ADAPTABLE ROBOTIC GRASPING

Brendan Menzies

Bachelor of Engineering
Mechatronic Engineering

Department of Mechatronic Engineering
Macquarie University

November 7, 2016

Supervisor: Subhas Mukhopadhyay

ACKNOWLEDGMENTS

I would like to acknowledge my academic supervisor Professor Subhas Mukhopadhyay for providing his knowledge in the field of study and for his guidance throughout the project. I would also like to thank PhD student Anindya Nag for his help on working with sensors and equipment.

STATEMENT OF CANDIDATE

I, Brendan Menzies, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Electronic Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment an any academic institution.

Student's Name: Brendan Menzies

Student's Signature: B. Menzies

Date: 07/11/2016

ABSTRACT

Automation is being integrated into a wider range of tasks every year as the technology increases to allow its development and growth. A machine that is able to capably perform more than one task, or be able to adapt to unknown circumstances is more beneficial than a machine that cannot. This project investigates the capabilities of low cost tactile sensors being used to determine different properties of gripped objects to allow for an adaptable grasping system. These properties include shape, material type and contact area. A circuit was developed to excite the sensors and condition the sensor signal so that the information can be read by an Arduino Uno microcontroller. Alongside the sensor research, a program was made using Matlab to control a five degrees of freedom robotic arm, with a user interface. The program was also used to control the gripper based on the sensor data. The tests performed on the sensors investigated the sensor characteristics, variable force tests, force response test, phase shift tests and object shape identification tests. With the system developed and tested, there was no strong indication that any of the force-sensing resistors or interdigitated capacitive sensors could determine object properties. With some modifications, this system could be used to investigate other types of sensors to determine their capabilities in object determination.

Contents

List of Figures

List of Tables

Chapter 1

Project Introduction

1.1 Introduction

Automation has become a vital part of our ability to produce, manufacture and transport most of our resources around the world. Automation is used for its advantages over human labour, such as the cost over time being cheaper, the faster work rate, and the longer work hours that can be performed. While these aspects exceed human efforts, not all parts of human labour can be replaced. One of these factors is our ability of adaptation. Automated robots are generally designed to perform specific tasks within a limited set of parameters. Tasks that fall outside of the parameters will likely fail, or not be performed.

Early automation used clean, simple shapes, sizes and weights, such that manipulating these objects could be done reliably without the need for feedback from the end effector. Designs like this allowed for large margins of error. Current robots use various sensors to determine where the robot is and the status of the object being manipulated. Information from the sensors is fed into a controller which decides on the action the robot will take to meet its goal. Systems like this also commonly use vision systems to monitor the environment, which may not be a practical approach in all cases of this type of automation.

This project aims to investigate a selection of low cost sensors to determine whether they can gather enough information about an object to assist with grasping. The two sensor types investigated are interdigitated capacitive sensors and force-sensitive resistor sensors. With the developed circuit, a sensor can obtain the voltage change and the phase shift when a force is applied. This information may provide enough information to determine the shape of the object, and therefore the force to grasp it. This could lead to a simple and cost effective way for a robot to determine and grasp separate shapes differently.

1.2 Project Overview

This section will give a brief overview to the project.

1.2.1 Project Scope

The main goals that the project should achieve are as follows:

- Research, design and prototype an adaptable robotic grasping system.
- Investigate different tactile sensors viability in the system.
- Program a controls system and user interface for the robotic arm.
- Integrate a controls system for the griper.

1.2.2 Project Outcomes

The outcomes from this project include:

- Test results from sensors
- Program to control robotic gripper
- Signal processing circuit
- Sensor control program

Chapter 2

Background

This background will provide some detail about some of the research areas surrounding the project.

2.1 Robotic Grasping

2.1.1 General

Robotic grasping is a wide area of study that covers several aspects of automation. Basic robotic grasping uses strict object shapes for the gripper to grasp, which minimises the chance of error in the task. Adding basic proximity or tactile sensors allow this sort of system to have basic automation. This approach is reliable and economic for simple grasping tasks, but fails when any factors are introduced to the system that the gripper cannot account for.

In order to have a grasping system that can work in unknown environments, it becomes a more expensive endeavour, such as the 3DOF robotic arm by Dollar and Howe [4]. This arm uses separate joints in the end effector which are able to conform to the grasped objects shape. This gripper is also designed to be robust and can accommodate a variety of objects. Limitations of this design are its cost and the need to manufacture special fingers so that the end effector can grip a large variety of objects.

Another method to enhance grasping capabilities is to add a learning function to the system, such as the one developed by Saxena and others [11]. This approach takes a still image of the object and the system tries to learn the best approach to pick up the object. Some examples of the objects that this system were able to grasp were wine glasses, keys, phones, knives, and other household objects. Limitations of system like this are that objects similar to these are needed to be modelled prior to the object being identified. Another limitation is that the object needs to be visible for the image to be taken.

Other methods of grasping also use optical feedback, but instead of imaging the object once, constant images are taken so that changes in the state of the object can be seen [3] [12]. This method is beneficial as it allows a program to be made that can adjust the end effector whilst the object is being carried in response to any shift in the objects position.

Since this is a smaller project, the budget has to be kept in mind. The robotic arm used will be one that is being used for teaching, which is beneficial as the robotic arm is the most expensive part of the project costs.

2.1.2 Dynamixel Servos

Research into the Dynamixel motors done by Vacek and others [14] on a 5DOF robotic arm investigates the characteristics of the AX and MX series of the Dynamixel servo motors. It is seen that the MX series provides more accurate positioning according to parameters as it uses a full PID controller, which the AX series does not have.

2.2 Tactile Sensors

Tactile sensors are able to convert mechanical energy into electrical signals. Tactile sensors will be used in the project to determine states and characteristics of objects to be gripped by the grasping system. There are many different types of tactile sensors which employ different methods of sensing tactile information [9].

2.2.1 Force Sensitive Resistors

Force-sensitive resistors (FSR), or force sensing resistors, are sensors that change their resistance in relation to the force applied to the sensor. The main benefits of using FSRs is that they are easily available, easy to use and are cheap in comparison to other types of tactile sensors. The principal behind FSRs is based on voltage division, seen in Equation 2.1 and Figure 2.1.

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in} \tag{2.1}$$

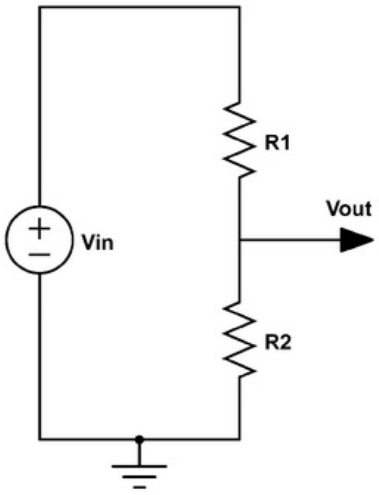


Figure 2.1: Voltage Division Circuit

If R_1 is the FSR in Equation 2.1, it can be seen that as R_1 is adjusted by applying force, V_{out} will also be adjusted proportionally. This is the basis of how an FSR is implemented into a circuit.

The construction of a FSR consists of two main parts, the electrodes and the piezoresistive separator. The circuit is connected through the electrodes and the piezoresistive material restricts the current flow. The force applied to the sensor determines the resistance of the piezoresistive material, and thus how much current is passed through the sensor. The two FSR sensors used in the project have two separate methods of positioning the electrodes. The Flexiforce sensor has the electrodes on the top and bottom of the sensor with the piezoresistive material in between, seen in Figure 2.2, while the Intertek sensor uses interdigitated electrodes with the piezoresistive material filling the empty space, seen in Figure 2.3. This difference will give each sensor different characteristics.

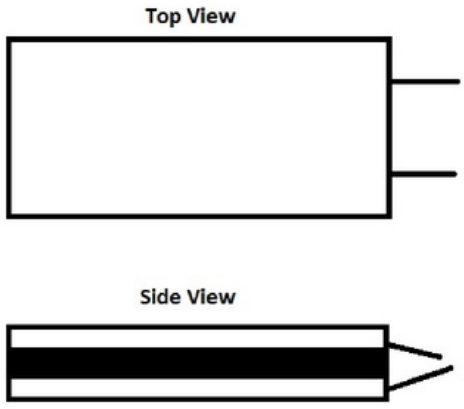


Figure 2.2: Flexiforce Sensor Diagram

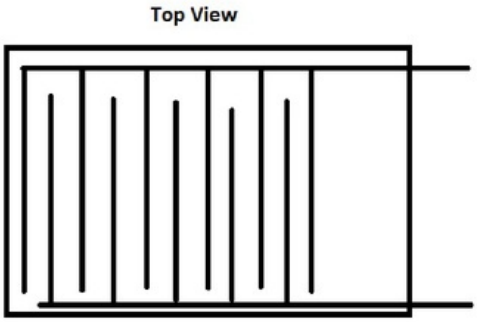


Figure 2.3: Intertek Diagram

An investigation into the characteristics of some commercial FSRs by Hollinger and Wanderley [6], looks at two of the FSRs that are similar to the ones used in this project. These are made by the same manufacturers, but are not the same sensors. These are made by Tekscan and Intertek. The factors looked into are resistance drift and hysteresis, or lag, of the sensors. These tests determined a few factors, firstly that the Flexiforce sensor provided a more repeatable result than the Intertek sensor, however the Intertek sensor performed better where a quick change needed to be measured. Another experiment on the two main manufacturers by Vecchi and others [15], focused on applying the sensors to biomechanics and motor control applications. They found similar results to Hollinger and Wanderly, where the Flexiforce performed better in terms of linearity, repetition of results and time drift. The tests did not however address the issue of the Flexiforce sensors response time to high frequency loads.

A nonlinear model of the Flexiforce and Intertek sensors was investigated by Lebosse and others [7]. By studying the nonlinear properties of the sensors, through observing the static and dynamic properties of both sensors, they are able to create a model to compensate for these factors. The model found that the sensors nonlinearity is more severe for the dynamic behaviour of the sensors. Once the sensors are fully integrated into the adaptable grasping system, this would be beneficial to implement to improve accuracy.

2.2.2 Interdigitated Sensors

Some of the sensors used within the project are based on the work of A. Nag [10]. These sensors are thin film sensors that use interdigitated electrodes on a flexible thin film. These sensors have uses other than tactile sensing [16]. An alternating current is passed across the sensor and the voltage is measured across the resistor in parallel. Elements that will effect the circuit are the frequency and voltage of the source, as well as the value of the sensor resistor, R_s. The circuit for the sensor can be seen in Figure 2.4.

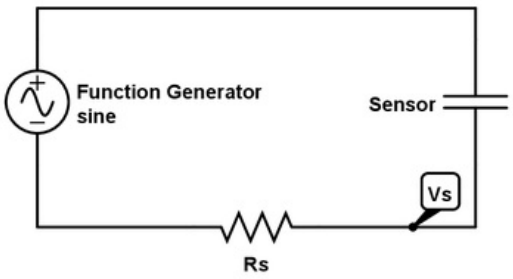


Figure 2.4: Interdigitated Sensor Circuit

The sensor behaves similar to a capacitor. Since the film is flexible, the capacitance changes as the sensor is distorted from mechanical contact. The standard equation for capacitance of a capacitor is as follows:

$$C = \frac{\epsilon d}{A} \tag{2.2}$$

Where, C is the capacitance of the material.
ϵ is the relative permittivity.
d is the distance between electrodes.
A is the area of the sensor patch.

When the patch is contacted, the values of A and d change. This gives the equation:

$$C' = \frac{\epsilon d'}{A'} \tag{2.3}$$

This change in capacitance can be measured and thus the mechanical impact on the circuit can be seen.

2.2.3 Other Tactile Sensors

Force/Torque sensors [8] use strain gauges on a shaft to detect the movements of the shaft in response to mechanical changes, seen in Figure 2.5.a. The tip of the sensor is covered with a rubber cover to aid gripping.Another tactile sensor type is a tactile matrix sensor [8] which uses an array of smaller sensors to give an idea of the position of the force, seen in Figure 2.5.b. The sensor uses a conductive rubber which alters its conductivity in response to force applied to the sensor. The array of these sensors is arranged in a 16 by 16 grid, giving 256 different sensors. This allows the determination of the force location and even the contact size to be measured.

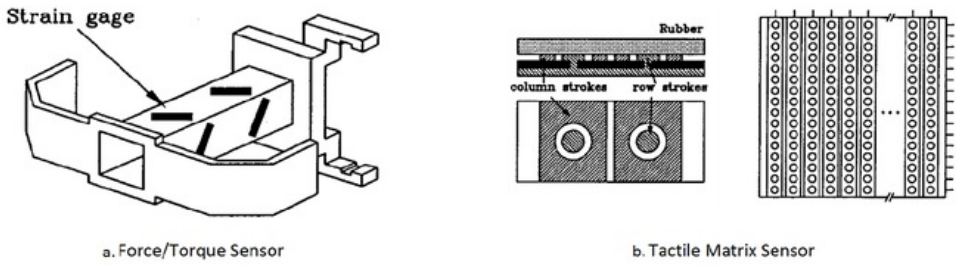


Figure 2.5: The Force/Torque Sensor and Tactile Matrix Sensor

The center of pressure (CoP) sensor [5] is another thin film type sensor, seen in Figure 2.6. This sensor is advantageous due to its physical flexibility and the ability to distinguish the force position and distribution across the sensor. Through processing the information, it is possible to compute the grasping force required and the friction coefficient of the object being grasped. By analysing this information and feeding it back into the gripper, it is possible to accommodate for slip detection. The response time of the sensor is around 1ms, which is ideal for use in feedback. The sensor also only requires four wires to operate. The sensor uses a pressure sensitive material between to films of electrodes. As pressure is applied to the sensor, the current between the electrodes changes and can be measured.

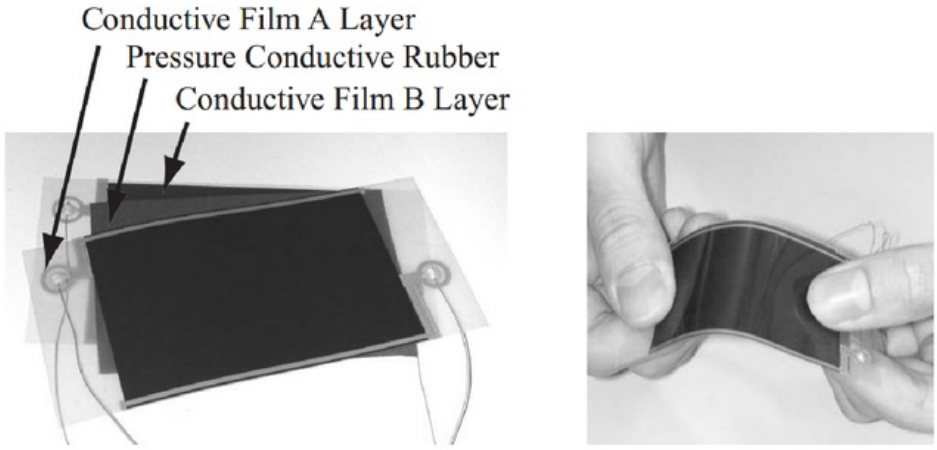


Figure 2.6: Center of Pressure Sensor

These methods of tactile sensing are not ideal for the current project due to the higher cost of these sensors in comparison to FSRs and the interdigitated sensors.

Chapter 3

Design of the System

3.1 Introduction

This chapter looks at the specifics to the design of the adaptable grasping system.

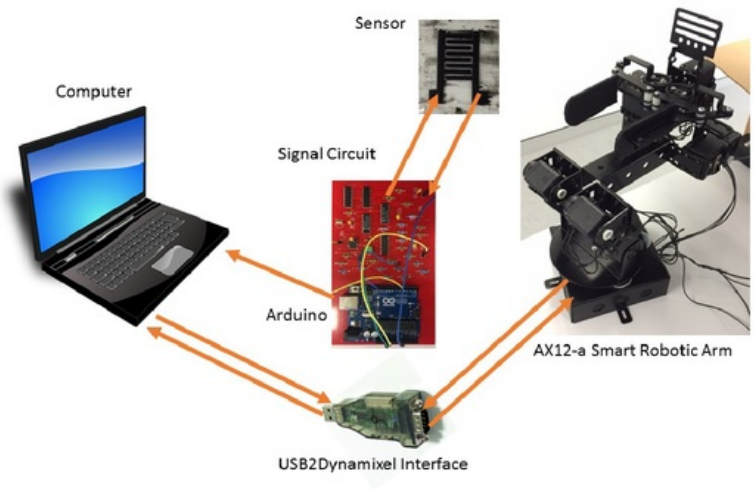


Figure 3.1: The Adaptable Robotic Grasping System

Figure 3.1 shows how the components of the system connect to one another, and the direction of communication between the systems. The overall system is comprised of several components, each performing distinct functions within the system. The components are as follows:

- Microcontroller
- Direct Digital Synthesis of Sinusoidal Wave
- Sensors
- Signal Circuits
- Robotic Arm
- Servos
- Controls

3.2 Microcontroller

The microcontroller used for the system is the Arduino Uno SMD version, which can be seen in Figure 3.2. The microcontroller is used for the generation of the sinusoidal waveform that is needed as an excitation signal for the sensor. By using an Arduino for the wave generation and not a function generator the system is significantly more portable. In addition, it is possible to implement wireless data transmission between a computer and the Arduino, which would allow the user to remotely operate the system, provided the user also connected to the robotic arm in a similar manner. The actual microcontroller is the ATmega328 chip mounted to the Arduino PCB. The difference between the SMD edition and the regular editions of the Arduino Uno is how the controller chip is mounted to the PCB. The SMD version uses a surface mount ATmega328 while the other editions use a through hole mount.

3.2.1 Microcontroller Specifications

The specification for the microcontroller are as follows:

Input Voltage:	7-12 V
Operating Voltage:	5 V
Max Operating Frequency:	20 MHz
Clock Frequency:	16 MHz
Digital I/O Pins:	14 pins, 6 of which are PWM
Analog Input Pins:	5
DC Current per I/O Pin:	40 mA
DC Current for 3.3 V Pin:	50 mA
Timers:	3
Flash Memory:	32 KB
SRAM:	2 KB
EEPROM:	1 KB

Table 3.1: Arduino Uno SMD Edition Specifications

Figure 3.2: Arduino Uno SMD Edition

3.3 Direct Digital Synthesis of Sinusoidal Wave

Direct Digital Synthesis (DDS) is a method of generating a sinusoidal waveform using discrete digital points. This is achieved by breaking up an analog waveform into a set of discrete points and storing them in a lookup table. The discrete points used can be seen in figure 3.3, where the points are stored in hexadecimal values. This lookup table is called from within the Arduino program. The Arduino program was not developed for this project, but using a code for DDS found from an online resource [1].

```
const int sinewave_length=256;
const unsigned char sinewave_data[] PROGMEM = {
0x80,0x83,0x86,0x89,0x8c,0x8f,0x92,0x95,0x98,0x9c,0x9f,0xa2,0xa5,0xa8,0xab,0xae,
0xb0,0xb3,0xb6,0xb9,0xbc,0xbf,0xc1,0xc4,0xc7,0xc9,0xcc,0xce,0xd1,0xd3,0xd5,0xd8,
0xda,0xdc,0xde,0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xed,0xef,0xf0,0xf2,0xf3,0xf5,
0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfc,0xfd,0xfe,0xfe,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xfe,0xfe,0xfd,0xfc,0xfc,0xfb,0xfa,0xf9,0xf8,0xf7,
0xf6,0xf5,0xf3,0xf2,0xf0,0xef,0xed,0xec,0xea,0xe8,0xe6,0xe4,0xe2,0xe0,0xde,0xdc,
0xda,0xd8,0xd5,0xd3,0xd1,0xce,0xcc,0xc9,0xc7,0xc4,0xc1,0xbf,0xbc,0xb9,0xb6,0xb3,
0xb0,0xae,0xab,0xa8,0xa5,0xa2,0x9f,0x9c,0x98,0x95,0x92,0x8f,0x8c,0x89,0x86,0x83,
0x80,0x7c,0x79,0x76,0x73,0x70,0x6d,0x6a,0x67,0x63,0x60,0x5d,0x5a,0x57,0x54,0x51,
0x4f,0x4c,0x49,0x46,0x43,0x40,0x3e,0x3b,0x38,0x36,0x33,0x31,0x2e,0x2c,0x2a,0x27,
0x25,0x23,0x21,0x1f,0x1d,0x1b,0x19,0x17,0x15,0x13,0x12,0x10,0x0f,0x0d,0x0c,0x0a,
0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x03,0x02,0x01,0x01,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x02,0x03,0x03,0x04,0x05,0x06,0x07,0x08,
0x09,0x0a,0x0c,0x0d,0x0f,0x10,0x12,0x13,0x15,0x17,0x19,0x1b,0x1d,0x1f,0x21,0x23,
0x25,0x27,0x2a,0x2c,0x2e,0x31,0x33,0x36,0x38,0x3b,0x3e,0x40,0x43,0x46,0x49,0x4c,
0x4f,0x51,0x54,0x57,0x5a,0x5d,0x60,0x63,0x67,0x6a,0x6d,0x70,0x73,0x76,0x79,0x7c};
```

Figure 3.3: Lookup Table for the DDS Sine Wave [1]

The output from the Arduino will be a square wave pulse width modulation (PWM) signal. This signal is too sporadic to use as an excitation signal for the sensor. A low pass and high pass filter are used as a band pass filter to produce the smooth sinusoidal signal from the PWM signal, which can be used as the excitation signal for the sensor.

3.4 Sensors

Five different sensors have been investigated to assess the validity of each sensors integration for use in adaptable grasping. Ideally the sensors will be able to provide information of the force of the gripper and details on the object being gripped, such as shape or size, using the phase shift of the output signal. The Flexiforce sensor and the Intertek piezoresistive sensor are FSRs, while the other three sensors are thin film sensors that use interdigitated electrodes to sense contact with the sensor. These will be referred to as:

- Flexiforce A401 Sensor
- Intertek piezoresistive Sensor
- PDMS (Polydimethylsiloxane) Sensor
- Graphene Sensor
- Aluminium Sensor

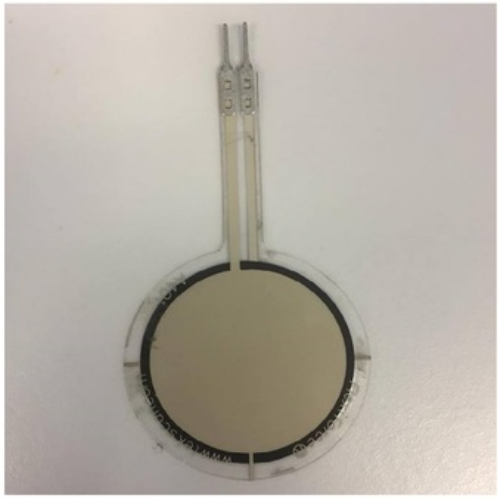


Figure 3.4: Flexiforce A401 Sensor

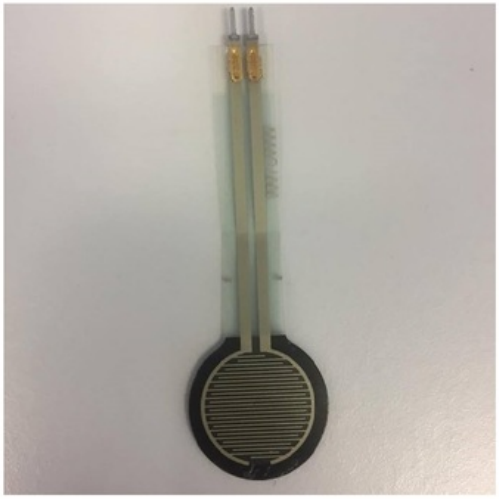


Figure 3.5: Piezoresistive Sensor

Figure 3.6: PDMS Sensor

Figure 3.7: Graphene Sensor

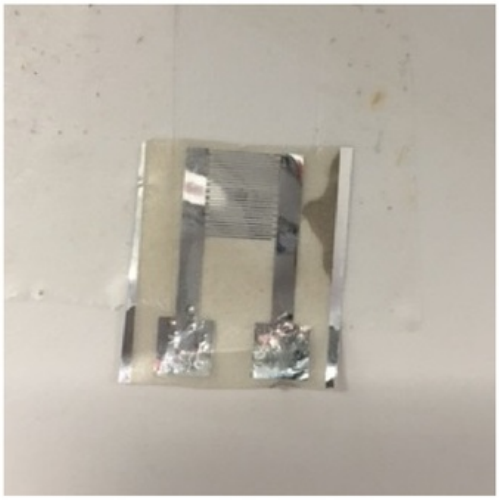


Figure 3.8: Aluminium Sensor

The PDMS sensor is made of PDMS backing and a mixture of carbon nanotubes and PDMS for the electrodes. The graphene sensor uses graphene for the electrodes and polyimide as the backing material. The aluminium sensor uses aluminium as the electrodes and a PET film as the backing.

3.5 Signal Circuits

There are several different sections of the signal circuit that perform different functions, including buffers and filters that prevent the sections interacting unintentionally. The sections include:

- Power Circuit
- Sensor Circuit
- Rectifier Circuit
- Zero Crossing Detector (ZCD)
- Phase Shift Detecting Circuit
- Low-Pass/High-Pass Filter Circuit
- Buffer Circuit
- Amplifying Circuit
- Voltage Inverter Circuit

3.5.1 Power Circuit

The power circuit, seen in in Figure 3.9, is used to convert the PWM signal from the Arduino into a smooth sinusoidal waveform. A low-pass filter and a high-pass filter are used together to create a bandpass filter. The low-pass filter eliminates the quick response of the PWM signal which leaves the sinusoidal signal. The high-pass filter eliminates the DC offset from the input signal, leaving a bipolar sinusoidal input signal at a frequency of 870 Hz. To calculate the correct components for the correct cut-off frequencies the following formulae is used:

$$f_c = \frac{1}{2\pi RC} \tag{3.1}$$

Where f_c is the cut-off frequency for the filter, R is the value of the resistor, and C is the capacitance of the capacitor. If a resistor value of 10 KΩ is selected arbitrarily, the equation can be rearranged to find a value of C_1, so that frequencies above 700 Hz are passed through.

$$C_1 = \frac{1}{2\pi R f_c} = \frac{1}{2\pi 10000 * 700} = 22.7nF \tag{3.2}$$

Similarly, to filter frequencies below 1 KHz, a value of 10KΩ is used as the resistance and the value of C_2 can be found.

$$C_2 = \frac{1}{2\pi R f_c} = \frac{1}{2\pi 10000 * 1000} = 15.9nF \tag{3.3}$$

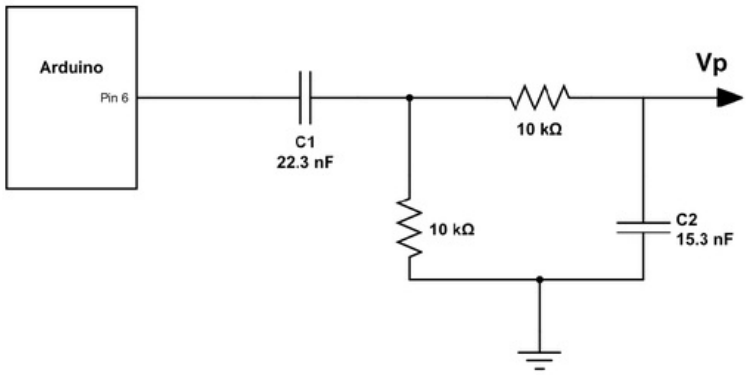


Figure 3.9: The Power Circuit

3.5.2 Sensor Circuit

The sensor circuit is where the sensors are connected to the circuit, seen in Figure 3.10. The power circuit feeds the sinusoidal signal into the sensor, then through a resistor in series and back to ground. The voltage at the node between the sensor and the sensor resistor R_s is altered as the resistance of the sensor changes. This is due to voltage division, and can be seen in the following formula.

$$V_s = \frac{R_2}{R_1 + R_2} V_p \tag{3.4}$$

Where V_p is the input voltage, R_1 is the varying resistance of the sensor, R_2 is the set resistor, and V_s is the output voltage. Since the input voltage V_p is sinusoidal in nature, the output voltage V_s is also sinusoidal. The rest of the circuits will be manipulating the voltage V_s and comparing it to the input.

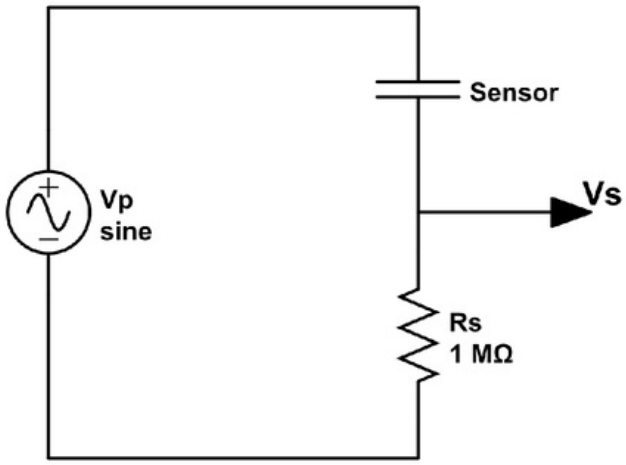


Figure 3.10: The Sensor Circuit

3.5.3 Rectifier Circuit

To read the signal, an Arduino microcontroller takes the signal into its analog input. The analog input cannot take a negative voltage, therefore the signal needs to be rectified to remove any negative parts. The rectifier circuit converts the bipolar sinusoidal wave into a rectified wave, bringing the negative halves of the curve to the positive side. Since the signal has a relatively low amplitude, a diode based rectifier would affect the signal through forward voltage drops. So the circuit used is an op-amp based circuit using the LM324 quad input chip. The circuit can be seen in Figure 3.11.

The top op-amp takes the sine wave in, and since it is referenced from ground to 5 V, the negative half of the sign wave is omitted. The diode allows the output to pass through to the input. This feedback allows only the positive half of the input and the negative halves are set to ground.

The bottom op-amp is a unity gain inverting amplifier. This takes the input and inverts the waveform, but since the op-amp has no negative reference, the negative half of this inverted wave is grounded. This half rectified waveform fills where the grounded parts of the above waveform sit.

The final op-amp on the right sums the two half rectified waveforms into a complete rectified wave, which is done using a summing amplifier.

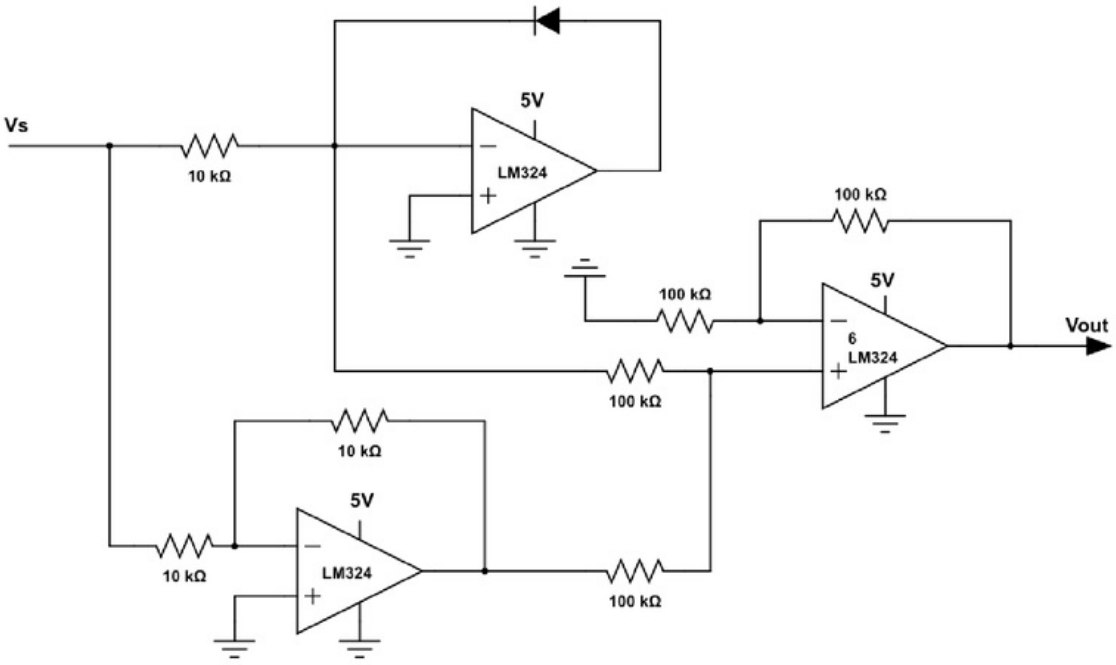


Figure 3.11: Rectifier Circuit

The amplitude of the wave changes as the sensor is interacted with. Unfortunately the Arduino cannot read an alternating signal easily. A filter can be used to transform the rectified sine wave into a DC signal. This is done using a low pass filter, where the filtered frequency is around one tenth of the signal frequency. The circuit can be seen in Figure 3.12.

$$f_c = \frac{1}{10} f \approx 90Hz \tag{3.5}$$

$$C = \frac{1}{2\pi R f_c} = \frac{1}{2\pi 1000 * 90} = 1.68\mu F \tag{3.6}$$

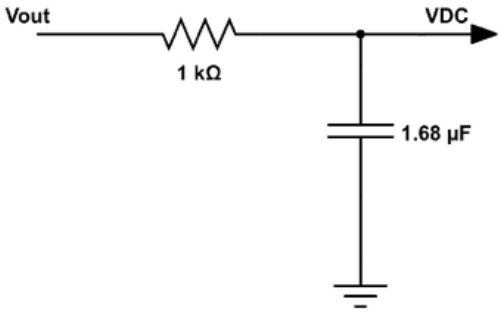


Figure 3.12: Filtering Circuit

The output of the filter will be a DC signal that changes depending on the sensor. The amplitude change of this signal is small, so it is fed into the amplifying circuit, so that a greater sensitivity can be achieved.

3.5.4 Zero Crossing Detectors

The zero crossing detectors (ZCD) are used to convert the sinusoidal waveforms into square waves. The ZCD is made using the Schmitt Trigger, which is a hysteresis comparator circuit. These square waves are then passed to the phase shift detecting circuit to be processed.

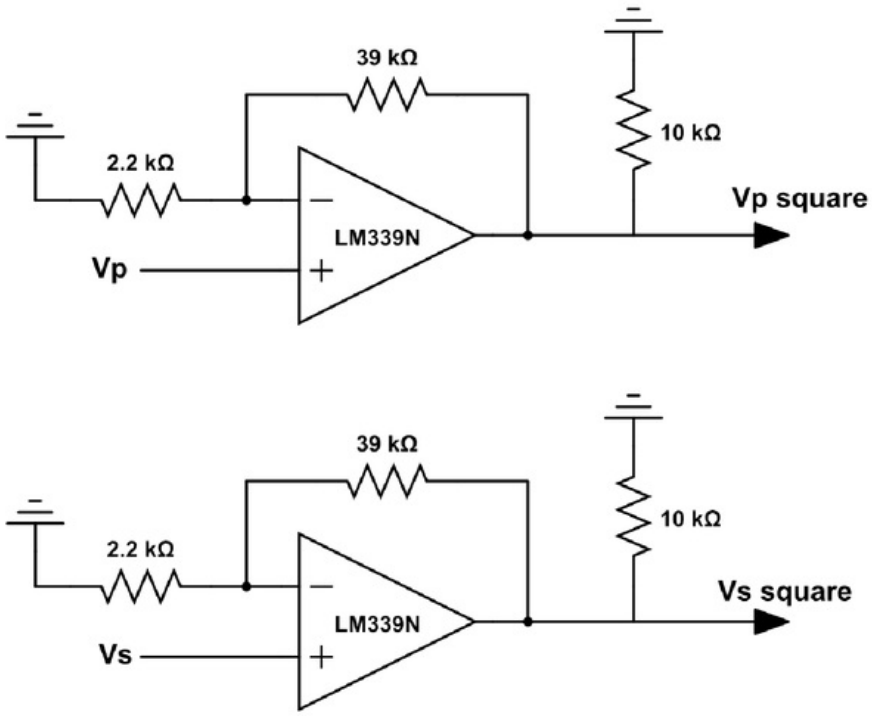


Figure 3.13: Zero Crossing Detector Circuits

3.5.5 Phase Shift Detecting Circuit

The phase shift circuit takes the square wave outputs from the ZCDs and feeds them into a NAND gate, seen in Figure 3.14. As the phase shift either increases or decreases, the pulse width output from the NAND gate is changed. When the same same filter used in subsection 3.5.3 is applied to this output, a DC voltage which relates to the phase shift can be obtained. This DC voltage is required to be fed into the Arduino, however the signal is too small, so it must be fed into an amplifier beforehand.

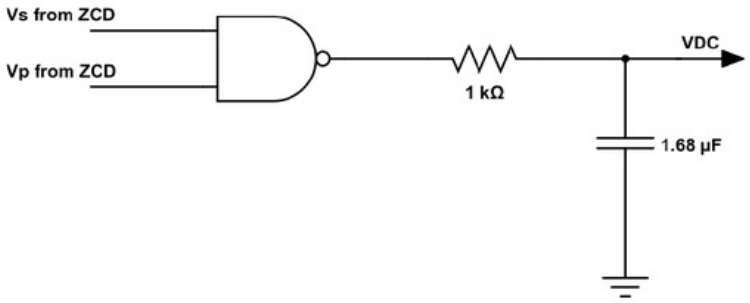


Figure 3.14: Phase Shift Detection Circuit

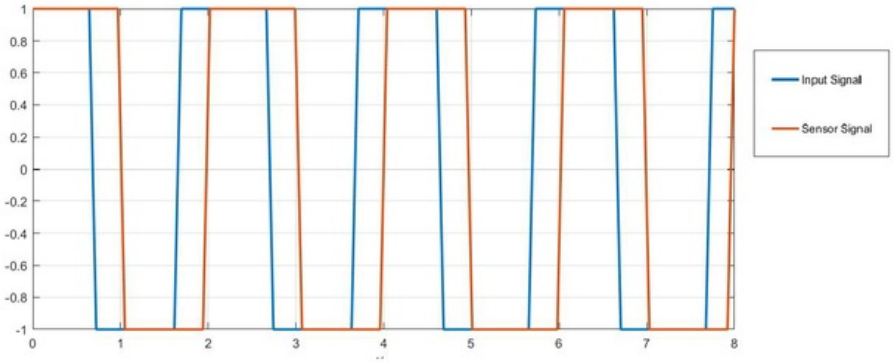


Figure 3.15: The Input Signal and Sensor Signals From the ZCD

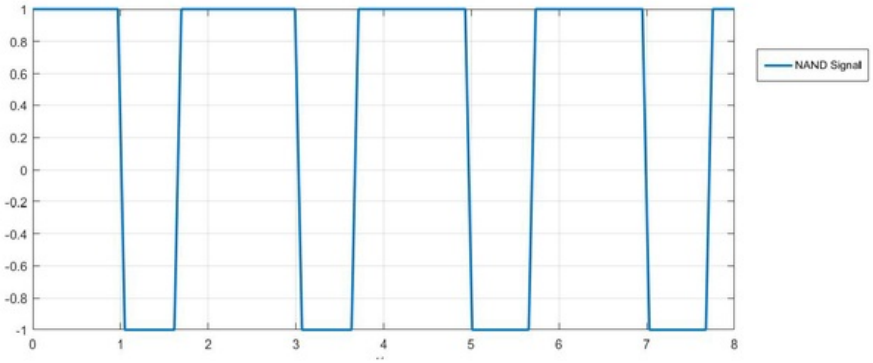


Figure 3.16: The Signal After Passing Through the NAND Gate

The input signals can be seen in Figure 3.15, where the output has a phase shift, relative to the input. Figure 3.16 shows that the signal remains high unless both the input signal and the sensor signal are low. The pulse width of this NAND gate signal changes, and therefore the DC voltage changes after the signal has been filtered.

3.5.6 Buffer Circuit

The buffer circuit is a simple circuit using an op-amp which can relay a signal while stopping other circuit elements from altering the original circuit (circuit loading). The collection of the signal circuit uses four of these buffer amps, by utilising a quad input LM324 op-amp. One buffer is used to retain the original sinusoidal signal from the power circuit, another is used to retain the sinusoidal signal from the sensors voltage. The last two are used to transfer the output from the two ZCDs to the NAND gate. Since the output has a high output impedance and a low input impedance, the signal can be transferred to the next circuit.

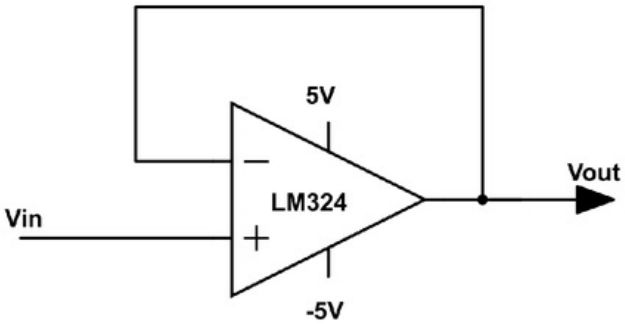


Figure 3.17: Buffer Amplifier Circuit

3.5.7 Amplifying Circuit

The amplifier circuit is required to increase the available range of the DC output from rectifier circuit, as well as the phase shift detecting circuit. Without the amplifier, the DC voltage change is small and difficult to determine smaller changes in voltage. The amplifier uses an LM324 op-amp with the signal to be amplified into the non-inverting input. The output is connected to the inverting input through resistor R_2, this input is connected to ground through resistor R_1. The circuit can be seen in Figure 3.18. The gain of the amplifier depends on the values of the resistors R_1 and R_2. This relationship can be seen in Equation 3.7.

$$Gain = 1 + \frac{R_2}{R_1} \tag{3.7}$$

The signal needed to be amplified, however too much amplification would result in the signal clipping at the maximum value the Arduino can input, 5 V. To fit within these limitations, a gain of four was determined to be sufficient. The values for the resistors was calculated below.

$$4 = 1 + \frac{R_2}{R_1} \Rightarrow R_2 = 3R_1 \tag{3.8}$$

Looking at standard resistance values, the desired gain can be approximated with resistor values of $R_1 = 22K\Omega$ and $R_2 = 68K\Omega$.

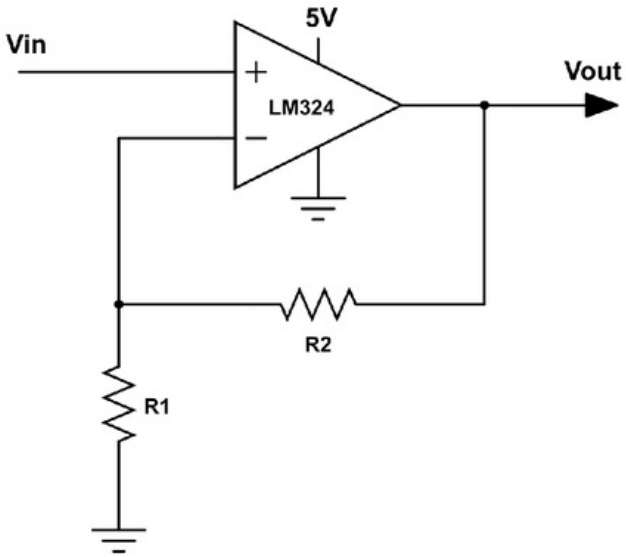


Figure 3.18: Amplifier Circuit

3.5.8 Voltage Inverter Circuit

The voltage inverter circuit uses a DC-DC converter to convert the 5V output from the Arduino into a -5V output. This is necessary for the LM324 chip which supports the four buffer amplifier circuits as the Arduino is only capable of outputting a positive voltage. If the buffer amplifiers do not get a negative supply, any voltage that is put through the amplifier that is below 0V will be clipped at 0V. Since sinusoidal signals are being passed through, it is necessary to go below 0V. The circuit is based on the TC7662B IC chip,

which is a DC-DC converter, commonly used as a voltage inverter. The circuit connections can be seen in Figure 3.19.

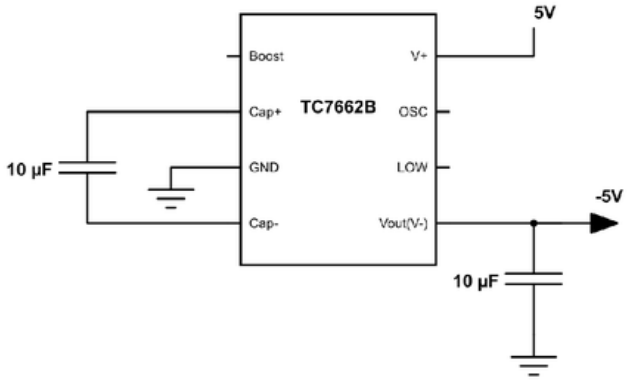


Figure 3.19: Voltage Inverter Circuit

3.5.9 PCB Design and Production

The circuit was connected via a breadboard initially, some images of this can be seen in Figure 3.20.

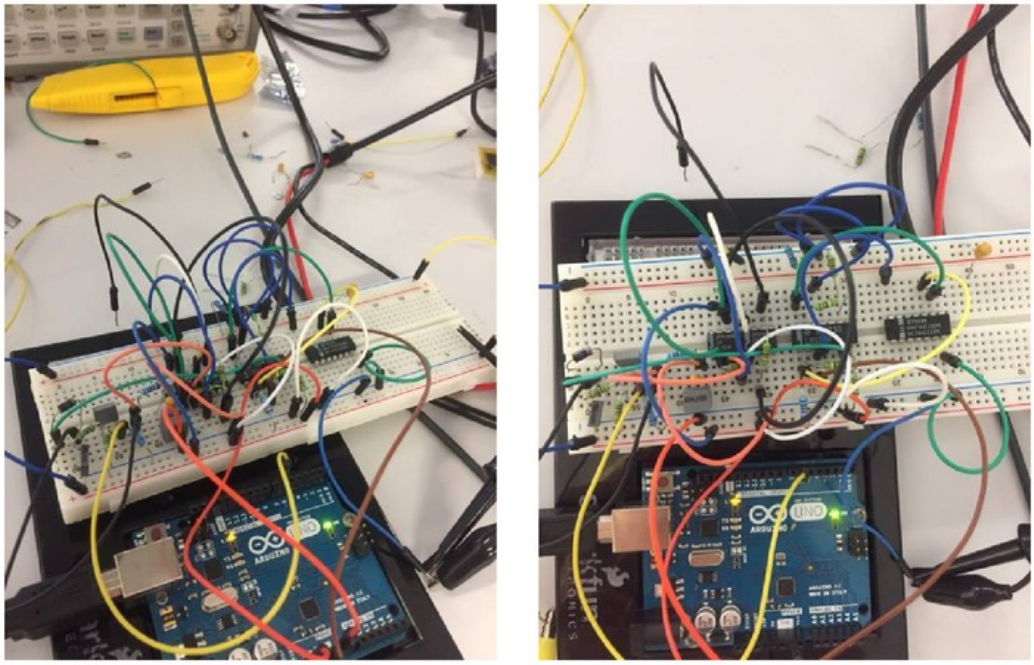


Figure 3.20: The Circuits Assembled on a Breadboard

The issue with this set-up was about the reliability of the circuit. Since the circuit is using an AC voltage of close to 1 kHz, interference can be introduced to the circuit as the wires can act as antenna to transmit and receive. This interference would affect any results of the system. Additional interference may be introduced from the breadboard if the internal connections are faulty, or if any of the tracks are not insulated from an another correctly. To solve this issue, the circuit was recreated into DesignSpark PCB, which can take a circuit and transfer the circuit to a PCB. Figure 3.21 shows the circuit assembled in DesignSpark PCB, Figure 3.22 shows the PCB design in full and Figures 3.23 and 3.24 show the top and bottom layer of the PCB respectively.

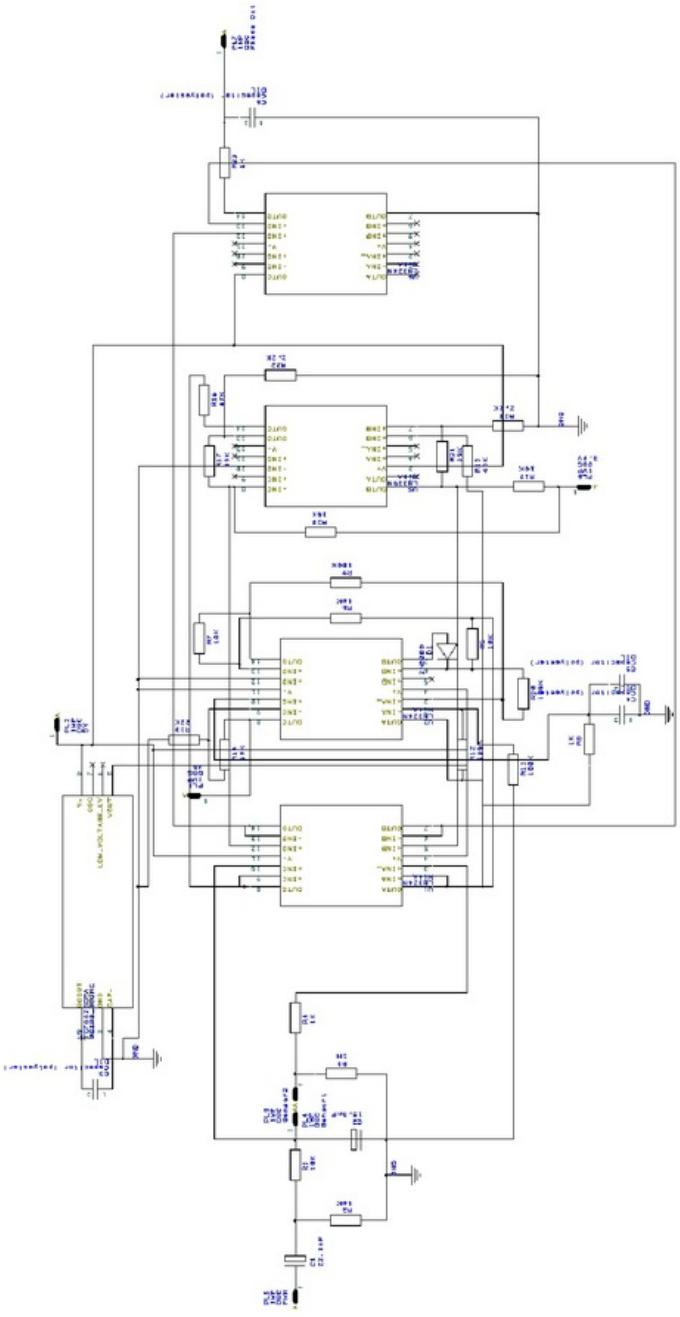


Figure 3.21: The Circuit Made in DesignSpark PCB

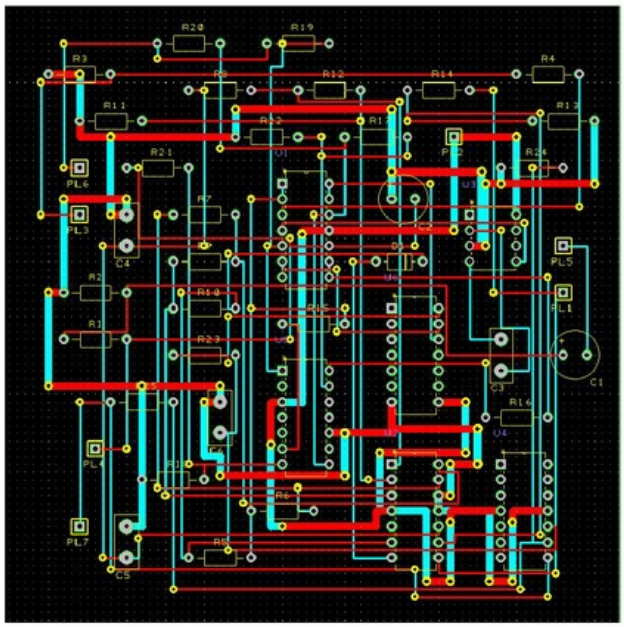


Figure 3.22: The PCB Design With Both Layers

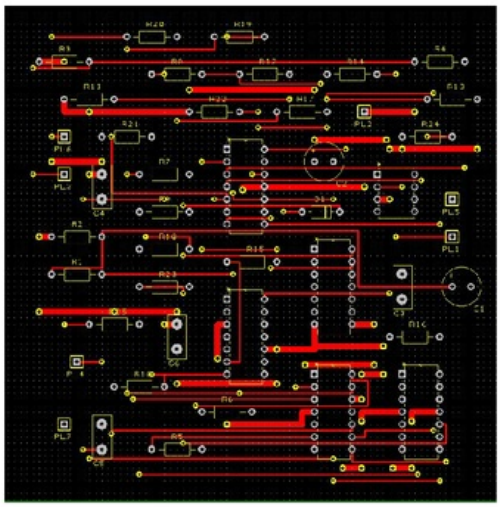


Figure 3.23: Top Layer of the PCB

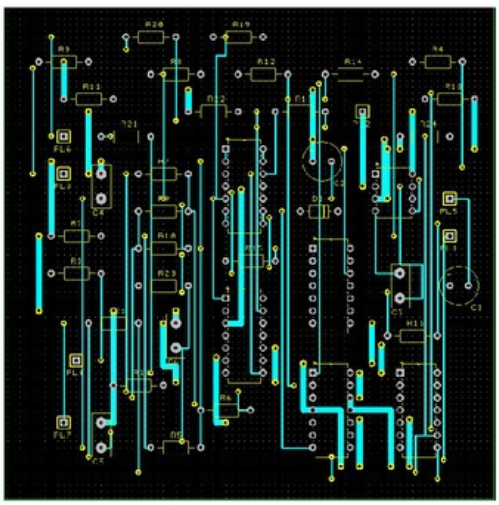


Figure 3.24: Bottom Layer of the PCB

This PCB design was then sent off to be produced. Figure 3.25 shows the empty board that was recieved, Figure 3.26 shows the PCB after it has been populated with the components, and the connections to the Arduino board, which can be mounted to the top. The circuit should now be free from any interference that may have be caused by using a breadboard.

Figure 3.25: The Empty PCB

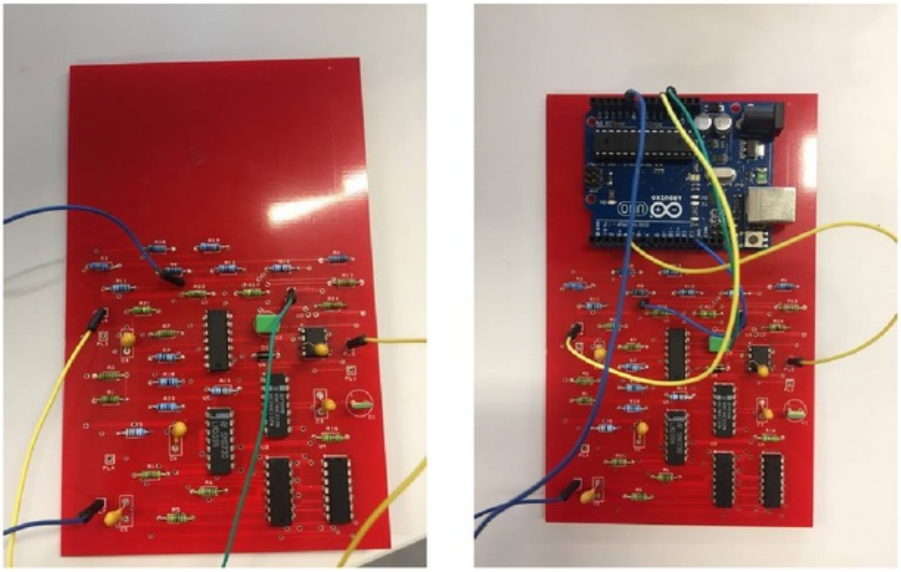


Figure 3.26: The Populated PCB

3.6 Robotic Arm

The robotic arm used was supplied by Macquarie University as a teaching kit, the AX-12/18 smart robotic arm. It has five degrees of freedom, driven by seven separate AX-12A Dynamixel servos. The body of the robotic arm is made from anodised aluminium sections, connected through the servos. The base of the unit is designed to be mounted at three points before use.

Figure 3.27: The AX-12/18 Smart Robotic Arm

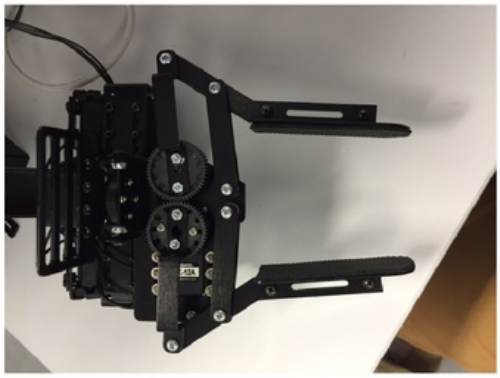


Figure 3.28: Open Gripper

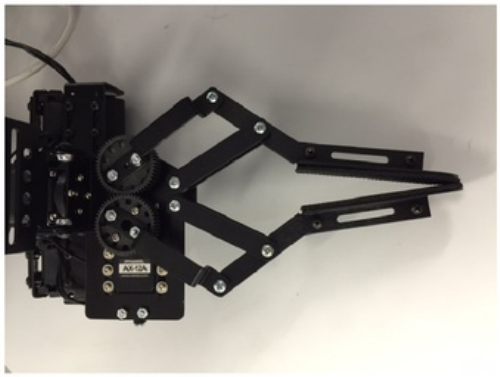


Figure 3.29: Closed Gripper

The gripper included in the robotic arm is an angular gripper with a design which tries to minimise the effect that the angle has on gripping.

3.6.1 Servos

The robotic arm uses the Dynamixel AX-12A smart actuator servos, made by Robotis. Some important specifications are:

- Weight: 54.6g
- Dimension: 32mm x 50mm x 40mm
- Resolution: 0.29°
- Gear Reduction Ratio: 254:1
- Stall Torque: 1.5N.m (at 12.0V, 1.5A)
- No Load Speed: 59rpm (at 12V)

Figure 3.30: The AX-12A servo.

The servos have two connection ports, one for input and one for output. The servos can connect together in series and only connect to the computer via one cable. To allow this, each servo must be assigned an ID from 0-253. The servos require a voltage of 9 - 12V in order to function. They may draw high a high current and therefore cannot be powered via USB interface. They must have an external power supply. The USB2Dynamixel is used to connect the power and the data together.

The AX-12A has several features regular servos do not have. There is an internal register inside the servo which allows you to read and write data to the servo. This allows a variety of values to be read from the servo during operation, such as current position, torque, speed, etc. Additionally parameters can be set for each individual servo this way. For example the maximum movement speed, the angle limits for clockwise and counter clockwise rotation, as well as the torque limit before the unit shuts down fore safety. Being able to control these variables is useful to prevent the unit from potentially damaging itself during use.

The register within the servo is split into two portions, the EEPROM and the RAM. The contents within the RAM are erased when the servo is turned off, or disconnected from power, the EEPROM contents, however, are saved. This means that long term data storage of information such as the servo ID, baud rate, and angle limits are stored on the EEPROM so that these variables do not need to be input every time the servo is turned on. The disadvantage of the EEPROM is that it is slower to access this data than it is to access other storage mediums, such as RAM. The RAM holds temporary data for the servo, such as the goal position, moving speed, and any present values of the servo. The disadvantage of using RAM is that the information stored can only be held when the power is on. The table for the addresses of the servos can be seen in figures 3.31 and 3.32, for the EEPROM and RAM respectively.

Area	Address (Hexadecimal)	Name	Description	Access	Initial Value (Hexadecimal)
EEPROM	0 (0X00)	Model Number(L)	Lowest byte of model number	R	12 (0X0C)
	1 (0X01)	Model Number(H)	Highest byte of model number	R	0 (0X00)
	2 (0X02)	Version of Firmware	Information on the version of firmware	R	-
	3 (0X03)	ID	ID of Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel	RW	1 (0X01)
	5 (0X05)	Return Delay Time	Return Delay Time	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit	RW	3 (0X03)
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature	RW	70 (0X46)
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage	RW	60 (0X3C)
	13 (0X0D)	the Highest Limit Voltage	Highest Limit Voltage	RW	140 (0XBE)
	14 (0X0E)	Max Torque(L)	Lowest byte of Max. Torque	RW	255 (0XFF)
	15 (0X0F)	Max Torque(H)	Highest byte of Max. Torque	RW	3 (0X03)
	16 (0X10)	Status Return Level	Status Return Level	RW	2 (0X02)
	17 (0X11)	Alarm LED	LED for Alarm	RW	36(0x24)
	18 (0X12)	Alarm Shutdown	Shutdown for Alarm	RW	36(0x24)

Figure 3.31: The AX-12A servo EEPROM addresses [2]

3.6.2 Interface

The kit uses the USB2Dynamixel interface to connect to a computer. This USB device connects into the USB port of the computer and provides a port to connect to the servos. Only one servo needs to connect to the device, as they are all connected in series. The external power supply brides the connection between the USB adaptor and the servo at the base of the arm. This power supply provides 12V DC to the servos and can supply up to 5A if necessary.

3.7 Controls and Programming

The robotic arm is controlled through Matlab, and is connected to a computer via a USB interface. The Matlab code displays a GUI for the user to use and has a variety of options for moving the arm. Sliders allow each joint to be moved, as well as input boxes for specific angles. Information such as the current angle can be read. Currently the gripper is simply controlled by a goal angle, but once the sensor is implemented, the information will be taken from an Arduino to the computer and the angle will be adjusted based on that information.

RAM	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	CW Compliance margin	RW	1 (0X01)
	27 (0X1B)	CCW Compliance Margin	CCW Compliance margin	RW	1 (0X01)
	28 (0X1C)	CW Compliance Slope	CW Compliance slope	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	CCW Compliance slope	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Lowest byte of Goal Position	RW	-
	31 (0X1F)	Goal Position(H)	Highest byte of Goal Position	RW	-
	32 (0X20)	Moving Speed(L)	Lowest byte of Moving Speed (Moving Velocity)	RW	-
	33 (0X21)	Moving Speed(H)	Highest byte of Moving Speed (Moving Velocity)	RW	-
	34 (0X22)	Torque Limit(L)	Lowest byte of Torque Limit (Goal Torque)	RW	ADD14
	35 (0X23)	Torque Limit(H)	Highest byte of Torque Limit (Goal Torque)	RW	ADD15
	36 (0X24)	Present Position(L)	Lowest byte of Current Position (Present Velocity)	R	-
	37 (0X25)	Present Position(H)	Highest byte of Current Position (Present Velocity)	R	-
	38 (0X26)	Present Speed(L)	Lowest byte of Current Speed	R	-
	39 (0X27)	Present Speed(H)	Highest byte of Current Speed	R	-
	40 (0X28)	Present Load(L)	Lowest byte of Current Load	R	-
	41 (0X29)	Present Load(H)	Highest byte of Current Load	R	-
	42 (0X2A)	Present Voltage	Current Voltage	R	-
	43 (0X2B)	Present Temperature	Current Temperature	R	-
	44 (0X2C)	Registered	Means if Instruction is registered	R	0 (0X00)
	46 (0X2E)	Moving	Means if there is any movement	R	0 (0X00)
	47 (0X2F)	Lock	Locking EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Lowest byte of Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Highest byte of Punch	RW	0 (0X00)

Figure 3.32: The AX-12A servo RAM addresses [2]

3.7.1 Matlab Control Program

The control program for the robotic arm is made in Matlab R2016a edition. The code is spread across two separate Matlab files, one is for the physical control of the servos on the arm, and the other is used to interact with the GUI. The interface for the program can be seen in Figure 3.33, which has been made in the Matlab tool set GUIDE (GUI development environment) with some additional code added. In order to correctly code for the Dynamixel servos, DLL file (dynamic-link library) must be used, which are found on the manufacturers site. This library allows access to the EEPROM and RAM on the servos, as seen in Section 3.6. The syntax of the commands to the Dynamixel servos are as follows:

```
1 calllib('dynamixel',READ OR WRITE, ID, FUNCTION, VALUE);
```

The Dynamixel DLL library is called, either the read or write word command is used, depending on whether data is being sent or received. The ID of the servo determines which servo is affected. The function is a numerical value which corresponds to addresses in the servos memory. The list of functions can be seen in Figures 3.31 and 3.32. Finally, the value is used when writing to the servo, its action is dependent on the function used. It is usually for setting a value between 0 and 1023, but some functions have a greater range.

GUI

The GUI has several functions which the user can interact with to control the robotic arm. The five joints of the arm are displayed as a slider and a designated angle. The first joint is the base of the arm (servo 1), the second joint is the next two servos up (servos 2 and 3), the third joint is the two servos after that (servos 4 and 5), the fourth joint is the wrist of the arm (servo 6) and the fifth joint is the gripper (servo 7). The user has several different options to control the position of the joints. The slider can be either clicked and dragged to a set position, with the sides of the slider setting the angle limits, or the user can click the arrows on the side of the slider to move the servo by its smallest increment, 0.29°. Additionally a text box is provided to the right of the slider where the user can input a value, in degrees, for the joint to move to, when the 'Set Angle' button is pressed. The numbered text to either side of the sliders indicate the set angle limits each joint has. The text underneath each joint name indicates the current goal angle that has been set, in a numerical value.

Underneath the sliders four different values are displayed to the user: 'Sensor Reading', 'Phase Shift', 'Loop Iteration Count' and 'Gripper Load'. The sensor reading displays the voltage received by the Arduino from the sensor, after the signal has been conditioned by the circuit. The phase shift is similar to the sensor reading, but for the value of the phase shift. The loop iteration displays how many loops the program has performed since entering the main loop of the program. This is useful for two main reasons, firstly to see

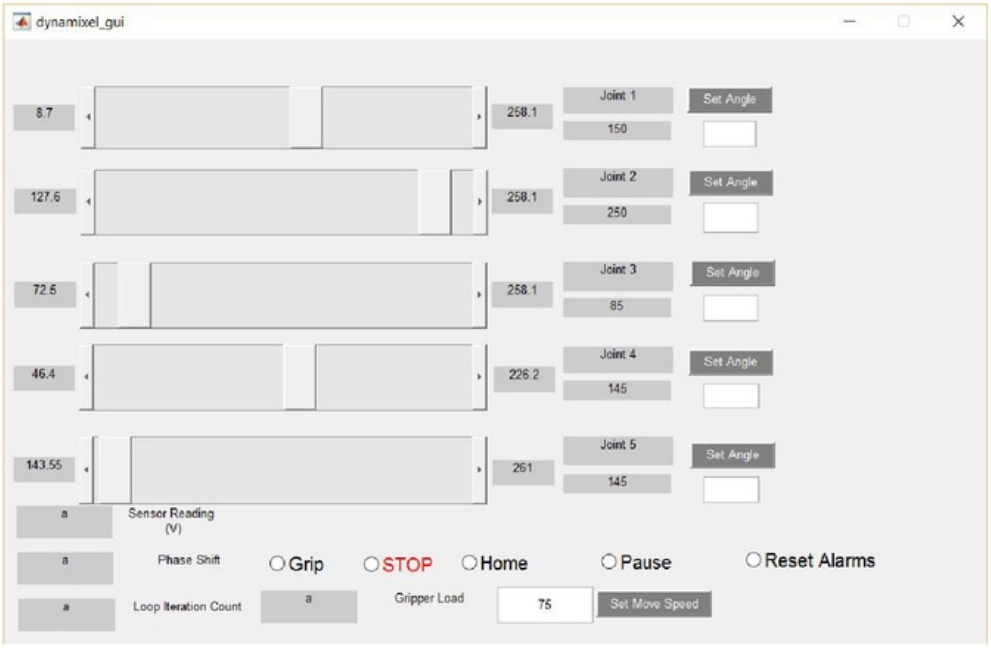


Figure 3.33: The Dynamixel Control GUI

how fast the program is progressing as well as knowing how long is left before the program performs a shutdown from a timeout. The gripper load reads a value stored in the servo that estimates the load on the servo. It should be noted that this is not accurate enough to measure anything useful, it is used as a indication of the direction of the load.

Next to the value outputs are several radial buttons that can perform functions to the arm. These are: 'Grip', 'Stop', 'Home', 'Pause', and 'Reset Alarms'. The grip button changes the grippers mode, so that it is no longer controlled by the slider, but a function that increments the gripper position until the sensor detects that an object has been adequately grasped. Toggling the button again returns the gripper to slider control mode. The stop button starts the shutdown sequence of the program. This sequence correctly closes any COM ports that are open, takes the torque off the servos and closes the GUI. The home button returns the joints of the arm to their default position. The pause button pauses the loop iterations and takes the torque off the servos until the pause button is pressed again. The reset alarms button is used to reset the gripper servo in case of it tripping the overload alarm. This should be used with caution as continued use over its limit can damage the servo. Finally there is a 'Set Move Speed' button and associated input box. Here the user can decide the speed at which all of the joints move at, from a range of 0 to 1023, where the default is 75.

The GUI code uses callback functions to perform actions within the program. These callbacks are only called when the interface element is interacted with, for example when the slider is moved or when a button is pressed. The slider movement code can be seen in Figure 3.34 and the code for toggling the status of a button can be seen in Figure 3.35.

```
% --- Executes on slider movement.
function s1_slider_Callback(hObject, eventdata, handles)
global s1pos;
handles=guidata(hObject);

set(handles.s1_value, 'string', num2str(get(handles.s1_slider, 'value')));
s1pos = (get(handles.s1_slider, 'value'))/0.29;


guidata(hObject, handles);
```

Figure 3.34: Slider Callback Code

```
% --- Executes on button press in stop_button.
function stop_button_Callback(hObject, eventdata, handles)

global program_loop;

if program_loop == 1
    program_loop = 0;
else
    program_loop = 1;
end
```

Figure 3.35: Button Toggle Callback Code

Program Initialisation

The program initialisation is performed as the program starts. This sets the baud rate and COM port of the USB2Dynamixel adaptor, as well as the Arduino. Here global values of the servo limits are stored, as well as some other values such as a program termination iteration limit, initial positions for the servos, grip thresholds, etc. Matrices to store the sensor voltage data and phase shift data are also created here.

Before data can be sent and received by the servos, they must be initialised within the code. Figure 3.36 shows the command to initiate the connection. Once connected

the angle limits for the clockwise and counter-clockwise directions are set from a matrix that stores these values. A loop cycles through each servo and sets the limits. The initial moves speed is set to all of the servos in a similar manner. Finally the gripper servo is set to a higher limit of torque than the other servos, as it is required for gripping heavier items. Increasing the other servos is not advised as it may have negative effects if the gripper gets caught on an object.

```
%Servo Setup ------------------------------------------------------------

%initialises the connection
res = calllib('dynamixel', 'dxl_initialize', DEFAULT_PORTNUM, DEFAULT_BAUDNUM);

%set angle limits
for i = 1:7
    id = i;
    calllib('dynamixel','dxl_write_word',id,P_CCW_ANGLE_LIMIT,Limits(i,1));
    calllib('dynamixel','dxl_write_word',id,P_CW_ANGLE_LIMIT,Limits(i,2));
end

%write move speed
for i = 1:6
    id = i;
    calllib('dynamixel','dxl_write_word',id,P_MOVE_SPEED,move_speed);
end

%move speed for gripper
calllib('dynamixel','dxl_write_word',7,P_MOVE_SPEED,175);

%torque limit for gripper
calllib('dynamixel','dxl_write_word',7,P_TORQUE_LIMIT_L,1023);
```

Figure 3.36: The Initialisation of the Dynamixel Servos

Main Loop

The main loop is where the program flows through until one of two conditions are met, either the user presses the 'Stop' button or the program reaches the loop iteration shut-down count. The main loop communicates and translates information between the GUI (the user) and the servos. The start of the loop can be seen in Figure 3.37. At the start the loop reads the value of the sensor voltage and phase shift voltage and stores them into their respective matrices. The 'handles' of the GUI are updated each iteration, these are the information that can be taken from the GUI, such as slider position, which is needed to set the goal positions.

```
while (program_loop == 1) && (zz < shutdown_timer+1) %main loop, continues until ProgramLoop is false i.e. stop buton

    vout = readVoltage(a,'A0');
    phase = readVoltage(a,'A1');
    voltage_matrix(zz) = vout;
    phase_matrix(zz) = phase;

    handles = guihandles(dynamixel_gui);
```

Figure 3.37: The Start of the Main Loop

Next the program checks whether or not the program has been paused, or the user has changed the speed of the servos. Reading and writing to the servos is a time consuming task, which is why the speed is only addressed if the user has changed the speed, and not every iteration. This can be seen in Figure 3.38.

```
if change_speed == 1;
    %write move speed
    for i = 1:6
        id = i;
        calllib('dynamixel','dxl_write_word',id,P_MOVE_SPEED,move_speed);
    end
    change_speed = 0;
end

while program_pause == 1
    %Disables torque on motors when stop is pushed
    for i = 1:7
        id = i;
        calllib('dynamixel','dxl_write_word',id,P_TORQUE_ENABLE,0);
    end
    pause(0.1);
end
```

Figure 3.38: The Movement Speed and Pause Functions

The goal positions are also only addressed when they change due to the read and write speed of the servos. Previous versions of the program wrote the speed each iteration, and this dramatically slowed the program down. This is an issue as a quick response time is necessary to gather the sensor data and adjust the arms grip in time to affect the gripped object. Now the previous set goal position is stored at the end of the main loop and this is compared to the current goal position. Only if these two differ do the servos get addressed, seen in Figure 3.39. The goal position is changed in the GUI side of the program when action occur that would change the goal position. These include: moving the slider, setting a goal through the text box, or pressing the home position button.

```
%write goal position
if prev_s1pos ~= s1pos
    calllib('dynamixel','dxl_write_word',1,30,s1pos);
end
if prev_s2pos ~= s2pos
    calllib('dynamixel','dxl_write_word',2,30,s2pos);
    calllib('dynamixel','dxl_write_word',3,30,s2pos);
end
if prev_s4pos ~= s4pos
    calllib('dynamixel','dxl_write_word',4,30,s4pos);
    calllib('dynamixel','dxl_write_word',5,30,s4pos);
end
if prev_s6pos ~= s6pos
    calllib('dynamixel','dxl_write_word',6,30,s6pos);
end
```

Figure 3.39: Setting the Goal Positions

When the 'Grip' button is pressed in the GUI, the gripper servo no longer responds to the slider, instead the gripper is controlled by the sensor voltage. This can be seen in Figure 3.40. A threshold variable is stored in the program, which is the corresponding minimum voltage for the gripper to have a reasonable grip on an object. If this is not reached, the goal position is incremented by 5, out of 1023, until this threshold is met. This mode remains active until the 'Grip' button is pressed again on the GUI and the servo will respond to the slider again.

```
%gripper control - gripper mode closes gripper until set voltage
if grip_button_var == 1;
    if (voltoutput < grip_threshold) && (voltoutput ~= 0)
        s7pos = s7pos+5;
    end
    calllib('dynamixel','dxl_write_word',7,P_GOAL_POSITION,s7pos);
else
    s7pos = (get(handles.s7_slider, 'value'))/0.29;
    calllib('dynamixel','dxl_write_word',7,30,s7pos);
end
```

Figure 3.40: The Gripper Control

Although every reading from the sensor voltage and phase shift is stored, they are not all displayed to the user in the GUI as the values change so fast, it is not very useful to the user. Instead, an average of a set number of samples is displayed to the GUI, which can be modified by the 'voltage_samples' variable. The more samples, the more accurate the measurement and easier it is to read, but also more time is taken to determine the value. A balance needs to be found so that the feedback to the gripper is not too slow.

```
if rem(zz, voltage_samples) == 0 %stores voltmax every x iterations and resets value
    average_matrix(1,voltage_samples) = vout;
    phase_matrix(1,voltage_samples) = phase;
    %mean(average_matrix,2);
    voltoutput = mean(average_matrix,2);  %average_matrix(1,1);
    phaseoutput = mean(phase_matrix,2);
    set(handles.s7_voltage, 'string', voltoutput); %sensor voltage
    set(handles.phase_shift_text, 'string', phaseoutput); %sensor voltage
      gripper_load = calllib('dynamixel','dxl_read_word',7,P_PRESENT_LOAD);
      set(handles.gripper_load_value, 'string', gripper_load - 1024);

else
    average_matrix(1,rem(zz,voltage_samples)) = vout;
    phase_matrix(1,rem(zz,voltage_samples)) = phase;
end
```

Figure 3.41: Sensor Information Display

At the end of the main loop the current goal position of each servo is stored so it can be used as the previous goal position for the next iteration. The iteration value in the GUI is updated and the variable storing the iteration count is incremented. All of this can be seen in Figure 3.42.

```
%store current goal positions for next loop
prev_s1pos = s1pos;
prev_s2pos = s2pos;
prev_s4pos = s4pos;
prev_s6pos = s6pos;

set(handles.iterations, 'string', zz); %iteration count

zz = zz+1; %increment iteration count
```

Figure 3.42: The End of the Main Loop

Program Termination

When either of the program termination conditions are met, the program begins its shutdown. The program ensures that all of the servos are still connected. Next the torque in all of the servos is disabled. Finally, the connection to the Arduino and the Dynamixel servos are both disconnected and the GUI is closed. Figure 3.43 shows this.

```
%Program Shutdown ------------------------------------------------------------

res = calllib('dynamixel', 'dxl_initialize', DEFAULT_PORTNUM, DEFAULT_BAUDNUM);

%Disables torque on motors when stop is pushed
    for i = 1:7
        id = i;
        calllib('dynamixel','dxl_write_word',id,P_TORQUE_ENABLE,0);
    end

%Terminates connection and closes gui
clear a;
calllib('dynamixel','dxl_terminate');
unloadlibrary('dynamixel')
close all force
%clearvars
```

Figure 3.43: Program Termination

Chapter 4

Experiments

4.1 Sensor Characterisation

The sensor characterisation tests are used to determine the operating conditions of the sensors, so that this can be replicated by the sensor circuit.

4.1.1 Method

Aluminium Sensor Characterisation

To find the optimal operating frequency of the aluminium sensor, a set input voltage was applied to the sensor and the frequency was varied using the HP Agilent 33120A function/arbitrary waveform generator. Measurements were taken using an Agilent DSO-X-2024A digital oscilloscope, seen in Table 4.1.

PDMS Sensor Characterisation

This test looked at the relationship between the input voltage and the output voltage of a sinusoidal wave using the PDMS sensor. The test was also carried across several frequencies. The waveform generator and the oscilloscope were used for this test.

4.1.2 Results

Aluminium Sensor Characterisation

Frequency	V_{In}	V_{Out}
1 Hz	2.13	0.64
10 Hz	2.13	1.33
100 Hz	2.13	1.97
1 kHz	2.13	2.13
10 kHz	2.13	2.13
50 kHz	2.13	2.09
100 kHz	2.13	2.01
500 kHz	2.13	1.85
1 MHz	2.13	1.77

Table 4.1: Aluminium Sensor Frequency Response.

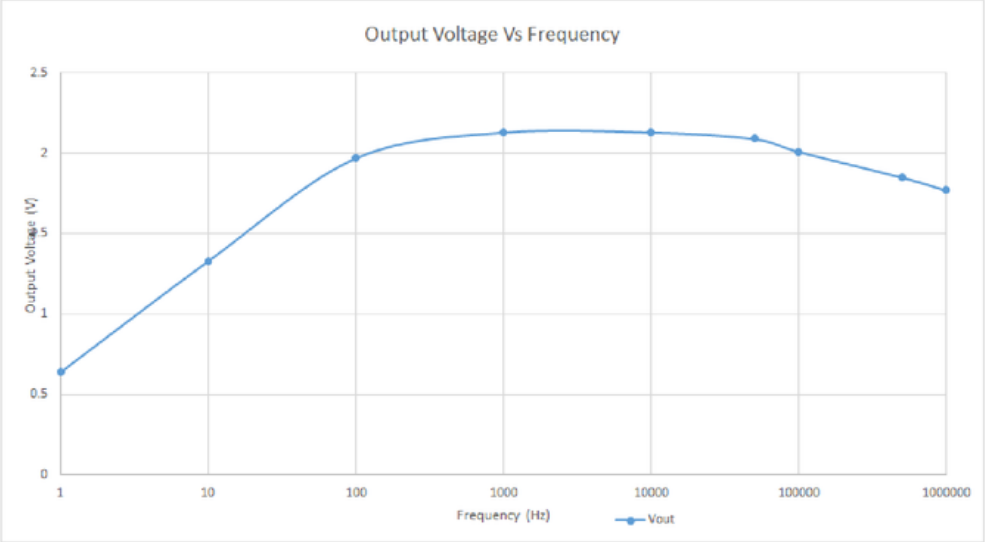


Figure 4.1: Aluminium Sensor Frequency Response Graph

PDMS Sensor Characterisation

4.1.3 Results

Frequency	V_{In} (mV)	V_{Out} (mV)		Frequency	V_{In} (mV)	V_{Out} (mV)
100 Hz	230	40		50 kHz	230	25
	430	56			430	40
	630	72			630	55
	840	92			840	70
	1040	110			1040	90
	1240	125			1240	105
	1480	140			1480	120
	1700	155			1700	140
	1880	177			1880	150
	2090	200			2090	190
500 Hz	230	33		100 kHz	230	25
	430	55			430	40
	630	75			630	55
	840	95			840	65
	1040	105			1040	80
	1240	125			1240	95
	1480	145			1480	110
	1700	160			1700	125
	1880	185			1880	140
	2090	205			2090	150
1 kHz	230	30		500 kHz	230	20
	430	50			430	32
	630	70			630	45
	840	90			840	60
	1040	110			1040	70
	1240	120			1240	85
	1480	145			1480	98
	1700	1170			1700	110
	1880	180			1880	125
	2090	200			2090	135
10 kHz	230	35		1 MHz	230	20
	430	50			430	32
	630	70			630	45
	840	90			840	60
	1040	110			1040	72
	1240	125			1240	85
	1480	140			1480	100
	1700	155			1700	111
	1880	170			1880	125
	2090	190			2090	139

Table 4.2: PDMS Sensor Characterisation Measurements

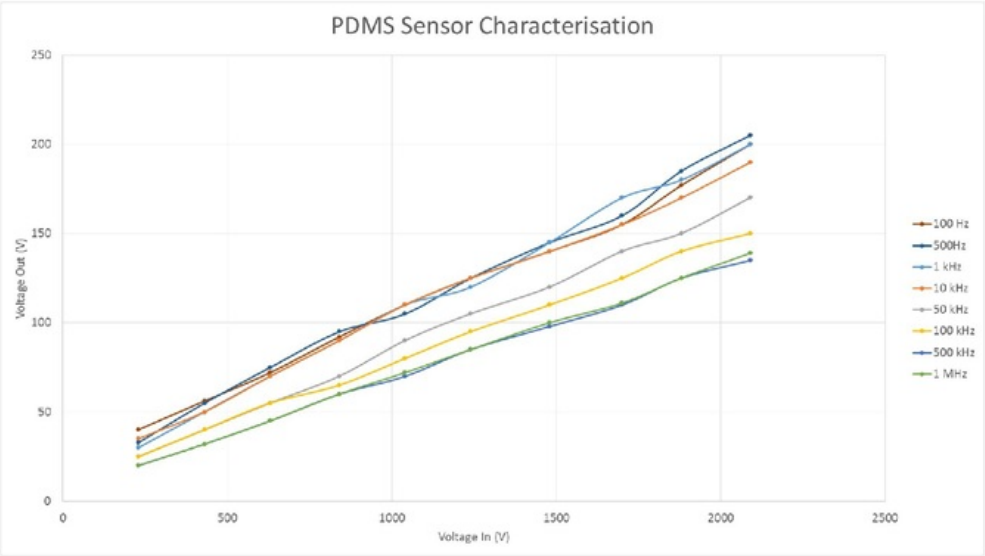


Figure 4.2: PDMS Sensor Characterisation Graph

4.1.4 Discussion

Aluminium Sensor

Looking at the results in Figure 4.1, the optimal frequency of the sensor is from 100 Hz to 100 kHz, as the peak output voltage spans this range.

PDMS Sensor

All of the tested frequencies show a linear relationship between the output voltage as the input voltage increases. The output voltage of the lower frequencies appears to be greater than the higher frequencies for the same input frequencies.

4.2 Sensor Force Test

The sensor force test is used to investigate the relationship between the force and the output voltage for the sensors.

4.2.1 Method

The sensors were tested to measure response under a known force. The test involved using a finger to press on the sensor until a weight was reached. Initially a mass carrier with disk weights was used to take these force measurements, as it would have provided a steadier reading. This did not work as the interdigitated sensors could not respond correctly to the weights and the piezoresistive sensors struggled to recognise the force due to the large surface area of the mass carrier base. To help improve the readings from the scale, three measurements were taken and the average result was used. The voltage reading is measured by the Arduino and displayed in the program, which is how the program reads the sensor voltage. Since the program is iterating quickly without the arm attached, the average of 20 samples is taken and displayed to the GUI, which helps improve accuracy.

The Flexiforce sensor was tested twice, once using a straight 5 V DC signal and again using the same AC signal as the other sensors. This was done because the manufacturer recommends using a DC signal, so any loss of functionality should be looked into. The reason AC would be better as the extra information of the phase shift can be obtained, which the DC signal cannot provide.

4.2.2 Results

Flexiforce A401 5V DC

Weight (Kg)	V_1	V_2	V_3	Average (V)
0	0.029	0.014	0.024	0.02
0.05	0.738	0.650	0.777	0.72
0.1	1.41	1.39	1.23	1.34
0.2	1.45	1.50	1.57	1.51
0.4	1.58	1.55	1.62	1.58
0.6	1.90	1.88	2.00	1.93
0.8	2.08	2.06	2.07	2.07
1	2.19	2.16	2.18	2.18
1.2	2.25	2.27	2.26	2.26
1.4	2.37	2.39	2.38	2.38
1.6	2.65	2.55	2.52	2.54

Table 4.3: Flexiforce A401 DC Voltage Force Test

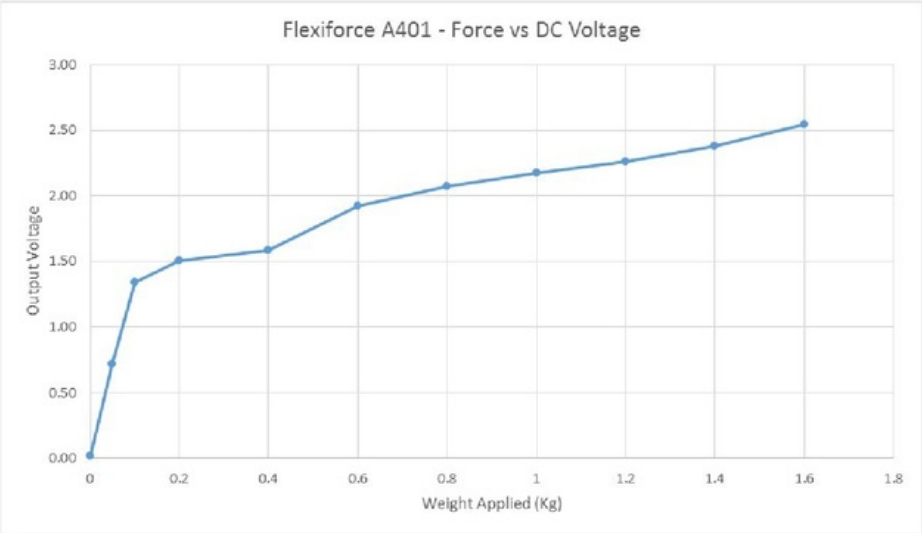


Figure 4.3: Flexiforce A401 DC Voltage Force Test

Flexiforce A401 5V AC

Weight (Kg)	V_1	V_2	V_3	Average (V)
0	0.733	0.696	0.727	0.720
0.05	1.70	1.74	1.69	1.71
0.1	1.71	1.89	1.83	1.81
0.2	1.92	1.83	1.93	1.89
0.4	2.12	2.08	2.16	2.12
0.6	2.19	2.24	2.18	2.20
0.8	2.28	2.30	2.32	2.30
1	2.47	2.37	2.46	2.43
1.2	2.54	2.45	2.54	2.51
1.4	2.65	2.64	2.62	2.64
1.6	2.68	2.68	2.69	2.68

Table 4.4: Flexiforce A401 AC Voltage Force Test

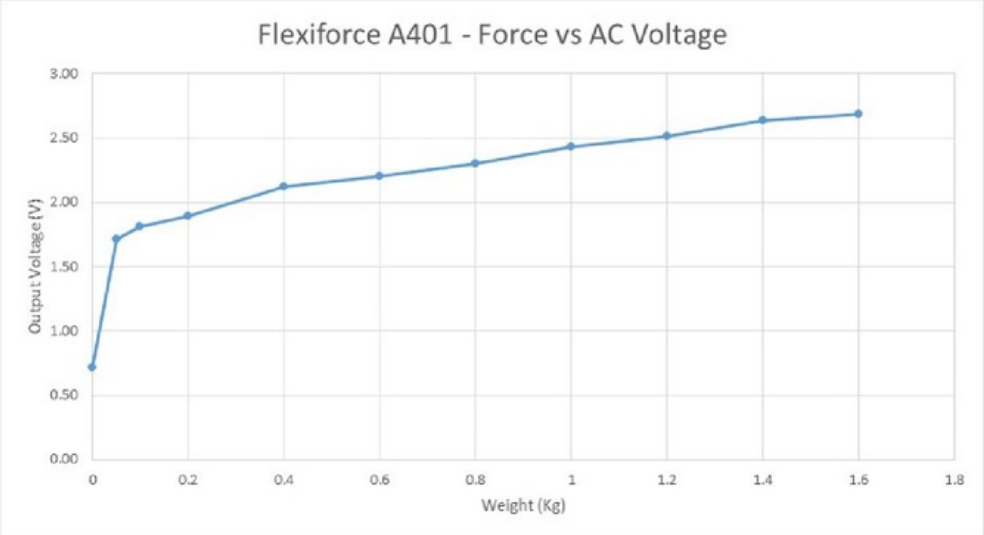


Figure 4.4: Flexiforce A401 AC Voltage Force Test

Piezoresistive Sensor 5V AC

Weight (Kg)	V_1	V_2	V_3	Average (V)
0	1.02	1.01	1.03	1.02
0.05	2.47	2.50	2.48	2.48
0.1	2.50	2.51	2.48	2.48
0.2	2.48	2.51	2.52	2.50
0.4	2.53	2.52	2.50	2.52
0.6	2.50	2.53	2.49	2.51
0.8	2.49	2.50	2.52	2.50
1	2.50	2.51	2.54	2.52
1.2	2.50	2.53	2.49	2.51
1.4	2.49	2.49	2.52	2.50

Table 4.5: Piezoresistive Sensor AC Voltage Force Test

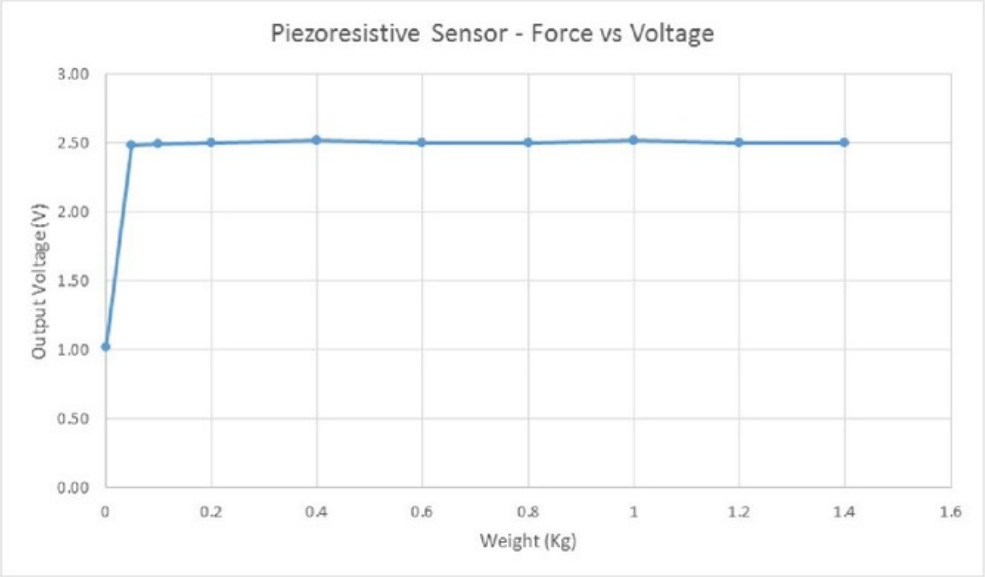


Figure 4.5: Piezoresistive Sensor AC Voltage Force Test

PDMS Sensor 5V AC

Weight (Kg)	V_1	V_2	V_3	Average (V)
0	0.123	0.129	0.127	0.130
0.05	0.358	0.466	0.461	0.430
0.1	0.543	0.566	0.619	0.580
0.2	0.675	0.709	0.687	0.690
0.4	0.865	0.846	0.849	0.850
0.6	0.879	0.996	0.917	0.930
0.8	0.950	0.916	0.920	0.930
1	0.967	0.979	0.962	0.970
1.2	1.02	1.00	1.03	1.02
1.4	1.00	1.02	0.98	1.00
1.6	1.04	1.30	1.01	1.12

Table 4.6: PDMS Sensor AC Voltage Force Test

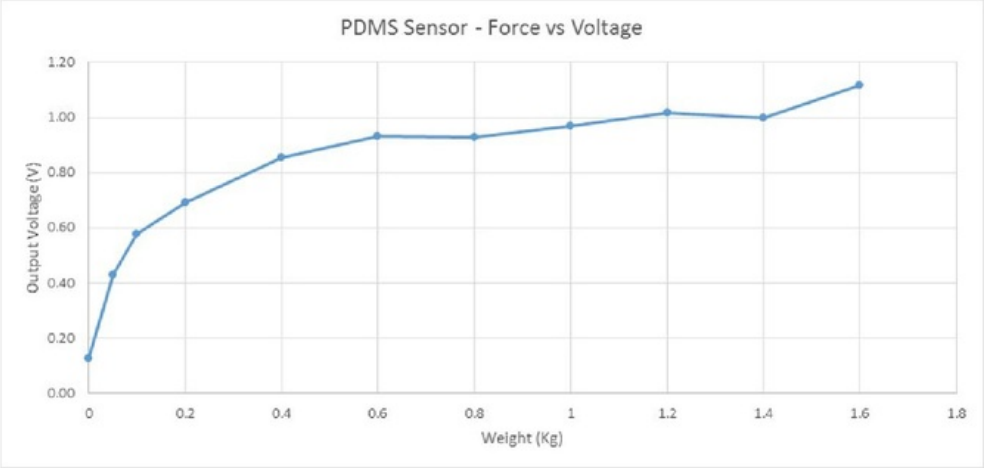


Figure 4.6: PDMS Sensor AC Voltage Force Test

Graphene Sensor 5V AC

Weight (Kg)	V_1	V_2	V_3	Average (V)
0	0.0502	0.0460	0.0488	0.050
0.05	1.50	1.56	1.49	1.52
0.1	1.98	2.01	2.03	2.01
0.2	2.25	2.20	2.22	2.22
0.4	2.40	2.33	2.32	2.35
0.6	2.36	2.30	2.39	2.35
0.8	2.35	2.33	2.29	2.32
1	2.40	2.39	2.35	2.38
1.2	2.29	2.37	2.30	2.32
1.4	2.38	2.35	2.38	2.37

Table 4.7: Graphene Sensor AC Voltage Force Test

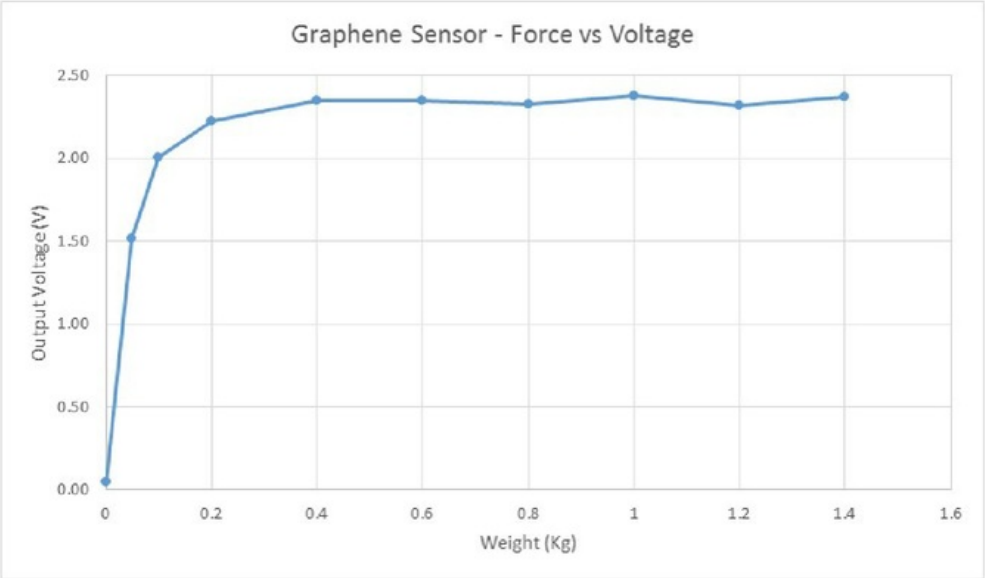


Figure 4.7: Graphene Sensor AC Voltage Force Test

Aluminium Sensor 5V AC

Weight (Kg)	V_1	V_2	V_3	Average (V)
0	0.0395	0.0415	0.0405	0.0400
0.05	2.25	2.34	2.37	2.32
0.1	2.33	2.34	2.37	2.35
0.2	2.33	2.34	2.37	2.35
0.4	2.38	2.36	2.41	2.38
0.6	2.24	2.26	2.27	2.26
0.8	2.28	2.31	2.32	2.30
1	2.17	2.22	2.20	2.20
1.2	2.42	2.40	2.41	2.41
1.4	2.43	2.41	2.42	2.42

Table 4.8: Aluminium Sensor AC Voltage Force Test

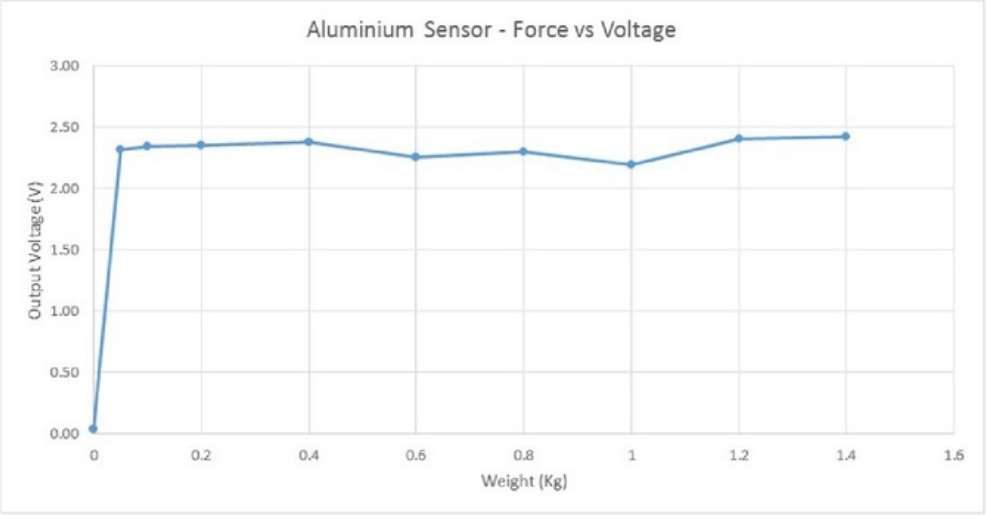


Figure 4.8: Aluminium Sensor AC Voltage Force Test

4.2.3 Discussion

Flexiforce A401 5V DC

The force versus voltage curve for the DC voltage, seen in Figure 4.3, appears logarithmic in form. The voltage change at lower loads increases significantly faster than at higher loads. The curve becomes closer to a linear relationship between 0.4 kg and the maximum recorded value of 1.6 kg. The DC voltage curve appears to be able to distinguish smaller forces with a better resolution than the same sensor excited with a sinusoidal signal, seen in Figure 4.4.

Flexiforce A401 5V AC

The force versus voltage curve of the Flexiforce sensor with the AC excitation signal appears to be less sensitive to smaller forces than the same sensor using a DC input. The AC input however, appears to act in a linear manner at a lower force than its DC counterpart. If the sensor is not used for small forces, this may be more beneficial to the user. Using the sensor for grasping, measuring small forces accurately would be beneficial. The AC input does provide another piece of information to be gathered from the sensor that the DC input cannot provide, the phase shift of the signal. This is the reason the AC signal will be used for most of the tests.

4.2.4 Piezoresistive Sensor 5V AC

The piezoresistive sensor does not appear to have any states between on or off. If there is a relationship between force and voltage, is does not present itself with this circuit. This sensor can still be used for tactile sensing as it is able to determine contact. The phase shift may also be a determining factor to this sensors usefulness.

PDMS Sensor 5V AC

This sensor has the best low force sensitivity of all of the sensors tested, and reaches a relatively linear state around 0.4 kg. Due to the material of the electrodes in the sensor and the resistance associated, this sensor has the lowest output voltage of all of the sensors. This means that any noise in the signal would have the greatest impact on the results of this sensor, rather than the others.

4.2.5 Graphene Sensor

The graphene sensor has a sharp relationship between force and voltage, increasing towards its peak value at 0.4 kg. This sensor seems to be more useful at much lower forces.

This result may be due to the high conductivity of graphene.

Aluminium Sensor 5V AC

The force versus voltage graph for this sensor is the least useful for gripping applications as the voltage reaches maximum as soon as the sensor is touched, at 0.05 kg. The result is relatively steady onwards. This seems to be due to the high conductivity of the aluminium electrodes used within the sensor.

Errors in Test

There are several aspects of this test that may have introduced errors into the results. Firstly, pressing and holding the sensor at the correct weight is difficult to stay steady. Therefore the measurements may have not been taken at exactly the correct weight. Additionally, the output voltage also appears to be related to the surface area of contact, across all sensors. For the larger weights the finger pressed down would have been pressed into a larger surface area, which may have had an effect on the results. Finally errors may have been introduced in either the calibration of the scales, or with the measurement of the voltage within the Arduino.

To help minimise the impact that these errors may have had on the test, many results were taken to obtain an average. For each recorded value, the Matlab program had taken the average of 20 voltage samples. Three of these recorded values were taken and an average was used. This helps reduce the errors introduced when trying to hold the sensor steady at the set weight.

4.3 Program Iteration Speed Test

The following tests, Sections 4.4, 4.5 and 4.6, are all logged through the Matlab program and are therefore bound by the speed of the program. This test is to measure the speed of the iterations so that a time base for the following tests can be established.

4.3.1 Method

The method to measure the time per iteration is to time several iterations of the program and take an average of five tests. The robotic arm is not attached for the following tests, as the communication to the servos is a time consuming task and will slow down the program. The program was timed for 800 iterations with a stopwatch and the average was taken from five runs.

4.3.2 Results

Sample No.	Time (s)
1	37.07
2	36.86
3	36.69
4	37.00
5	37.08
Average	36.94

Table 4.9: Program Iteration Speed Test

4.3.3 Discussion

As seen from the results in Table 4.9, the average speed of 800 iterations is 36.94 s. This means that it takes 46.175 ms per iteration, or 21.67 iterations per second. This will give a time basis for the following results.

4.4 Sensor Load Response Testing

The sensor load response is used to determine each sensors response to applied loads for varied times.

4.4.1 Method

To test the response of the sensors, a load was applied for a varied amount of time to the sensors. The load was a finger pushed onto the sensors, with an even force across all sensors. 1000 iterations of the program was run and the sensor voltage was recorded within Matlab. The times of when the load was applied can be seen below, in terms of the program iterations.

- 0 - 100: No load
- 100 - 150: Load applied
- 150 - 300: No load
- 300 - 400: Load applied
- 400 - 500: No load
- 500 - 700: Load applied
- 700 - 1000: No load

4.4.2 Results

Flexiforce A401

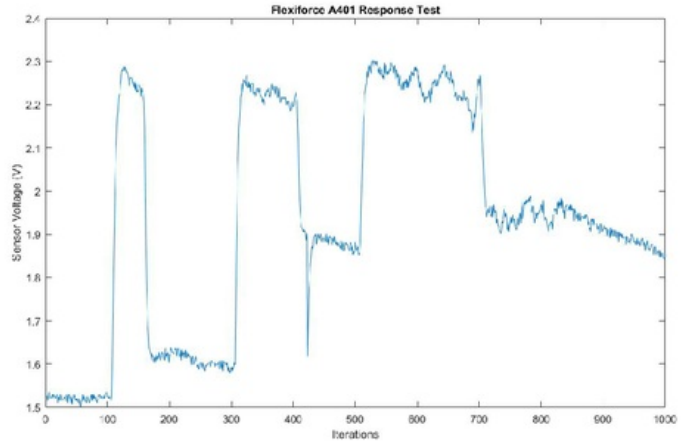


Figure 4.9: Flexiforce A401 Response Test

Piezoresistive Sensor

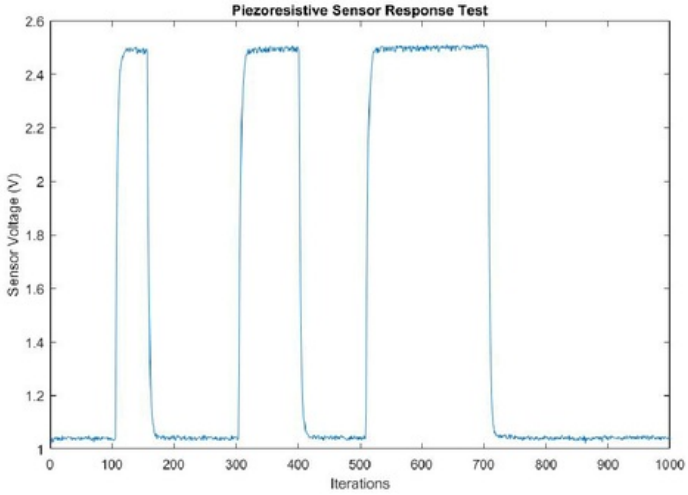


Figure 4.10: Piezoresistive Sensor Response Test

PDMS Sensor

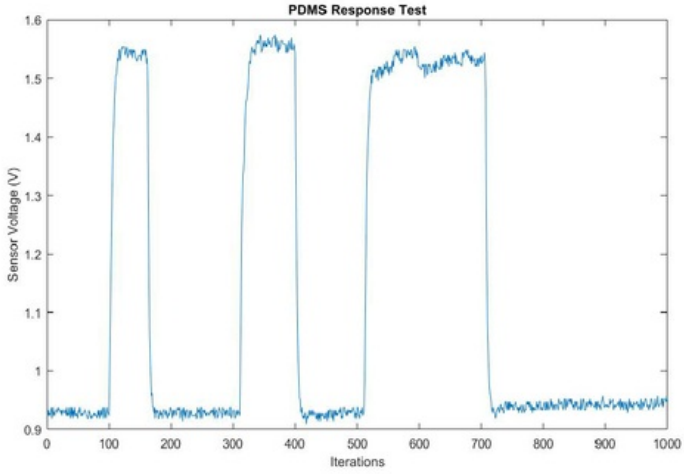


Figure 4.11: PDMS Sensor Response Test

Graphene Sensor

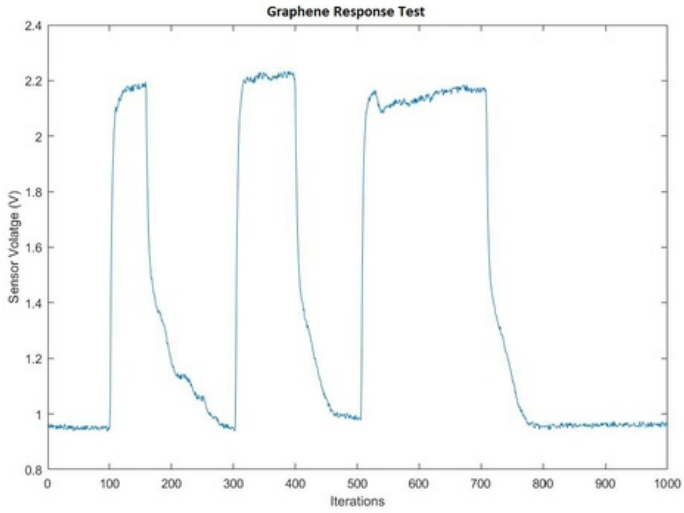


Figure 4.12: Graphene Sensor Response Test

Aluminium Sensor

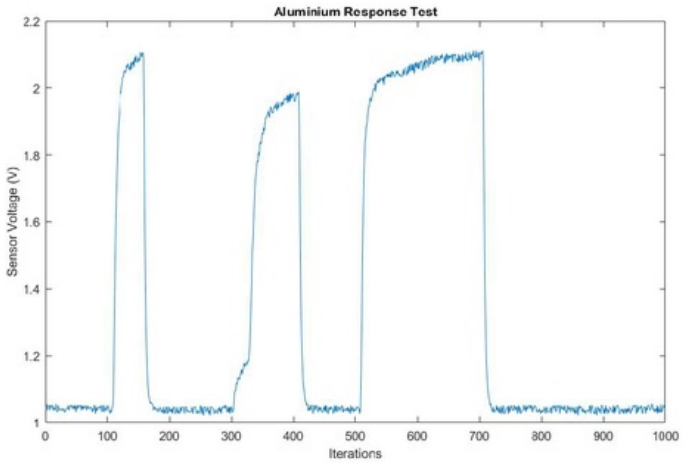


Figure 4.13: Aluminium Sensor Response Test

4.4.3 Discussion

Flexiforce A401 Sensor

Once the load is applied to the sensor at 100 iterations, you can see that when it is removed, the sensor does not return to its initial voltage immediately. The magnitude of this voltage offset seem to increase the longer the load is applied for. This can be seen with the 100 iteration load application and the 200 iteration. This is not an ideal response for a sensor that is working in a feedback system, as it does not promote quick changes to the signal. The reason the sensor responds this way may be due to the construction of the sensor. The piezoresistive layer between the electrodes is not recovering from its strained state quick enough, and thus the sensor appears to be partially activated. This effect seems to carry on to the phase shift signal measured, when can be seen in the following tests, Sections 4.5 and 4.6. If an object was gripped for a long period of time and then suddenly changed, the sensor may not be able to respond quick enough to detect this change. The fluctuations at the peaks of where the load is applied may be due to the distribution of the load not being steady on the sensor.

Piezoresistive Sensor

The piezoresistive sensor shows an ideal voltage signal, in terms of sensor response. The signal has a quick rise time and fall time, and unlike the Flexiforce A401, the voltage returns straight to its initial state. The length of the applied load seems to have no impact

on the sensor. The signal also seems noticeably less noisy in this sensor in comparison to the Flexiforce sensor. The peak voltage of the response also reaches a higher value, 2.5 V, compared to the Flexiforce sensor, 2.3 V.

PDMS Sensor

The PDMS sensor shows similar results to the piezoresistive sensor, albeit with a significantly lower voltage peak, 1.55 V versus 2.5 V, and with greater noise within the signal. The rise and fall time of the loads in comparable to the piezoresistive sensor. The voltage also returns to its initial value after the load is removed, unlike the Flexiforce sensor.

Graphene Sensor

The graphene sensor shows a less ideal response than the PDMS sensor or the piezoresistive sensor. In this signal the rise time has been affected slightly. Instead of a sharp change between unloaded and loaded, the peak has a slight curve towards the top. This increases the rise time before the peak value is reached. A similar effect can be seem when the load is removed. The fall time is curved at the base of the signal, instead of immediately returning to its initial value. This result is still better than the response from the Flexiforce sensor, however. The signal voltage is comparable to the piezoresistive sensor, which is desirable.

Aluminium Sensor

The rise time of the aluminium sensor is comparable to the graphene sensor, but the curve at the peak of the signal is more apparent. The fall time of the signal is significantly better than the graphene sensor, returning to the initial value immediately. This is comparable to the piezoresistive sensor and the PDMS sensor. The sensor voltage also peaks at a reasonable level, 2.1 V which isn't too far below the piezoresistive sensor, 2.5 V.

Errors in Test

Errors in the results may have come from a few sources. First, although the force was around the same for all of the tests, it is hard to ensure that the exact force is used across all sensors. Similarly, the contact area of the finger on the sensor is going to vary, based on the shapes of the sensors, as some are round while others are rectangular. Human response time would have affected how close to the right iterations the load was applied and removed, although these small errors would have had minimal impact on the results. Finally, errors with the Arduino measurement may have impacted the recorded results.

4.5 Sensor Phase Shift Test

The sensor phase shift is a test to determine the phase shift response from the sensor when a load is applied. This test compares all five sensors abilities phase shift response together, to determine which sensors show any sign of phase shift. The phase shift may be a factor in determining the type of load applied to the sensor, which will be addressed in the next test, Section 4.6.

4.5.1 Method

The method is similar to the one employed in Section 4.4, but is purely looking at the response in the phase shift. The test looks at the impact of a finger placed on the sensor, and the circular end face of plastic cylinder with a diameter of 15mm. Matlab is used to take in the measurement data. The sensor has no load initially, at 100 iterations a finger is placed on the sensor, it is held until the program reaches 200 iterations and then the load is removed. At 300 iterations the cylinder is applied to the sensor until the program reaches 400 iterations and the load is removed. The force of the finger and the cylinder will remain the same. This method will also allow the response after the load has been removed to be examined.

4.5.2 Results

Flexiforce A401

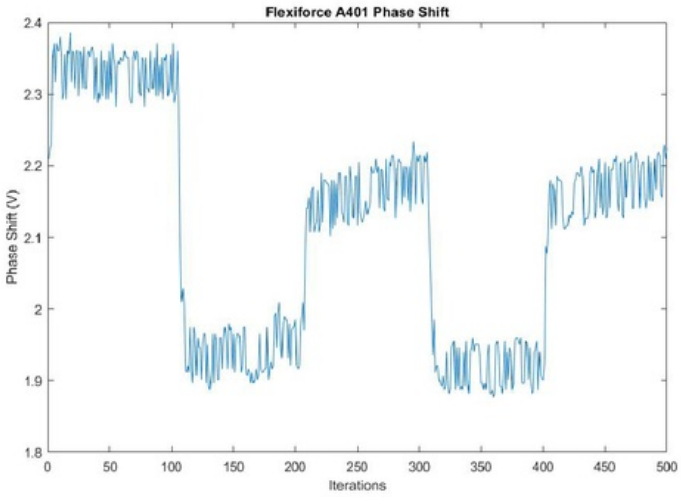


Figure 4.14: Flexiforce A401 Phase Test

Piezoresistive Sensor

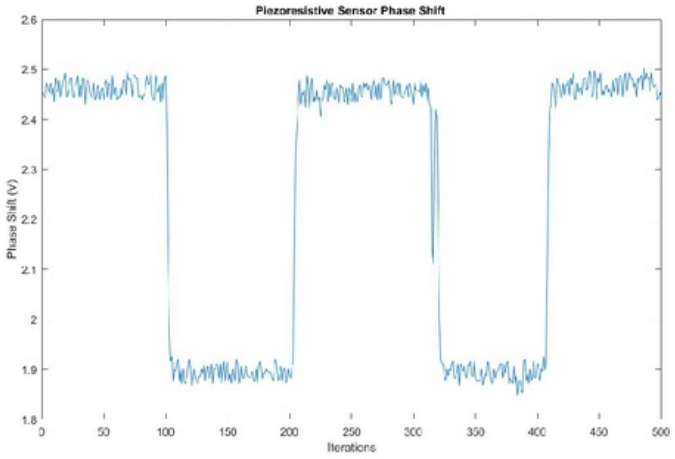


Figure 4.15: Piezoresistive Sensor Phase Test

PDMS Sensor

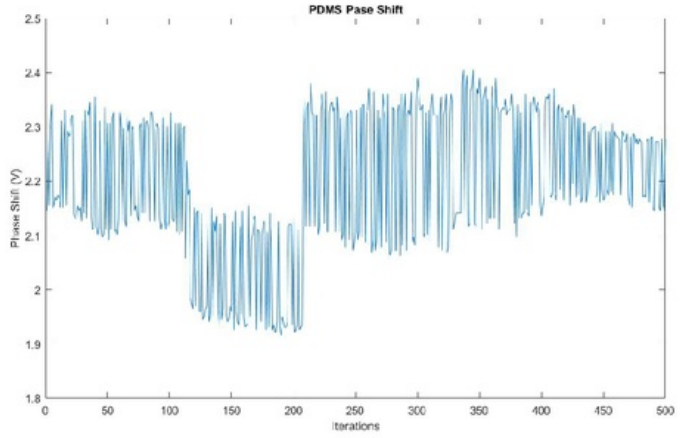


Figure 4.16: PDMS Sensor Phase Test

Graphene Sensor

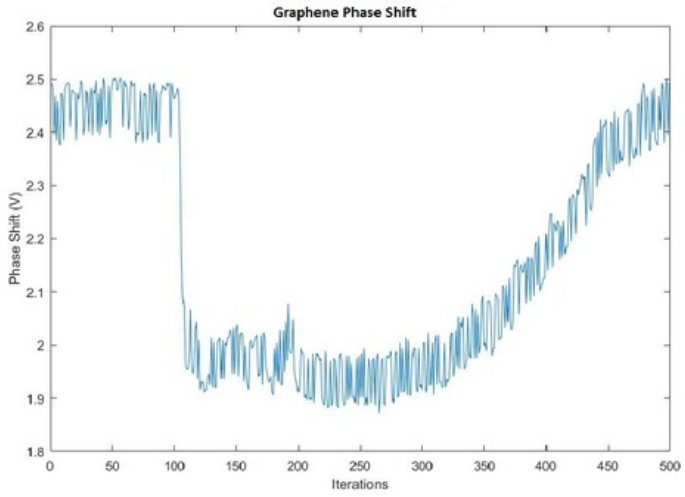


Figure 4.17: Graphene Sensor Phase Test

Aluminium Sensor

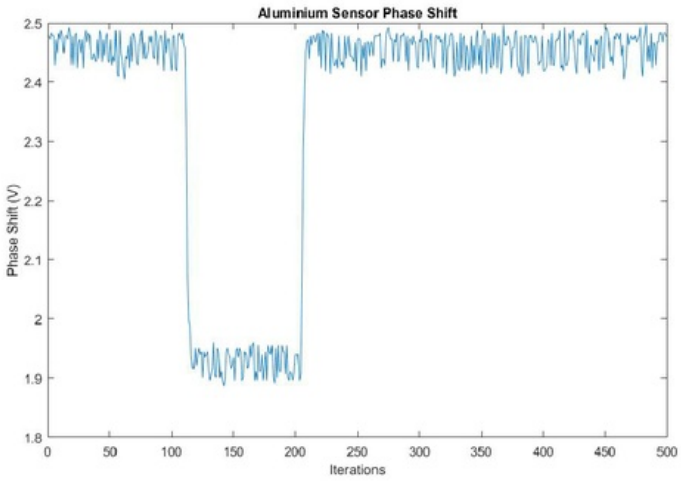


Figure 4.18: Aluminium Sensor Phase Test

4.5.3 Discussion

Flexiforce A401 Sensor

The phase shift graph for the Flexiforce sensor shows that it is receptive to both the finger load and the plastic cylinder. The phase shift response has a short rise and fall time, which is ideal. The issue of the sensor not returning to its initial state after the load has been removed, seen in Section 4.5, is still present for the phase shift response. Once the load is removed the phase shift increases to a certain level, then slowly begins to drift towards its initial state. This is not ideal for a sensor used in a feedback system, which requires short response times. The phase shift magnitude reached by both loads rests at similar levels. The signal shows significant noise at steady state, which may be an issue for obtaining an exact value. This may be reduced by taking the average of a few measurements, but this may compromise the response time. The change between the loaded and unloaded states of the sensor is significant enough to determine between the two states.

Piezoresistive Sensor

The piezoresistive sensor shows more desirable phase shift results than the Flexiforce sensor. The sensor returns to its initial state straight after the load is removed and the noise level is significantly less than the Flexiforce sensor. The sensor is receptive to both types

of loads, and both loads produce a similar phase shift response. The only issue with the piezoresistive sensor seems to be that it does not have any levels between loaded and unloaded.

PDMS Sensor

The phase shift response of the PDMS sensor provides not clear details. The signal noise is extreme, much higher than all of the other sensors, to the point where the signal overlaps between its loaded and unloaded state. A phase shift can be seen where the finger load was applied, but there is no real change when the plastic cylinder is applied.

Graphene Sensor

The graphene sensor responded well to the finger load applied, but when the load was removed, the phase shift did not return immediately to its initial state. Similar to the PDMS sensor, the graphene sensor does not appear to respond to the plastic cylinder, but it is difficult to tell as the sensor had not recovered from the first load when the second load was applied. The noise levels and the phase shift magnitude seem to be in similar levels to the Flexiforce sensor.

Aluminium Sensor

The aluminium sensor provides the best results from the three interdigitated sensors. There is a clear trough in the signal where the finger load was applied, then the signal quickly recovers to its initial state. The cylinder load does not appear to impact the signal at all. The noise levels and the phase shift magnitude appear to fall somewhere between the Flexiforce sensor and the piezoresistive sensor.

Errors in Test

The errors in this test can be attributed to the same errors seen in Section 4.4.

4.6 Object Shape Test

This test looks at the impact that the shape of the gripped object has on the sensor voltage or the phase shift of the signal. Different results for different shapes would allow the arm to identify between a few shapes, and therefore adjust the gripping method to allow for the best gripping results. This test was only performed on the Flexiforce A401 force sensing resistor and the piezoresistive sensor as the other sensors were not sensitive to non-conductive materials.

4.6.1 Method

The test involved six different shapes being applied to the two sensors, these were:

- Finger
- Circle
- Cylinder edge
- Sphere
- Thin edge
- Sharp point

The finger is used as a reference point, as most of the testing has been done with this shape. The circle is the circular face of a cylinder with a diameter of 15mm, and is pressed onto the sensor. The cylinder edge is the rounded length of the same cylinder pressed onto the sensor. The sphere is used to see the response of the contact point of a spherical object. The thin edge is a plastic ruler pressed onto the sensor. Finally, the sharp point is the tip of a pen pressed onto the sensor. All of the measurements were taken using a similar amount of force.

The measurements were taken from the Matlab program, using 300 iterations. The different shaped loads were applied at 100 iterations and removed at 200 iterations. This gives the initial state, the loaded state and the response to the loads. The sensor voltage and the phase shift voltage were plotted using Matlab.

4.6.2 Results

Flexiforce A401 Finger Test

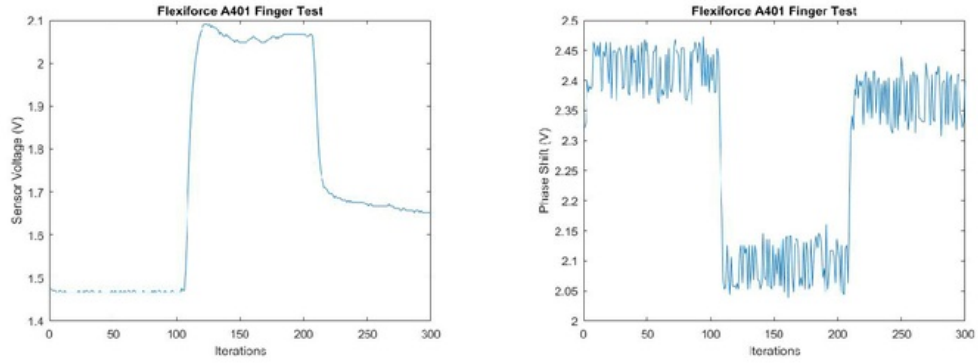


Figure 4.19: Flexiforce Finger Test - Sensor Voltage & Phase Shift

Flexiforce A401 Circle Test

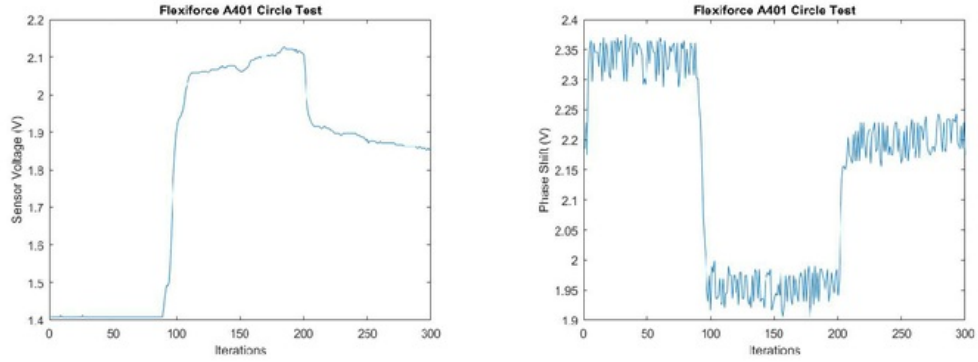


Figure 4.20: Flexiforce Circle Test - Sensor Voltage & Phase Shift

Flexiforce A401 Cylinder Edge Test

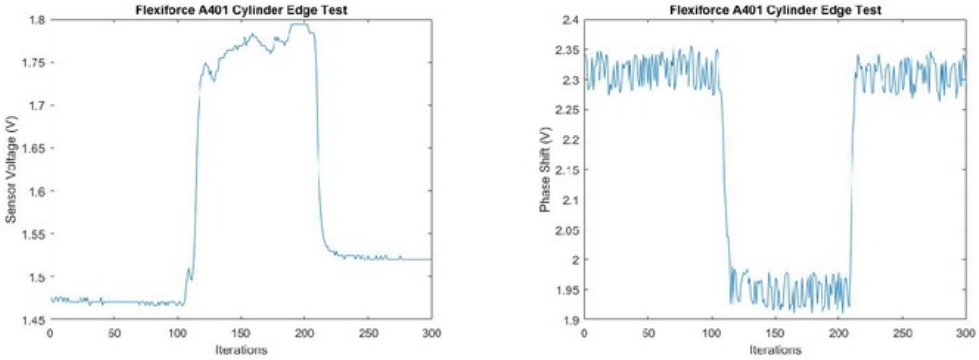


Figure 4.21: Flexiforce Cylinder Edge Test - Sensor Voltage & Phase Shift

Flexiforce A401 Sphere Test

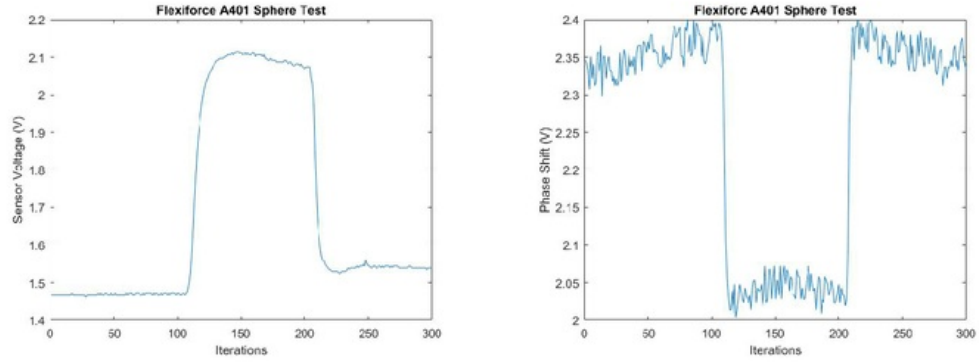


Figure 4.22: Flexiforce Sphere Test - Sensor Voltage & Phase Shift

Flexiforce A401 Thin Edge Test

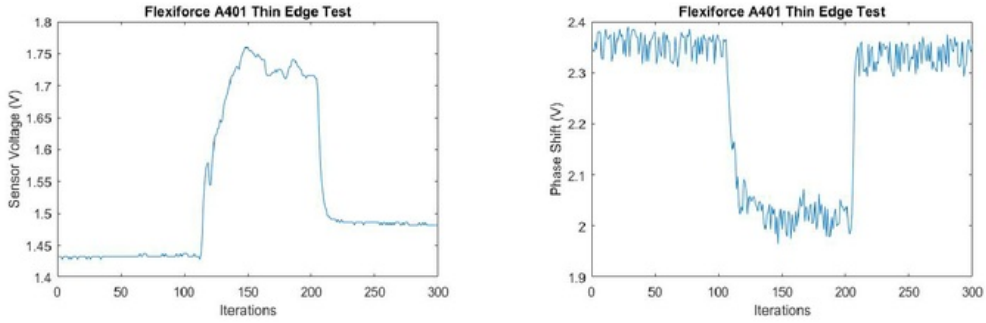


Figure 4.23: Flexiforce Thin Edge Test - Sensor Voltage & Phase Shift

Flexiforce A401 Sharp Point Test

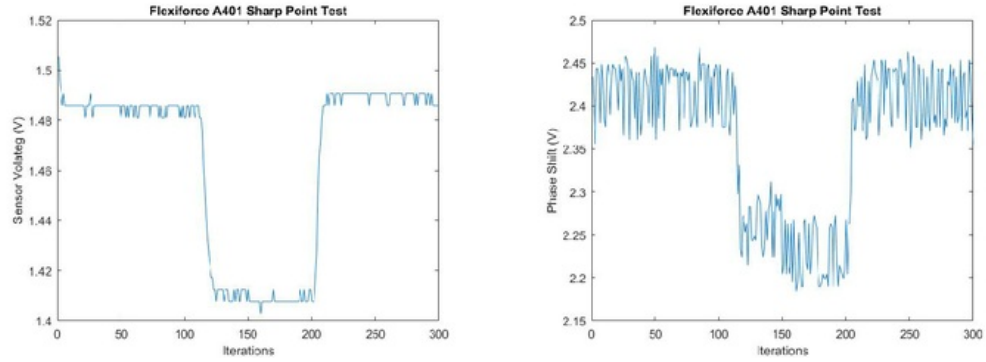


Figure 4.24: Flexiforce Sharp Point Test - Sensor Voltage & Phase Shift

Piezoresistive Sensor Finger Test

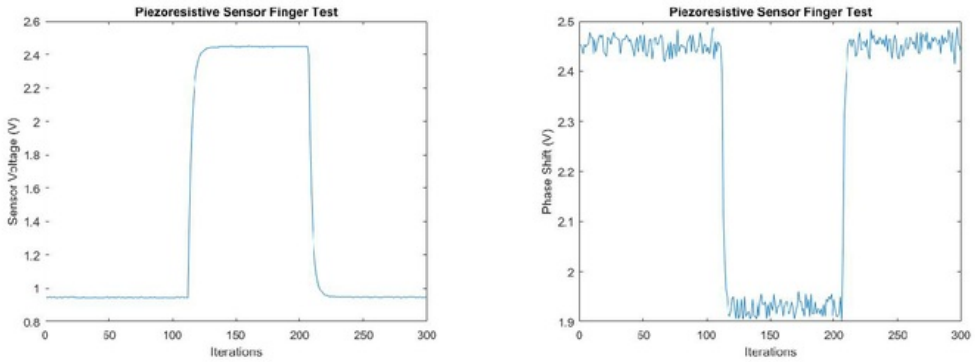


Figure 4.25: Piezoresistive Sensor Finger Test - Sensor Voltage & Phase Shift

Piezoresistive Sensor Circle Test

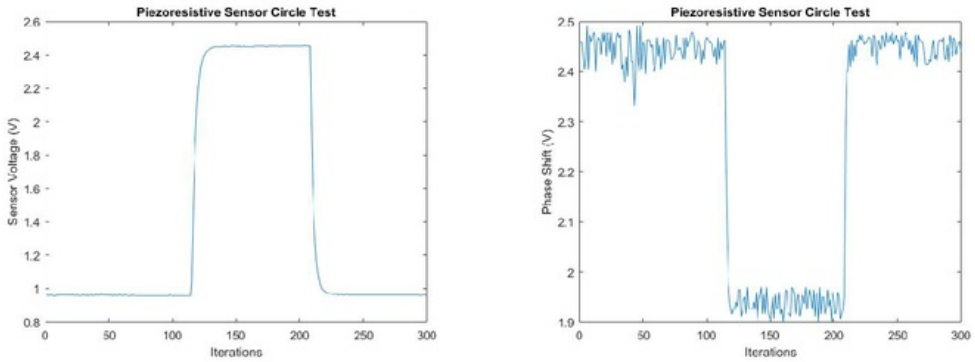


Figure 4.26: Piezoresistive Sensor Circle Test - Sensor Voltage & Phase Shift

Piezoresistive Sensor Cylinder Edge Test

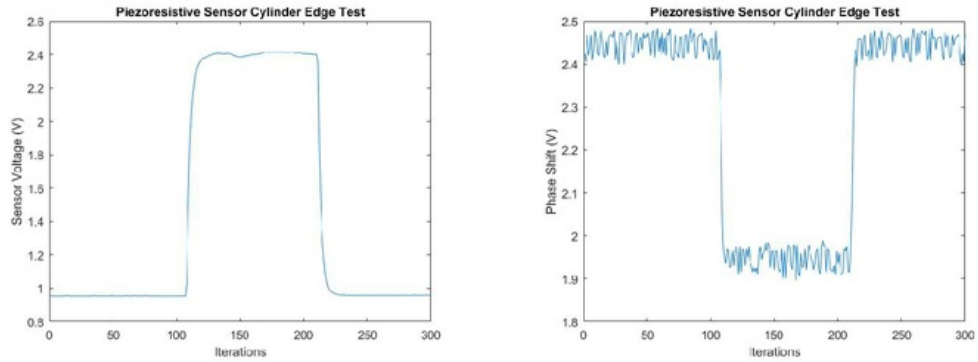


Figure 4.27: Piezoresistive Sensor Cylinder Edge Test - Sensor Voltage & Phase Shift

Piezoresistive Sensor Sphere Test

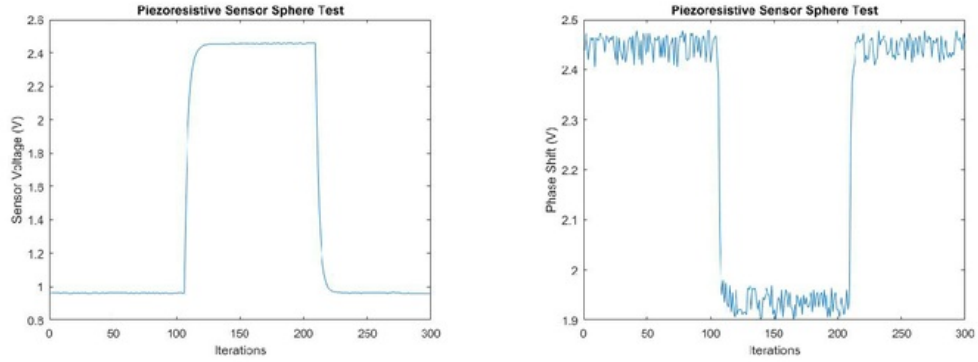


Figure 4.28: Piezoresistive Sensor Sphere Test - Sensor Voltage & Phase Shift

Piezoresistive Sensor Thin Edge Test

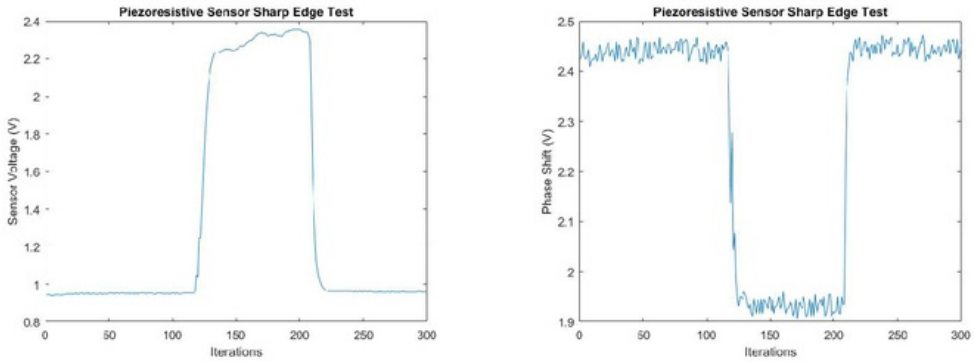


Figure 4.29: Piezoresistive Sensor Thin Edge Test - Sensor Voltage & Phase Shift

Piezoresistive Sensor Sharp Point Test

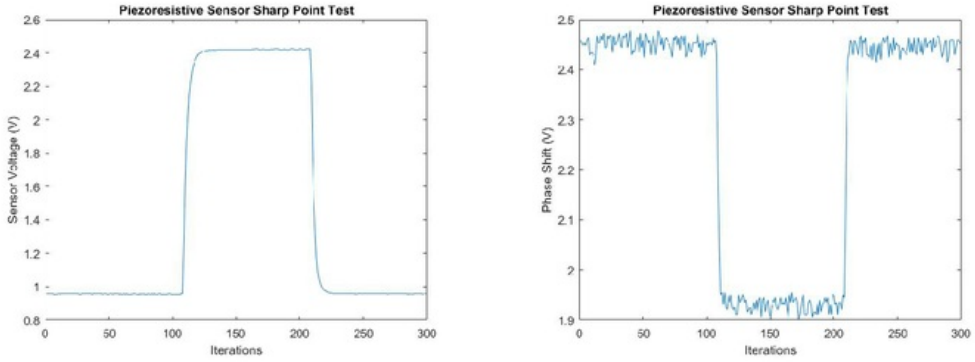


Figure 4.30: Piezoresistive Sensor Sharp Point Test - Sensor Voltage & Phase Shift

4.6.3 Discussion

Flexiforce A401 Sensor

With the finger test for the Flexiforce sensor, the results are similar to the response test for the sensor voltage, where after the load is removed from the sensor, the sensor does not return to its initial state immediately. There is a sharp decline to a point above the initial value, then a very slow decline can be seen. The phase shift results are similar to the sensor voltage as the phase shift after the load has been removed is slightly lower than the initial phase shift.

The circle test shows some interesting results. After the load has been removed, the sensor voltage stays noticeably higher than the finger test, sitting at around 1.9 V as opposed to 1.7 V. This may be due to the uniform spread of force across the large contact area of the circle in comparison to the finger which has an uneven contact area and the load distribution may not have been even. Another potential reason for this result may be due to the material type. The circle face is a hard plastic material in contrast to the soft and malleable finger. Although this result may be able to distinguish elements of a gripped object, it is not entirely useful for a quick assessment of a gripped object, or continued monitoring, as it requires the load to be applied and removed to see the results. Another noticeable aspect of these results is the phase shift response not return to its initial state.

The cylinder edge test and the thin edge test show similar results, which makes sense as they are similar shapes. The sensor voltage in both tests appears to be noisy when the load is applied. This may be due to the distribution of the force not being steady across the entire edge of the object, causing some fluctuations. Other than the noisy signal, the results from the sensor voltage and phase shift appear to fall in line with the standard load test when using a finger, with a slightly lower voltage, reaching just under 1.8 V for the cylinder edge and 1.75 V for the thin edge as opposed to around 2.1 V for the finger test. This smaller voltage may be attributed to the smaller overall contact area of both edge tests in comparison to the finger test. The phase shift in both edge tests reaches similar levels as the reference finger test.

The sphere test shows no significant differences between the reference finger test. The only difference is that after the load has been removed, the voltage returns closer to the initial state. This is likely to be due to the smaller contact area of the sphere in comparison.

The sharp point test shows some unexpected results. When the load was applied to the sensor the voltage decreased instead of the expected increase, as per all other tests. This decrease in voltage is much smaller in magnitude than the other results, being only 0.08 V decreased in comparison to the reference test results of 0.6 V. The change is small, but very distinct. This test was performed several times to verify the results, and this

occurred every time. Another interesting note on these results is that the phase shift also decreases, even though the sensor voltage decreased. These results indicate that the Flexiforce A401 sensor may be able to distinguish a sharp point of contact separately for other types of loads. While this has no obvious use at the moment for adaptable grasping, this may have an application in some more specific uses.

Piezoresistive Sensor

The piezoresistive sensor results show that the shape of the load applied to the sensor has no significant impact to the sensor voltage or the phase shift response of the circuit for this particular sensor. Unlike the Flexiforce A401, this sensor does not seem to have only two real states of operation, full load and no load, where the Flexiforce sensor can determine the force applied. What is worth noting is that the sensor voltage and the phase shift appear to be clearer signals than the signals produced by the Flexiforce sensor.

Errors in Test

There are a few ways that errors may have been introduced into this test which may have affected the results. Firstly is the force used to apply the load. While the forces applied each time were similar, it is difficult to get it exact using human input. This may have resulted in variations in the result. Similarly, the distribution needs to be applied evenly across the contact surface which is difficult to do by hand. Finally, errors in the Arduino measurements may have contributed small errors to the results.

Shape Test Summary

The results do not provide enough evidence to suggest that the sensor voltage or the phase shift can determine an objects shape while the load is applied. While the shape cannot be determined, it can be seen that the contact area of the object on the sensor does appear to affect where the sensor voltage returns to after the load has been removed. Since this can only be determined after the load has been removed, its application to adaptable grasping is not apparent. The results from the sharp point test on the Flexiforce A401 sensor may be useful for applications that benefit from identifying sharp points for other types of load on the sensor, however this is not useful for the current application.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis project investigated the use of low cost tactile sensors being used for adaptable grasping. The force-sensing resistors investigated were a Flexiforce A401 sensor and an Intertek piezoresistive sensor. The interdigitated capacitive sensors investigated were a PDMS and carbon nanotube sensor, a graphene sensor, and an aluminium sensor. A circuit was developed to provide an input signal for the sensors, and provide two pieces of information for the microcontroller. The circuit filtered the signal so that the a DC output voltage was supplied which related to the force on the sensor, and the phase shift of the sensor output was compared to the input, and also filtered into a DC voltage. The robotic arm control program was developed using Matlab to control all five axis of the arm, including the gripper. This program provided a user interface to interact with the arm, and to gather data about the arm and the sensors. The program allowed the gripper to grip to a voltage, correlating with the sensor force. This threshold was adjustable by a variable. Had the sensors provided object characteristic information, this variable could have been adjusted for certain object shapes, therefore adapting to the object. Various tests were performed with the five sensors, including sensor characterisation, variable force tests, force response tests, phase shift tests and object shape tests. Looking at these results together, there is no indication that these particular sensors in this developed system can determine different characteristics of separate objects. The system could be modified to test different types of sensor to see if they can provide this information.

5.2 Future Work

After performing the investigation into adaptable grasping, there are a few different areas where the research could be expanded, listed as follows.

Improvements

One area where the project could be improved is the servos used. The Dynamixel AX12-a servos are good servos to use in terms of accessibility and functions available straight from the servo, however they are not as powerful as desired. In terms of gripping, the servos tripped their maximum torque limit settings for heavier loads, which limited what tasks the robotic arm could perform. Stronger servo motors would open up different tasks and actions the arm could perform.

Sensor Investigation

As the sensors investigated were not suitable for the adaptable grasping application, investigating more sensors would be an area to proceed in. The reason that the sensors were used was due to their low cost, so a more expensive approach may be necessary. The center of pressure sensor would be a good option to investigate as the sensor is able to provide a pressure map of sorts. This would open up the sensor to implementing other features such as slip detection.

Mapping Signal Values

While this project mainly looked at change in voltage and phase shift of the sensor, a model could be made to more accurately determine the degree of phase shift or the value of the force on the sensor. Had there been more time for the project and the sensors provided object determination results, this would have been performed.

Slip Detection

Slip detection is another area that could be investigated. This would also increase the cost of the project as more sensors may need to be implemented, on top of the currently investigated sensors. Optical systems are typically used for this purpose, in tandem with other sensors. This would increase the complexity of the system. Other methods of slip detection include micro-vibration sensors and array matrix sensors, which would also be expensive options.

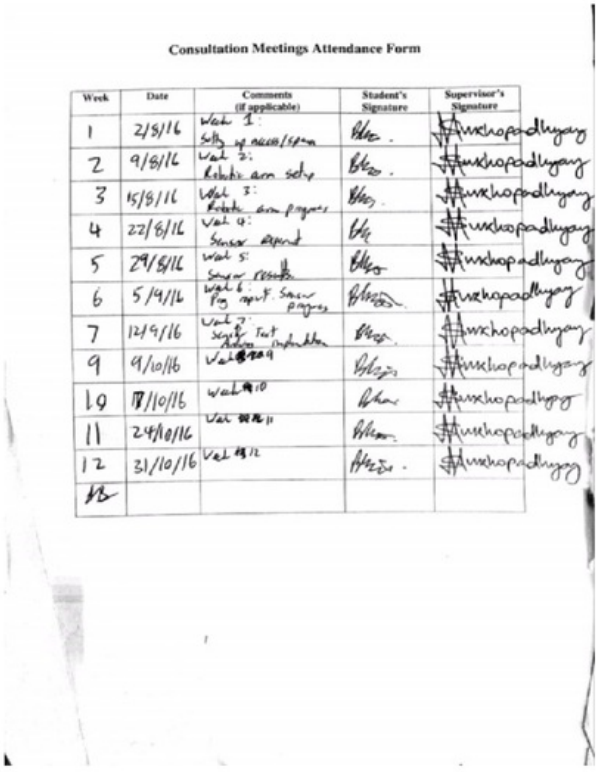
Chapter 6

Abbreviations

COM Port	Communications Port
CoP	Center of Pressure
DDS	Direct Digital Synthesis
DLL	Dynamic-link Library
DOF	Degrees of Freedom
EEPROM	Electrically Erasable Programmable Read-Only Memory
FSR	Force Sensitive Resistor
GUI	Graphical User Interface
GUIDE	GUI Development Environment
PCB	Printed Circuit Board
PDMS	Polydimethylsiloxane
PID Controller	Proportional-Integral-Derivative Controller
PWM	Pulse Width Modulation
RAM	Random Access Memory
USB	Universal Serial Bus
ZCD	Zero Crossing Detector

Appendix A

Meeting Attendance



Consultation Meetings Attendance Form

Week	Date	Comments (if applicable)	Student's Signature	Supervisor's Signature
1	2/8/16	Week 1: Setting up ROS/Gazebo		
2	9/8/16	Week 2: Robotic arm setup		
3	15/8/16	Week 3: Robotic arm progress		
4	22/8/16	Week 4: Sensor report		
5	29/8/16	Week 5: Sensor results		
6	5/9/16	Week 6: Prog. report. Sensor progress		
7	12/9/16	Week 7: Sensor Test Arduino Implementation		
9	9/10/16	Week 8 & 9		
10	17/10/16	Week 10		
11	24/10/16	Week 11		
12	31/10/16	Week 12		

Figure A.1: Meeting Attendance Sheet.

Appendix B

Project Timeline and Budget

B.1 Project Timeline

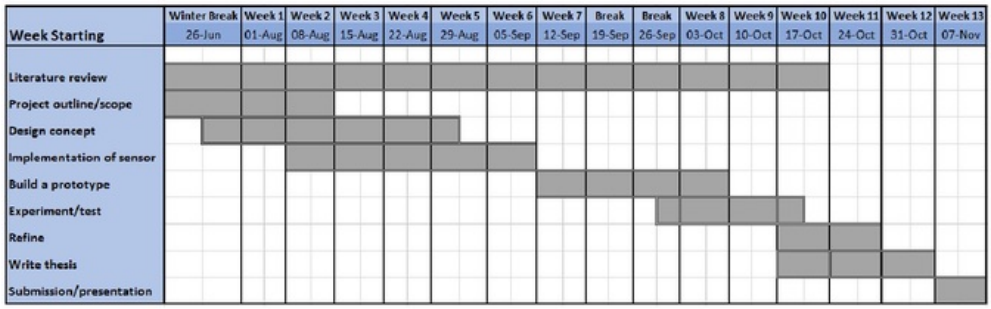


Figure B.1: The Proposed Project Timeline.

B.2 Project Budget

The project has only one cost associated with it as most of the parts used were already available from Macquarie University. The only bought component was printing the sensor circuit onto a PCB which cost $145.90.

Appendix C

Matlab Code

C.1 Control Program Code

The following is the code developed to control the robotic arm using Matlab. It calls upon the GUI code run the user interface. This is provided in the next section.

```
%Dynamixel Smart Robotic Arm Control Program
%Written By Brendan Menzies - 42849969
%Macquarie University - Mechatronic Engineering
%For use in Research Thesis - Intelligent Robotic Grasping

%Program Initialisation
    ----------------------------------------------------------------

loadlibrary('dynamixel','dynamixel.h');

DEFAULT_PORTNUM = 4; %COM Port Number
DEFAULT_BAUDNUM = 34; %1 for 1mbps, 34 for 57142 kbs

%Servo Limits
%global variables are shared with the gui
    global s1max; s1max = 890;
    global s1min; s1min = 30;
    global s2max; s2max = 890;
    global s2min; s2min = 440;
    global s3max; s3max = s2max;
    global s3min; s3min = s2min;
    global s4max; s4max = 890;
    global s4min; s4min = 250;
    global s5max; s5max = s4max;
    global s5min; s5min = s4min;
    global s6max; s6max = 780;
    global s6min; s6min = 160;
    global s7max; s7max = 900;
    global s7min; s7min = 495;

    global s1pos; s1pos = 150/0.29;
```

```
    global s2pos; s2pos = 250/0.29;
    global s4pos; s4pos = 85/0.29;
    global s6pos; s6pos = 145/0.29;

    global grip_button_var; grip_button_var = 0;
    global program_pause; program_pause = 0;
    global program_loop; program_loop = 1;
    global move_speed; move_speed = 75;
    global change_speed; change_speed = 0;

Limits = [s1max s1min;s2max s2min;s3max s3min;s4max s4min;s5max s5min;s6max
    s6min;s7max s7min]; %hard limits on servos, to prevent damage

%AX-12a function addresses
P_GOAL_POSITION = 30;
P_PRESENT_POSITION = 36;
P_MOVE_SPEED = 32;
P_MOVING = 46;
P_CW_ANGLE_LIMIT = 6;
P_CCW_ANGLE_LIMIT = 8;
P_TORQUE_ENABLE = 24;
P_TORQUE_LIMIT_L = 34;
P_TORQUE_LIMIT_H = 35;
P_VOLTAGE = 42;
P_PRESENT_LOAD = 40;

%other variables
grip_threshold = 1.5;
voltage_samples = 3;
voltoutput = 0;
shutdown_timer = 300;
prev_s1pos = 0;
prev_s2pos = 0;
prev_s4pos = 0;
prev_s6pos = 0;

average_matrix = 1:voltage_samples;
phase_matrix = 1:voltage_samples;
voltage_matrix = zeros([1 shutdown_timer]);
phase_matrix = zeros([1 shutdown_timer]);

%GUI Setup ------------------------------------------------------------------
%this stuff needs to only run once. The gui opening function loops.

handles = guihandles(dynamixel_gui);

%set start position
set(handles.s1_slider, 'value', 150);
set(handles.s2_slider, 'value', 250);
set(handles.s4_slider, 'value', 85);
set(handles.s6_slider, 'value', 145);
set(handles.s7_slider, 'value', 145);
```

```
%Display start position
set(handles.s1_value, 'string', 150);
set(handles.s2_value, 'string', 250);
set(handles.s4_value, 'string', 85);
set(handles.s6_value, 'string', 145);
set(handles.s7_value, 'string', 145);

s7pos = (get(handles.s7_slider, 'value'))/0.29; %sets s7 initial pos

%arduino setup
a = arduino('COM3') %#ok<NOPTS> - stops errors appearing

zz = 1; %counts loop iterations

%Servo Setup
    ------------------------------------------------------------------

%initialises the connection
res = calllib('dynamixel', 'dxl_initialize', DEFAULT_PORTNUM,
    DEFAULT_BAUDNUM); %#ok<*NASGU>

%set angle limits
for i = 1:7
    id = i;
    calllib('dynamixel','dxl_write_word',id,P_CCW_ANGLE_LIMIT,Limits(i,1));
    calllib('dynamixel','dxl_write_word',id,P_CW_ANGLE_LIMIT,Limits(i,2));
end

%write move speed
for i = 1:6
    id = i;
    calllib('dynamixel','dxl_write_word',id,P_MOVE_SPEED,move_speed);
end

%move speed for gripper
calllib('dynamixel','dxl_write_word',7,P_MOVE_SPEED,175);

%torque limit for gripper
calllib('dynamixel','dxl_write_word',7,P_TORQUE_LIMIT_L,1023);

%dynamixel_gui %loads GUI

%Program Loop
    ------------------------------------------------------------------

while (program_loop == 1) && (zz < shutdown_timer+1) %main loop, continues
    until ProgramLoop is false i.e. stop buton

    vout = readVoltage(a,'A0');
    phase = readVoltage(a,'A1');
    voltage_matrix(zz) = vout;
```

```
    phase_matrix(zz) = phase;

    handles = guihandles(dynamixel_gui);

    if change_speed == 1;
        %write move speed
        for i = 1:6
            id = i;
            calllib('dynamixel','dxl_write_word',id,P_MOVE_SPEED,move_speed
    );
        end
        change_speed = 0;
    end

    while program_pause == 1
        %Disables torque on motors when stop is pushed
        for i = 1:7
            id = i;
            calllib('dynamixel','dxl_write_word',id,P_TORQUE_ENABLE,0);
        end
        pause(0.1);
    end

    %write goal position
    if prev_s1pos ~= s1pos
        calllib('dynamixel','dxl_write_word',1,30,s1pos);
    end
    if prev_s2pos ~= s2pos
        calllib('dynamixel','dxl_write_word',2,30,s2pos);
        calllib('dynamixel','dxl_write_word',3,30,s2pos);
    end
    if prev_s4pos ~= s4pos
        calllib('dynamixel','dxl_write_word',4,30,s4pos);
        calllib('dynamixel','dxl_write_word',5,30,s4pos);
    end
    if prev_s6pos ~= s6pos
        calllib('dynamixel','dxl_write_word',6,30,s6pos);
    end

    %old gripper control - gripper val is based off pressure applied
    %     mapped_val = s7min + (vout-0)*(s7max-s7min)/2; %0 is voltage min,
     2 is voltage max from sensor
    %     calllib('dynamixel','dxl_write_word',7,P_GOAL_POSITION,mapped_val
    );
    %     %set(handles.s7_voltage, 'string', num2str(mapped_val));
    %     set(handles.s7_slider, 'value', mapped_val*0.29);

    %gripper control - gripper mode closes gripper until set voltage
    if grip_button_var == 1;
        if (voltoutput < grip_threshold) && (voltoutput ~= 0)
            s7pos = s7pos+5;
        end
```

```
179         calllib('dynamixel','dxl_write_word',7,P_GOAL_POSITION,s7pos);
    else
181         s7pos = (get(handles.s7_slider, 'value'))/0.29;
            calllib('dynamixel','dxl_write_word',7,30,s7pos);
183     end

185     %reset alarms
        if get(handles.alarm_button, 'value') == 1
187         calllib('dynamixel','dxl_write_word',7,P_TORQUE_LIMIT_L,1023);
            calllib('dynamixel','dxl_write_word',7,P_TORQUE_ENABLE,1);
189     end

191     if rem(zz, voltage_samples) == 0 %stores voltmax every x iterations and
        resets value
            average_matrix(1,voltage_samples) = vout;
193         phase_matrix(1,voltage_samples) = phase;
            %mean(average_matrix,2);
195         voltoutput = mean(average_matrix,2);  %average_matrix(1,1);
            phaseoutput = mean(phase_matrix,2);
197         set(handles.s7_voltage, 'string', voltoutput); %sensor voltage
            set(handles.phase_shift_text, 'string', phaseoutput); %sensor
    voltage
199 %         gripper_load = calllib('dynamixel','dxl_read_word',7,
    P_PRESENT_LOAD);
    %         set(handles.gripper_load_value, 'string', gripper_load - 1024);
201
    else
203         average_matrix(1,rem(zz,voltage_samples)) = vout;
            phase_matrix(1,rem(zz,voltage_samples)) = phase;
205     end

207     %         set(handles.s7_voltage, 'string', vout); %sensor voltage
        %         set(handles.phase_shift_text, 'string', phase); %phase shift
209

211     %store current goal positions for next loop
        prev_s1pos = s1pos;
213     prev_s2pos = s2pos;
        prev_s4pos = s4pos;
215     prev_s6pos = s6pos;

217     set(handles.iterations, 'string', zz); %iteration count

219     zz = zz+1; %increment iteration count
end
221
%Program Shutdown
    ------------------------------------------------------------------------
223
res = calllib('dynamixel', 'dxl_initialize', DEFAULT_PORTNUM,
    DEFAULT_BAUDNUM);
225
```

```
%Disables torque on motors when stop is pushed
    for i = 1:7
        id = i;
        calllib('dynamixel','dxl_write_word',id,P_TORQUE_ENABLE,0);
    end

%Terminates connection and closes gui
clear a;
calllib('dynamixel','dxl_terminate');
unloadlibrary('dynamixel')
close all force
%clearvars
```

C.2 GUI Code

The following is the code for the GUI portion of the Matlab code. It is called upon by the above control code and presents the user interface.

```
%GUI Program to accompay dynamixel_control program.
%Generated from GUIDE
%Other code written by Brendan Menzies - 42849969
%Macquarie University - Mechatronic Engineering
%For use in Research Thesis - Intelligent Robotic Grasping

function varargout = dynamixel_gui(varargin)

% Last Modified by GUIDE v2.5 14-Oct-2016 15:31:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @dynamixel_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @dynamixel_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

global k; k =1;
% End initialization code - DO NOT EDIT
```

```
32
% --- Executes just before dynamixel_gui is made visible.
34

36 function dynamixel_gui_OpeningFcn(hObject, eventdata, handles, varargin)

38  global s1max;
    global s1min;
40  global s2max;
    global s2min;
42  global s4max;
    global s4min;
44  global s6max;
    global s6min;
46  global s7max;
    global s7min;
48
%show min and max labels
50 set(handles.s1_max, 'string', num2str(s1max*0.29));
   set(handles.s2_max, 'string', num2str(s2max*0.29));
52 set(handles.s4_max, 'string', num2str(s4max*0.29));
   set(handles.s6_max, 'string', num2str(s6max*0.29));
54 set(handles.s7_max, 'string', num2str(s7max*0.29));
   set(handles.s1_min, 'string', num2str(s1min*0.29));
56 set(handles.s2_min, 'string', num2str(s2min*0.29));
   set(handles.s4_min, 'string', num2str(s4min*0.29));
58 set(handles.s6_min, 'string', num2str(s6min*0.29));
   set(handles.s7_min, 'string', num2str(s7min*0.29));
60
%set slider max and min values
62 set(handles.s1_slider, 'max', s1max*0.29);
   set(handles.s2_slider, 'max', s2max*0.29);
64 set(handles.s4_slider, 'max', s4max*0.29);
   set(handles.s6_slider, 'max', s6max*0.29);
66 set(handles.s7_slider, 'max', s7max*0.29);
   set(handles.s1_slider, 'min', s1min*0.29);
68 set(handles.s2_slider, 'min', s2min*0.29);
   set(handles.s4_slider, 'min', s4min*0.29);
70 set(handles.s6_slider, 'min', s6min*0.29);
   set(handles.s7_slider, 'min', s7min*0.29);
72
%set slider step
74 set(handles.s1_slider, 'sliderstep', [1/(s1max-s1min), 0.1]);
   set(handles.s2_slider, 'sliderstep', [1/(s2max-s2min), 0.1]);
76 set(handles.s4_slider, 'sliderstep', [1/(s4max-s4min), 0.1]);
   set(handles.s6_slider, 'sliderstep', [1/(s6max-s6min), 0.1]);
78 set(handles.s7_slider, 'sliderstep', [1/(s7max-s7min), 0.1]);

80 handles.output = hObject;
   guidata(hObject, handles);
82
```

```
% --- Outputs from this function are returned to the command line.
function varargout = dynamixel_gui_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


function s1_slider_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function s1_slider_Callback(hObject, eventdata, handles)
global s1pos;
handles=guidata(hObject);

set(handles.s1_value, 'string', num2str(get(handles.s1_slider, 'value')));
s1pos = (get(handles.s1_slider, 'value'))/0.29;


guidata(hObject, handles);



% --- Executes on slider movement.
function s2_slider_Callback(hObject, eventdata, handles)
global s2pos;
handles=guidata(hObject);

set(handles.s2_value, 'string', num2str(get(handles.s2_slider, 'value')));
s2pos = (get(handles.s2_slider, 'value'))/0.29;

guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function s2_slider_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function s4_slider_Callback(hObject, eventdata, handles)
global s4pos;
handles=guidata(hObject);

```

```
set(handles.s4_value, 'string', num2str(get(handles.s4_slider, 'value')));
s4pos = (get(handles.s4_slider, 'value'))/0.29;

guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function s4_slider_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function s6_slider_Callback(hObject, eventdata, handles)
global s6pos;
handles=guidata(hObject);

set(handles.s6_value, 'string', num2str(get(handles.s6_slider, 'value')));
s6pos = (get(handles.s6_slider, 'value'))/0.29;

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function s6_slider_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function s7_slider_Callback(hObject, eventdata, handles)
handles=guidata(hObject);

set(handles.s7_value, 'string', num2str(get(handles.s7_slider, 'value')));

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function s7_slider_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
182
% --- Executes on button press in stop_button.
184 function stop_button_Callback(hObject, eventdata, handles)

186 global program_loop;

188 if program_loop == 1
    program_loop = 0;
190 else
    program_loop = 1;
192 end
% Hint: get(hObject,'Value') returns toggle state of stop_button
194

196 % --- Executes on button press in pause_button.
function pause_button_Callback(hObject, eventdata, handles)
198
global program_pause;
200 % Hint: get(hObject,'Value') returns toggle state of pause_button
if program_pause == 0
202     program_pause = 1;
else
204     program_pause = 0;
end
206
% --- Executes on button press in alarm_button.
208 function alarm_button_Callback(hObject, eventdata, handles)


210


212 function s1_textb_Callback(hObject, eventdata, handles)


214


216 guidata(hObject,handles)


218
% --- Executes during object creation, after setting all properties.
220 function s1_textb_CreateFcn(hObject, eventdata, handles)

222 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
224 end


226


228 function s2_textb_Callback(hObject, eventdata, handles)

230 function s2_textb_CreateFcn(hObject, eventdata, handles)

232 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
```

```
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function s4_textb_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function s4_textb_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function s6_textb_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function s6_textb_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function s7_textb_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function s7_textb_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in s1_button.
function s1_button_Callback(hObject, eventdata, handles)

handles=guidata(hObject);

setAngle = str2double(get(handles.s1_textb, 'string'));
set(handles.s1_slider, 'value', setAngle);

```

```
set(handles.s1_value, 'string', num2str(get(handles.s1_slider, 'value')));
282
guidata(hObject, handles);
284
% --- Executes on button press in s2_button.
286 function s2_button_Callback(hObject, eventdata, handles)

288 handles=guidata(hObject);

290 setAngle = str2double(get(handles.s2_textb, 'string'));
set(handles.s2_slider, 'value', setAngle);
292
set(handles.s2_value, 'string', num2str(get(handles.s2_slider, 'value')));
294
guidata(hObject, handles);
296
% --- Executes on button press in s4_button.
298 function s4_button_Callback(hObject, eventdata, handles)

300 handles=guidata(hObject);

302 setAngle = str2double(get(handles.s4_textb, 'string'));
set(handles.s4_slider, 'value', setAngle);
304
set(handles.s4_value, 'string', num2str(get(handles.s4_slider, 'value')));
306
guidata(hObject, handles);
308
% --- Executes on button press in s6_button.
310 function s6_button_Callback(hObject, eventdata, handles)

312 handles=guidata(hObject);

314 setAngle = str2double(get(handles.s6_textb, 'string'));
set(handles.s6_slider, 'value', setAngle);
316
set(handles.s6_value, 'string', num2str(get(handles.s6_slider, 'value')));
318
guidata(hObject, handles);
320
% --- Executes on button press in s7_button.
322 function s7_button_Callback(hObject, eventdata, handles)

324 handles=guidata(hObject);

326 setAngle = str2double(get(handles.s7_textb, 'string'));
set(handles.s7_slider, 'value', setAngle);
328
set(handles.s7_value, 'string', num2str(get(handles.s7_slider, 'value')));
330
guidata(hObject, handles);
332
```

```
% --- Executes on button press in grip_button.
function grip_button_Callback(hObject, eventdata, handles)

global grip_button_var;

if grip_button_var == 0
    grip_button_var = 1;
else
    grip_button_var = 0;
end


% --- Executes on button press in home_button.
function home_button_Callback(hObject, eventdata, handles)
handles=guidata(hObject);

%set start position
set(handles.s1_slider, 'value', 150);
set(handles.s2_slider, 'value', 250);
set(handles.s4_slider, 'value', 85);
set(handles.s6_slider, 'value', 145);
set(handles.s7_slider, 'value', 145);

%Display start position
set(handles.s1_value, 'string', 150);
set(handles.s2_value, 'string', 250);
set(handles.s4_value, 'string', 85);
set(handles.s6_value, 'string', 145);
set(handles.s7_value, 'string', 145)

global s1pos; s1pos = 150/0.29;
global s2pos; s2pos = 250/0.29;
global s4pos; s4pos = 85/0.29;
global s6pos; s6pos = 145/0.29;

guidata(hObject, handles);



function move_speed_text_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function move_speed_text_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


```

```
384 % --- Executes on button press in move_speed_button.
    function move_speed_button_Callback(hObject, eventdata, handles)
386
    global move_speed;
388 global change_speed;

390 move_speed = str2num(get(handles.move_speed_text, 'string'));
    change_speed = 1;
```

Appendix D

Arduino Program

This is the program that was used to output a PWM signal from the Arduino. This was not developed for the project, made by G. Hill [1].

```
/*
 * sinewave_pcm
 *
 * Generates 8-bit PCM sinewave on pin 6 using pulse-width modulation (PWM).
 * For Arduino with Atmega368P at 16 MHz.
 *
 * Uses timers 1 and 0. Timer 1 reads the sinewave table, SAMPLE_RATE times a
second.
 * The sinewave table has 256 entries. Consequently, the sinewave has a
frequency of
 * f = SAMPLE_RATE / 256
 * Each entry in the sinewave table defines the duty-cycle of Timer 0. Timer
0
 * holds pin 6 high from 0 to 255 ticks out of a 256-tick cycle, depending on
 * the current duty cycle. Timer 0 repeats 62500 times per second (16000000 /
256),
 * much faster than the generated sinewave generated frequency.
 *
 * References:
 * http://www.atmel.com/dyn/resources/prod_documents/doc2542.pdf
 * http://www.analog.com/library/analogdialogue/archives/38-08/dds.html
 * http://www.evilmadscientist.com/article.php/avrdac
 * http://www.arduino.cc/playground/Code/R2APCMAudio
 * http://www.scienceprog.com/generate-sine-wave-modulated-pwm-with-avr-
microcontroller/
 * http://www.scienceprog.com/avr-dds-signal-generator-v10/
 * http://documentation.renesas.com/eng/products/region/rtas/mpumcu/apn/
sinewave.pdf
 * http://ww1.microchip.com/downloads/en/AppNotes/00655a.pdf
 *
 * By Gary Hill
```

```
* Adapted from a script by Michael Smith <michael@hurts.ca>
*/
#include <stdint.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#define SAMPLE_RATE 8000
// 8 ksps
/*
* The sinewave data needs to be unsigned, 8-bit
*
* sinewavedata.h should look like this:
* const int sinewave_length=256;
*
* const unsigned char sinewave_data[] PROGMEM = {0x80,0x83, ...
*
*/
#include "sinewavedata.h"
int outputPin = 6; // (PCINT22/OC0A/AIN0)PD6, Arduino Digital Pin 6
volatile uint16_t sample;
// This is called at SAMPLE_RATE kHz to load the next sample.
ISR(TIMER1_COMPA_vect) {
if (sample >= sinewave_length) {
sample = -1;
}
else {
OCR0A = pgm_read_byte(&sinewave_data[sample]);
}
++sample;
}
void startPlayback()
{
pinMode(outputPin, OUTPUT);
// Set Timer 0 Fast PWM Mode (Section 14.7.3)
// WGM = 0b011 = 3
(Table 14-8)
// TOP = 0xFF, update OCR0A register at BOTTOM
TCCR0A |= _BV(WGM01) | _BV(WGM00);
TCCR0B &= ~_BV(WGM02);
// Do non-inverting PWM on pin OC0A, arduino digital pin 6
// COM0A = 0b10, clear OC0A pin on compare match,
//
set 0C0A pin at BOTTOM (Table 14-3)
TCCR0A = (TCCR0A | _BV(COM0A1)) & ~_BV(COM0A0);
// COM0B = 0b00, OC0B disconnected (Table 14-6)
TCCR0A &= ~(_BV(COM0B1) | _BV(COM0B0));
// No prescaler, CS = 0b001 (Table 14-9)
TCCR0B = (TCCR0B & ~(_BV(CS02) | _BV(CS01))) | _BV(CS00);
// Set initial pulse width to the first sample.
OCR0A = pgm_read_byte(&sinewave_data[0]);
// Set up Timer 1 to send a sample every interrupt.
cli();
```

```
// disable interrupts
// Set CTC mode (Section 15.9.2 Clear Timer on Compare Match)
// WGM = 0b0100, TOP = OCR1A, Update OCR1A Immediate (Table 15-4)
// Have to set OCR1A *after*, otherwise it gets reset to 0!
TCCR1B = (TCCR1B & ~_BV(WGM13)) | _BV(WGM12);
TCCR1A = TCCR1A & ~(_BV(WGM11) | _BV(WGM10));
// No prescaler, CS = 0b001 (Table 15-5)
TCCR1B = (TCCR1B & ~(_BV(CS12) | _BV(CS11))) | _BV(CS10);
// Set the compare register (OCR1A).
// OCR1A is a 16-bit register, so we have to do this with
// interrupts disabled to be safe.
OCR1A = F_CPU / SAMPLE_RATE;
// 16e6 / 8000 = 2000
// Enable interrupt when TCNT1 == OCR1A (p.136)
TIMSK1 |= _BV(OCIE1A);
sample = 0;
sei();
// enable interrupts
}
void setup()
{
startPlayback();
}
void loop()
{
while (true);
}
sinewavedata.h
/* Sinewave table
* Reference:
* http://www.scienceprog.com/generate-sine-wave-modulated-pwm-with-avr-
microcontroller/
*/
const int sinewave_length=256;
const unsigned char sinewave_data[] PROGMEM = {
0x80,0x83,0x86,0x89,0x8c,0x8f,0x92,0x95,0x98,0x9c,0x9f,0xa2,0xa5,0xa8,0xab
    ,0xae,
0xb0,0xb3,0xb6,0xb9,0xbc,0xbf,0xc1,0xc4,0xc7,0xc9,0xcc,0xce,0xd1,0xd3,0xd5
    ,0xd8,
0xda,0xdc,0xde,0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xed,0xef,0xf0,0xf2,0xf3
    ,0xf5,
0xf6,0xf7,0xf8,0xf9,0xfa,0xfb,0xfc,0xfc,0xfd,0xfe,0xfe,0xff,0xff,0xff,0xff
    ,0xff,
0xff,0xff,0xff,0xff,0xff,0xfe,0xfe,0xfd,0xfc,0xfc,0xfb,0xfa,0xf9,0xf8
    ,0xf7,
0xf6,0xf5,0xf3,0xf2,0xf0,0xef,0xed,0xec,0xea,0xe8,0xe6,0xe4,0xe2,0xe0,0xde
    ,0xdc,
0xda,0xd8,0xd5,0xd3,0xd1,0xce,0xcc,0xc9,0xc7,0xc4,0xc1,0xbf,0xbc,0xb9,0xb6
    ,0xb3,
0xb0,0xae,0xab,0xa8,0xa5,0xa2,0x9f,0x9c,0x98,0x95,0x92,0x8f,0x8c,0x89,0x86
    ,0x83,
0x80,0x7c,0x79,0x76,0x73,0x70,0x6d,0x6a,0x67,0x63,0x60,0x5d,0x5a,0x57,0x54
```

```
    ,0x51,
0x4f,0x4c,0x49,0x46,0x43,0x40,0x3e,0x3b,0x38,0x36,0x33,0x31,0x2e,0x2c,0x2a
    ,0x27,
0x25,0x23,0x21,0x1f,0x1d,0x1b,0x19,0x17,0x15,0x13,0x12,0x10,0x0f,0x0d,0x0c
    ,0x0a,
0x09,0x08,0x07,0x06,0x05,0x04,0x03,0x03,0x02,0x01,0x01,0x00,0x00,0x00,0x00
    ,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x02,0x03,0x03,0x04,0x05,0x06,0x07
    ,0x08,
0x09,0x0a,0x0c,0x0d,0x0f,0x10,0x12,0x13,0x15,0x17,0x19,0x1b,0x1d,0x1f,0x21
    ,0x23,
0x25,0x27,0x2a,0x2c,0x2e,0x31,0x33,0x36,0x38,0x3b,0x3e,0x40,0x43,0x46,0x49
    ,0x4c,
0x4f,0x51,0x54,0x57,0x5a,0x5d,0x60,0x63,0x67,0x6a,0x6d,0x70,0x73,0x76,0x79
    ,0x7c};
```

Bibliography

[1] Pwm sine wave generation. [Online]. Available: http://web.csulb.edu/~hill/ee470/Lab%202d%20-%20Sine_Wave_Generator.pdf

[2] Robotis e-manual (v1.25.00). [Online]. Available: http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm

[3] P. K. Allen, A. Timcenko, B. Yoshimi, and P. Michelman, "Automated tracking and grasping of a moving object with a robotic hand-eye system," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 2, pp. 152–165, 1993.

[4] A. M. Dollar and R. D. Howe, "Simple, robust autonomous grasping in unstructured environments," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 4693–4700.

[5] D. Gunji *et al.*, "Grasping force control of multi-fingered robot hand based on slip detection using tactile sensors," *IEEE International Conference on Robotics and Automation*, 2008.

[6] A. Hollinger and M. M. Wanderley, "Evaluation of commercial force-sensing resistors," in *Proceedings of International Conference on New Interfaces for Musical Expression*. Citeseer, 2006.

[7] C. Lebosse, B. Bayle, M. de Mathelin, and P. Renaud, "Nonlinear modeling of low cost force sensors," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3437–3442.

[8] C. Melchiorri, "Slip detection and control using tactile and force sensors," *IEEE/ASME Transactions on Mechatronics*, vol. 5, no. 3, pp. 235–243, Sep 2000.

[9] B. Menzies, "Final project plan," 2016.

[10] A. Nag and S. Mukhopadhyay, "Tactile sensing from laser-ablated metallized pet films."

[11] A. Saxena, J. Driemeyer, J. Kearns, and A. Y. Ng, "Robotic grasping of novel objects," in *Advances in neural information processing systems*, 2006, pp. 1209–1216.

[12] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, 2008.

[13] P. A. Schmidt *et al.*, "A sensor for dynamic tactile information with applications in human-robot interaction and object exploration," *Robotics and Autonomation*, vol. 54, no. 12, pp. 1005–1014, Dec 2008.

[14] M. VACEK, J. ŽILKOVÁ, and M. PÁSTOR, "Regulation of dynamixel actuators in robot manipulator movement," *Acta Electrotechnica et Informatica*, vol. 14, no. 3, pp. 32–35, 2014.

[15] F. Vecchi, C. Freschi, S. Micera, A. M. Sabatini, P. Dario, R. Sacchetti *et al.*, "Experimental evaluation of two commercial force sensors for applications in biomechanics and motor control," in *5th Ann. Conf. of Int. FES*, 2000.

[16] L. Xie, "An electrochemical impedance spectroscopy based nitrate sensor for practical application: a project report submitted in partial fulfillment of the requirements for the degree of master of engineering in electrical and electronics engineering, school of engineering and advanced technology, massey university, palmerston north, new zealand," Ph.D. dissertation, Massey University, 2016.