A Control System and Application Program Interface (API) for an Elevation Over Azimuth Tracking Pedestal

James Callum Russell

A dissertation submitted to the Department of Electrical Engineering,
University of Cape Town, in partial fulfilment of the requirements
for the degree of Bachelor of Science in Electrical and Computer Engineering.

20 October 2008

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted in partial fulfilment for the degree of Bachelor of Science in Electrical and Computer Engineering at the University of Cape Town. It has not been submitted before for any degree or examination in this or any other university.

Signature of Author . . . .

Cape Town

20 October 2008

Abstract

This project concerns the design and implementation of a control system for an Elevation over Azimuth Tracking Pedestal as well as the creation of an Application Program Interface (API). The API provides a number of methods to control the movement and position of the pedestal on both its elevation and azimuth axes. The control algorithm used was created using the statistical analysis of key factors affecting the accurate positioning of the pedestal on its axes. The system was made to run in an Open Source environment (Ubuntu 8.04.1 LTS - Hardy Heron) with a Linux kernel version of 2.6.24-21. The accuracy of the azimuth axis positioning was calculated to be within 0.2915° with a standard deviation of 0.0544° and the accuracy of the elevation axis positioning was calculated to be within 0.0193° with a standard deviation of 0.0308°.

Acknowledgements

I would like to thank my family for their continued support and encouragement throughout my undergraduate degree at the University of Cape Town. Their advice and expert proof reading skills were invaluable to me during this project.

I would also like to thank Professor Inggs from the University of Cape Town for his supervision and guidance throughout this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background to Investigation

Many applications such as those in astronomy, radar and various military applications [6] require accurate tracking and positioning of objects in the sky. The use of an Elevation over Azimuth Tracking Pedestal is one method that accomplishes this. Elevation over Azimuth Tracking Pedestals have two axes of movement: azimuth (horizontal) and elevation (vertical). Its name stems from the fact that the elevation axis is situated above the azimuth axis. The Tracking Pedestal can rotate a full 360° on the azimuth axis but is usually limited to a 90° range on the elevation axis. This design has been in common use for decades, however it does have a major functional limitation. If the target being tracked passes directly over the pedestal, a 180° movement is required on the azimuth axis to continue tracking [7]. This means that the pedestal has to reacquire the target once the rotation is complete. Figure 1.1 shows the axes of movement of a typical Elevation over Azimuth Tracking Pedestal. The ϕ and θ symbols denote the angles of rotation from the 0° position for the elevation and azimuth axes respectively.

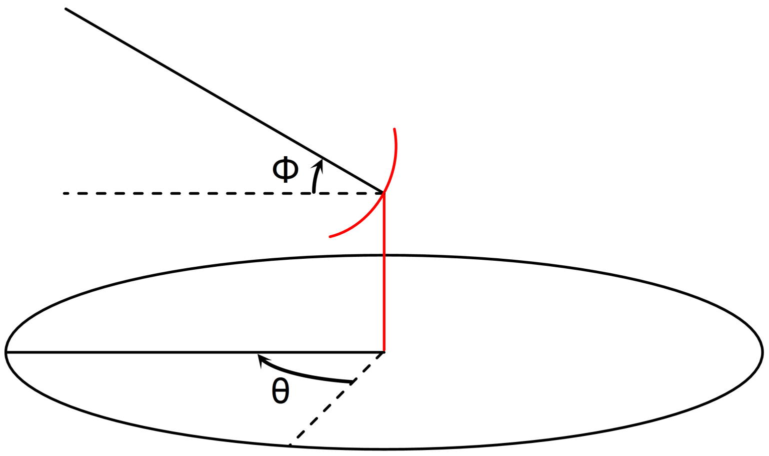

Figure 1.1: Axes of movement of a typical Elevation over Azimuth Tracking Pedestal

The Radar Remote Sensing Group at UCT has three Elevation over Azimuth Tracking Pedestals which were previously used on Navy ships to mount Electronic Warfare (EW) Radar Jammers. One of these pedestals was used for this project. Each axis of the pedestal used has a 28V DC motor to control the position of the antenna. Synchro Control Transformers are also present on each axis to calculate the angular position of the antenna. There is a single limit switch which is activated when the elevation axis moves out of bounds (i.e less than 0° or greater than 60°). The pedestal used for this project has a limited range of 60° on the elevation axis and if it was driven out of this range, it could be damaged.

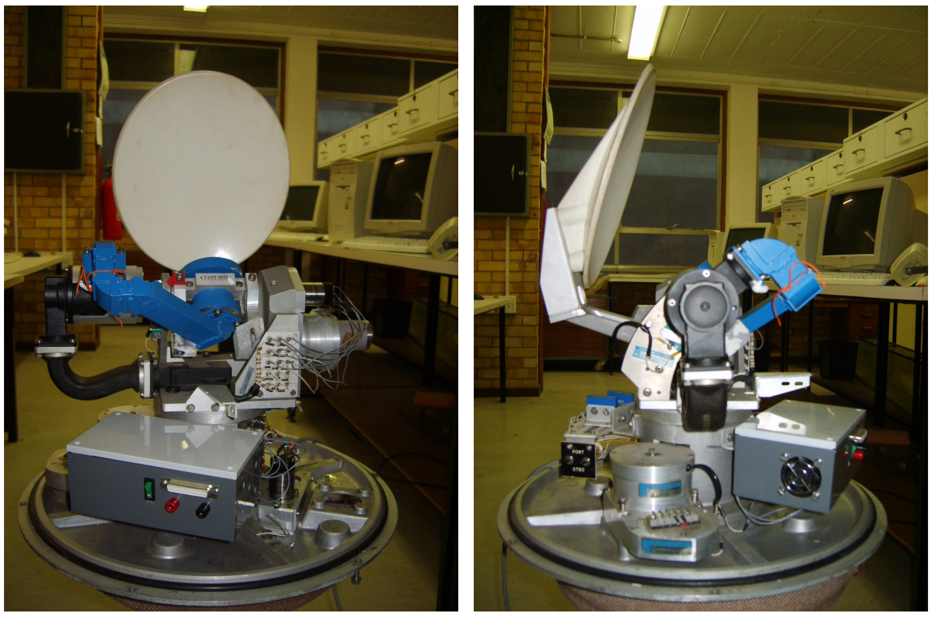
Figure 1.2: Elevation over Azimuth Tracking Pedestal

Previous students from UCT have used the pedestals for various projects, but have suffered inaccuracies in the movements of the axes or had trouble interfacing with the hardware. The author worked at the university for 6 weeks as per the requirements for the EEE3000X course (3rd year vacation work). The objective of this work was to correct problems had by Gregor George in his 2007 thesis ("*A Control and Measurement System for an Elevation over Azimuth Antenna Pedestal*" [8]), the most significant being the interfacing with the synchros. On completion of the work Prof. Inggs proposed this thesis as the knowledge gained would be invaluable in completing this project.

1.2 User Specification

The requirements of this project can be divided into the high-level user requirements and the more technical functional requirements. The user requirements specify the overall product to be produced, while the functional requirements give specifications relating to the way in which the system must function.

1.2.1 User Requirements

The requirements for this project are to produce hardware and an Application Program Interface (API) containing a collection of methods to allow effective control of the pedestal from a PC as well as allowing access to important data such as the current angular position. The movement of the pedestal must be as accurate as possible while still giving the best possible response. The API must be made for use in an open source environment.

1.2.2 Functional Requirements

The functional requirements of the system were split into the following 2 sections:

Control Circuitry

The control circuitry built for the pedestal must be able to:

1. Receive a digital input from the computer.
2. Use this data to control the speed and direction of the 28V motors.
3. Limit the movement of the pedestal on the elevation axis when the limit switch is activated.
4. Allow the elevation axis to move in the other direction out of the limited position.
5. Operate off a single 28V supply.

Application Program Interface

The API programmed for the pedestal must contain:

1. A public initialisation method to set up the 76CS1 Card and the data port used to connect to the control hardware.
2. Private methods to retrieve position and speed information from the synchros for each axis.
3. Private methods to control the movement of the pedestal by outputting the relevant data to control hardware.
4. Public methods to read information about the position and speed of the pedestal on each of its axes.
5. A public method to move the pedestal to a specific position with the best possible accuracy and the fastest response.

6. A public method to make the pedestal scan a specific angle range stopping at predefined intervals.
7. A public method to abort the current movement of the pedestal and stop its movement.

1.3 Project Scope

Figure 1.3 shows the proposed system. Due to time constraints, only the control system and the API were chosen for this project. The modules of the system which fall outside of this project scope are shown in red while those included in the project are shown in green.

This project does not include the modelling of the transfer function for the pedestal and thus does not use control methods to design a controller algorithm. An ad-hoc approach was used that relied on a robust software controller coupled with statistical data obtained from experimentation to give a very accurate and fast response.

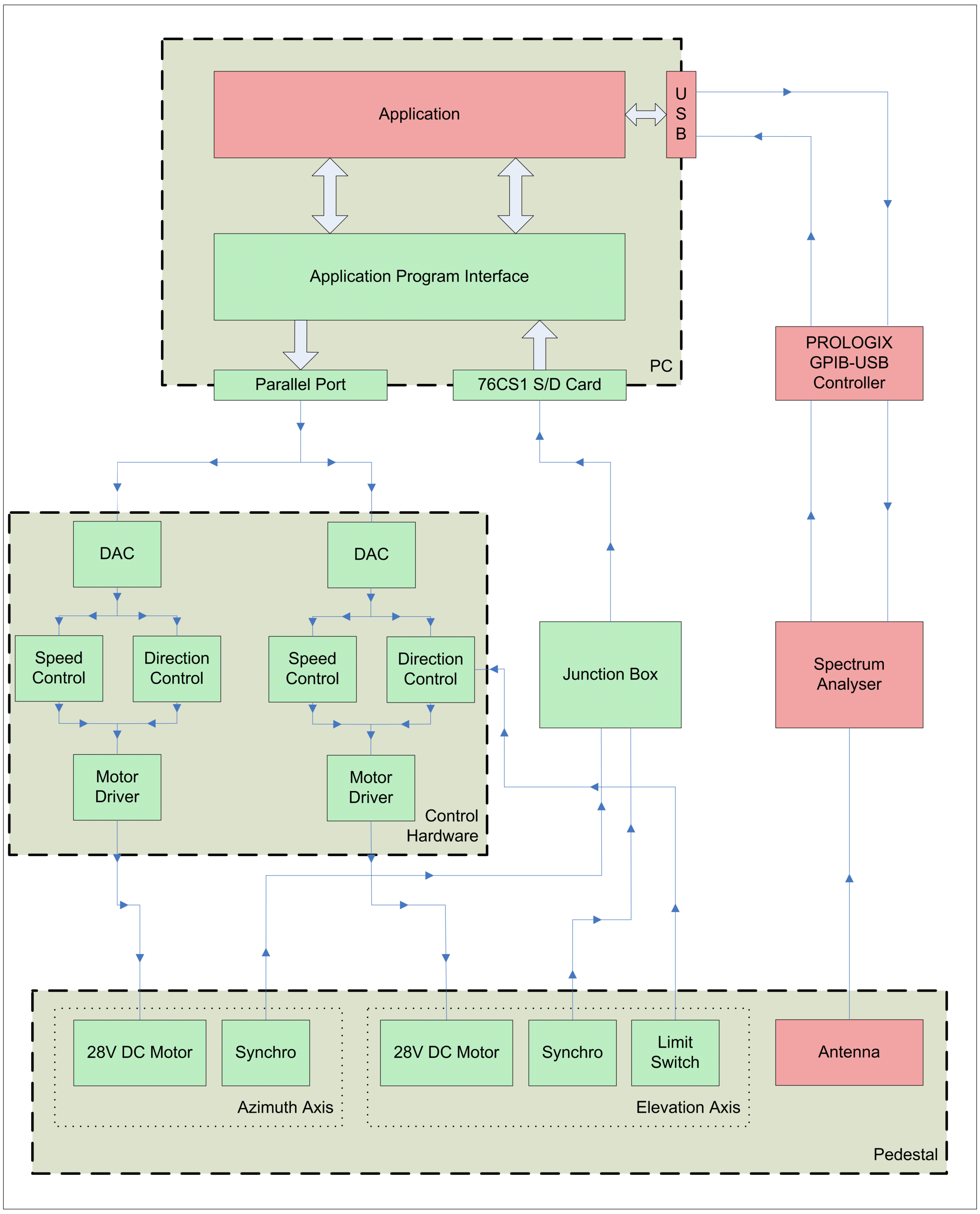

Figure 1.3: Proposed system showing scope of project

1.4 Requirements Analysis

From the user and functional requirements, the following observations can be made:

- Both the Parallel Port and the Universal Serial Bus (USB) are available to interface the computer with the control hardware.

- A Digital to Analogue Converter (DAC) must be built to change the digital signal from the computer port to an analogue voltage.
- The control circuitry must use this voltage to control the speed of the motor.
- Speed can be controlled by using Pulse Width Modulation (PWM) or variable voltage.
- The input to the control hardware must also be able to change the direction of each motor.
- The direction of each motor can be changed by using an H-Bridge.
- A suitable logic circuit must be implemented to allow the single Limit Switch to be used in both directions on the elevation axis.
- The synchros must be interfaced with the computer through the 76CS1 card.
- A suitable Linux distribution must be chosen in which to implement the API.
- Suitable drivers must be found to install the 76CS1 Card in Linux.
- Experiments must be conducted to determine the accuracy of the control system as well as the uncertainty in the measurements.

1.5 Dissertation Overview

CHAPTER 2 gives an overview of certain hardware on the pedestal, the 76CS1 PCI Synchro to Digital Card and the structure of the junction box used to connect the synchros to the 76CS1. Some problems involving the use of the synchros on the pedestal are shown and solutions proposed.

CHAPTER 3 contains the design and unit testing of the hardware for the control system. Two almost identical circuits were designed for the elevation and azimuth axes respectively. They both receive inputs from the Parallel Port and interpret these using a Digital to Analog Converter (DAC), this DAC output then sets the reference input of an operational amplifier Pulse Width Modulation (PWM) generating circuit. This PWM signal is used to control the speed of an LMD18200T H-Bridge Integrated Circuit (IC) which facilitates the bi-directional control of the motors. The elevation axis had the addition of a logic circuit to solve the problem of a single Limit Switch operating on both extremes of the axis. A 5V voltage regulator IC was used to provide a 5V supply for the PWM circuit and a 60mm fan. To ensure that every circuit would work, each module of the design was built and tested individually and then integrated into the bigger system. In order to simulate the Parallel Port input, a simple circuit was built using a 5V supply and 8 toggle switches which would send the appropriate high and low signals to the input of the DAC. This showed inadequacies in some approaches, especially in the use of the L298 Dual H-Bridge Driver IC.

CHAPTER 4 starts with the implementation of the hardware designed in Chapter 3 on a Printed Circuit Board (PCB). The hardware was then integrated with the pedestal and connected to the computer. Functional testing was performed to ensure that the total system functioned as predicted. The naming convention used for the different azimuth and elevation speeds is explained here.

CHAPTER 5 contains five experiments to which were used to help in the design of a software controller and also to quantify the error in the azimuth axis synchro readings due to backlash.

- **Experiment 1**: To determine the average overshoot for each azimuth axis speed over the full 360° range of movement.
- **Experiment 2**: To determine the average overshoot for each elevation axis speed over the full 60° range of movement.
- **Experiment 3**: To determine the average overshoot for each Software Pulse Width Modulated variation of the minimum azimuth axis speed over the full 360° range of movement.
- **Experiment 4**: To determine the average overshoot for each Software Pulse Width Modulated variation of the minimum elevation axis speed over the full 60° range of movement.
- **Experiment 5**: To approximate a value for the backlash in the gearing on the azimuth axis.

CHAPTER 6 shows the changes that were made to the 76CS1 drivers in order to have them compile on Ubuntu 8.04.1 with kernel version 2.6.24-21. It then gives a brief overview of how the thresholds were calculated for each speed. Two flow charts depicting the coarse motor speed control algorithm and the fine motor speed control algorithm are shown. A description of all the structures, variables and methods in the pedestal API is given.

CHAPTER 7 contains a number acceptance tests to evaluate whether the final system satisfies the user specification given in Chapter 1. A test to determine the positioning accuracy for each axis was also conducted. The accuracies were calculated to be:

- The azimuth axis had an average accuracy of 0.1023° with a standard deviation of 0.0544°.
- The elevation axis had an average accuracy of 0.0193° with a standard deviation of 0.0308°.

CHAPTER 7 analyses the results of Chapter 7 as well as other the functional test results from Chapter 4.

CHAPTER 8 draws appropriate conclusions from the testing conducted thoughout the project cycle. The following conclusions were made:

1. The accuracy of the movement on the azimuth axis was calculated to be 0.1023° with a standard deviation of 0.0544°. This measure could be improved by increasing the number of measurements taken.

2. The accuracy of the movement on the elevation axis was calculated to be 0.0193° with a standard deviation of 0.0308°. This measure could be improved by increasing the number of measurements taken.

3. The backlash in the azimuth gearing was approximated to be 0.1892°. This gave an uncertainty in the synchro reading of 0.1892°. Thus the total accuracy of the azimuth axis positioning was calculated to be 0.2915° with a standard deviation of 0.0544°.

4. This approximation for backlash was not very accurate as a lot of opportunities were available in the testing to introduce human error into the measurement.

5. The use of a non real-time Operating System for the API contributed to the error in the movements as real-time measurements could not always be obtained.

6. The controller implemented only used the current position of the pedestal to control the speed of movement. Thresholds were also defined according to experimental data which only related to the exact pedestal configuration used. If the weight of the antenna is changed, the dynamics of the system will be altered and the controller thresholds would be invalid.

7. The maximum speed of movement for the azimuth axis was not realised due to the azimuth control hardware not being able to output a 100% duty cycle due to the 3.7V Parallel Port inputs.

8. There are many non-linearities in the movement of each axis due to the wear of the gearing systems. Without conducting appropriate tests and deriving a control transfer function, the dynamics of the system cannot be predicted and a controller with the maximum possible accuracy and minimum response time cannot be constructed.

9. The overshoot for each speed was averaged over both directions, this posed a problem for the elevation axis as the weight of the antenna changes the overshoot depending on the current position. Thus the results for each direction were very different and averaging them gave the best solution for the controller designed in this project, but limited the response of the system because undershoot and overshoot occurred often and the fine motor control method had to recover from this error.

CHAPTER 9 gives recommendations to increase the functionality of the system. These recommendations correspond directly to increasing the accuracy and response of the system. The following recommendations were made for future work:

1. Use commercial backlash measuring equipment and procedures to obtain a very accurate measurement for the backlash in the gearing systems.
2. Use a real-time Operating System like RTLinux to allow for process priorities and scheduling algorithms for the system to be set. This will ensure the system receives real-time measurements from the synchros and can perform very accurate control.
3. Increase the resolution (number of bits) of each DAC. This will enable the system to run at more speeds and thus give better control. A separate Parallel Port for the elevation and azimuth axes would be best.
4. Use a driver circuit to ensure that the logic HIGH voltage from the Parallel Port enters the DAC as a 5V signal.
5. Model the movement of the axes using control theory to obtain a transfer function for the pedestal. This will allow for a controller which uses both speed and position to control the movement of the axes.

Chapter 2

Description of Existing Hardware

This chapter describes the already existing hardware that was used in this project. The synchros, Limit Switch and mechanical gearing all relate to the tracking pedestal whereas the 76CS1 Digital to Synchro Measurement card was a PCI card used in the computer to interpret the synchro outputs.

2.1 Synchros

Synchos have been used as part of electromechanical shaft angle positioning and measurement systems for over 50 years with very good reliability and cost effectiveness [9]. A synchro is a name given to a family of instruments which function as rotating transformers. These devices take in one or more time varying voltages and output another set of voltages whose phase and magnitude are used to determine the angular position of the shaft [10]. As the shaft of the synchro is turned, the angular position of its rotor winding in relation to its stator windings is represented by the AC voltage coupled from the primary winding to the secondary windings (rotor to stator in a Control Transmitter and stator to rotor in a Control Transformer) [11].

2.1.1 Control Synchros

Control synchros are used in servo systems to provide and deal with control signals where accurate angular transmission to a mechanical load is required. Control synchros only function as passive positioning or measurement devices and do not handle any motive power for driving the load [10].

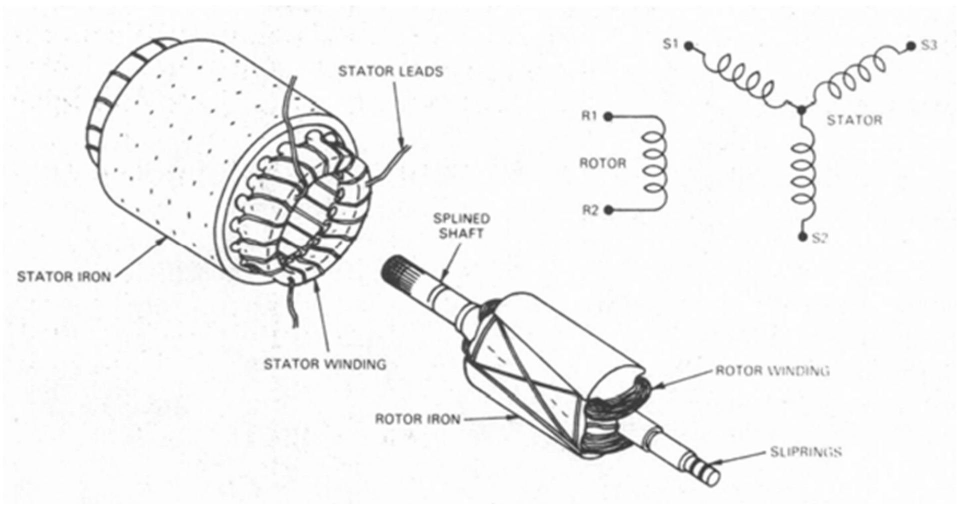

Figure 2.1: Internal structure of a synchro control transmitter and its electrical representation (taken from [1])

As shown in Figure 2.1, there are three stator windings which are connected in a star fashion 120° apart. These three windings are then brought directly to the output terminals S1, S2 and S3. The rotor is an extension of the shaft and its winding is usually connected via slip rings and brushes to the terminals R1 and R2 [10].

This arrangement forms a rotary transformer which if excited with an AC voltage (normally 60 Hz or 400 Hz) on the rotor, will induce corresponding voltages on the 3 S terminals corresponding to the angular position of the rotor [10].

If the rotor of a synchro transmitter is excited by applying a reference voltage across terminals R1 and R2 of the form [10]:

$$V_{ref} = A \sin \omega t$$

The voltages which will appear across the stator terminals will be [10]:

$$V_{S1-S3} = A \sin \omega t \sin \theta$$

$$V_{S3-S2} = A \sin \omega t \sin(\theta + 120^\circ)$$

$$V_{S2-S1} = A \sin \omega t \sin(\theta + 240^\circ)$$

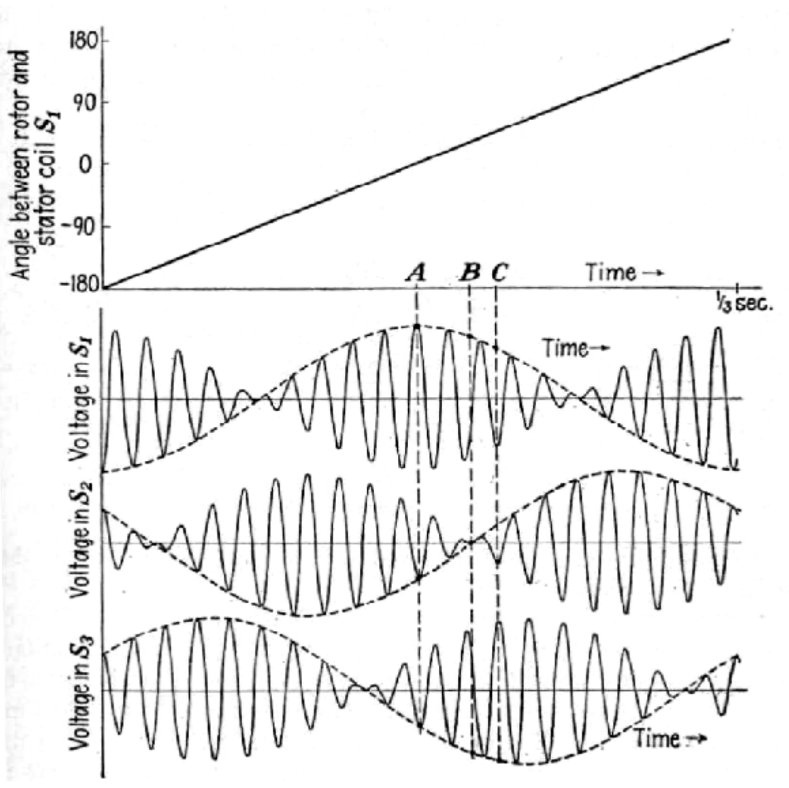

Figure 2.2: Synchro output voltages as shaft rotates at 3 RPM (taken from [1])

From Figure 2.2 it can be seen that the amplitude of the output voltage on the stator terminals corresponds to the amplitude of an envelope sinusoid ($\sin\theta$, $\sin(\theta+120°)$ & $\sin(\theta+120°)$) that the output voltages follow when the shaft is rotated.

2.1.2 Types of Control Synchros

The two most common types of control synchro are the Control Transmitter (CX) and the Control Transformer (CT) [10].

Control Transmitter

The rotor terminals of a CX are exited with an AC voltage and a corresponding voltage is induced on the stator terminals [10].

Control Transformer

A CT differs from a CX in that its stator terminals are excited with AC voltages. These voltages differ in phase by 120° and correspond to the required angular position of the shaft. An error signal is induced on the rotor terminals to signal that the position of the shaft does not match the AC voltage inputs on the stator terminals. As the shaft is turned, the error signal changes until a null voltage is present when the position of the shaft exactly matches the AC input voltages on the stator terminals [10].

2.1.3 Synchros on the Pedestal

The tracking pedestal uses 2 synchros, one on the azimuth axis and one on the elevation axis. These synchros are Control Transformers, which are usually used to receive a positioning signal from a CX and transmit an error signal to a control system corresponding to the difference between the shaft position and the required position. The 76CS1 card owned by UCT only has channels which can read in a signal from a CX.

This means that the CT's on the pedestal must be run backwards to essentially make CX's. This works in the same way as a regular CX, except that the reference voltage must be carefully chosen because the voltage on the R terminals is now stepped up to make the voltage on the S terminals (instead of stepped down in a regular CX). Thus for an output of 11.8Vrms as required by the 76CS1 [5], the reference voltage must be set at approximately 3.6V.

The ideal reference voltage can be checked by measuring the Vrms across S1 and S3 and turning the rotor until a maximum voltage is seen. This maximum voltage must be as close to, without exceeding, 11.8Vrms.

2.2 The 76CS1 Synchro to Digital Card

The 76CS1 is a PCI Synchro/Resolver Measurement and Simulation Board and has 8 S/D (Synchro to Digital) and 6 D/S (Digital to Synchro) channels [5]. The S/D channels read in the 3 stator voltages (S1, S2 and S3) as well as the reference voltage used (R1 and R2) and allow a user to display the angular position of a synchro. The D/S channels output 3 stator voltages (S1, S2 and S3) and read in the angular error (R1 and R2).

The part number of the 76CS1 card acquired by UCT is:

76CS1-040CSA-01

The part number indicates the following configuration:

76CS1–

04	4 channels of S/D
0	(0 D/S channels) Synchro format
C	Commercial Temp environment
S	Synchro format (all channels)
A	Programmable reference supply (0-28 Vrms, 360 - 10 kHz)
-01	Low voltage SYNCHRO input (11.8 Vrms signal input, 26.0 Vrms Reference input / Calibrated and tested @ 400 Hz) **[12]**

Thus only 4 S/D channels are available and require an ideal 11.8 Vrms signal input and at most a 26 Vrms reference input

2.2.1 Connecting Synchros to the 76CS1

The card uses a 78-pin D connector to interface with synchros. Table 2.1 shows the pin configuration of the 78-pin D connector in order to connect synchros to the 76CS1 card.

	Ch. 1 S/D	Ch. 2 S/D	Ch. 3 S/D	Ch. 4 S/D
S1	39	18	36	15
S2	58	76	55	73
S3	78	57	75	54
S4	19	37	16	34
RHi	38	17	35	14
RLo	77	56	74	53

Table 2.1: 76CS1 Pin Configuration for S/D Channels [5]

- RHi output on pin 21
- RLo output on pin 60

The reference signal that feeds the synchro must also feed back to the card RHi and RLo inputs for the specific channel, even if the 76CS1 card supplied the RHi and RLo signals. If this is not done there will be a phase lock problem. This would cause unstable measurements and random 180° shifts in the angle measurement [12].

2.3 The Elevation Axis limit switch

There is a single limit on the elevation axis which is used to limit the movement in both directions. Two metal arms are mounted on the axis of rotation 60° apart. When the axis moves to either side of the 60° boundary, one of the metal arms presses a switch mounted nearby. Figure 2.3 shows how this is implemented.

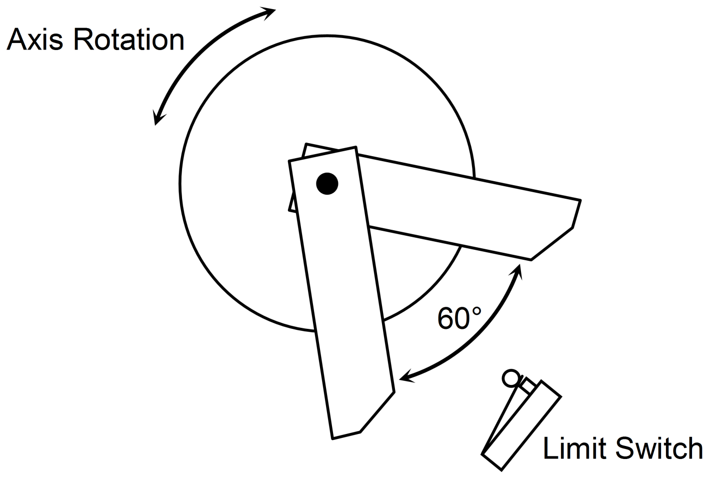

Figure 2.3: Limit switch implementation

2.4 Mechanical Gearing

The tracking pedestal uses gears in specific ratios to allow the motors to turn the pedestal on its axes and to ensure that one rotation of the pedestal on the azimuth axis corresponds to one rotation of the synchro monitoring the axis. A problem with this is that the physical coupling between two gears causes degradation. This degradation leads to a problem referred to as Backlash.

Backlash

The American Meteorological Society (AMS) defines backlash as the play or loose motion in an instrument due to the clearance existing between mechanically contacting parts [13]. In the pedestal's gearing system, this applies to the gap between the teeth of two contacting gears. This gap will cause a lag time from when the motor begins to spin, the pedestal begins to rotate and the synchro reading begins to register this movement. The synchro could also settle at a different position to that of the pedestal axis, this error in position is equal to the gap between the contacting teeth of the gears. In order to get a measure of the accuracy of the pedestal movements, the backlash in the gearing of the azimuth axis must be measured. No backlash measurements need to be done for the elevation axis as the synchro is coupled directly to the elevation axis of rotation and no gearing is used to connect the two.

Chapter 3

Hardware Design and Unit Testing

This chapter describes the design and unit testing aspects of the control hardware. It covers the details and testing of

- the computer to control hardware interface;
- the control hardware; and
- the motor control.

3.1 Unit Testing

> "*The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the interfaces between modules. Unit testing has proven its value in that a large percentage of defects are identified during its use.*" Microsoft Developer Network [14]

This software testing methodology was adapted and applied to the testing of the individual hardware units in the system. These units were designed, constructed and tested and only when all the units were complete were they integrated with eachother.

3.2 Interface Between the Computer and the Control Hardware

Two computer ports were available to communicate with the control hardware:

- The Universal Serial Bus (USB); and

- The Parallel Port

The Parallel Port was chosen as the interface between the computer and the control hardware. This was due to its eight available bits to transfer data and the ease in which it can be accessed from software.

3.3 Simulating the Parallel Port

In order to allow for testing of the hardware during the design stage, it was necessary to simulate the Parallel Port output. A simple circuit was constructed using a 5V supply and eight toggle switches to simulate the different output signals from the Parallel Port. Light Emitting Diodes (LED) were added to the design to give a visual representation of the output. The circuit diagram of the Parallel Port simulator is shown in Figure 3.1.

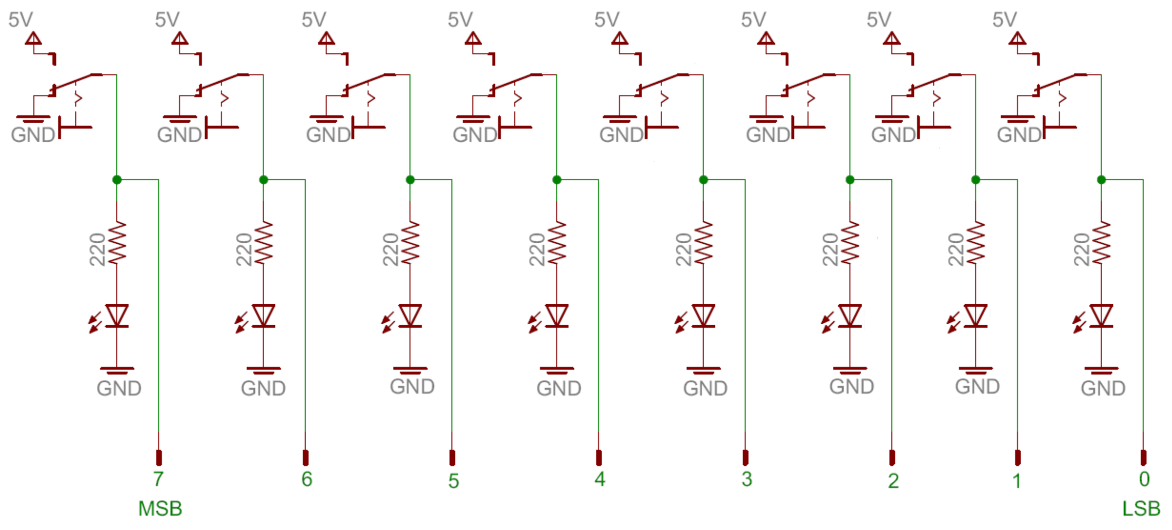

Figure 3.1: Parallel Port simulation circuit

3.4 Interpreting the Computer Output

The 8-bit signal from the Parallel Port was divided into two sets of 4-bit signals. Each signal controlling the motor on one of the axes. The three Least Significant Bits (LSB) in each 4-bit signal would control the speed of the motor and the Most Significant Bit (MSB) would determine the direction of the motor. This is illustrated in Figure 3.2.

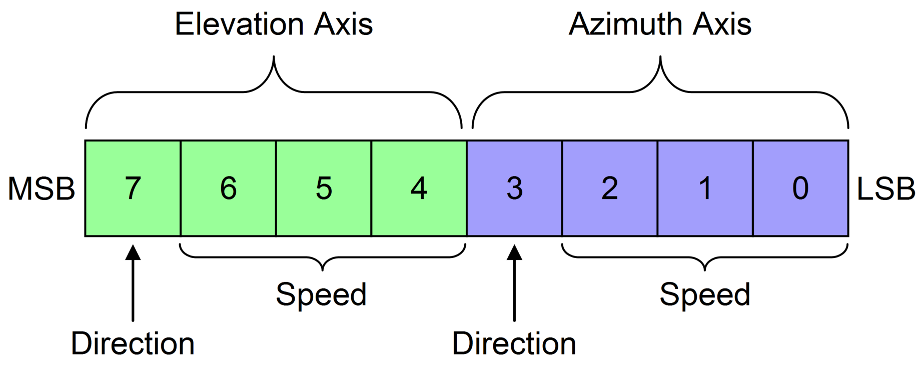

Figure 3.2: Parallel Port output to DAC input

In order to use each of the 3-bit speed inputs, two 3-bit Digital to Analogue Converters (DAC) were constructed. Off the shelf DAC Integrated Circuits (IC) were available but only with a greater number of inputs (i.e. more bits). A simple R-2R resistor configuration was used for each DAC and can be seen in Figure 3.3. Each of the inputs to the elevation axis DAC was controlled through an AND gate with a $\overline{STOP}$ input. This $\overline{STOP}$ input will come from the limit switch logic circuit and facilitates the limiting of the elevation axis movement. When the $\overline{STOP}$ input was set to 0V all DAC inputs were then set to 0V.

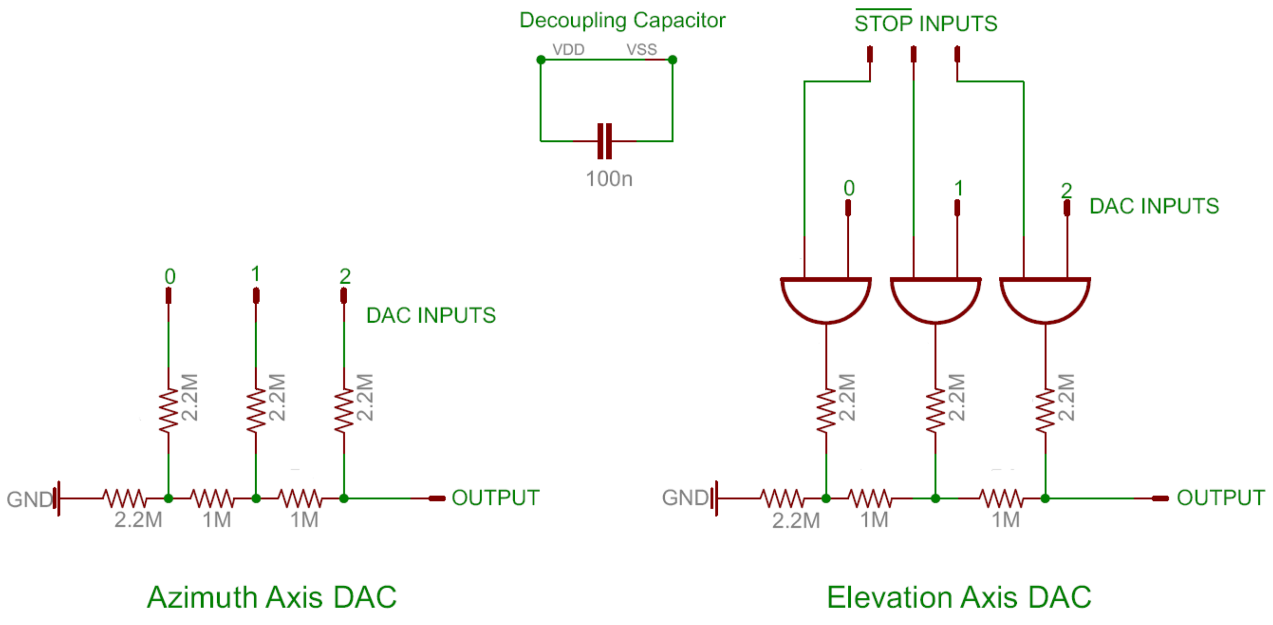

Figure 3.3: DAC circuit diagram

3.4.1 DAC Unit Test

The inputs in Table 3.1 were simulated using the Parallel Port simulator described in Section 3.3.

	Elevation Axis ($\overline{STOP} = 111$)	Azimuth Axis
Binary Input	Output (V)	Output (V)
000	0.05	0.047
001	0.54	0.62
010	1.16	1.145
011	1.45	1.744
100	2.205	2.173
101	2.52	2.775
110	3.18	3.33
111	3.5	3.93

Table 3.1: Simulated DAC inputs and corresponding outputs

With $\overline{STOP} = 000$, the output voltage for the elevation axis DAC was 0.047V for all binary inputs.

3.5 Controlling the speed of the motors

Two methods for controlling the speed of the motors were investigated:

- Variable voltage; and
- Pulse Width Modulation (PWM)

3.5.1 Variable Voltage Controller

The variable voltage controller was designed and constructed using a LM358 single supply operational amplifier (op-amp) and a TIP122 Darlington transistor to source the current as shown in Figure 3.4. This gave a 0-28V output for the corresponding 0-5V input from the DAC. A problem with this approach is that power is wasted because the circuit steps down the 28V supply voltage to the specified output and the voltage not applied to the motor is dissipated as heat.

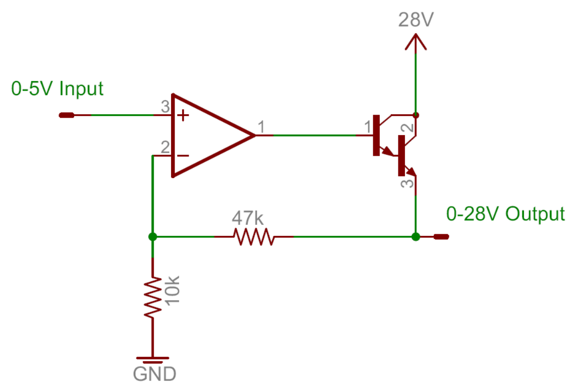

Figure 3.4: Variable voltage controller

3.5.2 PWM Controller

A PWM controller produces a chopped output to the motor of maximum voltage (28V) and varies the duty cycle of its output in order to vary the speed. This method uses power very efficiently as the full supply voltage is only used during every pulse which gives an average voltage output relating to the duty cycle of the PWM Controller. Thus the voltage is stepped down by essentially turning the supply voltage on and off and power is not wasted. For this reason, PWM was chosen as the method to control the motor speed. The PWM circuit was constructed using four op-amps (a single LM324 quad op-amp [15]), a single transistor (2N2222 [16]), resistors and capacitors (adapted from [2]). The circuit diagram can be seen in Figure 3.5.

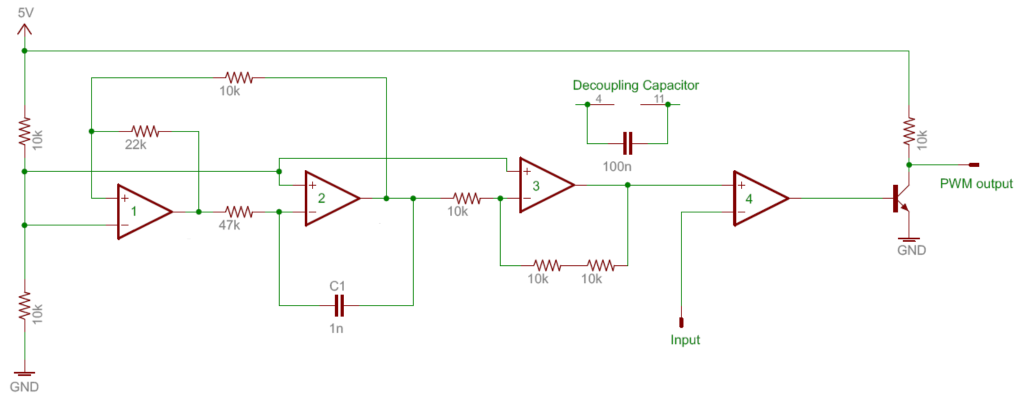

Figure 3.5: PWM circuit [2]

- Op-amps 1 and 3 act as a Schmitt Trigger and Miller Integrator and produce a saw-tooth waveform [2].
- Op-amp 3 acts as a low gain amplifier and moves the bottom of the waveform to just above 0V.
- Op-amp 4 functions as a comparator and compares the waveform outputted by op-amp 3 to the PWM Controller input voltage. If the waveform voltage is greater than the input voltage, the output of op-amp 4 is high, otherwise the output is low. This effectively outputs a square wave with a duty cycle that varies according to the PWM Controller input voltage.
- This output drives a transistor with a pull-up resistor which inverts the square waveform from op-amp 4 and moves the peak of the waveform to 5V and the trough to 0V to ensure each state is recognised as a logic HIGH and LOW voltage respectively.

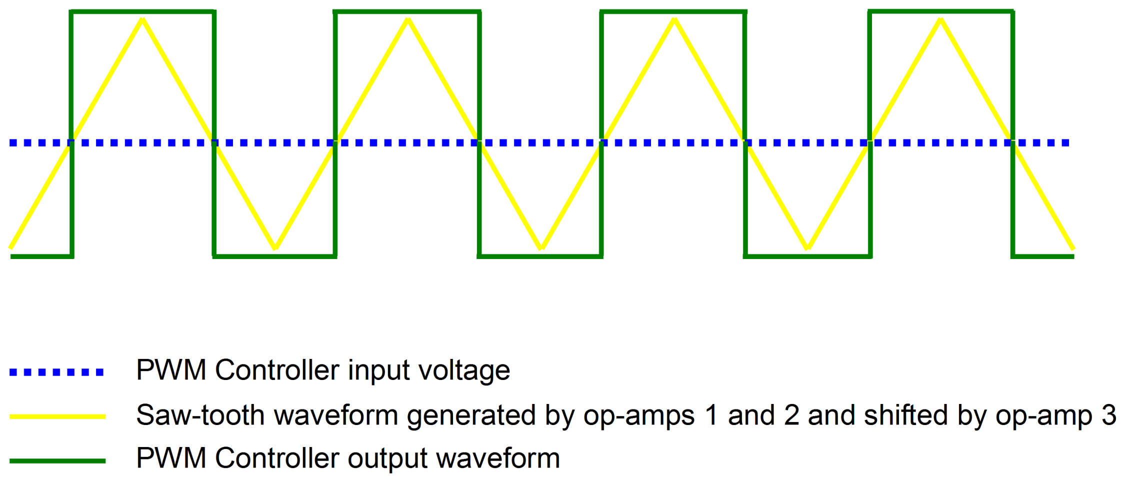
PWM Controller input voltage
Saw-tooth waveform generated by op-amps 1 and 2 and shifted by op-amp 3
PWM Controller output waveform

Figure 3.6: Visual representation of how the PWM controller works [2]

3.5.3 Software PWM

If the smallest speed obtained from the PWM controller is not low enough to facilitate accurate positioning of each axis, a method of further reducing the duty cycle of the motor output will be needed. Software Pulse Width Modulation can be used to achieve this.

Software PWM involves modulating the output of the Parallel Port with varying duty cycles. The Parallel Port is set to the lowest speed output for a certain amount of time then set to 000 for another period of time. The time that the port is set to the lowest speed in relation to the time that it is set to 000 determines the duty cycle of the outputted signal. This duty cycle from the Parallel Port will combine with the duty cycle of the PWM Controller to output a voltage to the motor with a duty cycle equal to the product of the two duty cycles. Figure 3.7 demonstrates the combining of the duty cycles.

3.5.4 PWM Unit Test

Table 3.2 shows how the duty cycle of the output to the motor changes as the input to the PWM controller is varied. The inputs correspond to the outputs generated in Section 3.4.1.

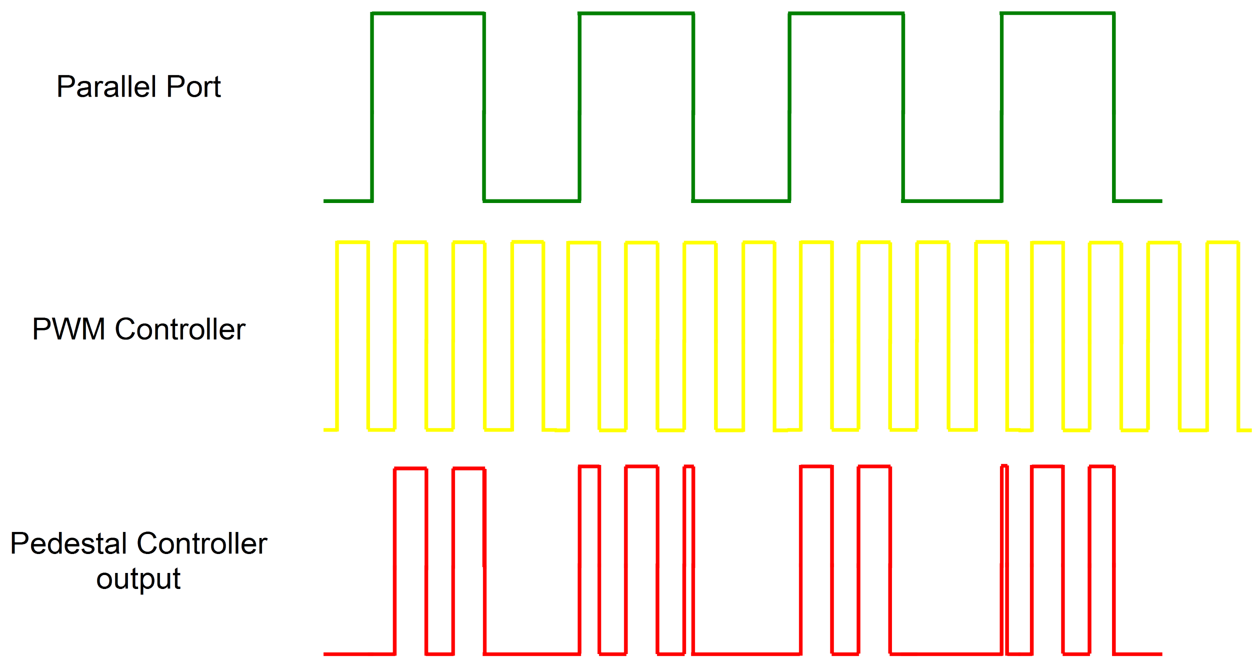

Figure 3.7: Combined duty cycle output

	Azimuth Axis
PWM Input (V)	Output +ve Duty Cycle (%)
0.05	0
0.54	32
1.16	43.5
1.45	58.9
2.205	69.9
2.52	85.2
3.18	100
3.5	100

	Elevation Axis ($\overline{STOP} = 111$)
PWM Input (V)	Output +ve Duty Cycle (%)
0.047	0
0.62	0
1.145	49.8
1.744	56.3
2.173	74.5
2.775	82.2
3.33	100
3.93	100

Table 3.2: PWM circuit inputs and corresponding outputs

With $\overline{STOP} = 000$, the output +ve duty cycle from the elevation axis PWM Controller was 0% for all voltage inputs.

3.6 Bi-directional Motor Control

In order to facilitate bi-directional control of the motors, a method was needed to reverse the direction of the current flow through each motor. This can be achieved using an H-Bridge circuit. It is possible to build an H-Bridge circuit using discrete components. However, it is easier to use off the shelf IC's which do not require sophisticated timing circuitry to prevent a short circuit. Short circuits can occur in H-Bridges if two in-line transistors are on at the same time. Figure 3.8 shows a basic H-Bridge circuit.

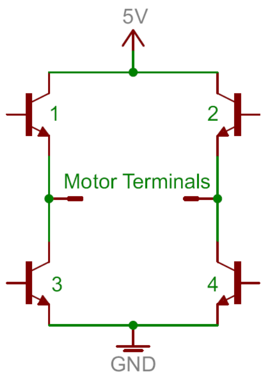

Figure 3.8: Basic H-Bridge circuit

Transistors 1 and 4 are turned on for current flow in one direction (transistors 2 and 3 are off) and transistors 2 and 3 are turned on for current flow in the opposite direction (transistors 1 and 4 are off). If either transistors 1 and 3 or 2 and 4 are on at the same time, a short circuit will occur (called shoot through current). In addition to protective logic to ensure that two in-line transistors are not turned on at the same time, a timing circuit is needed to ensure that sufficient time is given for one set of transistors to turn off before the other set is turned on. For this reason two H-Bridge IC's were investigated:

- The L298 Dual H-Bridge; and
- The LMD18200T H-Bridge.

3.6.1 L298 Dual H-Bridge IC

The L298 is a high voltage, high current dual H-Bridge IC. It accepts standard Transistor-Transistor Logic (TTL) voltage levels and also has an additional supply input to allow for lower voltage logic signals. Two enable inputs are available which enable or disable the device independently of the other input signals. The emitters of the lower transistors of each bridge are connected together and can be connected to ground or used for the connection of an external current sensing resistor. The L298 is designed to drive inductive loads such as DC motors, stepping motors, relays and solenoids [3].

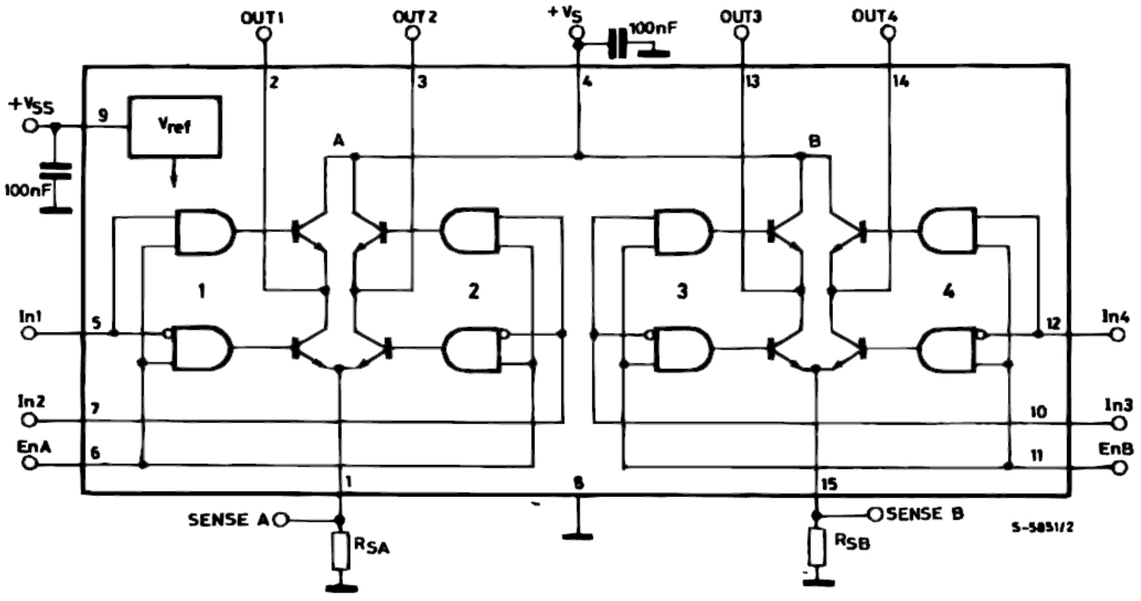

Figure 3.9: L298 block diagram [3]

- The In1, In2 and In3, In4 inputs were used to control the direction of the first and second H-Bridges respectively.
- The EnA and EnB lines were each connected to an output of one of the two PWM generating circuits.
- The Vss input is used as a reference for the AND gates in order to function for both TTL and lower voltage logic signals and was connected to 5V.
- The SENSE A and SENSE B pins were connected to ground as no monitoring or regulation of the motor current was needed.
- Protection diodes were placed from each motor output to ground and the supply in order to dissipate the back-emf generated by the turning motor.

L298 Unit Test

This unit test consisted of setting the PWM duty cycle and monitoring the motor output lines with an oscilloscope to see the waveform generated. The motors were connected and the speed observed. At low speeds the motor exhibited inconsistent movement and when monitored with the oscilloscope a variation in the duty cycle of the waveform on the motor leads was seen. The waveform also had an irregularly long fall time in relation to its frequency. At a low frequency of 125 Hz (8ms period) a fall time of 1.5ms was observed. The 1.5ms fall time persisted when the frequency of the PWM circuit was increased, this therefore would not allow the frequency to be increased any further as the pulse would not reach its trough before the leading edge of the next pulse. This slow fall time was thought to be due to the internal logic switching time or the protection diode recovery time. The switching time of the internal logic could not be changed for obvious reasons. The diodes were replaced with faster Shottky diodes however no improvement was seen. For these reasons, it was decided to use a different H-Bridge IC.

3.6.2 LMD18200 H-Bridge

The LMD18200 is a 3A single H-Bridge IC and is designed for motion control applications. It includes a PWM input to control speed, a Brake input which connects the output terminals together, a Current Sense output which allows for the connection of an external current sensing resistor and a Thermal Flag output which is raised when the IC exceeds a certain temperature and shuts down. The device is built using a multi-technology process which combines bipolar and Complementary Metal Oxide Semiconductor (CMOS) control circuitry with Double-diffused Metal Oxide Semiconductor (DMOS) power devices on a single IC [4].

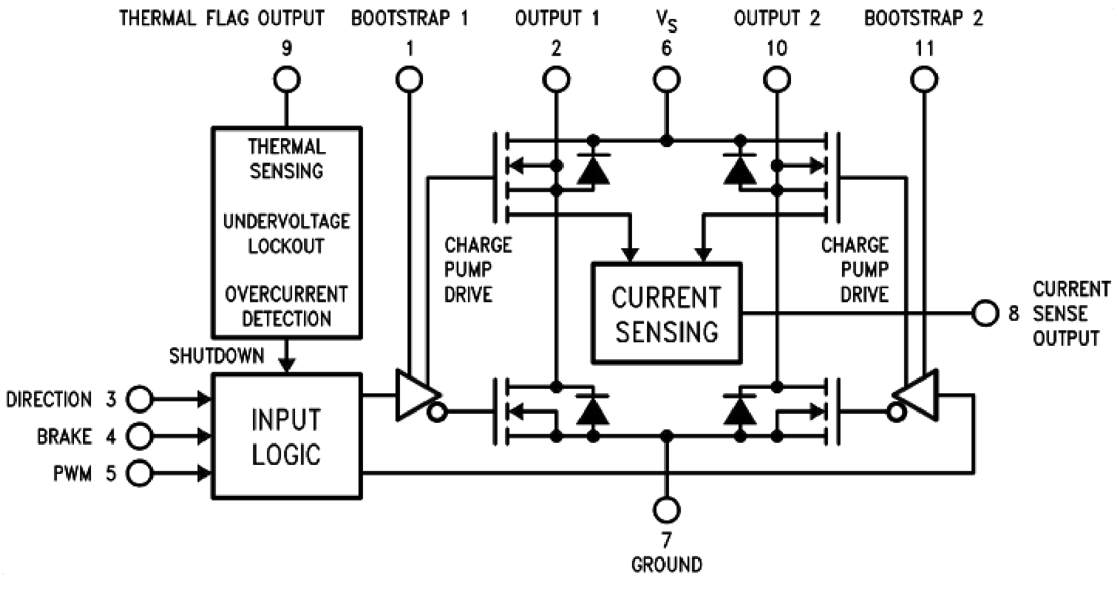

Figure 3.10: LMD18200 block diagram [4]

- The DIRECTION input was used to control the direction of the H-Bridge.
- The BRAKE input was not used and was connected to GND.
- The PWM input was connected to the output of the PWM generating circuit.
- The CURRENT SENSE OUTPUT was not needed and was connected to GND.

- The THERMAL FLAG OUTPUT was left unconnected.
- A 10nF capacitor was placed from BOOTSTRAP 1 to OUTPUT 1 and from BOOTSTRAP 2 to OUTPUT 2. This allows the MOSFET's to switch at frequencies above 1kHz [4] (up to 500kHz [17]).

LMD18200 Unit Test

The method for this unit test was identical to that for the L298. However, the LMD18200 gave a stable, constant square wave output to the motor with a duty cycle matching that of the PWM signal.

3.7 Limit Switch Logic

As discussed in Section 2.3, a single Limit Switch exists on the elevation axis. A suitable logic circuit was required to ensure that only a single direction of motion is allowed at each limit. RS Flip Flops were needed for the system to "remember" which direction triggered the Limit Switch to ensure that changing the direction while in a limited position does get interpreted as a limit condition for the other direction. Table 3.3 shows a state chart for the required logic circuit, where:

- X0 and X1 are RS Flip Flops
- Lim is 1 when the Limit Switch is pressed and 0 otherwise.
- Dir is the direction input for the controller
- $\overline{STOP}$ will go directly to the $\overline{STOP}$ input for the elevation axis DAC.

Present State		**Input**		**Next State**		**Output**
X0	**X1**	**Dir**	**Lim**	**X0**	**X1**	**$\overline{\text{STOP}}$**
0	0	0	0	0	0	1
0	0	0	1	1	0	0
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	1	1
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	1	0	0
1	1	1	0	0	0	1
1	1	1	1	1	0	1

Table 3.3: State chart for limit switch logic circuit

From Table 3.3 the following relationships were defined:

$$X0 = \overline{Dir}.Lim.\overline{X1}$$

$$X1 = Dir.Lim.\overline{X0}$$

$$X0_{reset} = \overline{Lim}$$

$$X1_{reset} = \overline{Lim}$$

$$\overline{STOP} = Dir.X0 + \overline{Dir}.X1 + \overline{Lim}$$

From these relationships, the circuit depicted by the circuit diagram in Figure 3.11 was built.

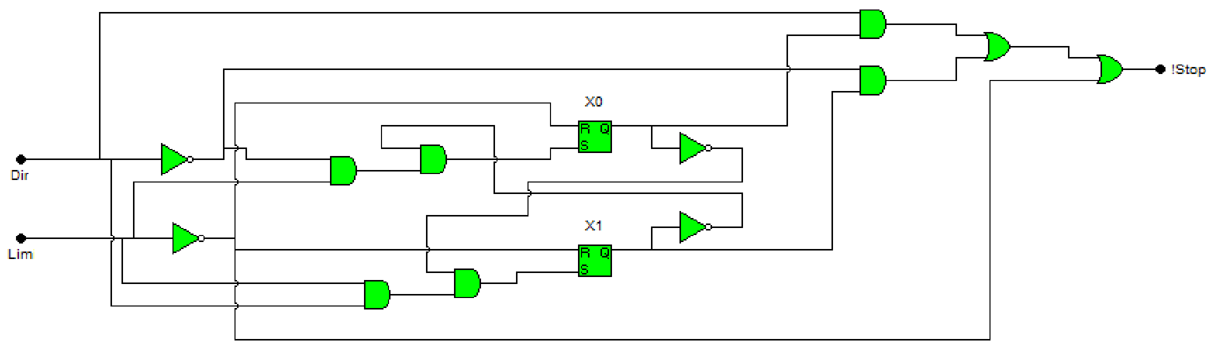

Figure 3.11: Limit switch logic circuit

3.7.1 Logic Circuit Unit Test

The inputs (Dir and Lim) were varied to match each of the cases in the State Chart. The intermediate states for X0 and X1 as well as the output ($\overline{STOP}$) matched the predicted values.

3.8 Voltage Regulator

The PWM generator and the Limit Switch logic circuit both need a 5V supply to operate. To satisfy this requirement whilst still running the system off a single 28V supply, a 5V voltage regulator was needed. A 7805 5V voltage regulator IC was chosen due to it ready availability. Because the input voltage of 28V was near to the rated absolute maximum of 35V for the IC [18] and the output voltage was low, a heat sink was needed. A SK104 heat sink was used and a layer of silicon heat transfer compound was added between the IC and heat sink for good heat transfer. Figure 3.12 shows the circuit diagram for the voltage regulator.

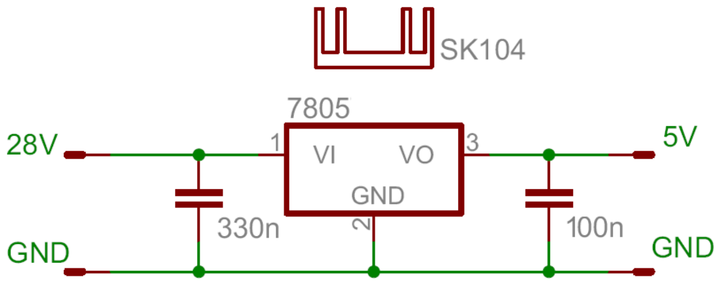

Figure 3.12: Voltage regulator circuit diagram

3.9 76CS1 Junction Box

It is time consuming and difficult to connect and disconnect wires from the 78 pin D connector on the 76CS1 card every time a synchro on the pedestal is added or removed. That is why a junction box was built to facilitate the easy addition to and removal from the S/D channels.

The junction box has a permanent connection to the 78 pin D connector to connect to the 76CS1 card and four 5-pin DIN connectors through which the synchros can be connected. The internal wiring of the box is also done in such a way to facilitate the feedback of the reference signal from the card so that the user only has to deal with connecting the five terminals of the synchro. The top and side views of the junction box are shown in Figure 3.13 and Figure 3.14 respectively.

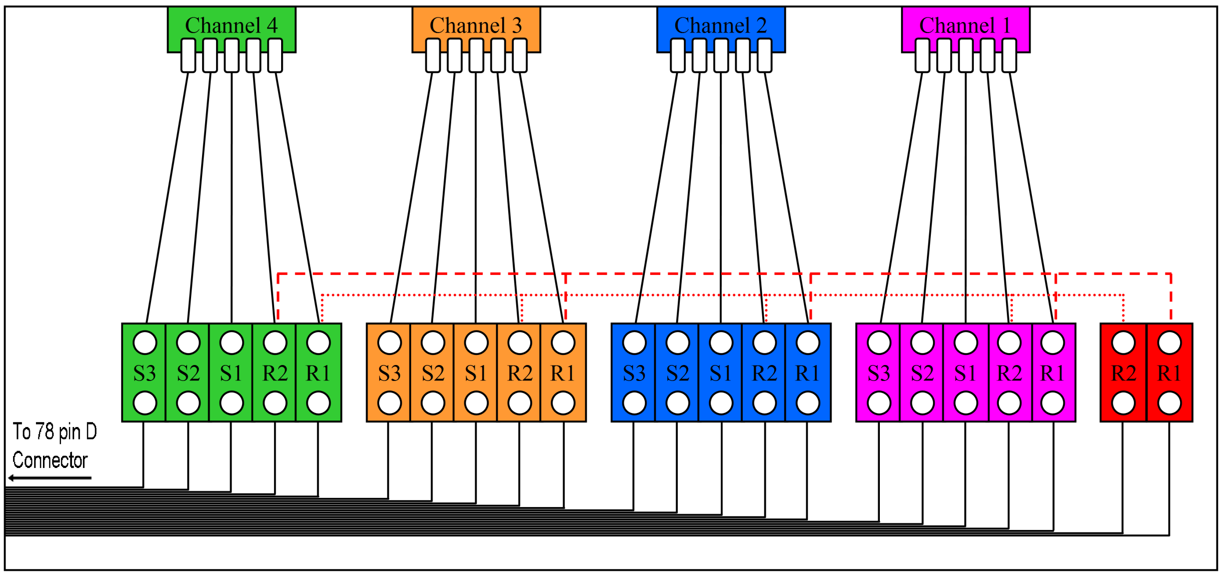

Figure 3.13: Top view of junction box showing internal wiring

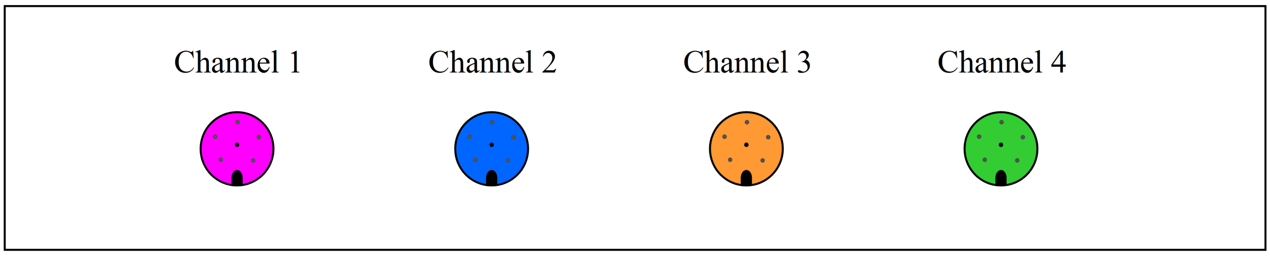

Figure 3.14: Side view of junction box showing panel mounted DIN sockets

Chapter 4

Hardware Implementation and Functional Testing

This chapter covers the implementation of the circuits designed in Chapter 3 as Printed Circuit Boards (PCB) as well as the functional testing of the control circuit as a whole to ensure that it matches the results from Chapter 3.

4.1 PCB Implementation

The implementation of the two PCB's involved selecting components, drawing the circuit for the total system in EAGLE Layout Editor and routing the tracks for the boards. The Limit Switch logic circuit was implemented on a separate PCB which was then mounted on top of the main PCB. Sections 4.1.1 and 4.1.2 give the circuit diagram and PCB layout for the main PCB and the limit switch logic PCB respectively. The accompanying CD contains a detailed bill of materials used for each PCB to allow for these boards to be recreated.

4.1.1 Control Circuit PCB

The board made for the control circuit had to contain two nearly identical circuits, one for each axis. Extra 5V-GND pins were added to allow the connection of a 5V fan or any other circuit. Figure 4.1 shows the schematic for the control circuit and Figure 4.2 shows the PCB layout and the routed tracks.

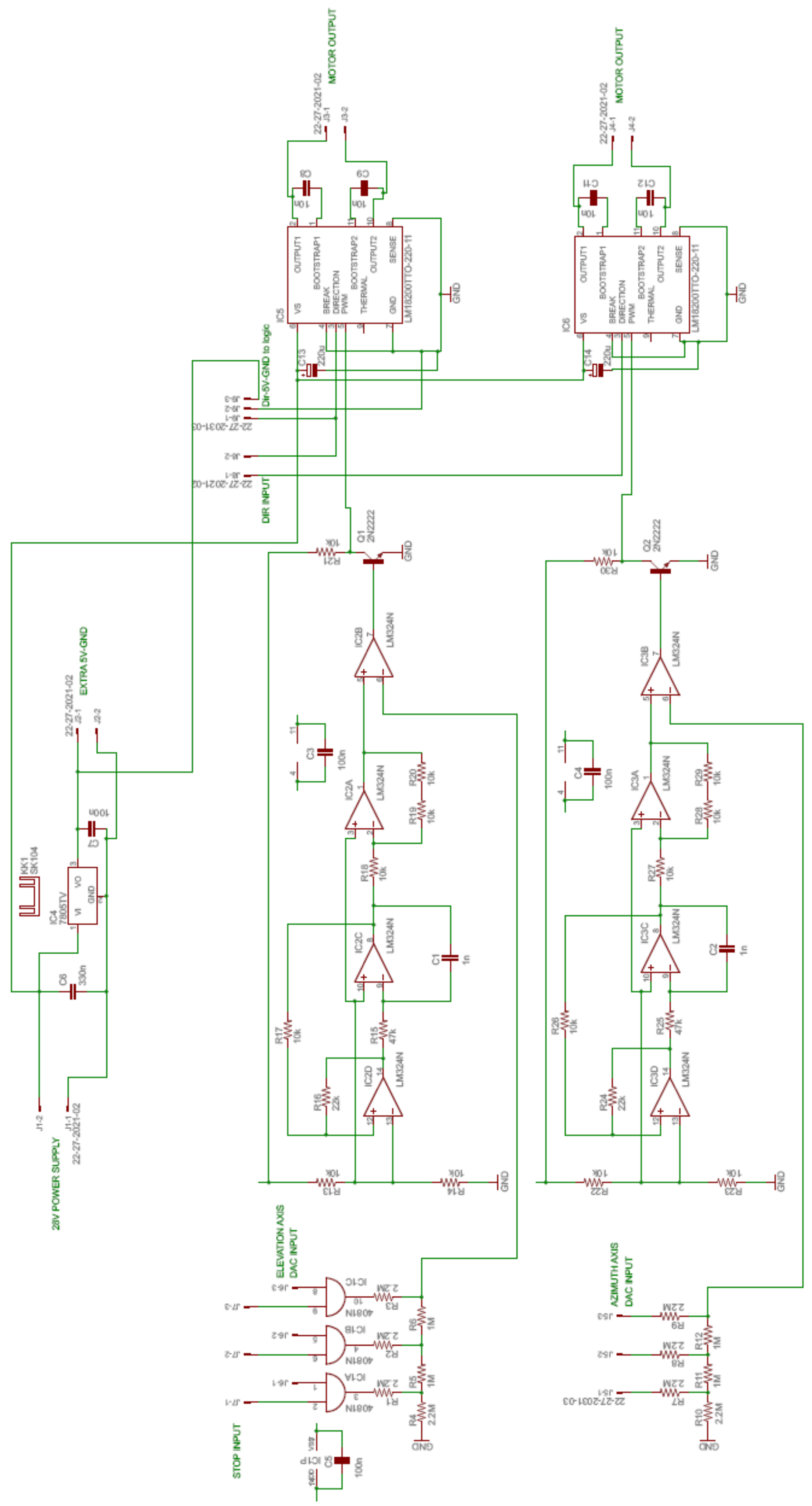

Figure 4.1: Control circuit schematic

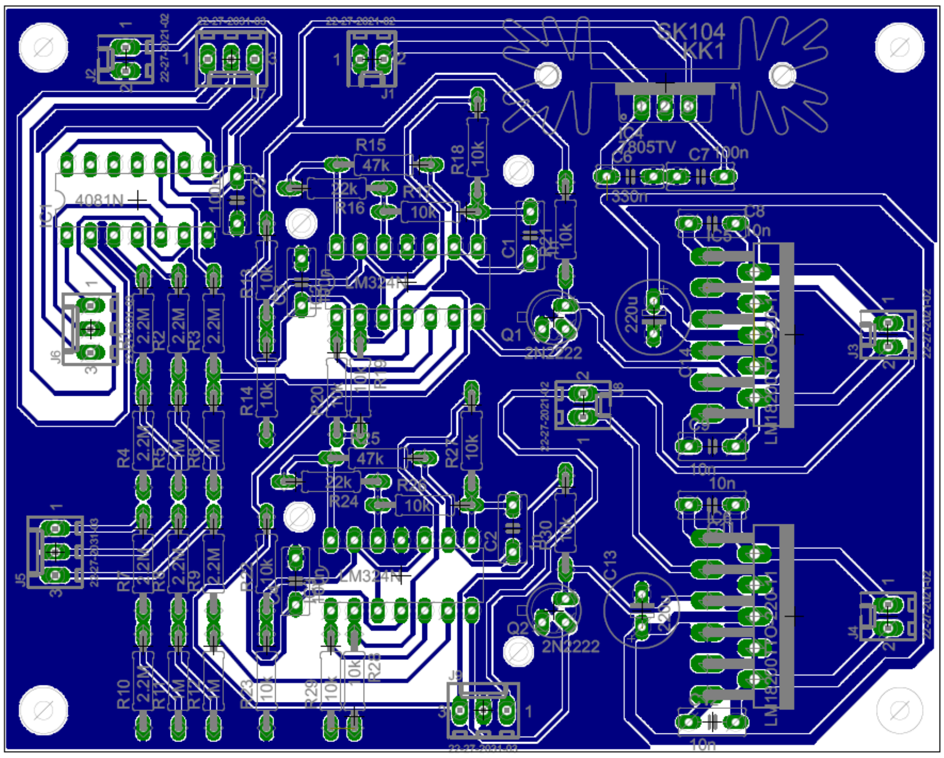

Figure 4.2: Routed PCB of control circuit

4.1.2 Limit Switch Logic PCB

The Limit Switch logic circuit was constructed on a separate PCB due to its routing complexity and to save space by mounting it above the control circuit PCB. Figure 4.4 gives the schematic of the circuit and Figure 4.3 shows the PCB layout and routed tracks. Due to the lack of components like resistors on the board which allow for easy routing, jumpers were needed to connect certain tracks. These jumpered tracks were limited to 5V and ground lines in order to ensure that any connection problems would not interfere with the logic signals. R1, R2, JP1, JP2, JP3, JP4, JP5 and JP6 were used as jumpers. The 5V and GND tracks on the schematic were connected by specifying that they connect to the same network. Thus some pins or wires on the schematic may appear to be unconnected.

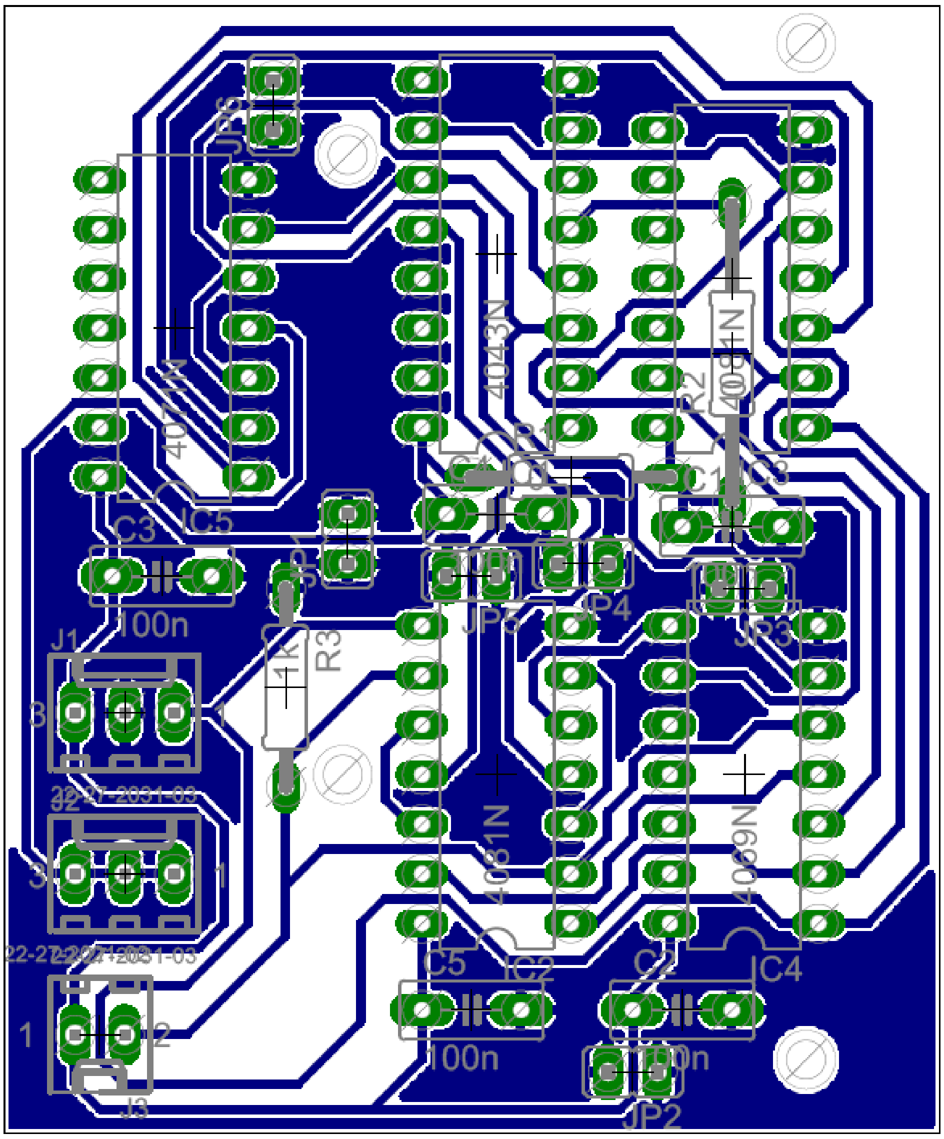

Figure 4.3: Routed PCB of logic circuit

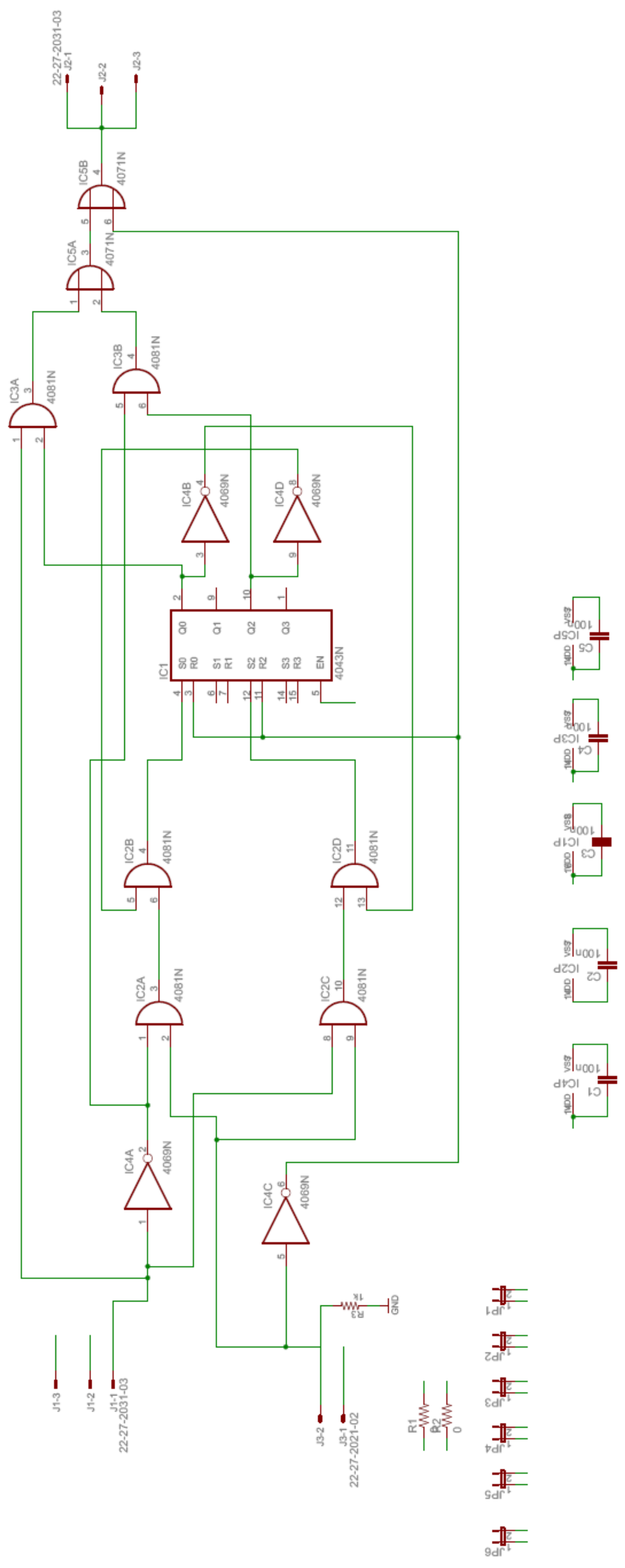

Figure 4.4: Limit Switch logic circuit schematic

For the remainder of this document, a naming convention is used for the speeds which corresponds to the decimal value of the binary number outputted to the Parallel Port to obtain that speed. In the case of the SPWM speeds, the speed which is being reduced is shown first followed by the a number corresponding to the the SPWM speed in relation to the other SPWM speeds (where 3 is the fastest and 1 the slowest).

For example:

- A DAC input of **101** on the azimuth axis is called **Speed 5**.
- The slowest **SPWM** of speed **010** on the azimuth axis is called Speed **2.1**.
- A DAC input of 101 on the elevation axis is called Speed 80.
- The slowest SPWM of speed **010** on the elevation axis is called **Speed 32.1**

4.2 Functional Testing

A functional test on a system concerns only the inputs and the outputs of the total system. It is often referred to as Black Box testing as the internal functioning of the system is unseen. It is used to verify whether a system gives the correct output for a specific input [19]. If the outputs are unexpected, the separate modules of the system can be investigated to determine the problem. Table 4.1 shows the Parallel Port inputs to the system and the pulse width modulated outputs to the motors. The X's in the control hardware inputs indicate that the value of that bit is negligible in this test. These bits are the direction bits and do not affect the output duty cycle to the motors.

Control Hardware Input	Elevation Axis Motor +ve Duty Cycle (%)	Azimuth Axis Motor +ve Duty Cycle (%)
X000X000	0	0
X001X001	0	0
X010X010	50	37.1
X011X011	57.1	44.7
X100X100	74.8	51.9
X101X101	83.1	62.8
X110X110	100	72.7
X111X111	100	82.7

Table 4.1: Functional test inputs and outputs

Chapter 5

Experimentation

A number of experiments were conducted to obtain results to aid in the design and tuning of the control software. The experiment to determine the backlash in the gearing of the azimuth axis gave a measure of the accuracy of the angular information received from the azimuth axis synchro.

5.1 Experiment 1

5.1.1 Aim

To determine the average overshoot for each azimuth axis speed over the full 360° range of movement.

5.1.2 Method

A test program was written that moved the pedestal on the azimuth axis at a predefined speed. At each degree on the axis, it recorded the position where it turned off the motor, then waited 2 seconds. It then recorded the position where it had settled, moved back 20° and moved to the next test point on the axis (1°further than the previous point). This program was run for each speed in both the clockwise (CW) and anti-clockwise (CCW) directions. The overshoot at each point can be calculated by subtracting the position where the motor was turned off from the position where the pedestal settled on the azimuth axis.

5.1.3 Results

With 360 measurements recorded per test, 2 directions and 6 speeds, a total of 4320 measurements were taken. The full results are contained on the accompanying CD. Table 5.1 shows a summary of the results.

Speed	Direction	Min. Overshoot (°)	Max. Overshoot (°)	Mean (°)	Variance ($^{\circ 2}$)	Standard Deviation (°)
2	CCW	0.3021	0.4450	0.3767	0.0007	0.0269
2	CW	0.2802	0.4339	0.3609	0.0009	0.0308
2	Both	0.2802	0.4450	0.3688	0.0009	0.0299
3	CCW	0.3845	0.5220	0.4520	0.0006	0.0245
3	CW	0.3730	0.5110	0.4414	0.0007	0.0264
3	Both	0.3730	0.5220	0.4467	0.0007	0.0260
4	CCW	0.4560	0.6040	0.5238	0.0007	0.0259
4	CW	0.4180	0.5713	0.5089	0.0007	0.0268
4	Both	0.4180	0.6040	0.5164	0.0007	0.0274
5	CCW	0.5550	0.7250	0.6411	0.0006	0.0246
5	CW	0.5550	0.6922	0.6254	0.0006	0.0249
5	Both	0.5550	0.7250	0.6332	0.0007	0.0259
6	CCW	0.6920	0.8080	0.7509	0.0006	0.0244
6	CW	0.6710	0.7910	0.7333	0.0006	0.0240
6	Both	0.6710	0.8080	0.7421	0.0007	0.0257
7	CCW	0.7910	0.9390	0.8647	0.0008	0.0286
7	CW	0.7306	0.9290	0.8469	0.0010	0.0309
7	Both	0.7306	0.9390	0.8558	0.0010	0.0311

Table 5.1: Results for Experiment 1

Figure 5.1 shows the Standard Normal Distribution Probability Density Function (PDF) for each speed.

The values for the Normal Distribution PDF were calculated using [20]:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{5.1}$$

where σ is the standard deviation, μ is the mean and x is the overshoot value.

These values were then normalised to a Standard Normal Distribution with $\sigma = 0$ and $\mu = 0$. This gave the following equation [20]:

$$n(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \tag{5.2}$$

where

$$z = \frac{x-\mu}{\sigma} \tag{5.3}$$

If $n(z)$ vs. z was plotted on the same axis for each speed, all the plots would be centered on 0 and an idea of the magnitude of the overshoot value for each speed would not be given. Thus a plot of $n(z)$ vs. x was done where z was calculated using Equation 5.3.

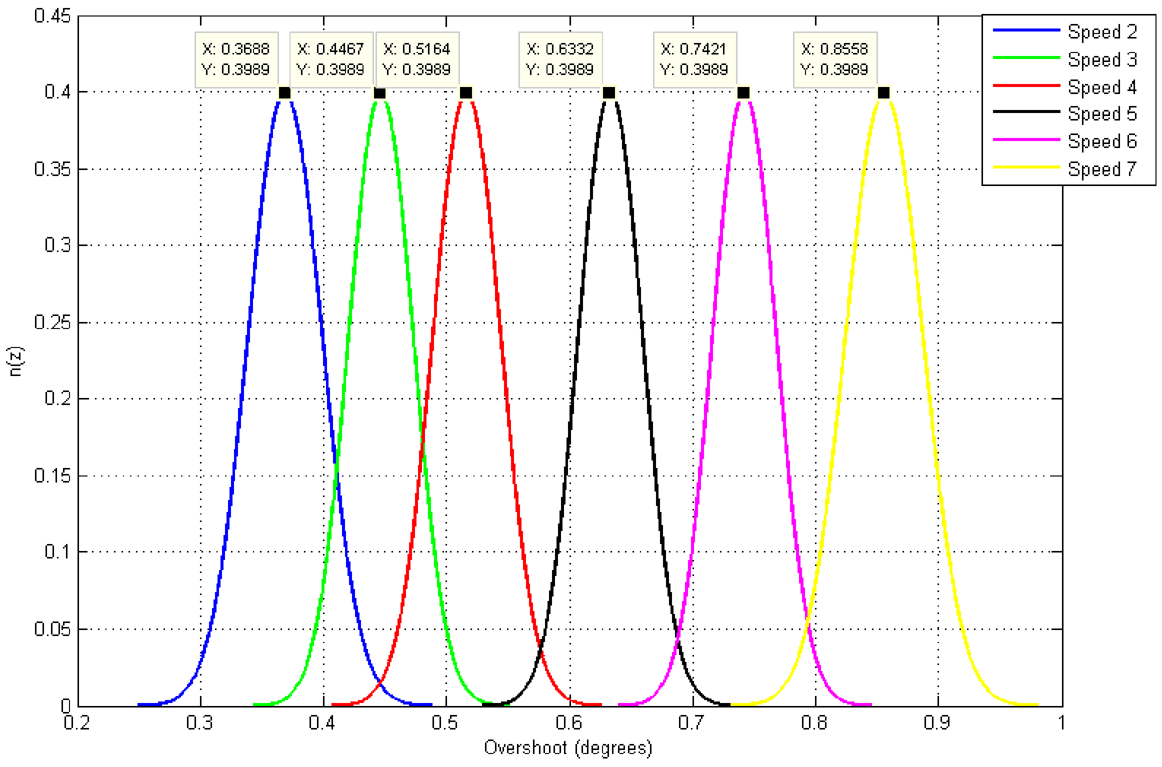

Figure 5.1: Standard Normal Distribution PDF of results for Experiment 1

5.1.4 Conclusions

The results from this experiment were used to create thresholds around the target position at which the speed must be reduced to ensure overshoot does not occur. The overshoot for the minimum speed was averaged as 0.3688°. This meant that the motor must be turned off at 0.3688° before the target position in order to stop within ±0.1196° ($\pm 4 \times \sigma$) from the target position 97.7% of the time [21]. This ±0.1196° range was too large and needed to be improved upon. For this reason, Software PWM was implemented in order to reduce the speed further.

5.2 Experiment 2

5.2.1 Aim

To determine the average overshoot for each elevation axis speed over the full 60° range of movement.

5.2.2 Method

A test program was written that moved the pedestal on the elevation axis at a predefined speed starting from 0°. At each degree on the axis, it recorded the position where it turned off the motor, then waited 2 seconds. It then recorded the position where it had settled, moved back 5° and moved to the next test point on the axis (1° further than the previous point). This program was run for each speed in both the clockwise and anti-clockwise directions. Speed 112 was not measured as its duty cycle was the same as Speed 96 (100%) and would have produced the same results. Due to the limitations in the movement of the pedestal on the elevation axis (0° to 60° only), the tests in the clockwise direction ran from 5° to 60° and the tests in the anti-clockwise direction ran from 0° to 55°. The overshoot at each point can be calculated by subtracting the position where the motor was turned off from the position where the pedestal settled on the elevation axis.

5.2.3 Results

With 56 measurements recorded per test for 5 speeds and 2 directions, a total of 560 measurements were taken. The full results are contained on the accompanying CD. Table 5.2 shows the average overshoot for each speed in each direction.

Speed	Direction	Min. Overshoot (°)	Max. Overshoot (°)	Mean (°)	Variance ($°^2$)	Standard Deviation (°)
32	CCW	1.4117	1.6973	1.5536	0.0022	0.0471
32	CW	1.8951	2.1533	1.9812	0.0021	0.0454
32	Both	1.4117	2.1533	1.7674	0.0483	0.2197
48	CCW	1.6260	2.0325	1.8619	0.0086	0.0926
48	CW	2.2083	2.5434	2.3891	0.0035	0.0593
48	Both	1.6260	2.5434	2.1255	0.0762	0.2760
64	CCW	2.5049	2.8290	2.6627	0.0039	0.0624
64	CW	2.9773	3.3014	3.1878	0.0047	0.0684
64	Both	2.5049	3.3014	2.9252	0.0739	0.2718
80	CCW	2.9498	3.2795	3.1371	0.0042	0.0645
80	CW	3.5046	3.8836	3.7013	0.0067	0.0819
80	Both	2.9498	3.8836	3.4192	0.0857	0.2928
96	CCW	3.7408	4.0155	3.8756	0.0034	0.0583
96	CW	4.3561	4.6582	4.5225	0.0035	0.0588
96	Both	3.7408	4.6582	4.1990	0.1091	0.3302

Table 5.2: Results for Experiment 2

Figure 5.2 shows the Standard Normal Distribution Probability Density Function (PDF) for each speed. The values for n(z) were calculated in the same manner as Experiment 1.

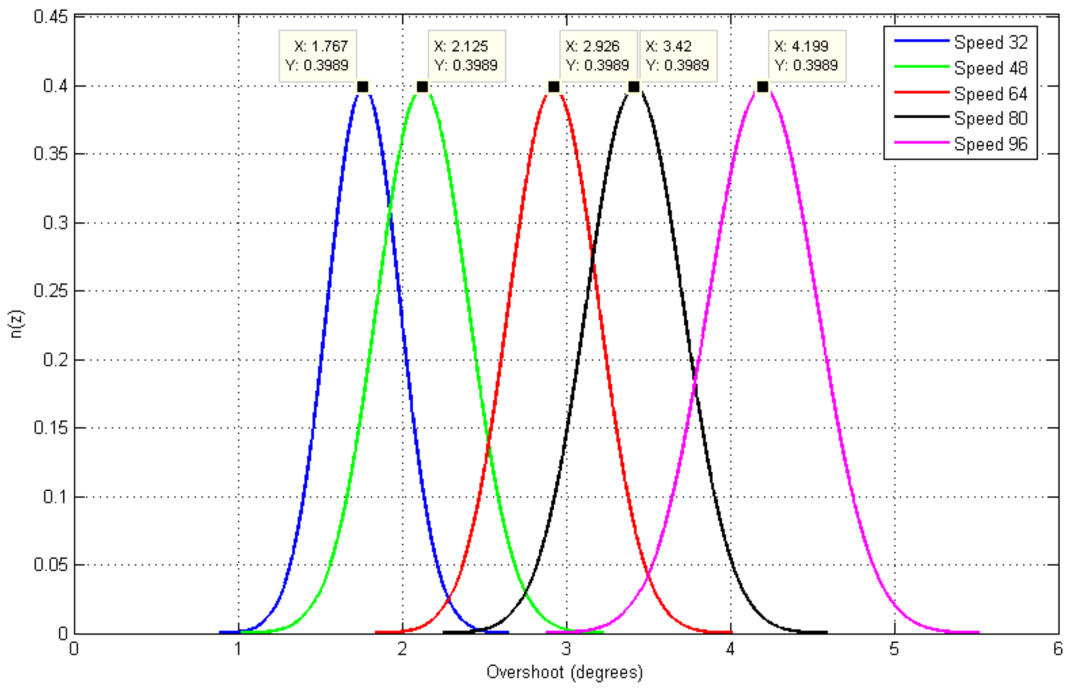

Figure 5.2: Standard Normal Distribution PDF of results for Experiment 2

5.2.4 Conclusions

The results from this experiment were used to create thresholds around the target position at which the speed must be reduced to ensure overshoot does not occur. The overshoot for the minimum speed was averaged as 1.7674°. This meant that the motor must be turned off at 1.7674° before the target position in order to stop within ±7.0696° ($\pm 4 \times \sigma$) from the target position 97.7% of the time [21]. This ±7.0696° range was too large and needed to be improved upon. For this reason, Software PWM was implemented in order to reduce the speed further.

5.3 Experiment 3

5.3.1 Aim

To determine the average overshoot for each Software Pulse Width Modulated variation of the minimum azimuth axis speed over the full 360° range of movement.

5.3.2 Method

A test program was written that moved the pedestal on the azimuth axis at each of the SPWM speeds starting from 0°. At each degree on the axis, it recorded the position where it turned off the motor, then waited 2 seconds. It then recorded the position where it had settled, moved back 5° and moved to the next test point on the axis (1° further than the previous point). This program was run for each SPWM speed in both the clockwise and anti-clockwise directions. The overshoot at each point can be calculated by subtracting the position where the motor was turned off from the position where the pedestal settled on the azimuth axis.

5.3.3 Results

Speed	Direction	Min. Overshoot (°)	Max. Overshoot (°)	Mean (°)	Variance (°^{2})	Standard Deviation (°)
2.1	CCW	-0.0330	0.1870	0.0160	0.0006	0.0248
2.1	CW	-0.0870	0.3190	0.0147	0.0017	0.0413
2.1	Both	-0.0870	0.3190	0.0154	0.0012	0.0341
2.2	CCW	-0.0280	0.1870	0.0182	0.0005	0.0216
2.2	CW	-0.0550	0.3020	0.0202	0.0020	0.0450
2.2	Both	-0.0550	0.3020	0.0192	0.0012	0.0353
2.3	CCW	-0.0440	0.0990	0.0300	0.0009	0.0295
2.3	CW	-0.0494	0.0880	0.0214	0.0008	0.0276
2.3	Both	-0.0494	0.0990	0.0257	0.0008	0.0289

Table 5.3: Results for Experiment 3

Figure 5.3 shows the Standard Normal Distribution Probability Density Function (PDF) for each speed. The values for n(z) were calculated in the same manner as Experiment 1.

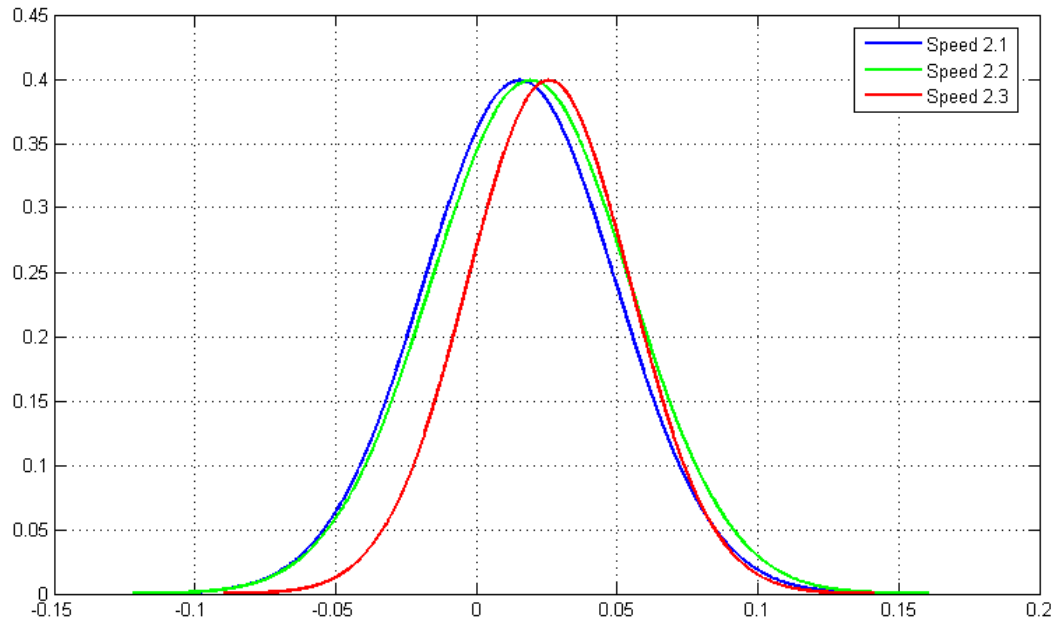

Figure 5.3: Standard Normal Distribution PDF of results for Experiment 3

5.3.4 Conclusions

The results from this experiment were used to create thresholds around the target position at which the speed must be reduced to ensure overshoot does not occur. The overshoot for the minimum speed was averaged as 1.016° with a standard deviation of 0.0341°.

5.4 Experiment 4

5.4.1 Aim

To determine the average overshoot for each Software Pulse Width Modulated variation of the minimum elevation axis speed over the full 60° range of movement.

5.4.2 Method

A test program was written that moved the pedestal on the elevation axis at each of the SPWM speeds starting from 0°. At each degree on the axis, it recorded the position where it turned off the motor, then waited 2 seconds. It then recorded the position where it had settled, moved back 2° and moved to the next test point on the axis (1° further than the previous point). This program was run for each SPWM speed in both the clockwise and anti-clockwise directions. Due to the limitations in the movement of the pedestal on the elevation axis (0° to 60° only), the tests in the clockwise direction ran from 2° to 60° and the tests in the anti-clockwise direction ran from 0° to 58°. The overshoot at each point can be calculated by subtracting the position where the motor was turned off from the position where the pedestal settled on the elevation axis.

5.4.3 Results

Speed	Direction	Min. Overshoot (°)	Max. Overshoot (°)	Mean (°)	Variance ($°^2$)	Standard Deviation (°)
32.1	CCW	-0.0110	0.0165	0.0030	0.00004	0.0061
32.1	CW	-0.0165	0.0824	0.0215	0.0004	0.0192
32.1	Both	-0.0165	0.0824	0.0122	0.0003	0.0170
32.2	CCW	-0.0110	0.0330	0.0073	0.0001	0.0099
32.2	CW	-0.0055	0.1263	0.0536	0.0006	0.0242
32.2	Both	-0.0110	0.1263	0.0304	0.0009	0.0297
32.3	CCW	0.0000	0.2087	0.0561	0.0017	0.0411
32.3	CW	0.0165	0.2198	0.1043	0.0016	0.0403
32.3	Both	0.0000	0.2198	0.0802	0.0022	0.0472

Table 5.4: Results for Experiment 4

Figure 5.4 shows the Standard Normal Distribution Probability Density Function (PDF) for each speed. The values for n(z) were calculated in the same manner as Experiment 1.

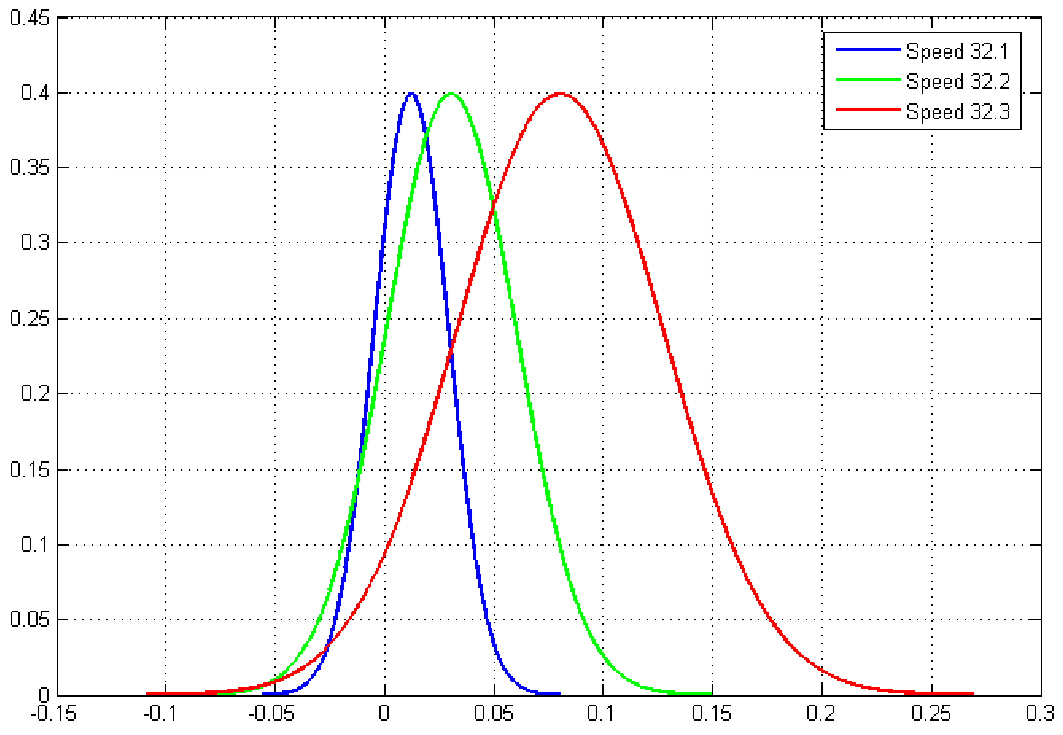

Figure 5.4: Standard Normal Distribution PDF of results for Experiment 4

5.4.4 Conclusions

The results from this experiment were used to create thresholds around the target position at which the speed must be reduced to ensure overshoot does not occur. The overshoot for the minimum speed was averaged as 0.0122° with a standard deviation of 0.017°. Due to the fact that the overshoot was averaged for both directions, and the antenna weight caused large overshoots in the CW direction and small overshoots in the CCW direction, the PDF for each speed covered a very wide range of angles.

5.5 Experiment 5

5.5.1 Aim

To approximate a value for the backlash in the gearing on the azimuth axis.

5.5.2 Method

A protractor was drawn using AutoCAD which had tick markings every 0.1° and an approximate distance of 1mm between 0.1° markings at the circumference. The pedestal was placed at the center of this protractor and a laser pointer was mounted on it. The pedestal was moved to 1° and the laser aligned with the 1° marking on the circumference of the pedestal. The pedestal was then moved to the 10°, 20°, 30°, 40°, 50°, 60°, 70°, 80°, 90° positions while returning to 1° between movements. At every stationary position, the synchro reading was taken and the actual position was read on the protractor. This test was performed twice to get a larger set of results.

5.5.3 Results

The full results for this experiment are contained on the accompanying CD. From the results, the value for backlash was calculated to be 0.0392° with a standard deviation of 0.0375°.

5.5.4 Conclusions

The value for backlash was calculated to be 0.0392° with a standard deviation of 0.0375°. This result gives a range of values in which the actual value for backlash can be found. To ensure that the real value for backlash is accounted for, a value of 0.1892° will be used. This value is equal to $\mu + 4\sigma$ and will ensure that the value for backlash is less than the approximated value, 97.7% of the time [21]. This is to ensure that the value for the accuracy of the synchros on the azimuth axis is a worst case value and encompasses nearly all other possible values for the backlash.

Chapter 6

Software Design

It was specified that the Pedestal Controller system run in an open source environment. Although the most common open source environment available for a PC is Linux, there are many distributions or “flavours” available. *Ubuntu 8.04 LTS - Hardy Heron* was chosen for this project because of its user-friendliness and large user base at UCT. The kernel version used was 2.6.24-21. Programs were written using gedit in GNOME and were compiled using the Makefile utility and g++ version 4.2.4.

6.1 Installing the 76CS1

The Linux drivers supplied by North Atlantic Industries for the 76CS1 card were compiled using Red Hat Enterprise Linux 4.0 for a kernel version of 2.6.9. This posed a problem as a number of commands and libraries had been depreciated in the newer kernels and consequently the kernel module, the driver program and test programs would not compile. Table 6.1 gives the changes that were made in order to compile the necessary files

The drivers were compiled and installed by following the instructions in the buildnotes.txt file that accompanied them.

Filename	Line Number	Phrase to replace	New phrase
Makefile	46	CFLAGS	EXTRA_CFLAGS
nai.c	175	SA_SHIRQ	IRQF_SHARED
”	22	<linux/config.h>	<linux/autoconf.h>
”	229	pci_module_init	pci_register_driver

Table 6.1: Changes made to 76CS1 files

6.2 Real-time programming

A real-time system is a computer system which requires that not only the results of an operation are correct, the results must also be produced within a specified time period [22]. Results returned after the time period has elapsed may be of no real value.

6.2.1 Hard Real-Time

A hard real-time system has the most stringent time requirements. It guarantees that critical real-time tasks will be completed within their deadlines [22]. In a hard real-time system, a late result is useless.

6.2.2 Soft Real-Time

A soft real-time system has less stringent timing requirements and a critical process will receive priority over other tasks until it is complete [22]. No guarantee is given that the deadline will be met. In soft real-time systems, a late result is less valuable but not useless.

6.2.3 The Pedestal Controller

The pedestal controller process needed to read in position information as often as possible in order to know the current position of each of the axes. This meant that if the pedestal was moving and one of the position read functions or the controller loop process was preempted by the operating system, a threshold could be missed causing the pedestal to overshoot its target position. For this reason the system needed to act as much like a hard real-time system as possible. This was very difficult using the regular Linux kernel as process and thread priorities and scheduling techniques cannot be specified by the programmer. Thus occasionally the OS would preempt the pedestal controller process causing a threshold to be missed. Also if the terminal used to run the program lost focus, the OS would put the process to sleep and because the data on the Parallel Port is latched, the pedestal would move at its current speed indefinitely. The "nice –15" command was used to execute the program and give the process the maximum priority of -15, however no improvement was noted.

Another Linux kernel is available for installation which would allow the priority of each process and the scheduling technique to be controlled. This kernel is called RTLinux and is a Linux kernel suited to real-time applications. Due to the time constraints of this project, the RTLinux kernel was not implemented.

6.3 Multithreaded Programming

A thread is flow of control within a process [22]. Threads allows multiple paths of control within a single process. Multithreaded programming has a number of benefits which can be categorised as follows [22]:

- **Responsiveness** - The interactivity of a process can be maintained while a part of it is blocked or executing a length operation.
- **Resource Sharing** - Threads share the memory and resources of the parent process to which they belong.
- **Economy** - Because threads share memory and resources, it is more economical to create and context-switch between threads of the same process than creating and context-switching between two separate processes.
- **Utilisation of multiprocessor architectures** - In multiprocessor architectures threads may be running in parallel on different processor whereas a single process can only run on one CPU regardless of how many are available.

6.3.1 PThreads

The PThread library provides an API for thread creation and synchronisation defined by the POSIX standard (IEEE 1003.1c) [22]. This library was used to create threads for methods that contained loops which would block the flow of the total process. This ensured that commands could still be sent to the pedestal controller while it was busy moving the pedestal. The most important of these commands being the Abort command and commands to receive the current axis positions.

6.4 Controlling the Movement of the Pedestal

To ensure no overshoot occurs and no change of direction is needed to recover, the speed of the movement when approaching the setpoint should be reduced accordingly. Thus in order to move the pedestal to a setpoint on either of its two axes, it is imperative that the speed of the movement is reduced or stopped before the position reaches a distance equal to the threshold for the current speed away from the setpoint. Figure 6.1 demonstrates this concept using an arbitrary threshold for each speed.

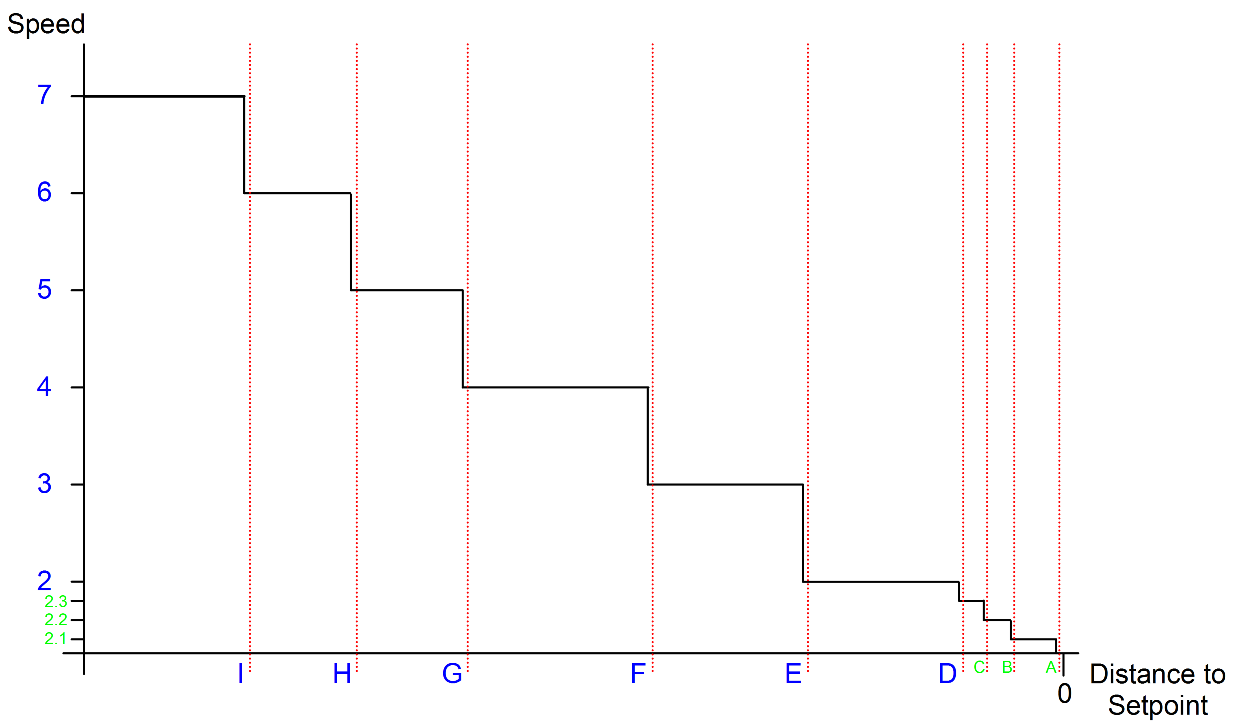

Figure 6.1: Speed vs. Distance to setpoint

Where A to C are the threshold values for each of the SPWM speeds and D to I are the threshold values for each regular PWM speed respectively.

These thresholds were defined as the sum of the mean overshoot for the current speed and all the slower speeds and are given in Table 6.2.

Azimuth Axis	
Speed	Threshold (°)
2.1	0.0154
2.2	0.0346
2.3	0.0603
2	0.4290
3	0.8757
4	1.3921
5	2.0253
6	2.7674
7	3.6232

Elevation Axis	
Speed	Threshold (°)
32.1	0.0122
32.2	0.0427
32.3	0.1229
32	1.8903
48	4.0158
64	6.9410
80	10.3602
96	14.5592

Table 6.2: Threshold value for each speed

6.5 Control Algorithm

In order to code the position control method for each axis, a suitable algorithm had to be designed. Figure 6.2 shows the flow chart for the control of the coarse (regular PWM) speeds.

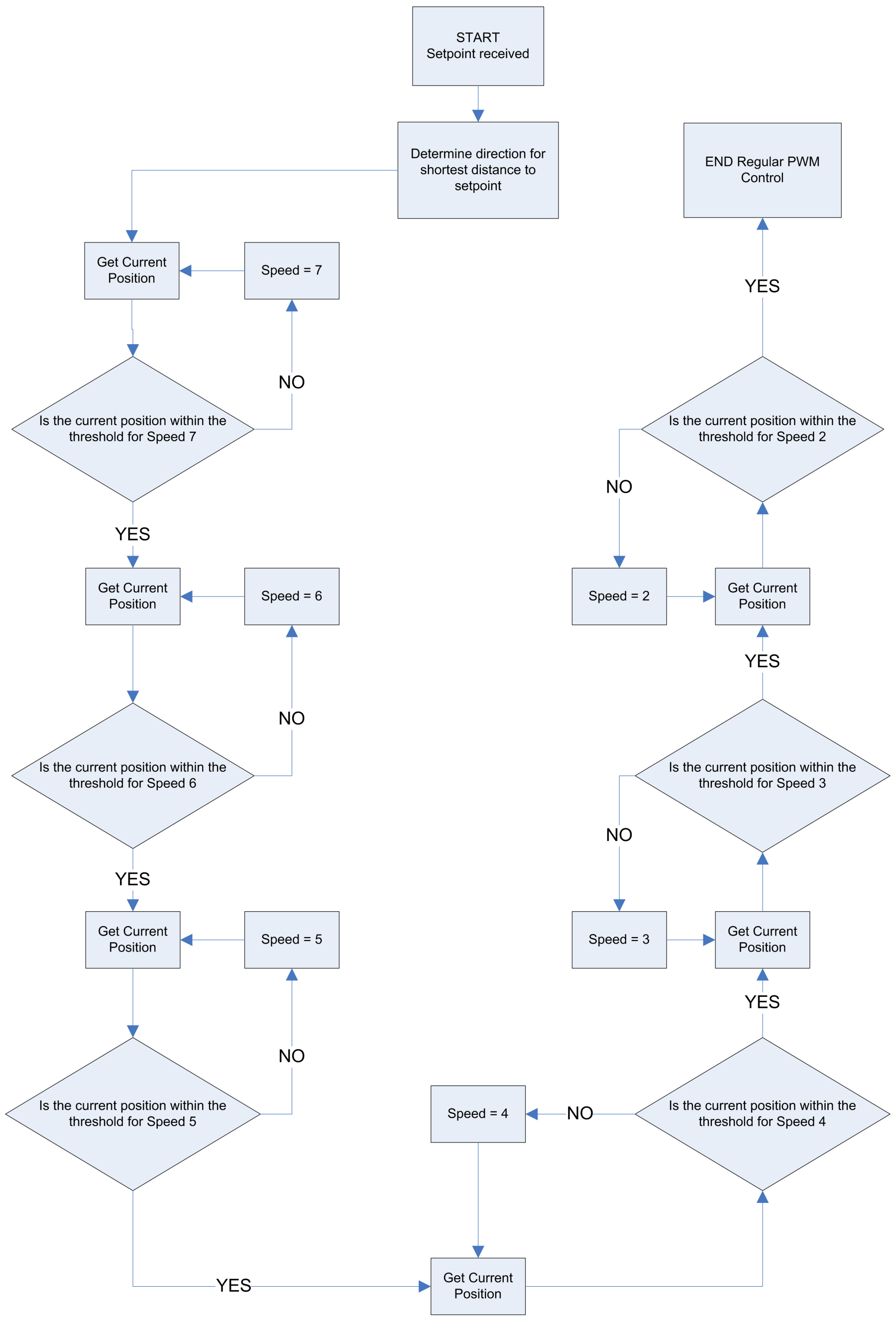
START
Setpoint received
Determine direction for shortest distance to setpoint
END Regular PWM Control
Get Current Position
Speed = 7
YES
NO
Is the current position within the threshold for Speed 7
Is the current position within the threshold for Speed 2
NO
YES
Get Current Position
Speed = 6
Speed = 2
Get Current Position
YES
NO
Is the current position within the threshold for Speed 6
Is the current position within the threshold for Speed 3
NO
YES
Get Current Position
Speed = 5
Speed = 3
Get Current Position
YES
NO
Is the current position within the threshold for Speed 5
Speed = 4
NO
Is the current position within the threshold for Speed 4
YES
Get Current Position

Figure 6.2: Flowchart of coarse speed control algorithm

To accurately position the pedestal on both of its axes, a method of fine motor control was needed. Software Pulse Width Modulation was used which allowed for each axis to move at very slow speeds. Figure 6.3 shows the flowchart for the control of the fine (Software PWM) speeds.

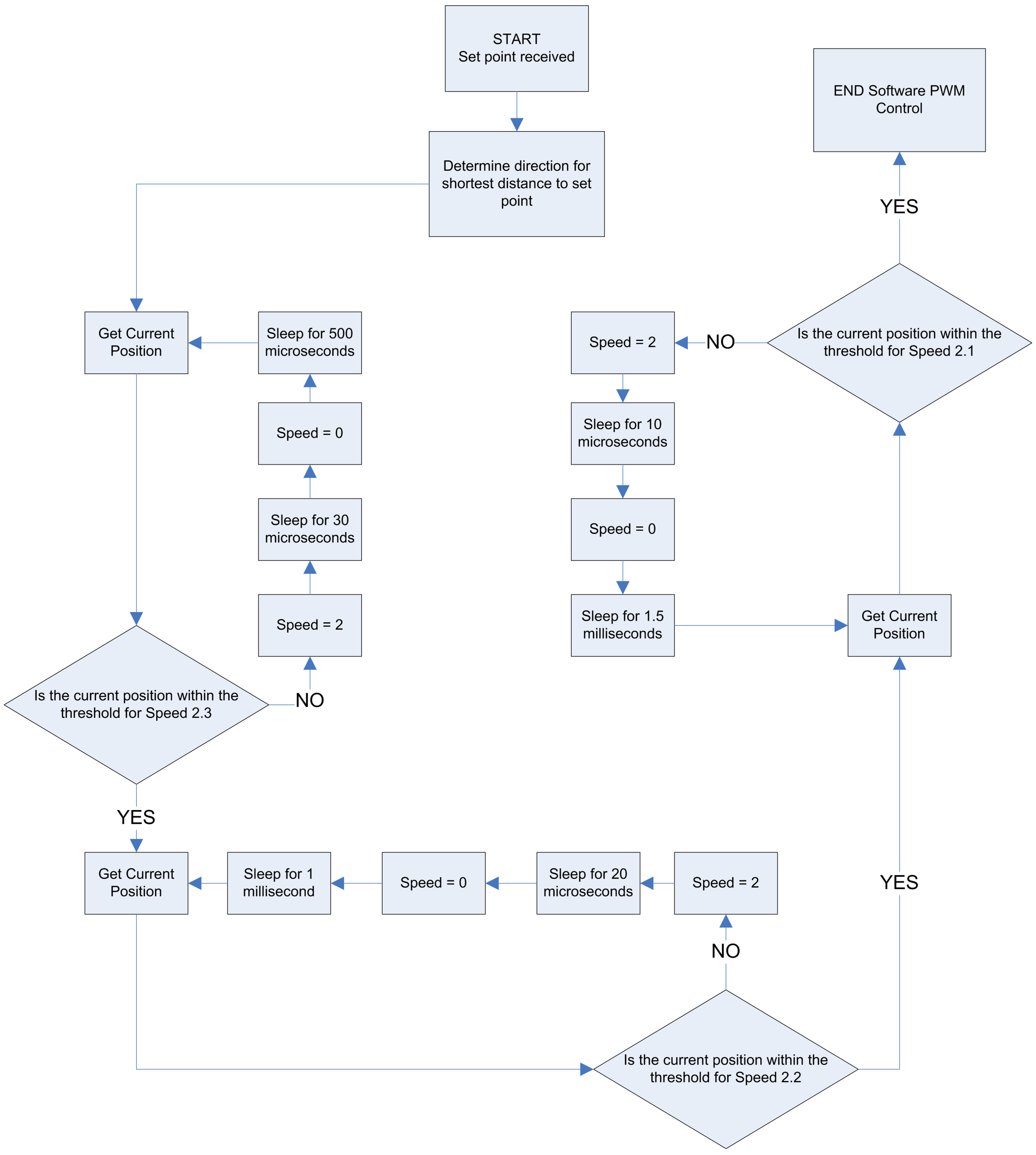
START
Set point received
Determine direction for shortest distance to set point
Get Current Position
Sleep for 500 microseconds
Speed = 0
Sleep for 30 microseconds
Speed = 2
NO
Is the current position within the threshold for Speed 2.3
YES
Get Current Position
Sleep for 1 millisecond
Speed = 0
Sleep for 20 microseconds
Speed = 2
NO
Is the current position within the threshold for Speed 2.2
YES
Get Current Position
Sleep for 1.5 milliseconds
Speed = 0
Sleep for 10 microseconds
Speed = 2
NO
Is the current position within the threshold for Speed 2.1
YES
END Software PWM Control

Figure 6.3: Flowchart of fine speed control algorithm

Both the coarse and fine speed control methods constantly check an integer called *Abort*. If *Abort* = 0, regular functioning continues, but if *Abort* = 1, the method outputs a speed of 0 to the motor and ends. The algorithms shown in Figure 6.2 and Figure 6.3 apply to both the azimuth and elevation axes. However on the elevation axis, the speed values will range from 16 to 112 in increments of 16 instead of from 2 to 7 in increments of 1.

6.6 Description of Code

This section describes the methods, structures and important variables contained in the pedestal control API. Only the section names correspond to actual code. All the method descriptions use pseudo-code to describe their function.

6.7 Structures

6.7.1 struct Positions

This structure contains two double values and is used to pass the theta and phi setpoints to the ped_ThrMoveTo thread.

6.8 Variables

The following variables are used globally in the pedestal controller and should not be accessed from outside of the pedestal.cpp file.

6.8.1 double AcoarseSpeeds[6]

This array contains the thresholds for the regular PWM azimuth axis speeds obtained from the results of Experiment 1 in Section 5.1. AcoarseSpeeds[5] and AcoarseSpeeds[0] correspond to the thresholds for Speed 7 and Speed 2 respectively.

6.8.2 double AfineSpeeds[3]

This array contains the thresholds for the SPWM azimuth axis speeds obtained from the results of Experiment 3 in Section 5.3. AfineSpeeds[2] and AfineSpeeds[0] correspond to the thresholds for Speed 2.3 and Speed 2.1 respectively.

6.8.3 double EcoarseSpeeds[6]

This array contains the thresholds for the regular PWM elevation axis speeds obtained from the results of Experiment 2 in Section 5.2. AcoarseSpeeds[6] and AcoarseSpeeds[0] correspond to the thresholds for Speed 112 and Speed 32 respectively.

6.8.4 double EfineSpeeds[3]

This array contains the thresholds for the SPWM elevation axis speeds obtained from the results of Experiment 4 in Section 5.4. EfineSpeeds[2] and EfineSpeeds[0] correspond to the thresholds for Speed 32.3 and Speed 32.1 respectively.

6.8.5 int A_chan, E_chan

A_chan and E_chan contain the channel number for the 76CS1 of the synchro used on the azimuth and elevation axes respectively.

6.8.6 double Theta, Phi

Theta and Phi contain the current position of the pedestal on the azimuth and elevation axes respectively.

6.9 Methods

6.9.1 int ped_Initialise(int a_chan, int e_chan)

This method opens the 76CS1 and gives it the following configuration:

- Sets the channels used for the elevation and azimuth axis synchros to e_chan and a_chan respectively
- Sets the a_chan and e_chan channels to Active;
- Sets the a_chan and e_chan channels to Synchro mode;
- Sets the voltage of the output reference signal to 3.6V.

The Parallel Port is also checked to determine whether the program can access it. The return value can take three values, namely:

1. The Parallel Port can be accessed but the 76CS1 setup failed;
2. The 76CS1 setup was successful but the Parallel Port cannot be accessed (most probably due to lack of root privileges);
3. The 76CS1 setup was successful and the Parallel Port can be accessed.

6.9.2 void* ped_Emoveto(void *arg)

This method only uses pointers to void in its call. Arguments are cast to the appropriate type to access the information, i.e. the set point is contained in arg. The **coarse speed** control loop depicted in Figure 6.2 is implemented in this method to control the elevation axis. Once the control loop has completed, the ped_Emovesmall method is called.

6.9.3 void* ped_Emovesmall(void *arg)

This method only uses pointers to void in its call. Arguments are cast to the appropriate type to access the information, i.e. the set point is contained in arg. The **fine speed** control loop depicted in Figure 6.3 is implemented in this method to control the elevation axis.

6.9.4 void* ped_Amoveto(void *arg)

This method only uses pointers to void in its call. Arguments are cast to the appropriate type to access the information, i.e. the set point is contained in arg. The **coarse speed** control loop depicted in Figure 6.2 is implemented in this method to control the azimuth axis. Once the control loop has completed, the ped_Amovesmall method is called.

6.9.5 void* ped_Amovesmall(void *arg)

This method only uses pointers to void in its call. Arguments are cast to the appropriate type to access the information, i.e. the set point is contained in arg. The **coarse speed** control loop depicted in Figure 6.3 is implemented in this method to control the azimuth axis.

6.9.6 int ped_GetTheta()

This method returns the current position of the pedestal on the azimuth axis.

6.9.7 int ped_GetPhi()

This method returns the current position of the pedestal on the elevation axis.

6.9.8 void ped_Abort()

This method raises the Abort flag (equal to 1) which causes any running movement thread to stop the motors and terminate.

6.9.9 int ped_MoveTo(double theta, double phi)

This method creates a thread which executes the ped_ThrMoveTo method. It passes a void pointer to the thread which contains the setpoint for each axis wrapped in the Positions struct. The Movement_Complete flag is set to 0 to signal that a movement is in progress. The Abort flag is also set to 0 to allow for the movement to be interrupted. The method then returns 1 to indicate that movement of the axes has begun. This is done to ensure that the control of the movement methods is not handled in the main process flow and is instead handled in a separate thread.

6.9.10 void* ped_ThrMoveTo(void *arg)

This method uses the methods described in Sections 6.9.2, 6.9.3, 6.9.4, 6.9.5. It takes a void pointer as its argument which is then cast to a Positions struct to obtain the setpoints for the azimuth and elevation axes and follows the following execution pattern:

1. Call ped_Amoveto(theta).
2. Wait for ped_Amoveto to complete its execution.
3. Call ped_Emoveto(phi).
4. Wait for ped_Emoveto to complete its execution.
5. Raise Movement_Complete flag (equal to 1).

6.9.11 int ped_raster(double theta1, double phi1, double theta2, double phi2, double Ainc, double Einc)

This method performs a Raster Scan of the box created by the theta1, phi1, theta2, phi2 coordinates. The pedestal moves along the azimuth axis in increments defined by Ainc. After each increment, the Raster_Stop flag is raised (equal to 1). The method then sleeps until the flag is lowered. It then moves to the next increment on the azimuth axis. When the azimuth axis reaches the other limit defined by the raster box, the elevation axis is incremented by Einc. This sequence continues until the opposite corner of the box is reached (theta2, phi2). Figure 6.4 shows this movement. The Raster_Stop flag is used to ensure that an application using the API has time to receive data from an antenna on the pedestal at each increment.

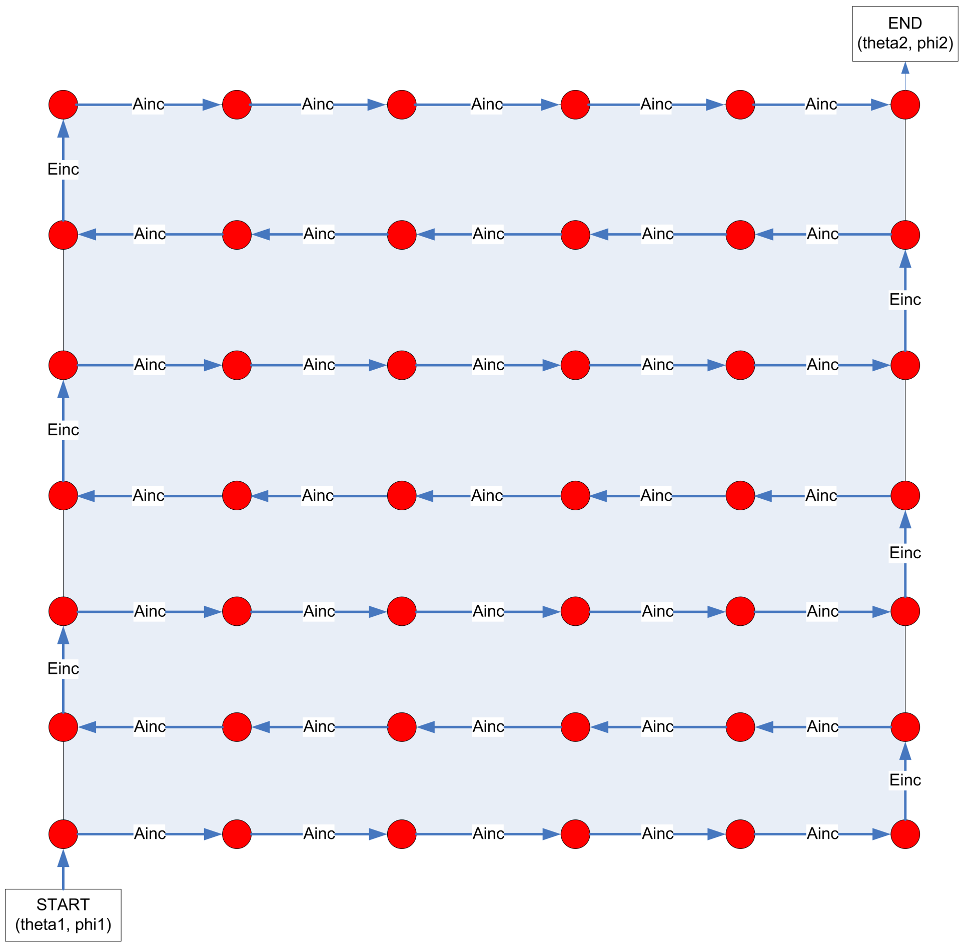

Figure 6.4: Raster Scan

Chapter 7

System Validation

This chapter performs a range of tests to produce results which can be used to validate the overall system in terms of the user specification.

7.1 Acceptance Testing

According to the Systems Engineering Guidebook for ITS [23], an Acceptance Test is a formal testing conducted to determine whether or not a system satisfies its acceptance criteria. In the context of this project, the acceptance criteria correspond directly to the user specification for the system. Multiple tests were conducted to determine whether the system satisfied the acceptance criteria. These tests began with the system in a specific state, an input was applied and the resulting output observed. A Pass/Fail assessment was given for every test.

7.1.1 Test 1

This test focuses on the ped_Initialise method and tests its various outputs.

Case 1

- **State:**
  - Program run with root privileges.
  - 76CS1 not open.
- **Input:**
  - ped_Initialise(1,2)

- **Output:**
  - Synchro channel 1 assigned to azimuth axis.
  - Synchro channel 2 assigned to elevation axis.
  - 76CS1 reference voltage set to 3,6V.
  - 3 is returned.
- **Pass/Fail**
  - PASS

Case 2

- **State:**
  - Program run without root privileges.
  - 76CS1 not open.
- **Input:**
  - ped_Initialise(1,2)
- **Output:**
  - Synchro channel 1 assigned to azimuth axis.
  - Synchro channel 2 assigned to elevation axis.
  - 76CS1 reference voltage set to 3,6V.
  - Parallel port cannot be accessed.
  - 2 is returned.
- **Pass/Fail**
  - PASS

Case 3

- **State:**
  - Program run with root privileges.
  - 76CS1 device not installed.
- **Input:**
  - ped_Initialise(1,2)
- **Output:**
  - 76CS1 cannot be opened.
  - 1 is returned.
- **Pass/Fail**
  - PASS

7.1.2 Test 2

This test focuses on the ped_MoveTo and tests a few variations of its inputs.

Case 1

- **State:**
  - Pedestal is stationary in any position.
- **Input:**
  - ped_MoveTo(40,55)
- **Output:**
  - Pedestal moves to 40° on the azimuth axis.
  - Pedestal moves to 55° on the elevation axis.
  - Movement_Complete flag = 1.
- **Pass/Fail**
  - PASS

Case 2

- **State:**
  - Pedestal is at 90° on the azimuth axis.
  - Pedestal is at 0° on the elevation axis.
- **Input:**
  - ped_MoveTo(90,60)
- **Output:**
  - Pedestal does not move on the azimuth axis.
  - Pedestal moves to 60° on the elevation axis.
  - Movement_Complete flag = 1.
- **Pass/Fail**
  - PASS

Case 3

- **State:**
  - Pedestal is at 120° on the azimuth axis.
  - Pedestal is at 40° on the elevation axis.
- **Input:**
  - ped_MoveTo(0,0)
- **Output:**
  - Pedestal moves past 0°, does a full revolution and stops at 0° the second time.
  - Pedestal moves to 0 on the elevation axis.
  - Movement_Complete flag = 1.
- **Pass/Fail**
  - FAIL

7.1.3 Test 3

This test focuses on the ped_GetTheta and ped_GetPhi methods.

Case 1

- **State:**
  - Pedestal is stationary
- **Input:**
  - ped_GetTheta()
- **Output:**
  - Current position of the azimuth axis is returned.
- **Pass/Fail**
  - PASS

Case 2

- **State:**
  - Pedestal is stationary.
- **Input:**
  - ped_GetPhi()
- **Output:**
  - Current position of the elevation axis is returned.
- **Pass/Fail**
  - PASS

Case 3

- **State:**
  - Pedestal is moving to a position.
- **Input:**
  - ped_GetTheta()
  - ped_GetPhi()
- **Output:**
  - Current position of the azimuth axis is returned from ped_GetTheta call.
  - Current position of elevation axis is returned from ped_GetPhi call.
- **Pass/Fail**
  - PASS

7.1.4 Test 4

This test focuses on the ped_raster method.

Case 1

- **State:**
  - Pedestal is stationary
- **Input:**
  - ped_raster(0,0,100,60,10,10)
- **Output:**
  - Pedestal moves to 0° on the azimuth axis.
  - Pedestal moves to 0° on the elevation axis.
  - Pedestal moves the specified 100° azimuth range in 10° increments.
  - The elevation axis is moved to 10°.
  - Pedestal moves the specified 100° azimuth range in -10° increments.

  - The elevation axis is moved to 20°.
  - This continues until the pedestal reaches a position of 100° on the azimuth axis and 60° on the elevation axis.

- **Pass/Fail**
  - PASS

7.1.5 Test 5

This test focuses on the ped_Abort command.

Case 1

- **State:**
  - Pedestal is moving to a position.
- **Input:**
  - ped_Abort()
- **Output:**
  - Current movement stops and ped_ThrMoveTo terminates.
- **Pass/Fail**
  - PASS

Case 2

- **State:**
  - Pedestal is performing a raster scan.
- **Input:**
  - ped_Abort()
- **Output:**
  - Current movement stops and ped_raster terminates.
- **Pass/Fail**
  - PASS

7.2 Determining the Accuracy of the System

To quantify a result for the accuracy of the system, a test program was written which moved the pedestal to a number of positions on both the azimuth and elevation axes. The final position and the setpoint were recorded to determine the error in each movement. The full results are contained on the accompanying CD. The results were analysed and the following measure of accuracy was determined:

- The azimuth axis had an average accuracy of 0.1023° with a standard deviation of 0.0544°.
- The elevation axis had an average accuracy of 0.0193° with a standard deviation of 0.0308°.

Chapter 8

Analysis of Results

This chapter analyses the results obtained in the functional testing of the hardware and the system validation tests of Chapter 7. The validity of these results as well as the reason for any deviation from the expected results is also discussed.

8.1 Functional Testing of Hardware

The functional test results obtained in Section 4.2 summarised the functionality of the control hardware. The duty cycle outputs for the corresponding inputs to the azimuth axis DAC did not match the results obtained during the unit testing. Upon further investigation, it was found that the Parallel Port voltage used at the DAC inputs was in fact 3.7V and not the simulated 5V. This caused a lower voltage to be outputted by the DAC and thus a lower duty cycle was obtained from the PWM Controller. However, the duty cycle outputs for the corresponding inputs to the elevation axis DAC did match the results obtained in the unit testing. This is because, the AND gates used at the DAC inputs output a constant voltage and thus as long as the DAC inputs are reliably recognised as a logic HIGH or LOW, the AND gate outputs will be the same voltage.

8.2 Acceptance Testing

The PASS/FAIL assessment for each of the acceptance tests allows for a measure of how well the system satisfies its user specification. All the tests were assessed as a PASS except Test 2 Case 3. This test is a special case of the regular ped_MoveTo method. It demonstrates how the use of a non real-time system can cause the irregular functioning of a device or program that requires a real-time system. The reason that the set point was missed in the first revolution, was due to the fact that the program could not access the angle information from the card and thus missed the range in which it must stop. This was most probably due to the OS assigning another task a higher priority which preempted the pedestal controller when it was passing through this range.

8.3 Determining the Accuracy of the System

The accuracy of the system was quantified, although due to the statistical nature of the data on which the controller was designed, this accuracy took the form of a statistical range. 70 measurement were used to define this range on the elevation axis and 200 measurements were used on the azimuth axis. These sample sizes were large enough to ensure that a very accurate measure of accuracy was defined.

Chapter 9

Conclusions

The following conclusions have been drawn based on the functionality of the final system and the analysis of the results obtained throughout the project:

1. The accuracy of the movement on the azimuth axis in relation to the synchro reading was calculated to be 0.1023° with a standard deviation of 0.0544°. This measure could be improved by increasing the number of measurements taken.

2. The accuracy of the movement on the elevation axis was calculated to be 0.0193° with a standard deviation of 0.0308°. This measure could be improved by increasing the number of measurements taken.

3. The backlash in the azimuth gearing was approximated to be 0.1892°. This gave an uncertainty in the synchro reading of 0.1892°. Thus the total accuracy of the azimuth axis positioning was calculated to be 0.2915° with a standard deviation of 0.0544°.

4. This approximation for backlash was not very accurate as a lot of opportunities were available in the testing to introduce human error into the measurement.

5. The use of a non real-time Operating System for the API contributed to the error in the movements as real-time measurements could not always be obtained.

6. The controller implemented only used the current position of the pedestal to control the speed of movement. Thresholds were also defined according to experimental data which only related to the exact pedestal configuration used. If the weight of the antenna is changed, the dynamics of the system will be altered and the controller thresholds would be invalid.

7. The maximum speed of movement for the azimuth axis was not realised due to the azimuth control hardware not being able to output a 100% duty cycle due to the 3.7V Parallel Port inputs.

8. There are many non-linearities in the movement of each axis due to the wear of the gearing systems. Without conducting appropriate tests and deriving a control transfer function, the dynamics of the system cannot be predicted and a controller with the maximum possible accuracy and minimum response time cannot be constructed.

9. The overshoot for each speed was averaged over both directions, this posed a problem for the elevation axis as the weight of the antenna changes the overshoot depending on the current position. Thus the results for each direction were very different and averaging them gave the best solution for the controller designed in this project, but limited the response of the system because undershoot and overshoot occurred often and the fine motor control method had to recover from this error.

Chapter 10

Recommendations

The following recommendations are made for future work:

1. Use commercial backlash measuring equipment and procedures to obtain a very accurate measurement for the backlash in the gearing systems.

2. Use a real-time Operating System like RTLinux to allow for process priorities and scheduling algorithms for the system to be set. This will ensure the system receives real-time measurements from the synchros and can perform very accurate control.

3. Increase the resolution (number of bits) of each DAC. This will enable the system to run at more speeds and thus give better control. A separate Parallel Port for the elevation and azimuth axes would be best.

4. Use a driver circuit to ensure that the logic HIGH voltage from the Parallel Port enters the DAC as a 5V signal.

5. Model the movement of the axes using control theory to obtain a transfer function for the pedestal. This will allow for a controller which uses both speed and position to control the movement of the axes.

Bibliography

[1] Synchros and Resolvers [Training Presentation]. Gasking, J; North Atlantic Industries;.

[2] PWM Control MkII;. Available from: `http://www.cpemma.co.uk/pwm_erg.html`.

[3] L298 [Datasheet]. STMicroelectronics; 2000.

[4] LMD18200 [Datasheet]. National Semiconductor; 1999.

[5] PCI-76CS1 [Datasheet]. North Atlantic Industries; 2007.

[6] Model 033-2T and 066-2T Tracking Pedestals. L-3 Brashear; [cited 20 October 2008]. Available from: `http://www.l-3com.com/brashear/markets/033-2t.htm`.

[7] A COMPARISON OF TRACKING PEDESTAL GEOMETRIES [Manual]. ViaSat;. Available from: `http://www.viasat.com/files/assets/support/manuals/ComparisonofPedestalGeometries.pdf`.

[8] George G. A Control and Measurement System for an Elevation over Azimuth Antenna Pedestal. University Of Cape Town, South Africa; 2007.

[9] Synchro and Resolver Engineering Handbook. MOOG Components Group Inc.; 2004.

[10] Gasking J, Agosti R, Dayton D, Freilich A, Grandner W, Haber F, et al. Synchro and Resolver Conversion Handbook (Abridged Version). North Atlantic Industries; 2005.

[11] Johnson MW. VXI synchro/resolver and phase-angle voltmeter COTS resources for F-15 AADTS program ATE. In: Proc. IEEE AUTOTESTCON; 2000. p. 554–558.

[12] Personal Communication [Email]. Riccobono F, Shafy A; North Atlantic Industries; 2008.

[13] Gloassary of Meteorology. American Meteorological Society; [cited 20 October 2008]. Available from: `http://amsglossary.allenpress.com/glossary/browse?s=b&p=2`.

[14] Unit Testing. Microsoft Developer Network; [cited 20 October 2008]. Available from: `http://msdn.microsoft.com/en-us/library/aa292197(VS.71).aspx`.

[15] LM324 [Datasheet]. National Semiconductor; 1994.

[16] 2N2222A [Datasheet]. STMicroelectronics; 1999.

[17] A DMOS 3A, 55V, H-Bridge: The LMD18200 [Application Note 694]. National Semiconductor; 1999.

[18] KA78XX/KA78XXA [Datasheet]. Fairchild Semiconductor; 2001.

[19] Writing Functional Tests [Wiki]. Zope;. Available from: `http://wiki.zope.org/zope3/ftests.html`.

[20] Larson HJ. Introduction to Probability Theory and Statistical Inference. 2nd ed. John Wiley & Sons; 1974.

[21] The Normal Distribution. Department of Statistics, Stanford University; [cited 20 October 2008]. Available from: `http://www-stat.stanford.edu/~naras/jsm/NormalDensity/NormalDensity.html`.

[22] Silberschatz A, Galvin PB, Gagne G. Operating System Concepts. 7th ed. John Wiley & Sons; 2005.

[23] Systems Engineering Guidebook for ITS. U.S. Department of Transportation Federal Highway Administration; [cited 20 October 2008]. Glossary. Available from: `http://www.fhwa.dot.gov/cadiv/segb/views/document/Sections/Section8/8_1_1.htm`.

[24] Lagerberg A, Egardt B. Backlash Estimation With Application to Automotive Powertrains. 2007;15(3):483–493.

[25] Lee KM, Zhou D. A real-time optical sensor for simultaneous measurement of three-DOF motions. 2004;9(3):499–507.

[26] Villwock S, Pacas M. Time Domain Identification Method for Detecting Mechanical Backlash in Electrical Drives. IEEE Transactions on. 2003;p. 1. Accepted for future publication Industrial Electronics.

[27] Zhou J, Wen C, Zhang Y. Adaptive backstepping control of a class of uncertain nonlinear systems with unknown backlash-like hysteresis. 2004;49(10):1751–1759.

[28] CD4043B [Datasheet]. Texas Instraments; 2003.

[29] CD4081B [Datasheet]. Texas Instraments; 2003.

[30] LM358 [Datasheet]. STMicroelectronics; 2002.

[31] HCF4069UB [Datasheet]. STMicroelectronics; 2001.

[32] TIP122 [Datasheet]. STMicroelectronics; 2000.