orange tennis ball was used as the object to be detected. The bright orange colour of the ball was chosen to reduce the likelihood of background objects having the same colour, thereby ensuring the ball would be distinguishable.

As well as being able to use the vision system to control the puppets directly, it can also be used to record script files for use at a later date. This is an extremely easy way to create the scripts for more complex movements, such as getting the eyes of a puppet to follow a particular path. This is discussed further in section 8.6.

8.1 INTERACTION WITH ENVIRONMENT

By including the intelligent visual control, the system is able to interact with its environment in ways that are otherwise not possible. The puppet can automatically

track an object, such as a person or vehicle, with no operator intervention. This is a feature that is very useful in many potential scenarios such as interactive displays in shopping centres, amusement parks, exhibitions, etc. The visual control system is not just limited to controlling puppets in displays, but can be extended into other areas such as surveillance, where security cameras may be required to track people within its vicinity.

8.2 CONNECTING TO THE VIDEO SOURCE

The intelligent visual control system is not limited to just the use of web cameras as the source of visual input, as any device that has 'Video for Windows' (VFW) drivers can be used. This includes USB web cameras, video capture cards, as well as many other devices. By allowing the user to choose the video capture device that is to be used, some potential limitations of web cameras can be overcome. As web cameras use USB, they are limited to a maximum permissible cable length of five metres. This places restrictions on the placement of the camera in relation to the computer, which may limit the potential usefulness of the feature.

By allowing the use of other devices such as home video cameras connected to the computer through a capture card, the camera can be placed much further than five metres from the computer. By taking advantage of the use of standard video inputs, a VCR or DVD player can be used as the source for the video capture. This would allow pre-recorded video footage to be used to control the animatronic puppet.

8.2.1 EXTERNAL WEB CAMERA STIMULUS

The web camera used during the development of this system was a Logitech QuickCam Messenger. It was chosen as it is inexpensive, easily available web camera that is comparable with the majority of web cameras people currently own. The decision to use an inexpensive camera as opposed to a high quality professional video camera was based on several key factors. These include the target audience of the system, the cost

of the device, the required size of the video frame, and the quality of the video receive from the device.

The target audience and the cost of the device are somewhat linked together. The system is designed to be a viable option as both a professional puppet control system, while still being suitable for use in a home/hobby environment. If a professional level video camera was used, it would not be economically viable for a large number within the target audience due to the high costs associated with such cameras.

The minimum size of the video frame required is determined by the resolution of the available for the servo motors. As the positions of the motors are controlled by an 8-bit number, there are only 256 positions the motor can ever be. Based on this information, the minimum size of the video frame would be approximately 256 by 256 pixels. The frame sizes on most cameras are based on the VGA resolution of 640 by 480, with multiples and fractions of this normally available. As a result, the use of a frame 320 by 240 pixels is used. This frame size is used as it is available on all web cameras, and is only slightly smaller than the optimal 256 pixel height. This difference does not affect the system in a detrimental way, as the position of the calculated object is from a larger range than the range of servo positions when the puppet limits are taken into account. As such, this video frame size presents a viable option.

The quality of the video frames obtained from the camera is very important, as a noisy image can make the process of identifying an object more complex. The quality of the professional cameras is definitely superior to web cameras, but this is not to say that a web camera is not suitable. As long as a web camera can supply images with minimal noise, there is no reason they cannot be used. After considering all of these factors, the decision was made to use a web camera during development. To cater for sources that do supply a noisy image, pre-processing of the frame is included. All of the images within this section relating to the object tracking are from the Logitech QuickCam Messenger web camera used during development.

8.2.2 CHOOSING A METHOD TO INTERFACE WITH THE CAMERA

The main considerations when deciding the method to be used to access the camera include the ease of use for both the user and the developer, operating system support for the method(s) used, and the features provided by the method(s) used. Some of the options that were considered were the Logitech QuickCam API, 'Video for Windows' interface, and DirectShow (a component of DirectX).

The Logitech QuickCam API is provided by Logitech, and allows full access and control over all QuickCam web cameras. Its main benefits are programmatic control of all camera settings, and support for all QuickCam cameras. This method does have some very limiting drawbacks though, including support of web cameras manufactured by Logitech only. This would result in the user having to purchase specific equipment to be used for this software, and eliminates the possibility of using other capture devices such as video capture cards and standard video cameras.

Video for Windows (VFW) adds support to the Windows environment for a very large range of video capture hardware, allowing the majority of video capture devices to be used. It has been built into the windows operating system since Windows 95 as the dynamic link library (DLL) 'msvfw32.dll' and 'avicap.dll'. As it is still available in the current Windows XP operating system, alternate methods of video capture do not have to be used for various versions of Windows. The ease of implementation using the AVICap DLL is this methods major selling point. It provides a simple method of accessing the video frame buffer and allowing a preview to be displayed on the screen directly through hardware, removing the need for the software to display each frame manually.

The downside to the VFW interface is the lack of programmatic access to the various settings of the capture device, including brightness, contrast, and white balance control. It also is an aging system, with Microsoft ceasing further development of this API for

the latest release of Windows. Even though Microsoft has ceased development of this API, it is still a widely documented and used method of video access, and as such is still an accepted method.

The final method that was considered was using the DirectShow component of DirectX. This method allows the greatest flexibility in terms of capturing and processing the image. It uses a method of custom filters that can be cascaded to perform a diverse range of functions. As with VFW, this method also supports automatic rendering of the video to screen. The major disadvantage of DirectShow is the complexity in creating the filters, and getting them to interface correctly, which can be a very complex process.

The decision was made to use the VFW interface, as it allowed for a diverse range of capture devices to be compatible with the software, while still providing a relatively simple method of access to the video frame buffers. The decision to use VFW instead of the more powerful DirectShow was not an easy one. In the end, the deciding factor was the complexity involved in programming the DirectShow filters, and the timeframe for development.

8.2.3 INITIALISING THE CAPTURE INTERFACE

To enable the software to utilise the web camera as an input, it must first be initialised. The initialisation includes creating a capture window, binding it to the correct driver, setting up a 'call-back' for the incoming frames, and setting the preview mode. These steps are explained below.

The first step is to create a capture window by using the `capCreateCaptureWindowA` command provided by AVICap. This creates a window for the display of the video with the specified properties, and returns the handle of the window, `lwndc`. The properties of the window can be set by passing the function various parameters. These properties include the visibility of the window, whether it is a child, the presence of a border, and if it is a fixed or movable window.

As the video is to appear to be embedded in this software, and not an external window, the window is created as a child of the main form. By removing the border and fixing its position within the main form, it appears to the user that the video is actually a part of the main form. This is the only method that can be used to embed the video within the software, as the capture window must have a unique handle to allow the AVICap interface to function.

Now that the capture window has been created, the call-back must be configured for the new frame event. This instructs the system to call the specified function every time a new frame is received from the camera. This can be achieved by calling the `capSetCallbackOnFrame` function, and passing it the handle to the capture window and a pointer to the frame processing function. The use of this call-back removes the need for the software to constantly poll the AVICap interface to determine when a new frame is ready to be processed, resulting in the complexity of the system to be reduced. It is analogous to an interrupt handler function, and is used to process each frame that is received from the camera.

On computer systems that cannot process the video frame in less time than the specified preview rate, it will automatically drop frames from the camera. This ensures that the call-back will not be called while it is still processing the previous frame. If this were to occur, the data from the previous frame would be pushed onto the stack, quickly filling the systems memory and crashing the system. The result of dropping frames to prevent this type of failure only has the effect of reducing the frame rate of the camera, allowing the software to be run on computers of varying power and computational speed.

With the capture window created and the call-back set, the capture window must be connected to the driver for the camera. This allows the capture window to actually receive the video data from the camera. By sending the `capDriverConnect` function the handle to the capture window and the driver index to connect to, the specified driver is bound to this capture window. The driver index can range from 0 to

9, with the lowest indexes being populated first. The order of the drivers is prioritised to ensure that 'Plug And Play' capture devices, such as web cameras, appear first. As a result, an index of 0 can be used as the default drive to bind to the capture window, with the user being able to modify this within a settings window. This allows the system to connect to the most likely driver, while still allowing the user full control of which driver is used.

The driver is not necessarily limited to being used with a particular capture device. For example, the default 'Microsoft WDM Image Capture (Win32)' driver used to connect to the Logitech QuickCam web camera, in Windows XP, can also be used to connect to many other image capture devices. This driver is actually a VFW to WDM wrapper for image capture devices that only support the WDM (Windows Device Manager) drivers. As Windows XP requires devices to support WDM drivers, this software can be used with the majority of image capture devices such as USB web cameras, all of which can be accessed through the default driver.

With the capture window successfully bound to the driver, the video capture mode must be configured. AVICap was primarily designed to capture video, and audio, data from an input device and encode it into an AVI format file. As we do not want to save the video to disk, but use it to find the location of an object in the cameras field of view, we must enable the preview functionality of the AVICap interface. The preview option sends a pointer to the video frame buffer to a user-defined function in the software. This buffer can be processed and modified, with the resulting contents of the buffer being displayed in the capture window configured previously.

The preview functionality can be enabled by sending `capPreview` the capture window handle and the 'true' Boolean parameter. The maximum preview rate, in milliseconds per frame, can be sent to the `capPreviewRate` function to limit the cameras maximum frame rate. As mentioned previously, the frame rate is a maximum only, as the camera will automatically lower its frame rate if the image analysis takes too long.

The last step is to scale the preview window to the desired size on screen. This stretches the video to fit the specified region with out affecting the video frame buffer at all. This allows a frame size to be used that is suitable for analysis, while still displaying it on screen at an appropriate size. In this software, the dimensions of the captured frames and the preview window are actually equal, as a size of 320 by 240 pixels was an adequate size when embedded within the main form. The scaling is set by calling the `capPreviewScale` function provided in the AVICap API. The code used for this initialisation of the web camera can be found on the CD attached to the back of this document.

8.3 Real Time Analysis Of The Video Stream

The process of tracking the location of an object can be divided into two main steps: finding the object, and calculating its position. A flowchart showing the process used within these steps can be seen below in Figure 8.1.

For each frame that is received from the camera, the object must first be identified, and then its location calculated. The object detection is based on a colour match algorithm, where the colour of each pixel in the image is compared to the target colour. If the pixel is similar in colour, it is considered part of the object. Once the pixels that belong to the object are identified, the objects location can be calculated by averaging all of the x coordinates and y coordinates. This process finds the centre of mass of the object, assuming each pixel is considered to be of equal mass, or weighting.

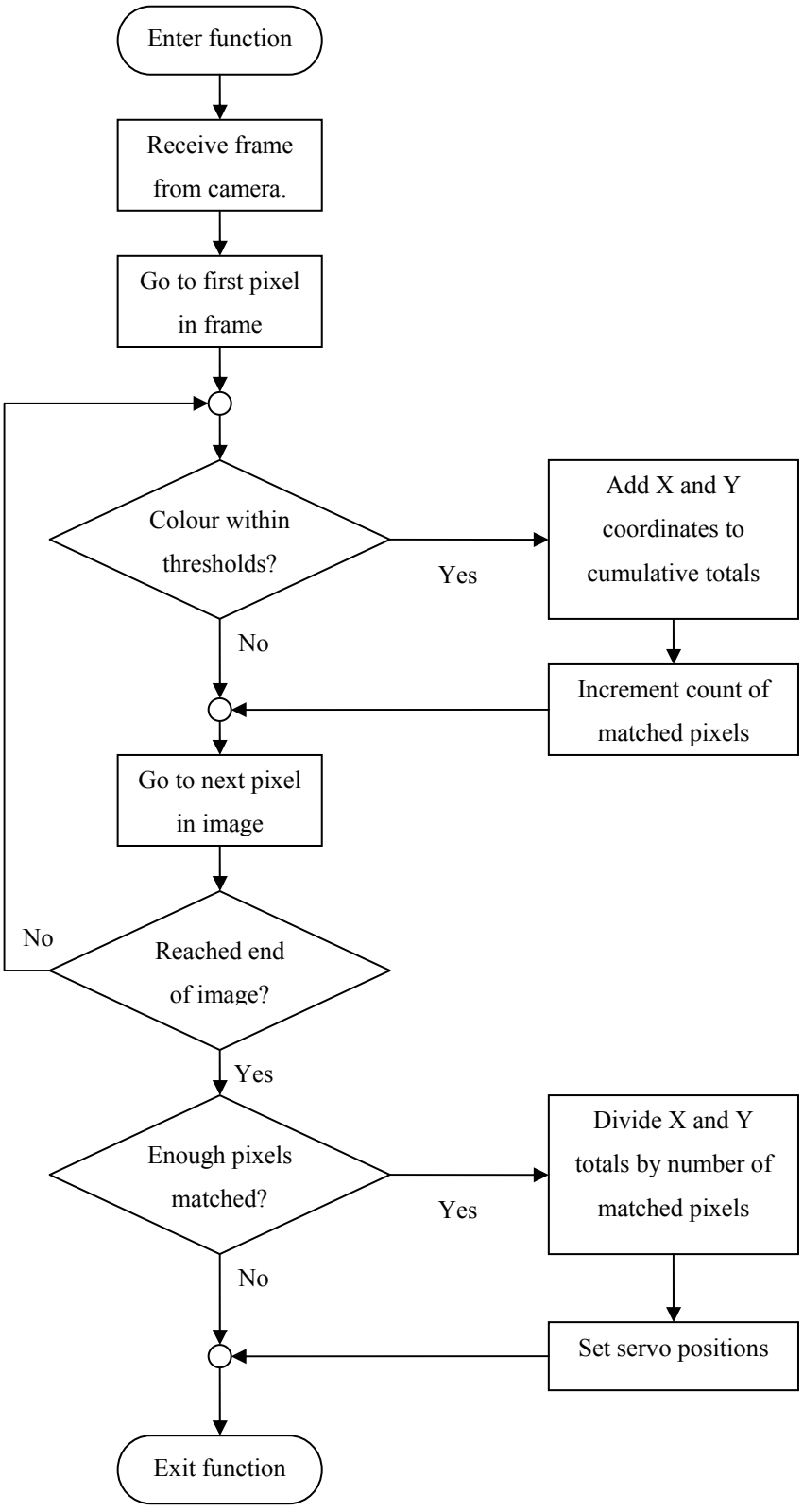


Figure 8.1 - Process Used To Detect An Object And Calculate Its Location

8.3.1 OBTAINING THE FRAME FROM THE WEB CAMERA

With the call-back functionality of the AVICap API, the custom function `MyFrameCallback` will be called every time a new frame is available from the camera. The system passes this function two parameters: the capture window handle, and a pointer to a video header structure (`VIDEOHDR`) containing the captured frame information. This structure contains the address of the video buffer, the length of this buffer, how many bytes of the buffer were used, as well as some other information not relevant to this application.

The frame buffer contains byte triplets containing the red, green and blue colour information for each pixel in the preview frame. The pixel data in the frame buffer is ordered from right to left, bottom to top, as you move down through the buffer. The order of the byte triplets containing the colour component, or sub-pixel, information is blue, green, and then red. This is the opposite of normal logic, as the majority of information in our lives is stored left to right, top to bottom. Figure 8.2 shows the coordinate system, within the image, used by VFW.

Figure 8.2 - Coordinate System Used By VFW Within The Image

The data in the buffer is stored in a one dimensional array starting at the location specified by `lpData` property of the video header, with the length frame data specified by the `dwBytesUsed` property. To be able to work with the data in the buffer, it must first be copied into an array the same size as the frame data. This is done be using the `RtlMoveMemory` subroutine accessible through the kernel32 API. This is a core function provide by the kernel of the Windows operating system.

By specifying the destination array as a three dimensional array, it allows the pixels to be addressed much easier than using a one dimensional array. The first dimension is the colour index of the sub-pixel, i.e. 0, 1 and 2 for blue, green and red respectively. The second and third dimensions are the y and x coordinates respectively. This is somewhat counterintuitive, as most 2D colour information is stored in the opposite order, and is to do with Microsoft's implementation of the VFW functionality. This is a more elegant solution than initially used, as the x-y coordinate does not have to first be converted to an index, thereby reducing the time taken to access the pixel colour data.

8.3.2 PRE-PROCESSING OF THE FRAME

The image received from the Logitech camera contains a considerable amount of noise. This noise can dramatically affect the results of the object detection, as demonstrated in Figure 8.3, when compared to an image that has been filtered before processing. The figure shows the results of the analysis, with black pixels being below the thresholds, white pixels being above the thresholds, and red pixels considered a match. (Note that not all pixels are analysed.) This noise is randomly distributed, producing significantly different values for some pixels compared to a noiseless image. This results in the calculated location of the object changing from frame to frame.

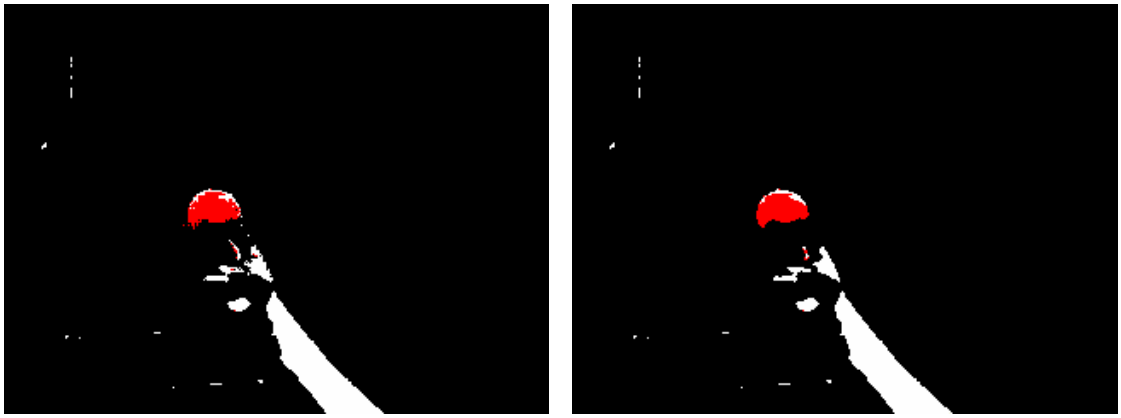


Figure 8.3 - Comparing Analysis With And Without Noise (3x3 Gaussian Filter Applied)

To eliminate the noise, a blur or softening filter must be applied to the image before analysis occurs. This is to ensure that erroneous pixels that are between the threshold values do not affect the result of the analysis. To apply this filter, a kernel is applied to each pixel in the image. The kernel is a weighting of the pixel in question, along with its neighbours, that is used to calculate the value of the pixel in the output image. It is often represented as a matrix with the relative weightings of each pixel in the various positions of the matrix. Shown below in Figure 8.4 is a sample kernel that averages a 3x3 pixel area for each pixel in an image.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Figure 8.4 - Sample 3x3 Kernel That Applies A Mean Filter On The Image

By combining information from multiple pixels into one output pixel, high frequency changes such as noise can be filtered out, at the cost of resolution. The affect on the resolution of the image is based on the size and severity of the filter used, and as such an acceptable compromise must be found between strength of the filter, resolution of the resulting image, and the processing time required to apply the filter.

The filter kernel is applied as a matrix convolution. The output image *y,* the result of the convolution of input image *g* with the kernel *h*, can be calculated using Equation 8.1, below. Note the filter matrix *h* has a width and height of 'M' in the equation below. This filter must be applied separately to each colour component, as they are independent of each other.

$$y[r,c] = \left[\frac{1}{\sum_{i,j} h[i,j]} \right] \cdot \sum_{j=0}^{M-1} \sum_{i=0}^{M-1} h[j,i] \cdot x[r-j, c-i]$$

Equation 8.1 - Gaussian Filter Using Matrix Convolution

The inclusion of the normalising factor in Equation 8.1 is to ensure the output values *y[r, c]* will always range between 0 and 255. The equation may look confusing and difficult to implement, but it is quite the opposite. A graphical representation of an example is shown in Figure 8.5. The value of the final pixel at (1, 1), in this example of applying Equation 8.1, can be calculated using the equation below. Note that pixel positions that are outside the image border are assigned a value of 0 for the purposes of the filter.

$$\text{Output}(1,1) = \frac{(2\times 0 + 9\times 0 + 4\times 0 + 7\times 0 + 5\times 17 + 3\times 24 + 6\times 0 + 1\times 23 + 8\times 5)}{(2+9+4+7+5+3+6+1+8)}$$

Equation 8.2 - Calculating The Output At (1, 1) For The Example In Figure 8.5

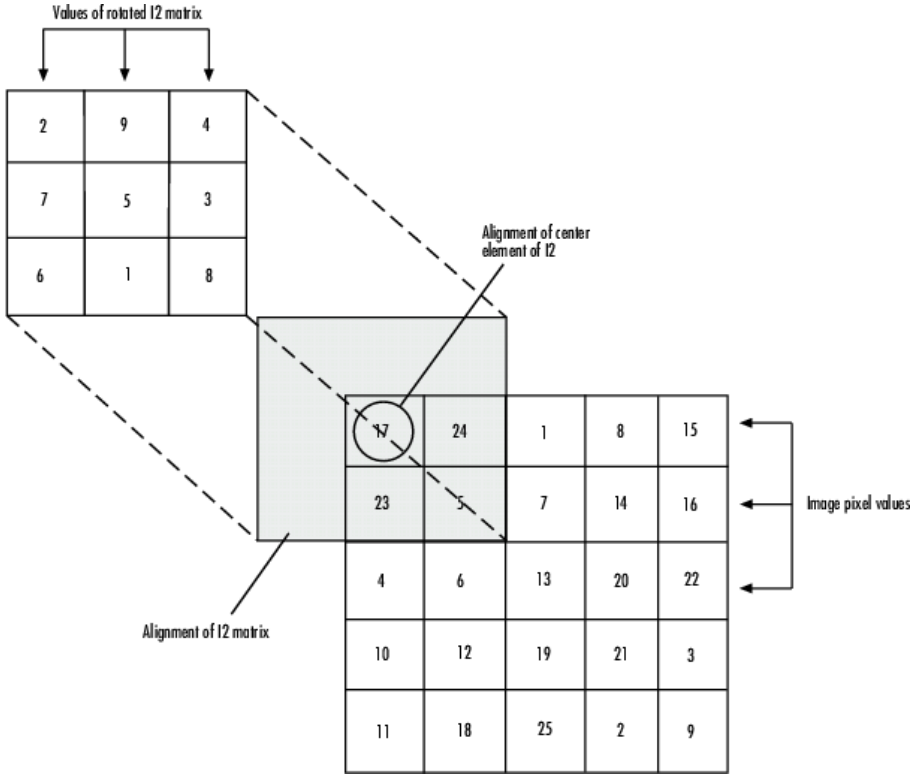


Figure 8.5 - Example Of Computing the (1, 1) Output of A Convolution

(Source: www.mathworks.com [15])

8.3.2.1 MEAN FILTERING

The mean filter is one of the simplest filters that can be used to smooth an image. It works by replacing each pixel with the mean value of its neighbours, including itself. Each neighbour is of equal weighting, and as such the coefficients in the kernel are always 1. Figure 8.4 shows the kernel for a mean filter with a width of three pixels. As the coefficients are always 1, the number of operations that occur per pixel is halved, as no multiplications are required. As a result of this, the mean filter can be applied to an image relatively quickly. As can be seen below in Figure 8.6, the mean filter can be used to remove some noise from an image.

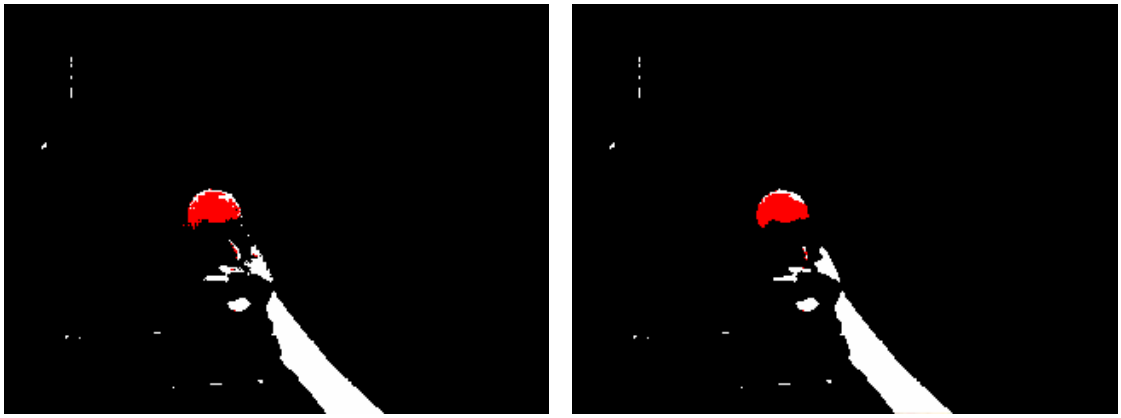


Figure 8.6 - Effect Of Applying A 3x3 Mean Filter

8.3.2.2 GAUSSIAN FILTERING

One of the more advanced filter's that performs this function is the Gaussian Blur. Its kernel is circularly symmetrical, resulting in it affecting lines and edges identically in all directions. The higher order filters result in a heavier blurred output image. The following kernel is used for a second order Gaussian filter. If this was applied in its current form, it would produce values greater than 255, so the output is divided by the sum of the coefficients, i.e. 16. This would result in 1/4 of the final value being comprised of the original pixel, 1/8 for each of the horizontal and vertical neighbours, and 1/16 of each of its diagonal neighbours. The result of applying this kernel to an image can be seen in Figure 8.8.

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & 1 \end{pmatrix}$$

Figure 8.7 - Second Order Gaussian Kernel

Figure 8.8 - Effect Of Applying A Second Order Gaussian Filter

When comparing the Gaussian and mean filters, an important characteristic are their frequency responses. Both the mean and Gaussian filters are considered low pass filters, but as can be seen in Figure 8.9 the Gaussian filter operates symmetrically irrespective of the direction of the frequency. This is not the case for the mean filter, as it has ripples parallel with the x and y axis. From these frequency response plots, it is clear that the Gaussian filter results in more uniform filtering, and will result in less noise remaining in the image after filtering.

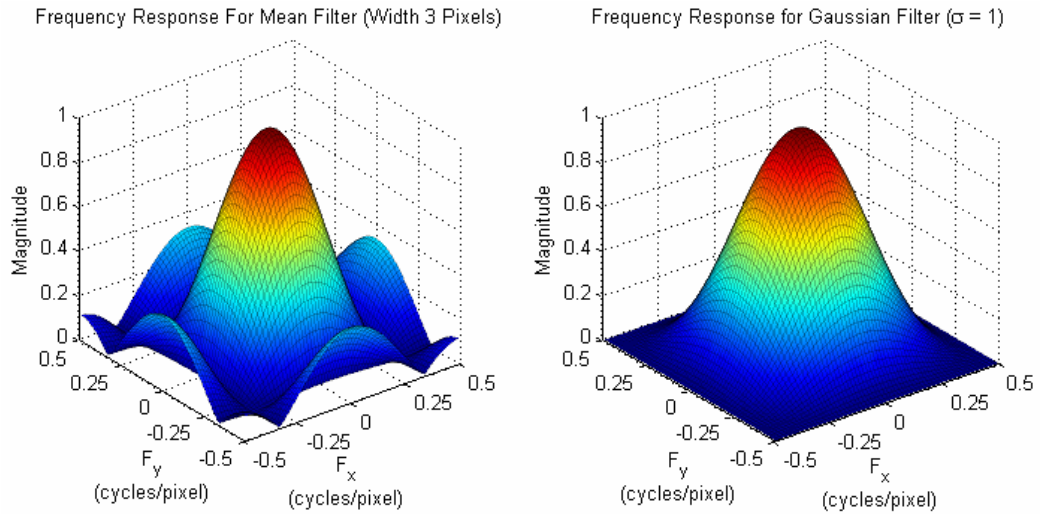


Figure 8.9 - Frequency Response of Mean and Gaussian Filters (Each 3 By 3 Pixels)

A useful characteristic of Gaussian kernels is that they can be separated, or decomposed, into the product of horizontal and vertical vectors. This can allow for

optimisation within code to reduce the time taken to apply the filter. The coefficients of the horizontal and vertical vectors can be found in the Table 8.1 (below). Note that the order of a filter is one higher than the index in the coefficient table below 0.

Index N	Coefficients	Sum of Coefficients = 2^N
0	1	1
1	1 1	2
2	1 2 1	4
3	1 3 3 1	8
4	1 4 6 4 1	16
5	1 5 10 10 5 1	32
6	1 6 15 20 15 6 1	64
7	1 7 21 35 35 21 7 1	128
8	1 8 28 56 70 56 28 8 1	256
9	1 9 36 84 126 126 84 36 9 1	512
10	1 10 45 120 210 252 210 120 45 10 1	1024
11	1 11 55 165 330 462 462 330 165 55 11 1	2048

Table 8.1 - Decomposed Gaussian Filter Coefficients

(Source: *An Introduction To Digital Image Processing*)

To optimise the code for the Gaussian filter, the vertical and horizontal vectors must be applied separately. This results in a reduction in the number of memory fetches that occur while applying the filter. The horizontal vector of the decomposed kernel is applied first, with the result stored in an intermediate image. The vertical vector is then applied to this intermediate image, with the output being the final image.

It can be seen that by decomposing the matrix and applying its horizontal and vertical vectors separately, the number of memory fetches is reduced from $2N^2$ to $4N$ for an N^{th} order filter (Note: the factor of two is due to multiplying by the coefficient). For example, for second orders filters it would reduce from 9 to 6 memory accesses. For a third order it would reduce from 16 to 8, and a firth order would reduce from 25 to 10. From these figures it is obvious that for higher order systems (i.e. larger kernels), the execution time can be improved dramatically by decomposing the matrix and applying its horizontal and vertical vectors separately.

The size of the kernel that is required is highly dependent of the level of noise that is present in the image. For images with a low signal to noise ratio, a larger filter is required, whereas images with high signal to noise ratios require less filtering. For the Gaussian filter, larger kernels significantly increase the processing power required to maintain a particular video frame rate. As a result, the smallest effective kernel should be used for the given environment. A comparison of various Gaussian kernel sizes can be seen in Figure 8.10 below. As you can see, the larger kernel sizes result in an image that is more blurred. Although this removes more noise, the resulting image will produce a lower precision calculation of the objects position. Therefore, as mentioned previously, the smallest kernel that still removes most noise should be used. In this system, a second order (3 x 3 kernel) Gaussian filter is used.

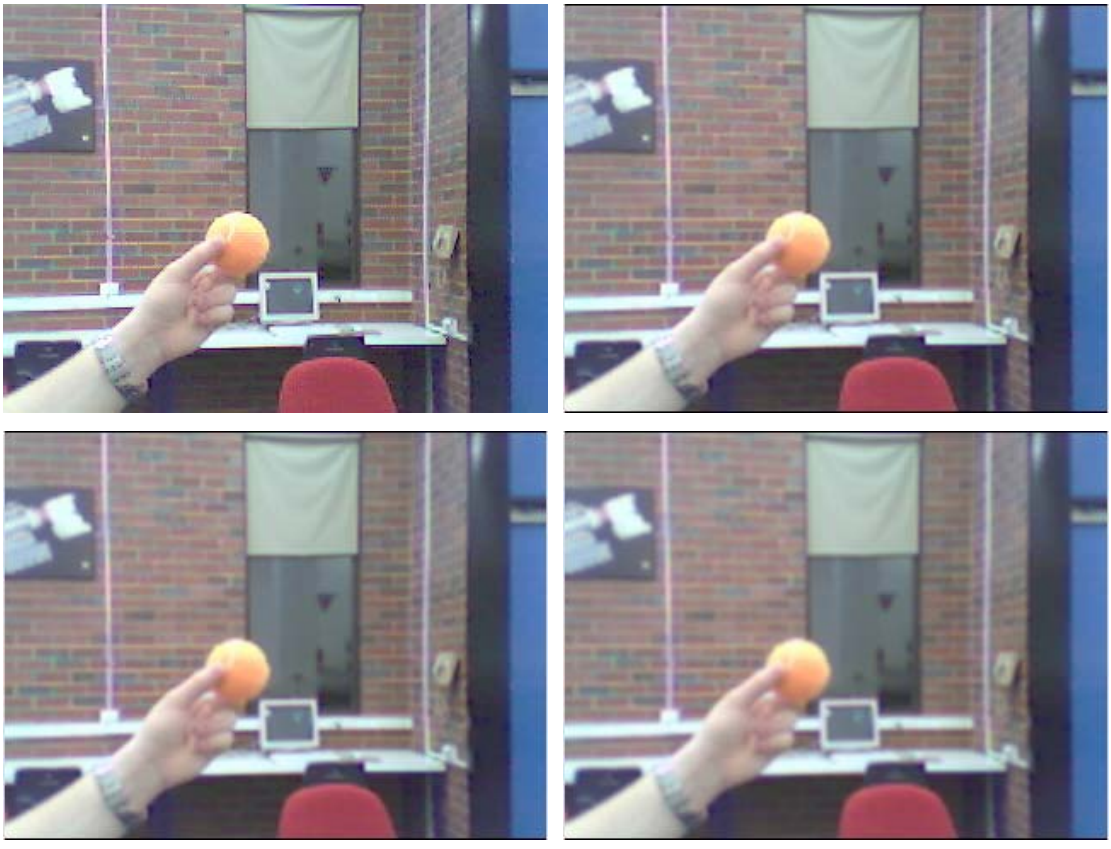


Figure 8.10 - Comparison Of Effects Of Various Gaussian Kernel Sizes.
From Left To Right: Original Image, 3x3, 5x5, 7x7 Kernels.

8.3.2.3 BACKGROUND SUBTRACTION

To enable the vision system to function correctly in many environments, a static background image can be subtracted from each frame from the camera. This will ensure that the software will not find the location of an object in the background that is not significant. This feature has is included within the software, but is not enabled by default. It is not enabled by default, as the user must ensure that they obtain an image to use as the background that does not contain any possible foreground, or trackable, objects. This feature can easily be re-enabled within the video analysis settings window when required. The result of the subtraction would be close to 0 if current pixel was very similar to the background pixel, would head towards -255 as the pixel got darker than the background, and head towards 255 as the pixel got lighter than the background. As you can see, this does not match the RGB24 colour space that requires each component to be between 0 and 255.

To convert the result of the subtraction into a compliant image format, two techniques can be used. The first is to scale the -255 to 255 range up to 0 to 255. This is done by dividing the result by two, then adding an offset of 128. The output image would appear grey (RGB [128,128,128]) in areas of the image that matched the background. This can make analysis of the resultant image more complex.

To overcome this problem, the absolute value of the result of the subtraction can be taken. This would result in all areas of the image that match the background being black, and all areas that are significantly different to the background would be very bright. This would allow a simple threshold to be applied to find all areas that are significantly different to the background image. These areas can then be processed further to find the object.

The other major issue of the background subtraction is the output image appears to be a double image in areas that are different to the background. This is demonstrated in Figure 8.11 below. If an image of just the differing areas is required, for example for a GUI, then a threshold can be applied to the subtraction result (using the absolute value

technique). If above a given threshold, the image is significantly different so the pixel from the original frame can be loaded into a new image. If below the threshold, then a black pixel can be loaded into this new image. This will produce an image with a black background and the differing areas will be from the true image. This image may also then be used as the input to the object detection. In Figure 8.11 you can see an example of blacking out the background.

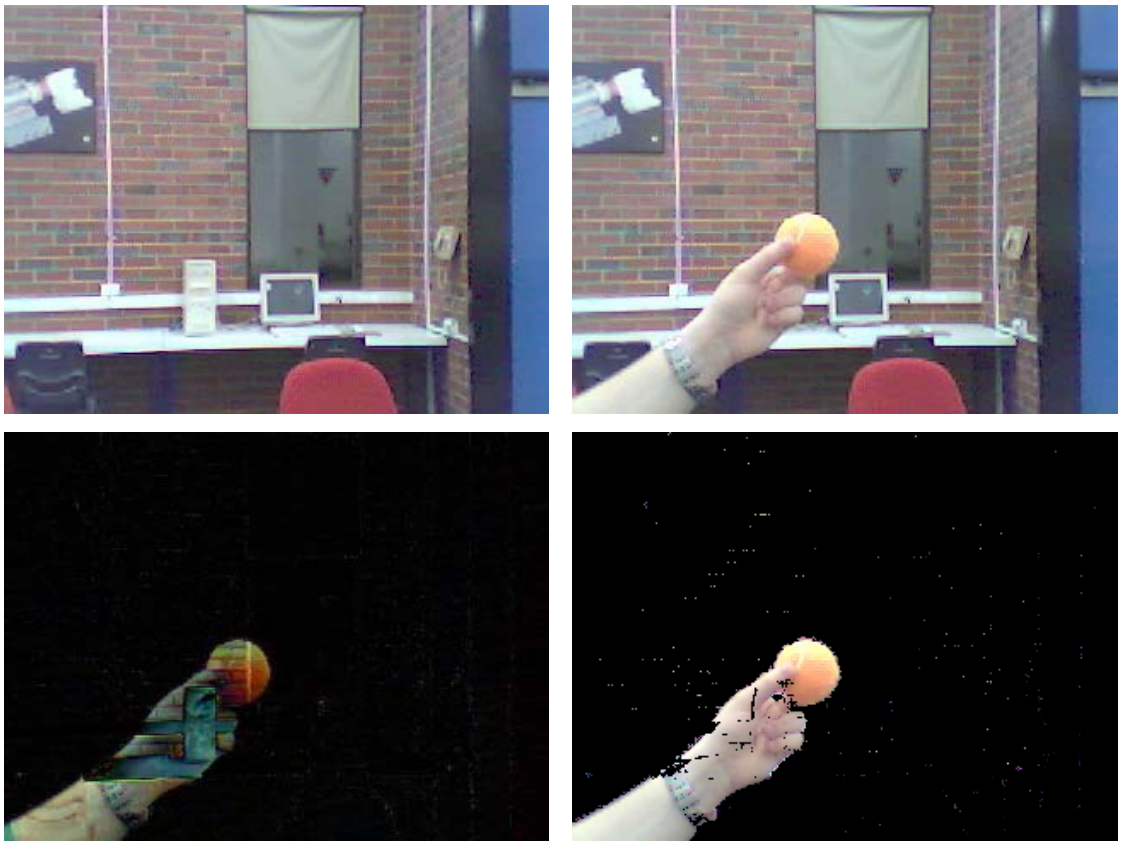


Figure 8.11 - Various Stages Of The Background Subtraction Process.
From Top Left: Background Image, Source Image, Absolute Different, and Foreground Objects.

Now that all static background areas have been effectively removed, the analysis can be performed. The thresholds used do not have to be set as precisely as normal, as most areas within the image have been removed. If the background has areas that are identical in colour to the object being tracked, they will not affect the analysis now that they are removed. This is very useful when being used in areas with visually 'busy' as long as the tracked object does not move in front of the identical coloured area. In this

case, the object would be considered part of the background and removed from the analysis.

This method of background subtraction is only useful for static backgrounds. If the background changes over time, for example trees moving in the breeze, those areas will not be removed before the analysis. This is not a problem in most scenarios, as long as the object to be detected is not an identical colour as the moving areas of the background. This is a future improvement that could be made within the system.

8.3.3 DETECTING THE OBJECT

Now that the image has been filtered to remove any noise, it can be analysed to find the location of an object. For the purposes of testing, an orange tennis ball was used as the object to be detected. The bright orange colour of the ball was chosen to reduce the likelihood of background objects having the same colour, and being mistaken for the ball.

Detecting the ball in the frame can be achieved using various methods including RGB thresholds or hue-saturation-brightness (HSB) thresholds. Both of these techniques are similar in that they look for pixels with colours in certain ranges.

8.3.3.1 RGB THRESHOLDS

This method detects the ball by comparing the red, green and blue components of each pixel with minimum and maximum thresholds for that particular colour component. This was the first technique that was used to detect the ball, as the image is already in RGB format. If the colour of the pixel is above all of the minimum thresholds and below all of the maximums, it is considered a match and is used when calculating the position of the ball. Figure 8.12 shows an image before and after the application of the thresholds shown in Table 8.2. All pixels that are below any of the thresholds are

shown as black, all pixels above any maximum are white, and all pixels that are between all minimums and maximums are shown as red.

Figure 8.12 - (a) Original Image (b) Result Of Applying Thresholds

Colour	Minimum	Maximum
Red	223	255
Green	100	255
Blue	90	191

Table 8.2 - Thresholds Used In Figure 8.12

To ensure that any residual noise in the image does not result in an object being falsely identified when the ball is not actually in the image, a minimum number of pixels must match before an object is considered detected. This minimum value can be set through the video settings window in software. By introducing this minimum match value, a small or partially obscured object may be ignored by this software. This also has the effect of simplifying the process of tuning the thresholds.

8.3.3.2 HUE-SATURATION-BRIGHTNESS THRESHOLDS

This technique is almost identical to the RGB thresholds, except it uses hue, saturation and brightness HSB thresholds. After spending some time working with the RGB technique, it was realised that it was very difficult to tune the thresholds as humans do

not tend to think about how much red, green and blue is present in a particular colour. When a person describes a colour, they tend to specify a general colour, and modify it with words such as ‘light’, ‘dark’, ‘vibrant’ and ‘washed out’. These descriptions translate quite simply into the HSB colour-space, with the general colour being the hue, the light/dark being brightness, and vibrancy being saturation. This greatly simplifies the process of tuning the thresholds, as we can actually understand what each component does.

Figure 8.13 below shows the HSB colour cone, which is one way of visualising the HSB colour space. The hue is represented by the angle around the central axis, saturation is represented by the horizontal distance from the central axis, and brightness (sometimes referred to as value, ‘V’, as in the figure below) is represented by the vertical distance from the point of the cone.

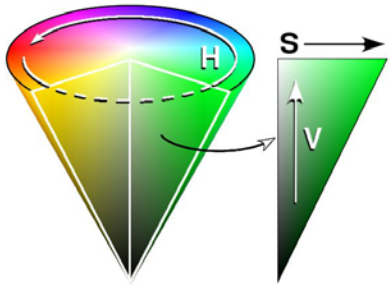


Figure 8.13 - Hue Saturation Brightness Colour Space Represented As A Cone
(Source: http://en.wikipedia.org/wiki/HSB_color_space)

By using the HSB colour space, the thresholding can be more successful as the pixels included within the allowed region are more closely related to the base, or target, colour. When HSB thresholds are used, a narrower hue band can be used, while still allowing for significant changes in the brightness and saturation components due to shading and lighting conditions. The figure below shows the difference between regions defined within the RGB and HSB colour spaces.

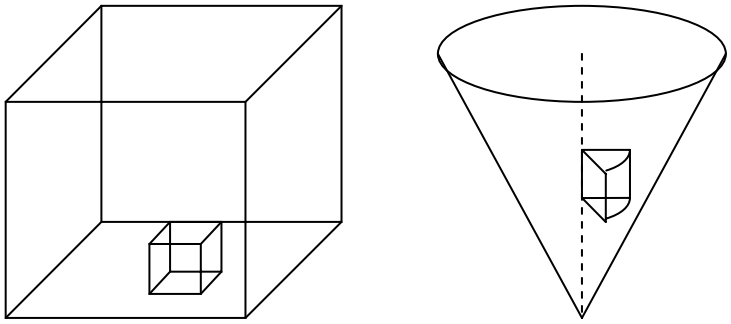


Figure 8.14 - Defining Regions In The RGB And HSB Colour Spaces

As seen below in Figure 8.15, the use of HSB thresholds greatly improves the results of the analysis. In the image using RGB thresholds, shadowed areas of the ball are not considered part of the object, whereas when using HSB thresholds almost every pixel within the ball is included with only a small area of one of the finger tips being erroneously included.

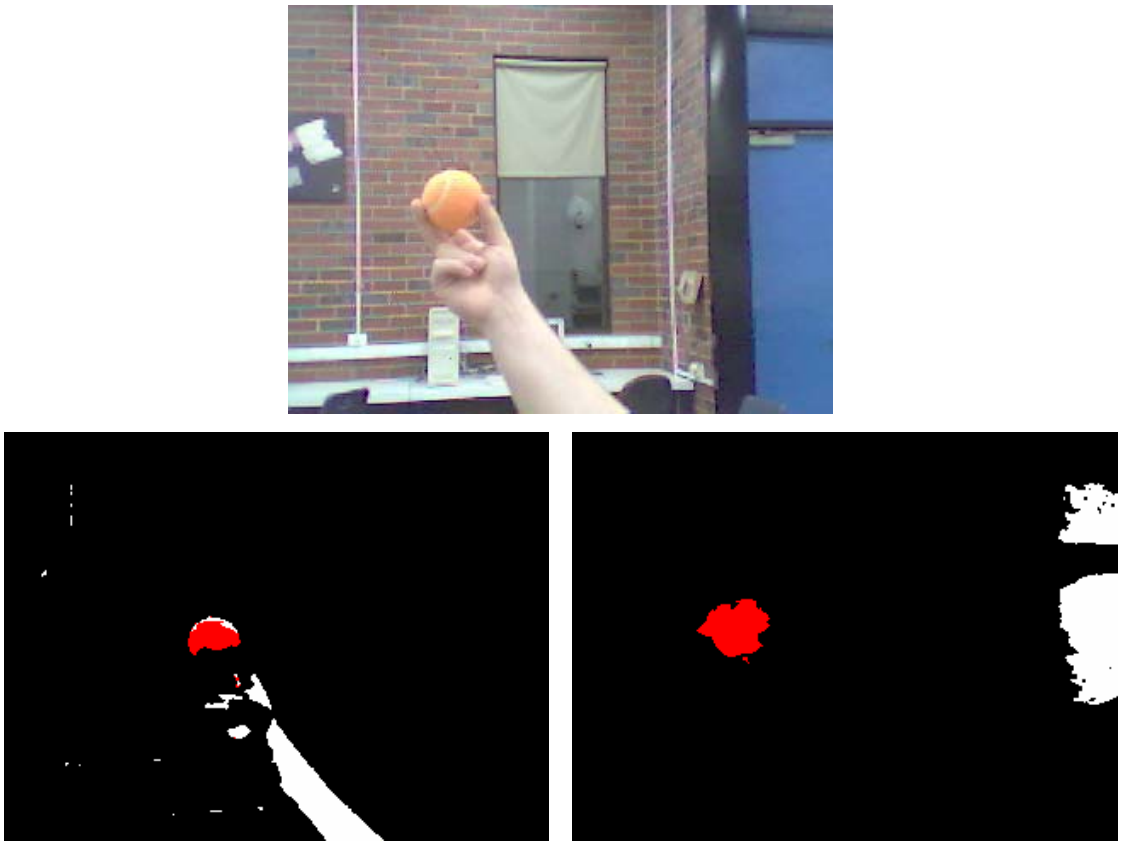


Figure 8.15 - (Top) Original Image, (Left) Using RGB Thresholds, (Right) Using HSB Thresholds

Colour	Min	Max		Component	Min	Max
Red	237	255		Hue	16°	63°
Green	142	255		Saturation	66	150
Blue	105	185		Brightness	185	255

Table 8.3 - Thresholds Used In Figure 8.15

8.3.3.3 RGB TO HSB CONVERSION

Converting between the RGB and HSB colour spaces is not a linear process. The following equations can be used to convert from RGB to HSB, where R, G and B are between 0 and 255, resulting in S and B also being between 0 and 255. The hue component ranges from 0° to 360°, with red, green, blue corresponding to 0°, 120° and 240° respectively. The *Max* and *Min* variables used in the following equations are defined as the maximum and minimum of the red, green and blue components.

$$H = \begin{cases} 60 \times \dfrac{G - B}{Max - Min} + 0, & \text{if } Max = R \\ 60 \times \dfrac{B - R}{Max - Min} + 120, & \text{if } Max = G \\ 60 \times \dfrac{R - G}{Max - Min} + 240, & \text{if } Max = B \end{cases}$$

$$S = Max - Min$$

$$Br = Max$$

Equation 8.3 - RGB To HSB Conversion

Note that if *Max* and *Min* are equal, then the hue is undefined. This is due to a saturation of zero, which corresponds to the central line of greys within the colour cone. Also, if *Max* is zero, then the hue and saturation are not defined. This corresponds to the bottom point of the cone, where brightness is equal to zero.

8.3.3.4 MANUALLY TUNING THE THRESHOLDS

For this technique of using thresholds to function correctly, it must be successfully tuned to only include the regions of the colour space that the object exists within. If the thresholds are too relaxed, a large number of pixels will match even though they do not form part of the object (false positives). In this situation, the software may believe it has successfully found the object even though it may not appear in the image, or its calculated position may not be aligned with the true location of the object. If the thresholds are too tight, then many of the objects pixels will not be considered a match (false negatives). This could result in the total number of matched pixels being below the minimum match count, resulting in the object not being detected at all. These scenarios can be seen below in Figure 8.16, bottom left and right respectively, with the optimal thresholds being used in the top left image.

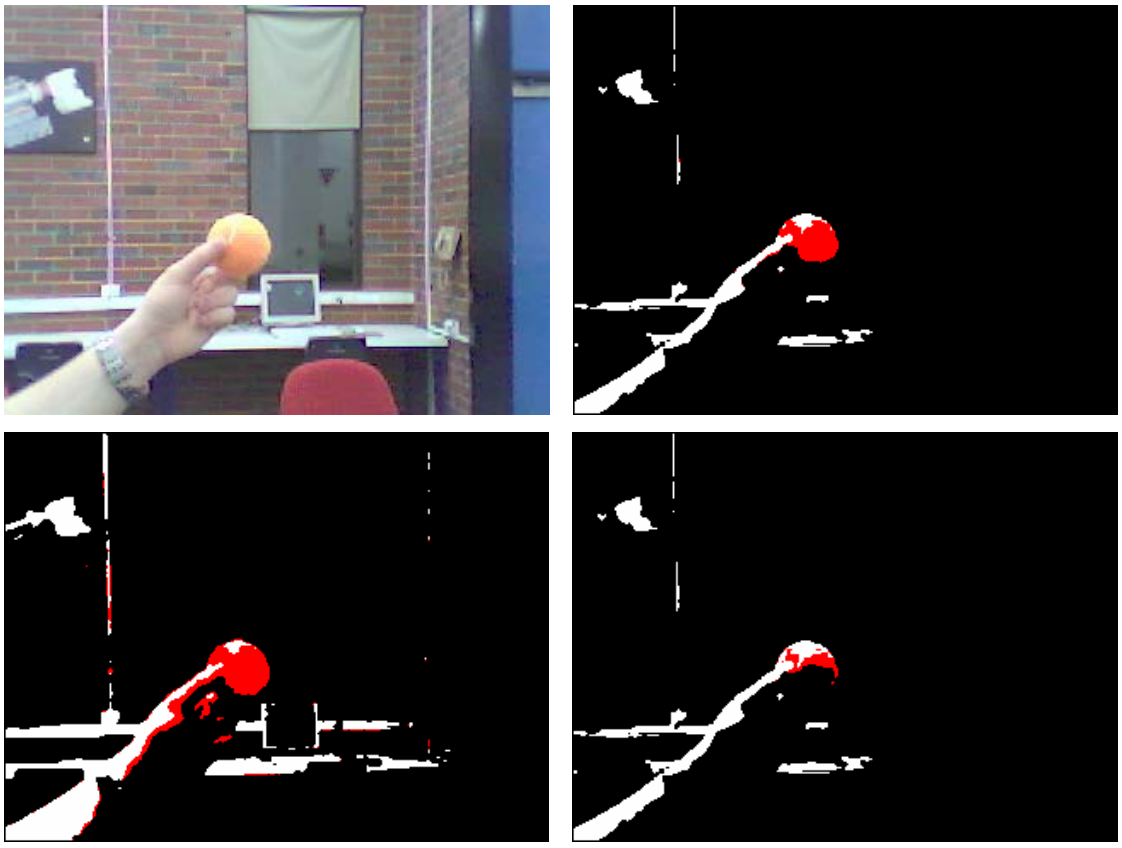


Figure 8.16 - Tuning The Thresholds: (Top Row) Source Image, Optimal Thresholds, (Bottom Row) Relaxed, Restricted Thresholds.

Ideally, no false positives or false negatives are desired but this is an almost impossible situation to achieve, resulting in a compromise to be reached. The most likely scenario that can be achieved is to eliminate all false positives, and minimise the number of false negatives. This would ensure that no stray pixels affect the calculation of the objects position, while trying to maximising the number of pixels within the object that are actually used in the calculation. This can be a difficult goal when using the RGB colour space, but is not unreasonable when using the HSB colour space. This is primarily due to the different shapes of the included regions within each colour space, as described previously.

The process of tuning the thresholds can be much simpler when using the HSB thresholds, as the pixels within the thresholds are more closely related than when using RGB thresholds. The optimal RGB and HSB thresholds for an image are compared in Figure 8.15

8.3.4 CALCULATING THE OBJECT'S POSITION

After the thresholds have been applied to the image, all the pixels that are considered part of the object are now known. This allows the program to calculate the centre of the ball. The position is calculated by averaging the x and y coordinates of all the matching pixels, resulting in an average x and average y coordinate.

By simply averaging the x coordinates and the y coordinates, it is assumed that only one object is in the frame at any one time. This can cause errors in any circumstances where multiple objects are visible. This has been partially overcome through the use of an optimised scanning routine that first performs a rough scan of the image, then a fine scan on a smaller region of the frame.

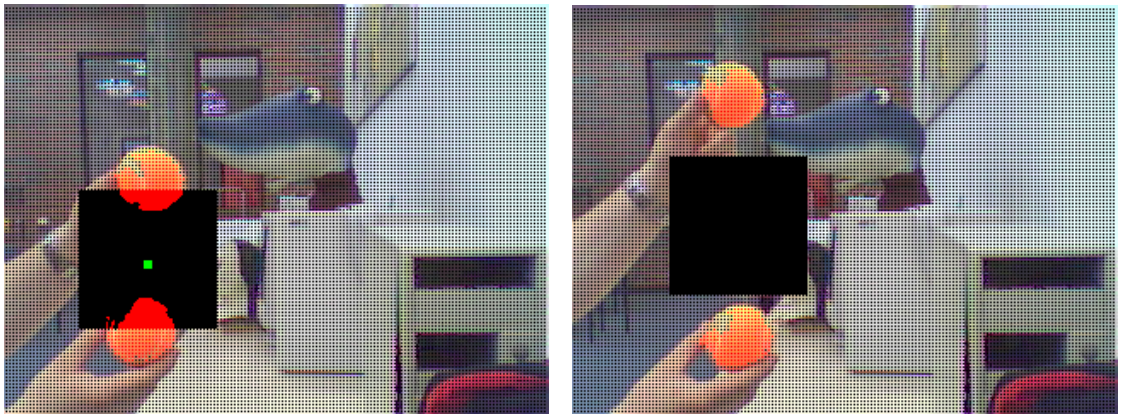
The process of scanning through every pixel can result in jerky movement of the puppet, as the rate at which the position is updated is too low. This is a result of the scanning process taking to long for each frame. To increase the frame rate of the image

analysis, the scan can be broken down into two stages. The first stage is to perform a rough scan of the image, only analysing a small portion of the total number of pixels. In this software, the rough scan checks every one in 5 pixels horizontally as well as vertically, resulting in only 1 in 25 pixels being processed. This can be used to calculate an approximate position of the object, allowing a fine scan to be performed only in this region.

The size of the fine scan can be configured through the video settings dialog in the software, allowing it to be optimised for the particular object being detected. The default size of this scan is a region 80 by 80 pixels, and is scanned for the object at full resolution. If the number of pixels that are matched within this region is not above a minimum count, the software does not consider an object to be found.

The side effect of this two stage scanning is noticed when two objects are present in the image. Previously the calculated position of the detected 'object' would be between the two actual objects, with the distance based on the size ratio of the two actual objects. For example, if one object was twice as large (in terms of number of pixels) as the other, the calculated 'position' would be two thirds of the way from the smaller to the larger object.

Now that a smaller region is scanned to calculate the final position of the object, there is an increased likelihood of no match occurring when multiple objects are present in the frame. This is due to the rough scanning approximating the position of the 'object' being part way between the two actual objects. The fine scan will then occur in an area that does not contain either of the actual objects. Even if part of one of the objects is within the fine scan region, there is an increased chance of the number of matching pixels within the region not being above the minimum count threshold. Figure 8.17(a) below demonstrates multiple objects that are close together, with some pixels from each ball matching, while (b) demonstrates the objects being further apart. As you can see, no pixels are found within the fine scan region, and as such no object is considered detected.



(a) Objects Close Together **(b) Objects Further Apart**

Figure 8.17 - Handling Multiple Objects Within The Image

(Note: This figure will be updated with new images)

The two stage scan of the image is not just applied to the threshold and calculation processes, but also to the Gaussian filtering. This can dramatically reduce the processing time of each frame, as the Gaussian filtering is a relatively time consuming process. By only filtering the pixels that will actually be used in the analysis, a significant number of calculations can be eliminated. For the scan resolutions mentioned previously, the rough scan will take 1/25th, or 4%, of the time taken to perform a full resolution scan, and the fine scan will take 8.33% of the time to perform a full scan. This results in an 87.6% reduction in the time to scan a single frame, or in other words an 8 times performance increase.

This two stage scanning process does have some negative aspects. If the object in the image is larger that the fine scan region, it may not be able to accurately calculate its true centre location. This situation can arise in several scenarios, such as the object being moved much closer to the camera than originally configured for, or the region being configured incorrectly to too small an area. The latter is easily dealt with by manually setting the fine scan to a larger size. The former is more difficult to deal with, as the object may be continuously moving from far away from the camera to up close.

In this situation, one possible solution is to increase the size of the fine scan at the cost of frame rate, or an alternate technique can be implemented in code that automatically

increases the size of the fine region when the number of matched pixels in the rough scan is above a given level. The fine scan resolution could also be reduced to counteract the increased processing time, resulting in a similar frame rate for these larger scan regions. This automated adjustment of the fine scan size and resolution has not been implemented, as it is outside the scope of this project and is only applicable in a small number of situations.

In the normal mode of operation, a maximum pixel count could also be set, along with the minimum count. This would ensure that if the thresholds were not set correctly, or the overall conditions changed, the software would not try to calculate the location of the object based on invalid, or meaningless, data.

8.4 AUTOMATED CALIBRATION OF THE WEB CAMERA

As discussed previously, manually adjusting the thresholds for the object detection can be a very time consuming task. To simplify the process, an automated calibration feature has been included within the software. There are multiple techniques that can be used to set the thresholds including finding the average colour within a region, and using a 'colour picker' tool.

The first technique involves the user holding the object such that it fills the specified region on the screen. The software then scans through the entire region, calculating the average colour and standard deviation of each colour component. The standard deviations are calculated to enable the software to easily set the upper and lower thresholds based on the average colour plus or minus a multiple of the standard deviation. An alternate to this is to use a fixed tolerance that the user can specify, instead of the standard deviation. The user can choose the shape of the region to be a circle, ellipse, or a rectangle.

As each pixel is scanned through during auto calibration, its coordinate is tested to check whether it is within the target region. If it is within the region, its red, green and

blue values are added to their respective collections, building up a list of colour data for further analysis. After all pixels have been checked, the mean and standard deviations can be found for each colour component, allowing the thresholds to be set. These can equations can be seen below in Equation 8.4 and Equation 8.5.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Equation 8.4 - Mean *(for data x_1, …, x_n)*

$$s_x = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n} x_i^{\,2}}{n} - \bar{x}^2}$$

Equation 8.5 - Standard Deviation *(for data x_1, …, x_n)*

The second technique that may be used to automatically set the thresholds involves the use of a colour picker tool. This method allows the user to define the target colour by selecting pixels within the video frame. This technique has several variations, including using a single pixel, using the average colour within a small region (3x3 or 5x5), or specifying multiple pixels. With the first two variations, after the target colour is found, the thresholds are then calculated by adding and subtracting a fixed tolerance. The third variation finds the tightest thresholds that include the specified pixel colours. The calculation of the tightest thresholds is simply performed by using maximum and minimum functions.

The colour picker tool gets the colour under the cursor when the user clicks a point on the screen. This is done through the use of several API calls that first retrieve the location of the cursor, and then get the colour of the pixel at that point on the screen. This does not restrict the user to selecting colours only within the video stream, but allows the user to select any pixel currently on the screen.

8.4.1 SENSITIVITY TO LIGHTING CONDITIONS

The process of applying a threshold to identify the object within the frame is sensitive to any changes in lighting conditions. If the lighting level changes during use, the colour of the object to be tracked will change, which could result in it no longer being within the current thresholds. This is more of a problem when using the RGB colour space, as it affects each of the colour components, whereas in the HSB colour space, the hue will still be the same. When using the HSB colour space, the brightness and saturation thresholds could be set slightly more relaxed than normal, allowing a variation in lighting levels. This would still allow some false positives through to the analysis, but significantly less than when using the RGB colour space. It should be noted that the RGB colour space is still able to be used for the thresholding, as it may be more appropriate in some situations.

8.5 ADJUSTING THE SETTINGS

Within the software, all of the settings related to the video analysis can be adjusted within a single window. These include thresholds, scan resolutions, and minimum pixel match requirements, which can all be set directly by entering the desired value. The display options area allows the user to choose whether to show the results of the analysis, or just show the original frame. These display options include masking the areas above, below and between the thresholds, and marking the calculated centre of the object. The settings dialog is shown in Figure 8.18, on the following page

As explained in section 8.2.2, the VFW interface does not allow for programmatic control of the video resolution and colour format. As such, the user is able to adjust these settings directly by clicking on the “Change Resolution” button. Within the dialog that appears, the user must select a resolution of 320 by 240 with a pixel depth of RGB24.

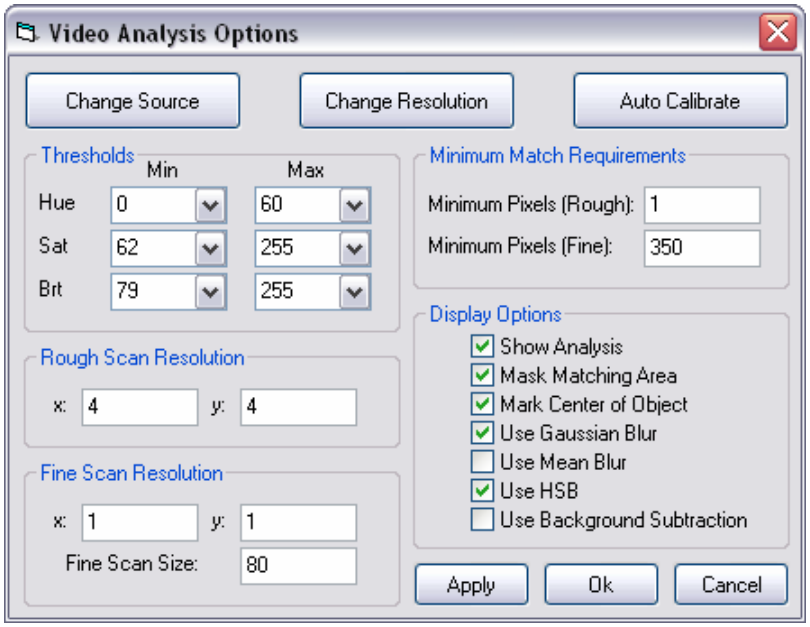


Figure 8.18 - Adjusting Video Analysis Settings Within The Software

If the user has multiple capture devices connected to their computer, they can choose which device to use by clicking the "Change Source" button within the video settings dialog. This allows the software to be used on computers that have other devices installed, such as a graphics card with video input.

8.6 Using The Web Camera To Record Scripts

With the ability to visually track an object using a web camera, as well as the ability to use script files to control the puppet, it is a logical extension to use the web camera to create, or record, scripts directly. During operation when using the web camera, the position of the detected object can be used to or record a script. This allows a simple method of creating scripts, without requiring them to be written manually. As each frame is analysed, the location is written to file, along with a time reference.

When no object is detected, there are two options that can be chosen from. These are pre-emptive and non pre-emptive movement. The pre-emptive mode does not add any lines to the script if no object is detected, resulting in the system interpolating between

the positions before and after the time span when no object is detected. The non pre-emptive mode adds an extra line to the script when an object is found, with the last known position but with the current time. This results in the animatronic holding its position until the object is redetected. The plot below compares the movement when each mode is used (see Figure 8.19).

The time reference used in the script is the time since the recording started in milliseconds. The current time is easily obtained within Visual Basic by reading the value of the `Timer` variable. By storing this when recording starts, the relative time value can be calculated each time data is written to the script file.

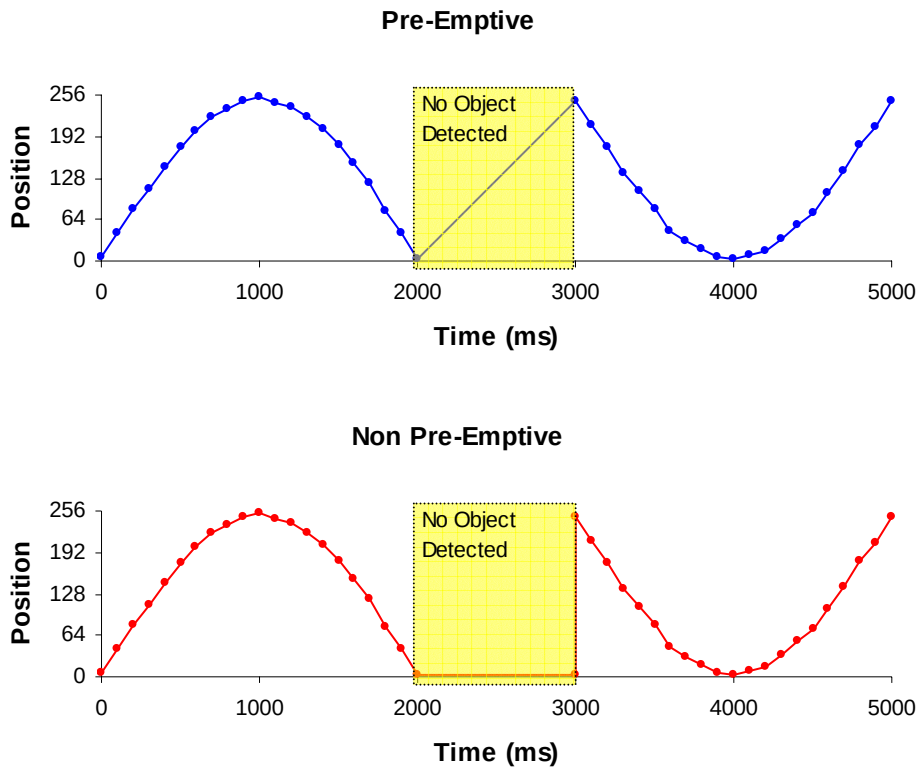


Figure 8.19 - Comparing Motion Profiles For Pre-Emptive And Non Pre-Emptive Modes

8.7 MINIMUM HARDWARE REQUIREMENTS

The intelligent video control system must be able to analyse the incoming video stream at a frame rate that allows for smooth motion of the connected animatronic. This introduces minimum requirements for the PC that is used to run the software. It has been found that a frame rate of 10 fps is sufficient to provide smooth motion, and as such can be used as to test if a PC is suitable for running this software.

It should be noted that this imposes a significant restriction on the level of PC to be used. The system requires at least an equivalent 1.2GHz Pentium based computer with 256 MB of RAM, and running Windows XP. It is recommended that an equivalent of a 1.6 GHz Pentium processor with 512 MB of RAM be used. This level machine is only required when using the vision control system, and a less powerful machine is perfectly suitable when only using the other modes of control within this software.

8.8 AREAS OF FURTHER DEVELOPMENT

If further development of the image analysis was to occur, there are several key areas that could be focused upon. These include handling multiple objects, and optimising the filtering speed and efficiency.

Currently, only one object can be handled at any one time, imposing restrictions on the colour of an object and the complexity of the environment in which the system may be used. To overcome this limitation, one method that could be investigated further is connected components labelling. This involves scanning through the thresholded image and determining the groups of interconnected pixels. These groups will be identified as different objects, allowing the position of each to be calculated separately. The process of identifying the individual objects within the image would be a very beneficial feature, with a minimum of implementation time required.

Another area that could be investigated for future development is the optimisation of the filtering process. One possible method that could be used to achieve this is recoding it using Single Instruction Multiple Data (SIMD) techniques. These include the use of MMX and/or SSE capabilities in all current processors. These instruction sets allow the processing of multiple pieces of data simultaneously, applying the same operation to each. This is ideal for applying the filter and the RGB to HSB conversion, as they are both very repetitive tasks.

MMX instructions work with integer data store in 64 bit registers. The 64 bits can be comprised of 8 byte values, 4 words or 2 double words. This packed data format is how MMX can work with data in parallel. If the data from the camera was received in RGB32 format, the use of MMX instructions would allow two full pixels to be operated on per instruction, as opposed to only 1 sub pixel per instruction. This can reduce the time taken to apply the filter to each frame, thereby increasing the possible frame rate on a given computer system. Another advantage of this is the reduction in minimum system requirements to be able to achieve a satisfactory performance of the system. Implementing this feature would be very beneficial as it would dramatically increase the efficiency of the image processing functionality, but would require a complete redesign of this section of the software.

The Gaussian filter can be applied more efficiently using MMX instructions if the kernel is broken down using a different approach to the current method. The 3 by 3 Gaussian kernel is actually the result of the convolution of a pair of 2 by 2 mean kernels. As a result of property, a second order (3 by 3) Gaussian filter can be achieved by cascading two first order (2 by 2) mean filters. This will have the same effect as the original Gaussian kernel, due to matrix convolutions being associative and commutative.

The application of two 2 by 2 mean filters is a much simpler task when using the MMX instructions, as it natively operates on eight bytes, or two pixels, at a time. This streamlines the pipelining, as it eliminates the need for any results to be accessed within

the same scan. The process of applying the each of the 2 by 2 mean filters is a two stage process, as for the optimised 3 by 3 Gaussian kernel. The output of the first mean filter is stored in a temporary image, which is used as the source for the second mean filter. This produces the same result as applying the 3 by 3 Gaussian, but is highly suitable for implementation using the MMX instructions.

Another benefit of using a mean filter is the kernel is that each element in the kernel is of equal weighting. This removes the need to multiply each pixel by a coefficient before summing, thereby reducing the processing overhead required to apply the kernel.

The level of complexity of the dual mean filter is given by Equation 8.6, where M is the width of the mean kernel, i.e. 2. This can be compared to the complexity of a decomposed Gaussian filter in Equation 8.7, where N is the width of the Gaussian kernel, i.e. 3.

$$Complexity_{(\text{Dual Cascaded Mean})} = 2M^2$$

Equation 8.6 - Level Of Complexity Of A Dual Mean Filter

$$Complexity_{(\text{Gaussian}-\text{Decomposed})} = 2N$$

Equation 8.7 - Level Of Complexity Of A Decomposed Gaussian Filter

From these equations, it can be seen that for a 2 by 2 mean filter, the dual mean filter has a complexity level of 8, while a 3 by 3 Gaussian filter has a level of 6. As these two filters produce the same results, the decomposed Gaussian filter would execute faster than the dual mean filters, but this is assuming they were both implement using the same instruction set. As the dual mean filter can be implemented using the MMX instruction set easily, whereas the decomposed Gaussian cannot, the dual mean filter can be made to execute more efficiently than the Gaussian.

9 CONCLUSION

The animatronic controller that has been designed and implemented is far superior to any existing controller available today. Previously, most servo controllers could only control up to 8 or 16 servo motors, and with limited positioning resolution. Some of these controllers could only operate a sub set of the attached servos simultaneously, due to the method in which the control algorithms and communications have been implemented. Also, a number of previous controllers are limited by the use of RS232 serial communication, which restricts the data rate such that good resolution with more servos is not possible. With this new servo controller that has been created, the data rate is much greater due to the use of USB communications, and so 27 servo motors can be controlled with full 8 bit resolution simultaneously with 256 position increments. The controller developed provides a far higher level of performance due to the signal decoding methods, compatibility due to the USB communications and functionality because of the number of available control methods.

Another advantage of this controller is the plug and play functionality built into the control interface software. When a new controller is attached to the host computer, the new animatronic is detected automatically, and is made available for immediate use with the correct settings. This greatly enhances the portability and ease of use of this controller, as the user does not have to reconfigure the controller every time they wish to use it. This is achieved by storing the settings on the actual microcontroller, instead of on the host computer.

A major feature of the software is the ability to choose between three separate modes of operation of the animatronic. The basic mode is direct control, and is usually the only control mode available with other available servo controllers. The scripted and intelligent visual control modes within the control interface software greatly enhance the functionality of this system. The scripted control with its ability to drive all of the

connected servos in an accurate and repeatable fashion, and with the motion profiles of each servo being predefined within the script file allows a versatile interface for applications such as stop motion. This software also allows the user to create and edit these scripts easily using the graphical user interface, and with the convenient click and drop interface, scripts are very simple to create.

The most advanced control mode available is the intelligent vision control. This mode allows the animatronic puppet to interact with its surrounding environment. The system detects and tracks objects of a specific colour and uses its location to control two servo motors. By using the generic VFW interface to obtain the video data stream, the system is able to work with most video capture devices available today, instead of being designed for a particular camera using a proprietary API. As described in section 8, the objects colour can be specified using either the RGB or HSB colour space, allowing for greater flexibility. One of the major focuses when developing the vision system was the speed and efficiency of the analysis. By using various techniques including filter kernel decomposition and two stage scanning, the algorithm can achieve 15 fps (the maximum speed of most web cameras) on an average computer, without the use of dedicated hardware such as custom frame grabber expansion boards. In addition, the intelligent vision control is capable of generating scripts based on the image capture, letting previously tracked paths to be replayed using the scripted control method.

While the final controller is capable of controlling a total of 27 servo motors, or a combination of servo motors and PWM driven devices, as well as containing three banks of digital I/O, there still is a large area for development. The required goal of developing a device capable of controlling the servo motors within a small robot head was achieved, with a large amount of extra functionality included.

The controller is in its current form capable of controlling far more than simply servo motors, although the sheer volume and precision of this control is significant enough to allow this controller to be used for far more than the original concept of a single four servo head. The controllers greatest short falling was the lack of development time

allocated for physical development, with too much time being dedicated to both adding functionality, as well as preparation of the controller to presentation standards. This aside, the development of the physical aspects of the controller are somewhat inhibited by available manufacturing processes and development capital.

All implemented methods have undergone extensive testing, however the testing time was not significant, and so while not problematic during typical operation, a more extensive testing and stability period, possible involving a beta release of the controller would provide a more comprehensive check, and allow greater confidence in the functionality. During testing however, the controller was able to operate both 27 servo motors simultaneously using both the direct and scripted control methods, as well as successfully operating both designated servos using the intelligent vision control system. In addition, the controller was able to operate constantly for a 2 seek period during display in an exhibition environment with touch screen control.

The GUI is capable of interpreting the functionality of specific controllers, and implementing all of the three control methods effectively, when operated with suitable equipment, including a compatible operating system and image capture device. The user interface operates intuitively, and the aim of reducing complexity for the user has been achieved. This however was also not allowed a great deal of testing with people not already familiar with the development of the program. While in presentation, only the standard touch screen interface was utilised, and so little work was done in exposing the rest of the interface to new users. While this could improve the immediate appearance of the program for possible consumers, additional development in this field would not improve the actual functionality of the system, and so was abstained from during this stage of creation.

The firmware of the controller has excelled expectations, and is not completely compatible with all specifications, and provides a product ready level of performance. There is room for improvement in the development of a suitable PCB, and development for implementation of all of the firmware's features. This aspect of the project was not

developed to its full potential, although it does provide a suitable level of control to demonstrate the core functionality of the controller.

Overall the controller and application have been an amazing success as far as achieving the original goal of controlling a four servo animatronic. The required functionality was obtained, along with both the required options, including direct and scripted control, as well as more complex, but beneficial goals such as the implementation of the USB communications standard, and the intelligent image analysis. This brought the controller to a level beyond that expected in the initial development stages, and provides far more opportunity as a platform for ongoing development.

9.1 FUTURE WORK

This system is by no means complete. There are many areas that could be looked into if further development was to occur, which were outside the scope of this project. Some of these areas include additional actuator types that could be controlled, alternate control methods using various types I/O, and further optimisation of the vision system.

9.1.1 ADDITIONAL ACTUATOR TYPES

The system that has been designed can be easily modified to enable PWM operation of the PCA modules, instead of outputting the multiplexed servo control signals. In PWM mode, the signal is approximately a 10.5 kHz pulse width modulate signal for using as the switching signal for a PWM device, such as a DC motor drive.

9.1.2 ALTERNATE METHODS OF I/O

Presently, the direct control mode cannot easily be used to update multiple servo's position simultaneously due to the limitations of PC human interface devices such as keyboards and mice. Generally they only allow for one or two servos to be changed at a time, as they only have two axes. To improve the ease of use of the direct control mode, the software could be extended to allow an alternate human interface device such as a joystick to be used. This would allow the user to control many servos directly using a simple input method.

9.1.3 VIDEO OPTIMISATION

The vision system is currently implemented using simple loops and compares which requires significant processing power when using large video frames. To improve the performance of the analysis, it can be implemented using Single Instruction, Multiple Data (SIMD) instructions available on all current CPUs. They allow multiple pixels to be processed in parallel, reducing the time taken to analyse each frame. This can be used to achieve higher frame rates, or to reduce the minimum system requirements for the system.

Another method that could be used to optimise the execution of the analysis is to use General Purpose Graphical Processing Unit (GPGPU) techniques. These involve using the resources of the graphics processor to perform the analysis, as they are perfectly suited to image manipulation tasks such as this. Their extremely parallel nature can greatly improve the performance of the system, thereby allowing more complex analysis algorithms to be implemented.

9.1.4 MULTIPLE STIMULUS RESPONSE

Currently, the vision system cannot be used to track multiple objects within the video frame. Work in this area could significantly increase the value of the entire system, as it could be used to control more than two axes at a time. There are many ways that multiple objects can be identified including connected components labelling. This method uniquely identifies each independent object in the video frame, allowing each to be tracked individually.

10 APPENDICES

10.1 APPENDIX A – LEGACY CONTROLLER

The legacy controller was set up using a dual 555 timer configuration to generate the required pulse width and duration. In this configuration one 555 timer is configured as a multivibrator to set the signal period of 20ms. This output is then used as the trigger for the second 555 timer configured as a single shot with variable pulse width. The pulse width of the second timer is varied by adjusting the charging current to the capacitor using a potentiometer.

To facilitate multiple outputs, a single 555 timer was configured for the multivibrator, shown in Figure 10.1 as X5. The output of this is then amplified using a small signal field effect transistor (FET), and fed to the inputs of as many 555 single shot timers as necessary.

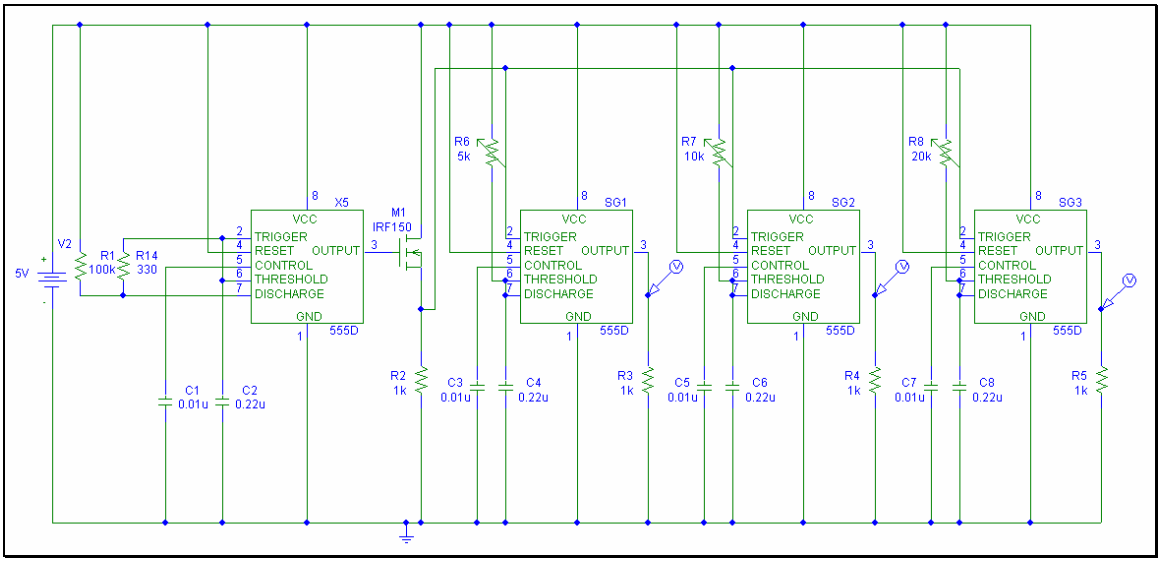


Figure 10.1 – Legacy Schematic

In Figure 10.1 three single shot timers have been implemented and labelled SG1, SG2 and SG3 respectively. These timers provide the pulse width variation by adjusting the potentiometers R6, R7 and R8. By varying these potentiometers, the voltage present at the timers Threshold pin is adjusted, causing the width of the output pulse to vary from between 0.7ms and 2.3ms. When these timers are run together, the signal outputs for 5kΩ, 10kΩ and 20kΩ potentiometers is given in the simulation shown in Figure 10.2.

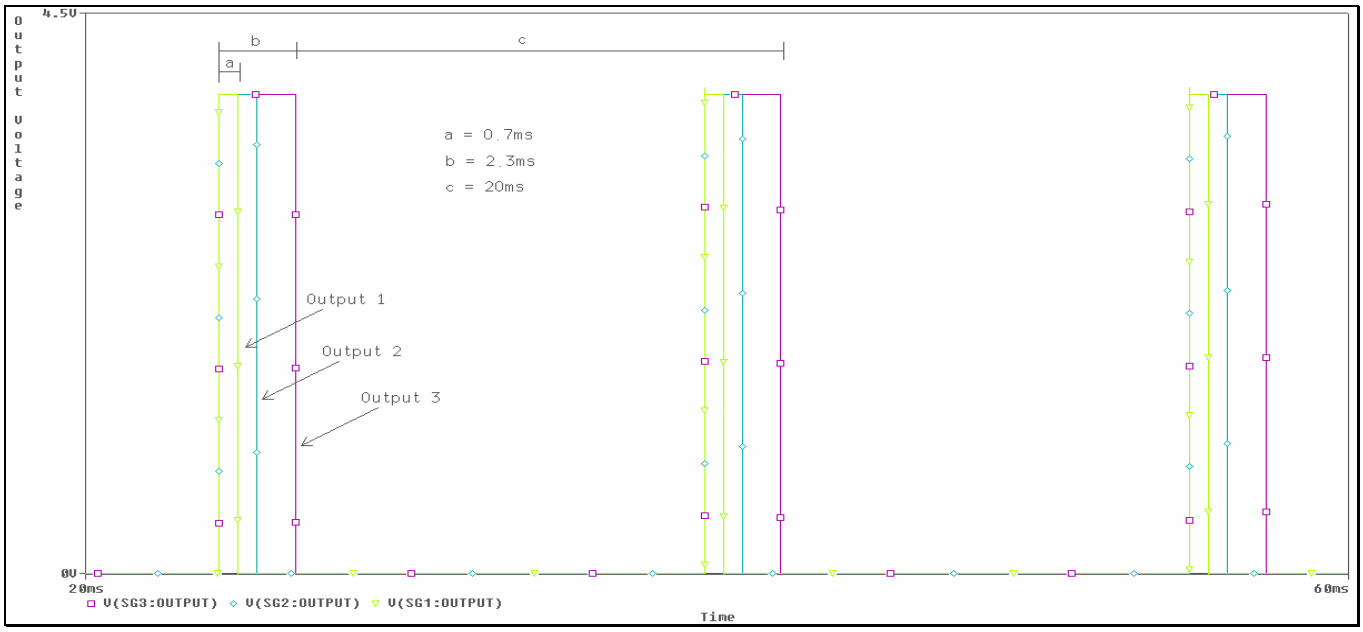


Figure 10.2 – Legacy Output Simulation

10.2 APPENDIX B – DESCRIPTORS

10.2.1 DEVICE DESCRIPTOR

The device descriptor is 18 bytes long, and consists of 14 fields. The descriptor is given in hexadecimal.

0x12 0x01 0x01 0x10 0x00 0x00 0x00 0x32 0x03 0xEB 0x01 0x00
0x01 0x00 0x01 0x02 0x03 0x01

This tells the host that the device conforms to USB specification 1.1, and that the control endpoint has a 32byte buffer. It specifies the controllers Vendor ID and Product ID, as well as the release number. It also tells the host the index of the strings containing manufacturer, product and serial information, and that the device has one configuration.

10.2.2 CONFIGURATION DESCRIPTOR

The configuration descriptor has a total of 9 fields in 10 bytes. The descriptor is given in hexadecimal.

0x00 0x29 0x01 0x01 0x00 0x80 0x50

This tells the host that the configuration, which was defined as the only possible configuration by the device descriptor, contains a total of 41 bytes of information in its descriptors. It identifies this configuration number as number one, and that in this configuration, only one interface exists. It then provides information that the controller is bus powered, and draws up to 160mA.

10.2.3 INTERFACE DESCRIPTOR

This descriptor relays information about each interface declared by the configurations descript, in this case, interface one. This descriptor has nine fields in nine bytes shown in hexadecimal.

0x09 0x04 0x00 0x00 0x02 0x03 0x00 0x00 0x00

This indicates that this is interface zero, and is a HID compliant interface support two endpoints in addition to control endpoint zero.

10.2.4 ENDPOINT DESCRIPTORS

There are two endpoint descriptors, each having 6 fields requiring 7 bytes of data. The information for each endpoint describes its specific capabilities. These endpoints are endpoint numbers 4 and 5 on the device.

10.2.4.1 ENDPOINT 4

This endpoint identifies itself as endpoint four, with an IN bound data direction. It then configures itself as in interrupt endpoint with a packet size of 32 bytes, and an interval of 255ms.

0x07 0x05 0x84 0x03 0x00 0x20 0x10

10.2.4.2 ENDPOINT 5

This endpoint identifies itself as endpoint five, with an OUT bound data direction. It then configures itself as an interrupt endpoint with a packet size of 32 bytes, and an interval of 16ms.

0x07 0x05 0x05 0x03 0x00 0x20 0xFF

10.2.5 STRING DESCRIPTOR

Each string descriptor contains information to allow the host to retrieve a string of information from the controller. The string descriptors contain all text in Unicode, and so 16bit values are used. The string descriptors contain the manufacturer name, product name and serial number of the device.

10.2.6 HID DESCRIPTOR

This is a variable length descriptor containing at least 7 fields in 9 bytes, byt can be expanded as more descriptors are added beneath it, however since this device supports only a single report descriptor, only the minimum 7 fields are used.

0x09 0x21 0x01 0x11 0x00 0x01 0x22 0x00 0x25

This informs the host that the controller is HID 1.11 compliant, only has a single subordinate report descriptor, and that the subordinate descriptor is a report descriptor. It then identifies that this subordinate descriptor has a length of 37 bytes.

10.2.7 REPORT DESCRIPTOR

The report descriptor defines what reports are sent to and from the device in addition to the standard control reports. The report descriptor is given in hex.

```
// HID Report 1
        0x06, 0xA0, 0xFF,       //Usage Page (vendor defined)
        0x09, 0xA5,             //Usage (Vendor Defined)

        0xA1, 0x01,             //Collection (Application)
// The Interrupt Input Report
        0x09, 0xA5,             //Usage (Vendor Defined)
        0x15, 0x80,             //Logical minimum (80h or -128)
        0x25, 0x7F,             //Logical maximum (7Fh or 127)
        0x75, 0x08,             //Report size (8 bits)
        0x95, 0x05,             //Report count (5 bytes)
        0x81, 0x02,             //Input (data, variable, absolute)

// The Interrupt Output Report
        0x09, 0xA5,             //Usage (Vendor Defined)
        0x75, 0x08,             //Report size (8 bits)
```

```
                0x95, 0x20,                     //Report count (32 bytes)
                0x91, 0x02,                     //Output (data, variable, absolute)
// The Feature Report
                0x09, 0xA5,                     //Usage (Vendor Defined)
                0x75, 0x08,                     //Report size (8 bits)
                0x96, 0xE8, 0x03,               //Report count (1000 bytes)
                0xB1, 0x02,                     //Feature (data, variable, absolute)

                0xC0,                           //End Collection (Application)

}
```

This provides all information regarding the data sent via USB. The input report, output report and feature report are all given usages, data boundaries, sizes and directions, and placed into a single collection object for the application.

This HID Report descriptor is tested using the HID report descriptor tool from the USB Implementers forum, with results given in Figure 10.5.

10.3 APPENDIX C – OVERTONE CRYSTAL

As a general rule, any crystal will oscillate at the lowest possible multiple of its fundamental frequency, and so, to inhibit these modes of vibration, an LC_3 overtone tank, shown in Figure 10.3, is required to cause the circuit to appear inductive at frequencies below the required overtone, and capacitive at frequencies surrounding and higher than the required overtone.

The series LC_3 branch is selected to become resonant at some frequency below the fundamental, causing the circuit to become inductive at the fundamental, and inhibiting oscillation at that frequency. The series LC branch also forms a parallel resonant circuit with C_2, and so these values are also matched so that this resonates at a frequency

approximately double the fundamental, causing the circuit to become capacitive at frequencies above this, encouraging the crystal to oscillate at its third overtone.

The crystals load capacitance is the equivalent capacitance resulting from C_1, C_2 and C_{stray}, where C_{stray} is simply the capacitance arising from circuit capacitance and is dependant on materials used. It is a reasonable approximation to assume C_{stray} = 5pF. The crystals pullability is a measure of its output frequency accuracy as a result of variations in the load capacitance. After considering these requirements, the crystal load capacitors C_1 and C_2 were chosen as 22pF. Then, the LC branch is chosen with L as 2µH and C_3 as 1nF. This causes the tank to resonate inductively at around 3.5MHz and capacitively at approximately 25MHz, giving a third overtone response.

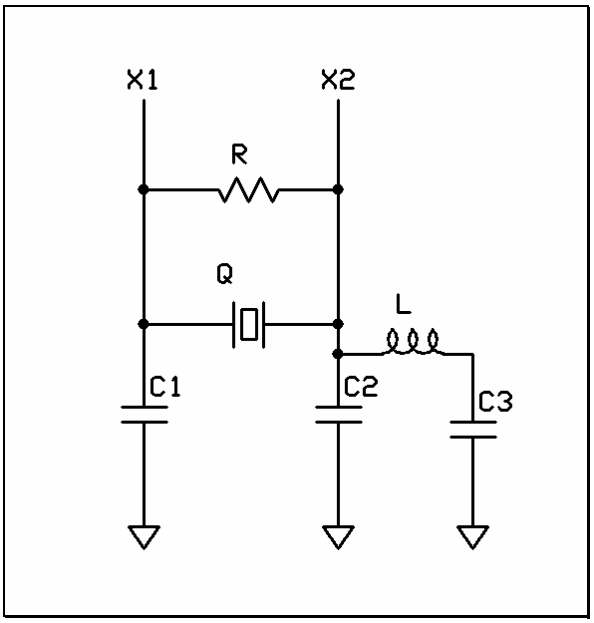


Figure 10.3 – LC Overtone Tank

10.4 APPENDIX D – FIRMWARE TESTING RESULTS

To ensure compatibility the free testing utilities provided by the USB Implementers Forum were used to evaluate the performance of the device under all necessary scenarios. This provides an accurate, current measure by which the device firmware can be ensured of compliance.

10.4.1 HID REPORT DESCRIPTOR TEST

The HID report structure is tested using the HID Report Descriptor Tool 2.4 from the USB implementers forum. The report was loaded into the testing program, as seen in Figure 10.4. This was then test using the programs Parse Descriptor menu item, with no errors being generated. The output windows generated by Parse Descriptor is shown in Figure 10.5.

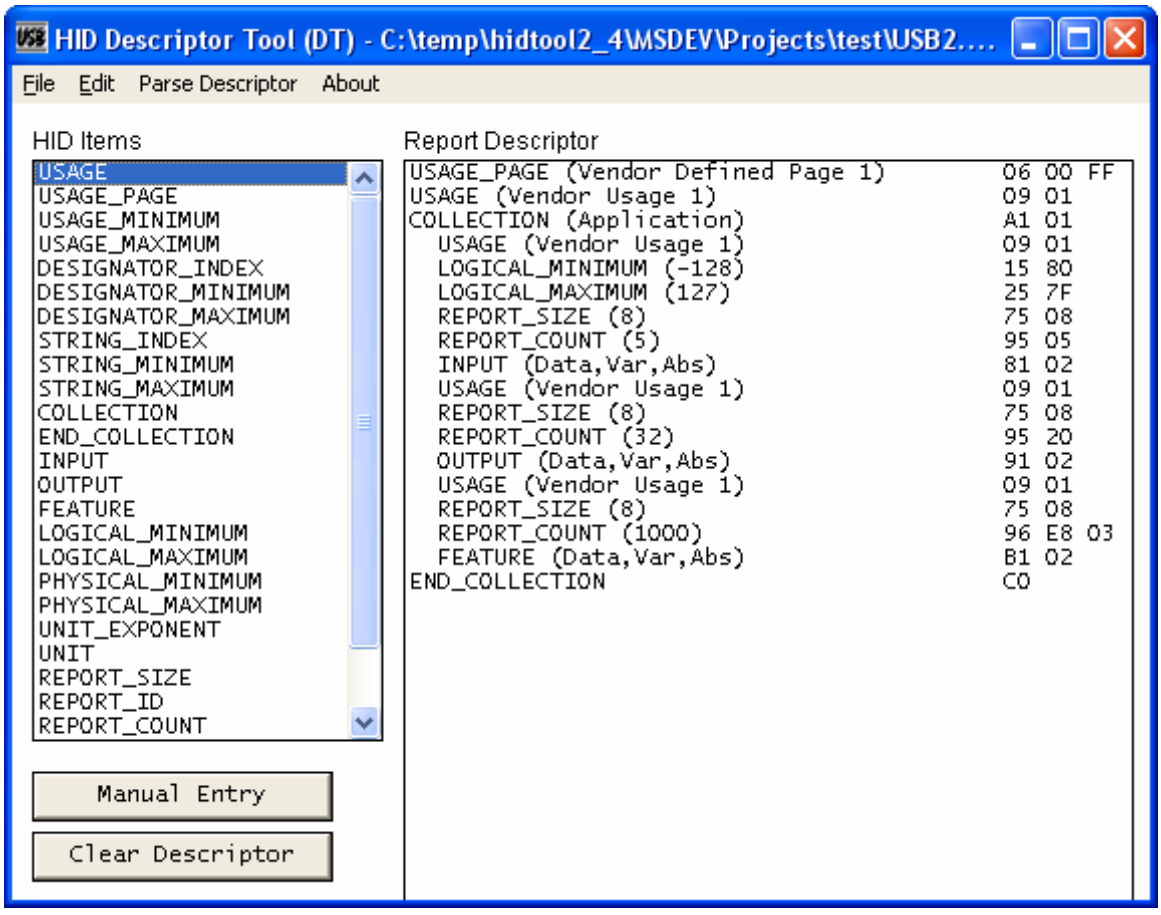


Figure 10.4 – HID Descriptor Tool 2.4 input

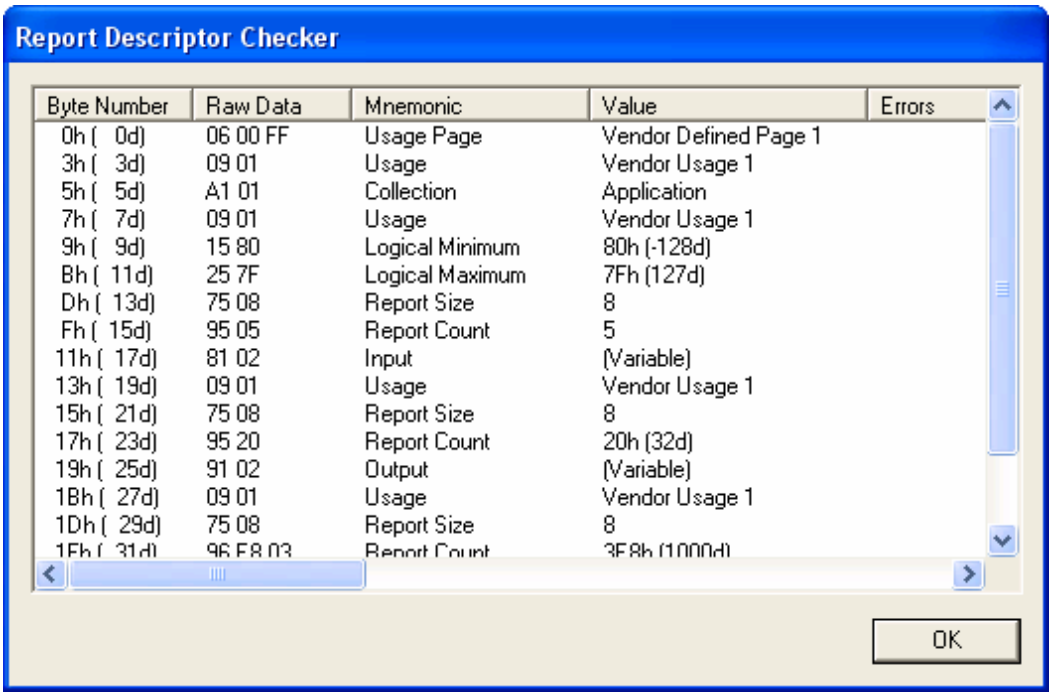


Figure 10.5 – HID Descriptor Tool 2.4 Result

10.4.2USB PROTOCOL ANALYSIS TEST

To test the ability of the device to comply with the USB 2.0 specifications Chapter 9 testing requirements, as well as the USB HID 1.1 specifications protocol requirements, the USB Implementers Forum released a USB Command Verifier, available free for download from their website, www.usb.org. The results of the Chapter 9 compliance and HID 1.1 compliance tests are given below in Figure 10.6 and Figure 10.7 respectively.

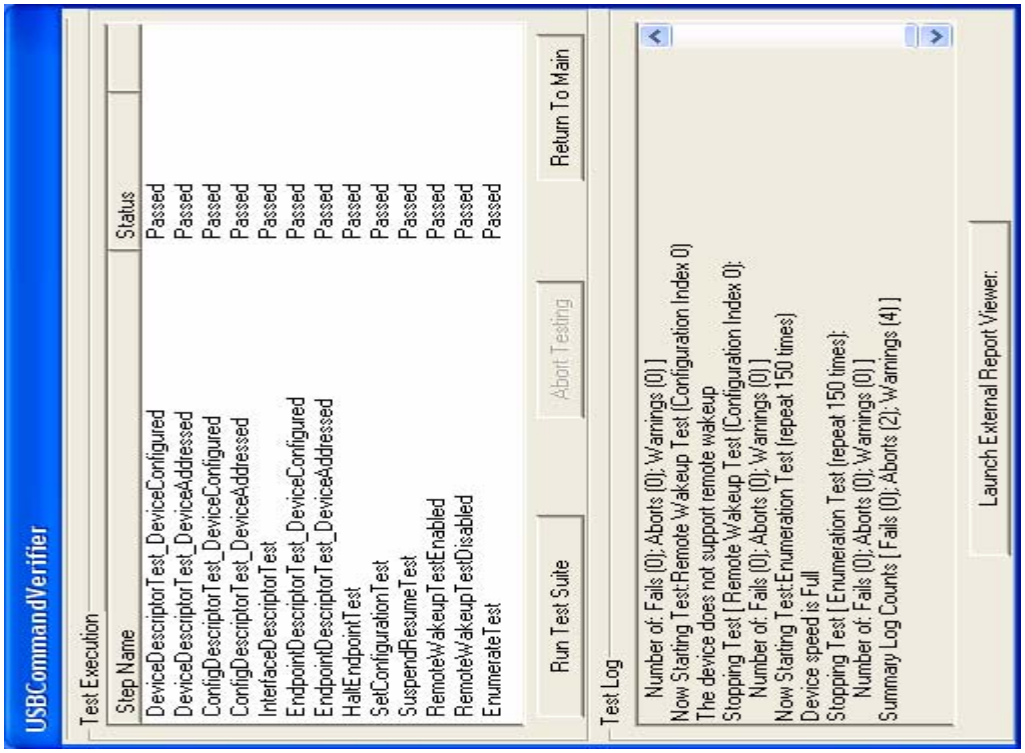


Figure 10.6 – USB 2.0 Chapter 9 Compliance Test Results

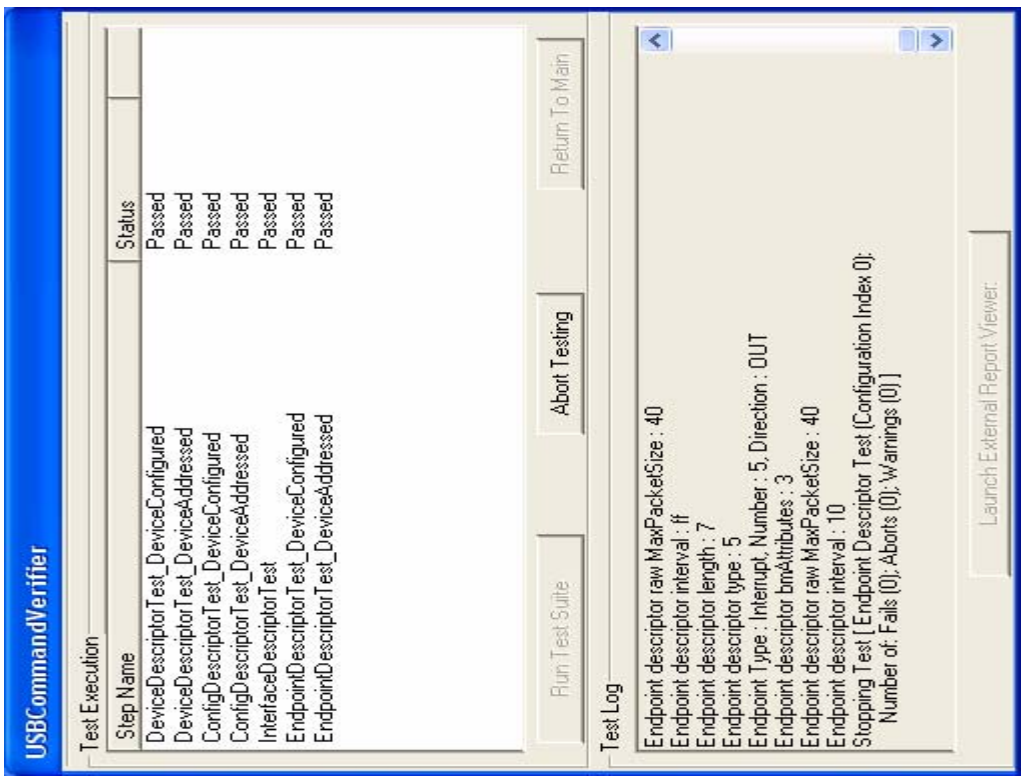


Figure 10.7 – USB HID 1.1 Compliance Test Results

11 REFERENCES

[1] USB Implementers Forum, Universal Serial Bus Specification [online] revision 2.0, 2000, USB Implementers Forum, Available From: http://www.usb.org/developers/docs/usb_20_02212005.zip [10th October 2005]

[2] USB Implementers Forum, Device Class Definition for Human Interface Devices (HID) [online] revision 1.11, 2001, USB Implementers Forum, Available From: http://www.usb.org/developers/devclass_docs/HID1_11.pdf [10th October 2005]

[3] USB Implementers Forum, HID Usage Tables [online] revision 1.12, 2005, USB Implementers Forum, Available From: http://www.usb.org/developers/devclass_docs/Hut1_12.pdf [10th October 2005]

[4] VASTIANOS, G., *16 Channel Servo Controller for Robotic Applications* [online], Seattle Robotics, Available From: http://www.seattlerobotics.org/encoder/200106/16csscnt.htm#0. [10th October 2005]

[5] Atmel Corp., 8-bit Flash Microcontroller with Full Speed USB Device AT89C5131-L [online] 2004, Atmel Corporation, Available From: http://www.atmel.com/dyn/resources/prod_documents/doc4338.pdf [10th October 2005]

[6] Atmel Corp., USB Microcontrollers AT89C5131 Errata Sheet [online] 2005, Atmel Corporation, Available From: http://www.atmel.com/dyn/resources/prod_documents/doc4380.pdf [10th October 2005]

[7] Atmel Corp., AT89C5131A Starter Kit – Hardware User Guide [online] 2004, Atmel Corporation, Available From: http://www.atmel.com/dyn/resources/prod_documents/doc4245.pdf [10th October 2005]

[8] Atmel Corp., AT89C5131A Starter Kit – Software User Guide [online] 2004, Atmel Corporation, Available From: http://www.atmel.com/dyn/resources/prod_documents/doc4246.pdf [10th October 2005]

[9] Microsoft Corp., HID Parser Functions [online] 2005, Microsoft Corporation, Available From:http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceddk5/html/wce50grfhidparserfunctions.asp [10th October 2005]

[10] AXELSON, J., 2001, *USB Complete* 2nd ed., Lakeview Research, Wisconsin

[11] National Semiconductors, CD4017BM/CD4017BC Decade Counter/Divider with 10 Decoded Outputs [online] 1988, National Semiconductors, Available From: http://www.national.com/ds.cgi/CD/CD4017BC.pdf [10th October 2005]

[12] National Semiconductors, 3A Low Dropout Positive Regulators [online] 2005, National Semiconductors, Available From: http://www.national.com/ds.cgi/LM/LM1085.pdf [10th October 2005]

[13] PATIN, F. *An Introduction to Digital Image Processing*. 2003

[14] DRISCOLL, M. T. *A Machine Vision System for Capture and Interpretation of an Orchestra Conductor's Gestures*.

[15] SOLIMAN, S & SRINATH, M, *Continuous and Discrete Signals and Systems*, 2nd edition, Prentice Hall, Upper Saddle River, New Jersey, 1998.

[16] *Computing The Output Of Convolution [Image]*. Retrieved 21 September, 2005 from: http://www.mathworks.com/access/helpdesk/help/toolbox/vipblks/2dconvolution.html

[17] *The Art of Assembly Language: Vol 4, Ch 11: The MMX Instruction Set*. http://webster.cs.ucr.edu/AoA/Windows/HTML/TheMMXInstructionSet.html

[18] Intel® Architecture Optimization Reference Manual, http://www.intel.com/design/pentiumii/manuals/245127.htm

[19] PATWARDHAN, B. *Introduction to the Streaming SIMD Extensions in the Pentium III*. http://www.x86.org/articles/sse_pt1/simd1.htm

[20] *Hypermedia Image Processing Reference: Connected Components Labelling*. Department of Artificial Intelligence, University of Edinburgh. http://www.cee.hw.ac.uk/hipr/html/label.html

[21] BRADLEY, J. M. *Mathematical Formulae and Statistical Tables for Tertiary Students*. Perth, Western Australia, 2001.

[22] Network Working Group, *Request for Comments: 1321. The MD5 Message-Digest Algorithm*, April 1992. http://rfc.net/rfc1321.html