HAND GESTURE CONTROLLED ROBOTIC ARM

Solomon Riley

Bachelor of Engineering
Mechatronics

Department of Electronic Engineering
Macquarie University

November 13, 2017

Supervisor: Prof. Suhbas Mukhopadhyay

ACKNOWLEDGMENTS

I would like to firstly acknowledge my parents for raising me to the best of thier abilities and providing me with my foundational education that was able to be the platform I needed for my further studies. I would also like to acknowladge my supervisor Prof. Suhbas Mukhopadhyay for teaching me the fundamentals and for his continuous support.

STATEMENT OF CANDIDATE

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

Student's Name: Solomon Riley

Student's Signature: 

Date: 13th November 2017

ABSTRACT

Over the years, robots have been designed to perform various tasks much quicker and more efficient than humans. Though these robots may be fast and capable, controlling them can be slow and cumbersome. Hand gesture control is an emerging method of controlling robots in a simplified and more efficient manner. Using hand gesture control over other control options is beneficial as it incorporates natural human movement to achieve the same or even more control in an uncomplicated fashion. The project seeks to develop a cheap and simple method of hand gesture control for a robotic arm with five degrees of freedom (DOF). The proposed method is a glove fitted with various sensors that will interpret hand gestures, process them and control the robot. A flex sensor, two accelerometers, and a gyroscope will be attached to the glove to achieve this control. The result is a cheap and simple prototype that has full control over the 5 DOF of the robotic arm. The concepts of this project can be applied to many areas of industry with possibilities such as military applications, medical procedures or even handling hazardous waste such as nuclear etc.

Contents

List of Figures

List of Tables

Abbreviations

GND	Ground
VCC	Voltage
CS	Chip Select
Int	Interrupt
SDO	Serial Data Out
SDA	Serial Data
SCL	Serial Clock
I2C	Inter-integrated Circuit
SPI	Serial Peripheral Interface
SS	Servo Select
ACC1	Accelerometer 1
ACC2	Accelerometer 2
ADC	Analogue to Digital converter

Chapter 1

Introduction

Robots aid us in many ways every single day. They are designed to be proficient at completing arduous repetitive tasks quickly, efficiently and with great precision. Robots can complete tasks deemed too dangerous for humans and can operate in extreme conditions where humans would be unable to effectively complete tasks. Though these robots have their advantages, robot-human collaboration could be considered a weakness and an area of improvement [4]. Various methods of control for robot currently exist: manual control with joysticks, or buttons, and autonomous control with micro-controllers With the aid of a prototype input device, the possibility of creating an improved human-robot interface using hand gestures will be explored. The success of this project could mean the advancement in areas such as handling of dangerous or harmful substances. Medical procedures in locations at which a physician can complete the procedure without the need to physically access the area [5], and completing tasks in harsh environments unsuitable for human occupation, which can be impacted by this projects outcome.

1.1 Project Overview

This project aims to create a fully functional prototype controller for a hand gesture controlled system. A hand will perform certain movements and the controller will be able to understand those movements and direct a robotic arm to follow accordingly. The robotic arm is fully articulated with 5 degrees of freedom. Joint 1: baseplate rotation, joint 2: elbow articulation, joint 3: wrist articulation, joint 4: wrist rotation, and joint 5: gripper articulation as shown in figure 1.1. Four sensors will be attached to a glove and this glove will read the hand gestures and direct the robot. Two accelerometers are used to measure x-axis, y-axis, and z-axis movement for joints 2, 3, and 4 a flex sensor is used to detect finger movement, and a gyroscope is used to rotate the arm. These sensors will measure acceleration and rotational movements of the hand and transfer the signals to the robotic arm.

Figure 1.1: Robotic Arm

1.1.1 Objectives

The first objective is the overall goal for the project while the subsequent objectives contribute to the final goal.

- Hand gesture control of the robotic arm
- Accelerometer control of wrist articulation and rotation
- Accelerometer control of elbow articulation
- Gyroscope control of base plate rotation
- Flex sensor control of gripper articulation

Chapter 2

Background and Related Work

2.1 Introduction

Existing information on this topic must be appraised to gain a knowledge base for which to begin the project. This chapter evaluates a few past projects that have similar goals and objectives. These projects will be critically evaluated and placed in context with this current project. Decisions made as a result of the study of similar projects with be clearly stated and explained.

2.2 Method of Gesture Input

A project by Peter Girovsk and Mat Kundrt using dynamixel motors to control a robotic arm employs the use of a glove to achieve gesture control [1]. This "data glove" as it has been named, is able to house a total of four sensors and the electronics that are required to operate them. Figure 2.1 shows this glove with the sensors and electronics.

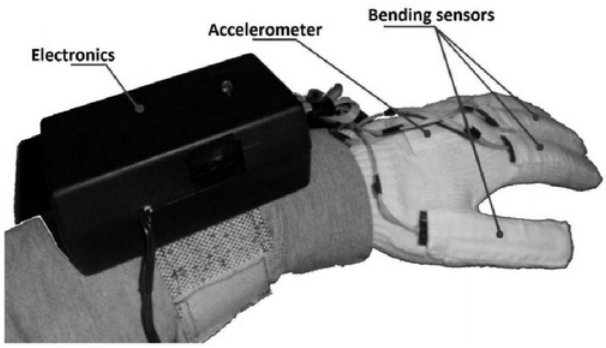

Figure 2.1: Data Glove [1]

Another project uses a glove to achieve control of an animatronic hand. This glove was able to house six sensors. five of the sensors control the gripping and relaxing of fingers and the sixth sensor controls the orientation of the hand [2]. Figure 2.2 shows this glove with the sensors attached.

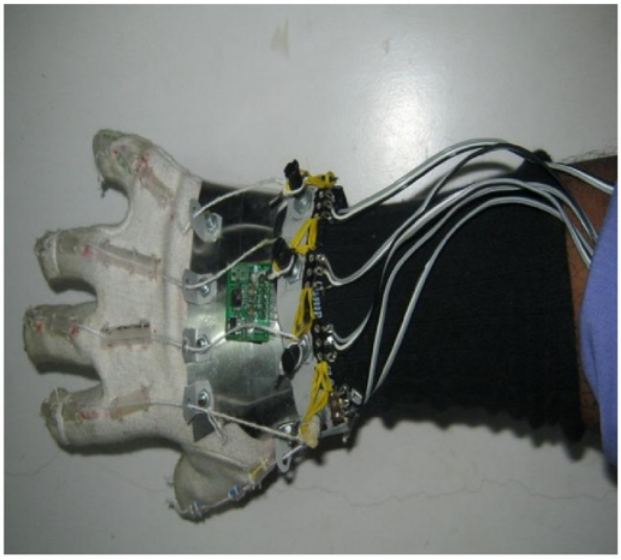

Figure 2.2: Glove housing six sensors [2]

In a separate project, a control method was developed for operation of a military mine detection and removal robot. The original control for this robot included joysticks, mechanical switches, and push buttons housed in a large heavy case. The new control method is a wireless glove fitted with motion sensors on the back of the hand and fingertips [3].

With the proven usefulness of gloves as a method of control, the nominated method of input will be a glove. The glove will house four sensors and the resulting electronics that are required for full functionality.

2.3 Sensors

Girovsk and Kundrt opted to use three unidirectional piezoresistive elements and a three axis accelerometer [1]. The glove that has been designed to accommodate for 6 sensors uses 5 potentiometers to control the flexion and extension of each finger. An accelerometer is used to control the orientation of the hand [2]. The wireless glove designed to control

the mine detection robot uses three accelerometers on the tips of the thumb, pointer and middle finger as well as an additional hand orientation motion sensor on the rear of the hand. Figure 2.3 shows these sensors.

Figure 2.3: Accelerometers on tips of fingers and rear of hand [3]

As the robotic arm for this project only has one gripping motion, this can be controlled with one piezoresistive element as in [1]. The orientation of joints three, and four will be controlled with two separate accelerometers. An additional gyroscopic sensor will control the orientation of joint one as in [3].

2.4 Software and Communications Protocol

The first mentioned project created the control software in in C-mex for matlab. The three piezoresistive sensors, and all three axes of the accelerometer are measured and a UART signal is sent to the Dynamixel motors in the robotic atm after each measurement [1]. The use of UART for Dyanmixel will be interchanged with (Serial Peripheral Interface) SPI as well as the control software being created in Arduino as opposed to matlab. The SPI interface is superior to UART as the UART only operates between 0.3-1Mbps while SPI can operate at up to 50Mbps. Arduino has been nominated for its simplicity with implementing new sensors.

2.5 Cost

The cost of production of these systems can be consolidated and and made cheaper by implementing a few changes. In [2] 6 sensors are being used. the proposed design for the current prototype only uses four sensors. In the design for [3] the same amount of sensors are being used however the system is wireless as well. This can be made more cost

effective by using piezoresistive sensors on the tips of fingers instead of accelerometers and removing the wireless capability of the system.

Chapter 3

Approach and Methodology

3.1 Introduction

This chapter is organised as follows. Section 3.2 details information regarding the robotic arm, accelerometer, flex sensor, and gyroscope implementation. In Section 3.3, the software components are described and a medium level description of the control code is given.

3.2 Hardware

3.2.1 Robotic Arm

The Robotic arm is actuated using 7 Dynamixel AX12-A servo motors from Robotis. These high performance motors communicate with each other via daisy chain method and can be programmed directly via Arduino with the Digital Servo Driver Shield or through matlab via USB2dynamixel. Within the compact frame of the dynamixel motor is a gear reducer, built in encoder with up to 1024 steps of precision, and a DC motor. Angular position, angular velocity, and load torque feedback is available to read from each motor. Ax-12a motors have two modes: joint, and wheel mode. Joint mode will be used for joints 2, 3, 4, and 5 in the robotic arm, while joint 1 will operate in wheel mode. Each motor is built in with a red status LED to alert the user of any errors. Due to its high quality manufacturing standard, the dynamixel motor can produce a high amount of torque for its size, and is resilient to external forces, making this a perfect motor for the robotic arm [6]. Individual parameters of each motor can be changed using USB2dynamixel, and the software provided by Robotis: RoboPlus. This program allows the user to change settings of the Dynamixel motor such as toggling between joint mode or wheel mode, changing motor ID, baud rate, angle limit, max torque, max speed, etc. All motor settings can be changed in other programs, however RoboPlus is the simplest way to edit settings.

3.2.2 Dynamixel Shield

The Dynamixel shield is a circuit board compatible with the Arduino Uno layout. This shield supports dynamixel Ax-12a motors over SPI interface (Digital 10,11,12,13). It has a UART interface in the event of a requirement for deeper development. There are 7 servo connector points. Dimensions of the board are: 59x53mm [7].

3.2.3 Accelerometer

Two ADXL345 3-axes accelerometers are used to control the robotic arm. The ADXL345 accelerometer is a suitable choice of sensor as it is small, lightweight, thin measuring 3mm×5mm×1mm, and consumes low power. Measurements for this sensor are at a high 13-bit resolution and can sense up to 16g of force. Digital output data is accessed through the Inter-integrated Circuit (I2C) digital interface on Arduino to allow for more than one accelerometer to be used for control [8]. As there are two accelerometers, they will act as slaves to the Arduino master and will be addressed accordingly. There are two different address types for the ADXL345 accelerometer. The first sensor "DEVICE A" is addressed as (0x1D) and the second sensor "DEVICE B" is addressed as (0x53). Therefore allowing both sensors to be operated on the I2C interface.

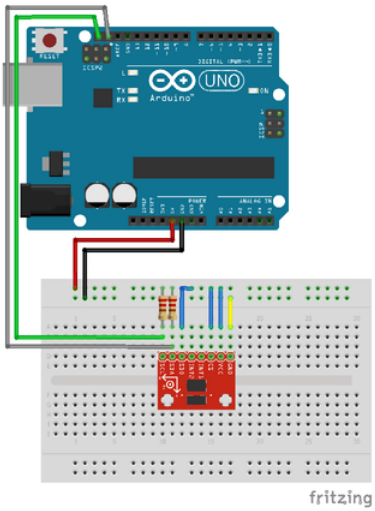

Figure 3.1: Accelerometer Circuit Diagram

3.2.4 Flex sensor

One 5.5cm flex sensor is used to actuate the gripper on joint 5 of the robotic arm. This flex sensor acts as a potentiometer when bent, increasing or decreasing resistance depending

on how far the sensor is flexed. The resistance value ranges between 10K ohm and 70K Ohm when flexed.

3.2.5 Gyroscope

One ITG-3200 gyroscope is used to control the base rotation. This gyroscope has three inbuilt 16 bit analogue-to-digital converters (ADC's) to read the three different rotational axes of movement. The ITG-3200 is operational between 2.6V and 3.6V, the I2C pins on the Arduino Uno howeve,r still transmit signal at 5V, so extended use without a logic level converter is not recommended. There is a VLOGIC pin (VIO) that acts as a reference for the logic signals coming in and out of the sensor. The VIO pin can be set to any voltage between 1.71V and the max input voltage VDD.

3.2.6 Implementation

Placement of sensors is crucial to gaining accurate control of the robotic arm. Both accelerometers are strategically positioned to best control the arm. The first accelerometer (ACC1) is fixed to the rear of the left hand on a 2.6 x 2.5 centimetre cut of vero board. ACC1 controls wrist joints 3 and 4 (motors 4, 5 and 6) on the arm. The X-Axis on the accelerometer is assigned to control wrist articulation while the Y-Axis controls the wrist rotation. These joints can be seen in Figure 1.1. The second accelerometer (ACC2), along with the Arduino board and gyroscope, is fixed to the rear of the wrist on two separate sections of veroboard. One section houses circuits for ACC2, and the gyroscope, and the second cut of vero board hold the Arduino and Dynamixel shield. The X-Axis on ACC2 controls the elbow articulation in joint 1 (motors 2, and 3). Initially, the veroboard was fixed to the glove with double sided tape. This method proved to be ineffective, as increased movement of the glove caused the boards to become loose or completely detached. Hot glue is now used to firmly fix the boards to the glove and acts as an adequate mitigation to the detachment issue.

ADXL345 accelerometers have 5 pins: ground (GND), voltage (VCC), chip select (CS), interrupt 1 (Int1), interrupt 2 (Int2), serial data out (SDO), serial data (SDA), and serial clock (SCL). GND, and VCC are connected respectively to the ground and 5v pins on the Arduino. CS is pulled up to HIGH to enable I2C mode. SDO controls the address type: 0x1D or 0x53 depending on if SDO is set to HIGH or LOW. SDA sends and receives information from the master and can be connected to either the SDA pin or analogue pin A4 on the Arduino. SCL receives the clocking signal from the master, and can be connected to either the SCL or analogue pin [8]. The SDA and SCL pins are both pulled up to HIGH through 5K6 resistors. ACC1 connects its SCL and SDA pins in parallel back to the ACC2 board and then into the Arduino. ACC1's SDO pin is set to HIGH and ACC2's SDO pin is LOW conversely. By connecting the SCL and SDA pins in parallel, this reduces the number of wires needed to connect to the Arduino. Only 4 wires are needed to connect both accelerometers to the Arduino.

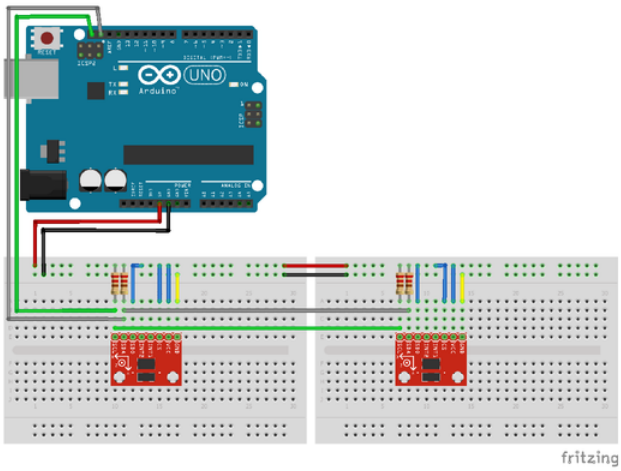

Figure 3.2: Two Accelerometer Circuit diagram

The flex sensor's position was originally set to be placed along the index finger on the inside of the left hand. This however has been revised and the sensor is now located on the rear of the middle finger. The connector pins are insulated with hot glue and fixed to the glove above the first knuckle of the middle finger. A little further along the sensor, hot glue is applied to fix the sensor to the glove above the base segment of the finger. The end of the flex sensor is enclosed in a housing above the middle segment of the finger. This allows free horizontal movement of the sensor. This housing is necessary due to the flex sensor being unable to stretch along the rear of the second knuckle. This housing allows the flex sensor to flex freely without being damaged. The first pin of the flex sensor is connected to the 5V pin of the Arduino. The second pin of the flex sensor is connected to ground through a 47K ohm resistor. The Sensor is connected to the Arduino as a voltage divider via any of the analogue pins as seen in Figure 3.3.

$$Vout = Vin * \frac{R1}{R1 + R2}$$

The flex sensor is printed on one side with a polymer ink that has conductive particles embedded in it. When the sensor is in its resting position, a resistance of roughly 30K Ohms can be read from the sensor due to the arrangement of the particles. When the sensor is curved away from the ink, conduction is decreased as the particles move away from each other, thus increasing the resistance. The opposite effect occurs when the sensor is bent towards the ink, the particles are closer together therefore, decreasing the resistance [9].

The ITG-3200 gyroscope as apposed to the ADXL345, measures rotational acceleration. This sensor is positioned on the rear of the wrist along with ACC2 to control the

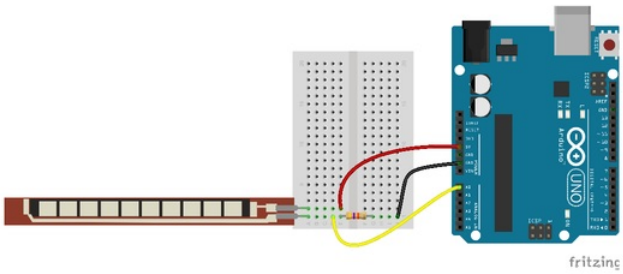

Figure 3.3: Flex Sensor Circuit Diagram

joint 1 base rotation. Figure 3.4 shows the circuit board of the ITG-3200, ACC1 and the Arduino.

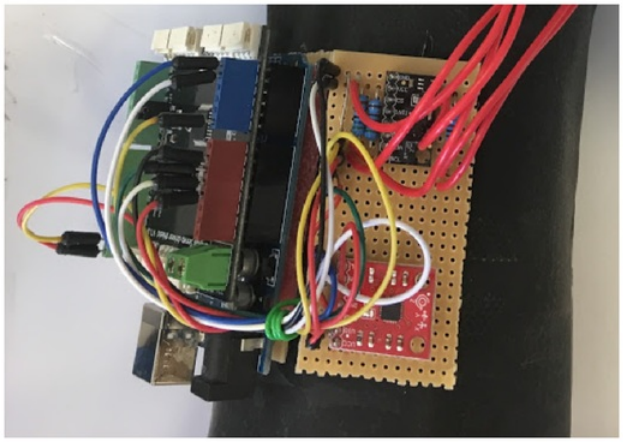

Figure 3.4: Arduino, Accelerometer 2, and Gyroscope Circuit Board

Because the ITG-3200 has a max voltage of 3.6V, the VCC pin on the sensor is connected to the 3.5V pin on the Arduino. VIO is connected straight directly to VCC as the max input voltage will be used as the logic reference [**?**]. The GND pin is connected to the ground rail for the flex sensor and then sent to the second ground pin in the shield. The SDA and SCL pins are connected to analogue pins 4 and 5 (A4 and A5) respectively. This can be seen in Figure 3.5.

The ITG-3200 measures rotational acceleration and as such, is positioned at the furthest point from the wrist. This position aids with gyroscope measurements and accuracy as it is closer to the point of rotation at the elbow.

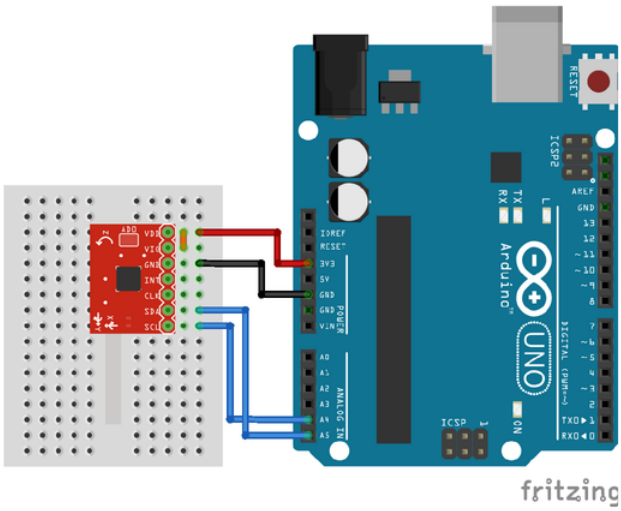

Figure 3.5: Gyroscope Circuit Diagram

3.3 Software

3.3.1 RoboPlus

Before coding can begin on the Arduino, each servo must be calibrated to the correct servo number and baud rate. This is done using the Dynamixel wizard within RoboPlus. RoboPlus is a software envoronment created by Robotis that contains the necessary software for controlling all Robotis products. Within the Roboplus software environment is the Dynamixel wizard. This section of RoboPlus allows firmware settings to be changed on the Dynamixel motors. The baud rate for each motor is set to 1000000 and number addresses are assigned to each motor. Motor 1 controls the base plate rotation, motors 2 and 3 control the elbow articulation, motors 4 and 5 control the wrist articulation, motor 6 controls the wrist rotation, and motor 7 controls the gripper movement. Each motor has a choice between joint mode and wheel mode Once the motors are calibrated, they can respond accordingly to commands from the arduino.

3.3.2 I2C

Both the Accelerometers and the Gyroscope operate on the I2C protocol. This communication protocol is selected as multiple slave devices can be controlled from the one master device. Alternately, SPI was considered as the communication protocol however as each additional device requires a new chip select I/O pin, I2C became the more favourable method of communication [10].

3.3.3 SPI

The Dynamixel Ax-12a motors are controlled on the SPI bus through a dynamixel digital device driver shield. The 7 dynamixel motors are daisy chained together and are connected to the dynamixel shield.

3.3.4 Software Architecture

The following is a description of the command blocks within the Arduino code. Firstly, global variables are declared and libraries are included. This section is divided in to initialisation for the accelerometers, motors, flex sensor, and gyroscope. Under the accelerometer section, Wire.h is included for I2C protocol and accelerometers are assigned addresses: 0x1D for ACC1 and 0x53 for ACC2. X, y, and z variables are set for both accelerometers and the register address for the "axis-acceleration-data" is stated. Two functions; writeTo, and readFrom are created to send data to and from the accelerometers: writeTo starts communication to the device, which then sends the register address, sends the value to be written, and then ends communication. readFrom starts communication, sends the address to be read from, requests the data from the specified address, and then ends communication. When reading data from the module, each axis is comprised of two bits of data, the data is broken down to three axes and saved to the variables x, y, and z. Libraries required for Dynamixel motors are now included. Dynamixel motors are controlled through SPI bus with the aid of a digital servo driver shield, therfore SPI.h in included. ServoCds55.h is the library that contains the commands necessary to control the Dynamixel motors and pins-arduino.h defines pin functions. Setup for the flex sensor needs only to define three global variables for effective use. The Gyroscope requires addresses for the device address, and x, y, z output address to be defined as characters so they can be sent to the sensor. Three functions are created to read the x, y, and z values of the gyroscope.

The setup function is now run. The I2C serial bus is joined and the baud rate is set to 1000000. Both Accelerometers are turned on by sending the value "24" to address: 0x2D. For the Dynamixel motors the SPI bus is joined and the SPI clock is set to one-eigth of the program clock speed. Servo select (SS) is set to high and the gyroscope is setup by sending the address to the device.

Within the loop function, x, y, and z values for the gyroscope are read first. Next, acceleration data from ACC1 and ACC2 are read using the readFrom function mentioned earlier. Each of the acceleration data variables need to be scaled in order to be understood by the motors. The accelerometers measure values and send them to the Arduino within a range of -260 and 260. Each motor has a range of 0-300 when in joint mode. This range is subject to change depending on where the motors are placed in the robotic arm. The maximum and minimum range values for each motor are shown in table 3.1. The x-axis on ACC2 controls elbow articulation and is scaled to a value between 80 and 260. The

x-axis on ACC1 controls wrist articulation and is scaled with respect to ACC2's x-axis so that the robotic arm follows the movements of the glove correctly. ACC1's y-axis controls wrist rotation and is scaled to output a value between 0 and 300. Figure 3.6 shows the co-ordinate axis of ACC1 on the hand. Base rotation is controlled by either the gyroscopes z or y axis. An if statement decides which axis to send to the motors depending on the orientation of the glove. If the glove is supinated, the y-axis values are sent, and if the glove is pronated, z-axis values are sent. Data is sent to the glove with the myservo.rotate command. This command takes two input arguments: the motor number and the speed of rotation. Speed of rotation can be either a positive or negative number depending on the direction of rotation. Motor velocity is fixed for the rest of the motors using the setVelocity command. Motor velocity can be anywhere between 0 and 300. The elbow, and wrist joints have a fixed velocity of 20 and the gripper has a velocity of 30. Here, a buffer smoothing method is used to stabilise the robotic arm. Even when stationary, the accelerometers give fluctuating signals, resulting in the robot becoming shaky when it should be completely stable. This buffer method is simply an if statement that checks to see if the signal has fluctuated above or below a given threshold. If the signal breaches the threshold, it is then allowed to be sent to the motor. Signals are sent to the motors using the myservo.write command. Myservo.write takes two input arguments: servo number and position. The flex sensor signal data is mapped in the same way to the accelerometers and gyroscope. The given range for the gripper motion is from 150-250. Buffer smoothing is also used to help stabilise the flex sensor signal.

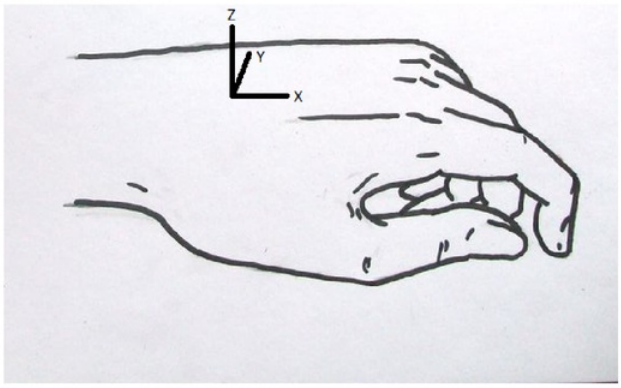

Figure 3.6: Co-ordinate Axis of ACC1 on Rear of Left Hand

Finally, two functions are created for the gyroscope to call on. The first function itgWrite, which initiates communication with the gyroscope and sets the register address. The second function itgRead, asks the gyroscope for data, waits for a response from the device, and returns it so that the data can be used.

Motor Number	Minimum Position	Maximum Position	Default Position
1	0	300	55
2, 3	80	260	260
4, 5	40	260	40
6	0	300	150
7	150	205	160

Table 3.1: Min and Max Ranges, Default Positions

Chapter 4

Experimental Procedures

Introduction In this chapter,a description of experimental procedures that were preformed to produce is given. These experiments are designed to produce results regarding sensor signals and error analysis.

4.1 Accelerometer offset test

This experiment is designed to gather data regarding the robotic arms response to signals sent by sensors on the glove. An accelerometer sensor will be fixed to different parts of the robot. This test sensor will be compared to the corresponding sensor on the glove.

A separate accelerometer testing circuit is constructed using a Sparkfun ADLXL345 accelerometer and an Arduino Uno. This circuit is fixed to the top of the wrist rotation joint (joint 4), as seen in Figure 4.1.

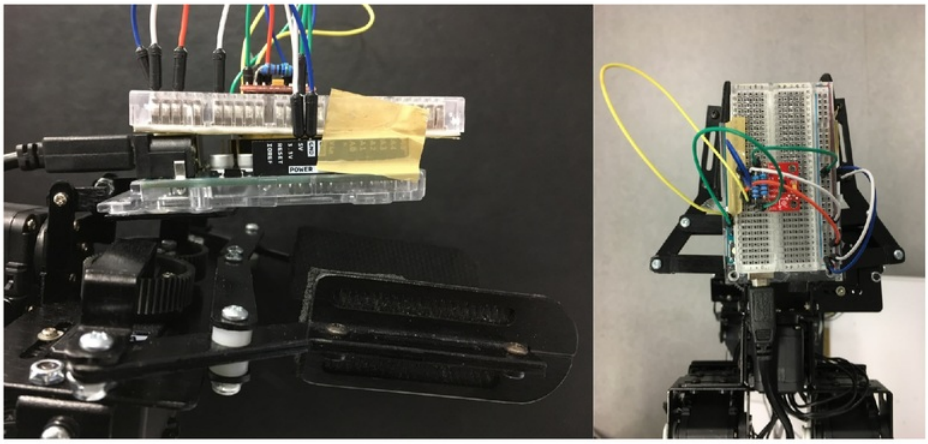

Figure 4.1: Placement of Accelerometer Testing Circuit

The accelerometers must now be calibrated. This is done by opening the serial monitor on Arduino for each circuit and matching the outputs. Firstly, the glove accelerometer is positioned in a natural horizontal position and the output is measured. Next, the test accelerometer is adjusted until its value is the same as the glove accelerometer. The accelerometer signals are scaled to the ACC1 x-axis scale, and sent to the serial plotter with buffer smoothing. Figure 4.2 shows the calibrated outputs of both accelerometers.

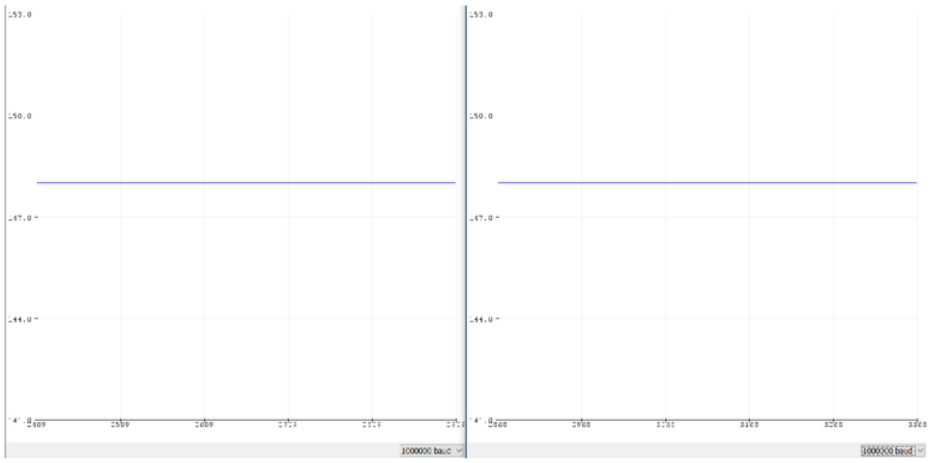

Figure 4.2: Glove Accelerometer (Left) and Test Accelerometer (Right) Calibrated to Show Same Static Output

This process is repeated when testing the elbow joint. Once the calibration is complete, movements are made with the glove sensor and compared with the test sensor. These results are detailed in the next chapter.

Chapter 5

Results and Discussion

5.1 Introduction

In this chapter, the final developmental prototype of the input device is exhibited. An analysis of results from experimental procedures in chapter 4 is also displayed and discussed. These results are gathered from sensor data on the glove and robot.

5.2 Glove

When the development of the software was complete, the glove was assembled. Figure 5.1 shows the final product of all the hardware elements.

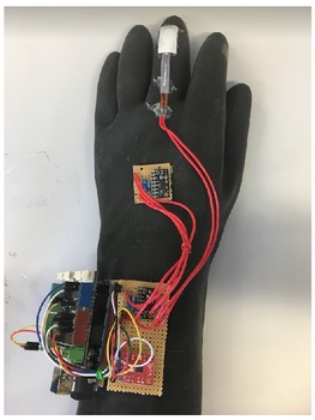

Figure 5.1: Final Glove Design

The four sensors can be clearly seen on the glove. Flex sensor positioned on the middle finger with the aforementioned housing (Figure 5.2), ACC1 on the rear of the palm (Figure

5.3), ACC2, and gyroscope on the end of the forearm (Figure 5.4)

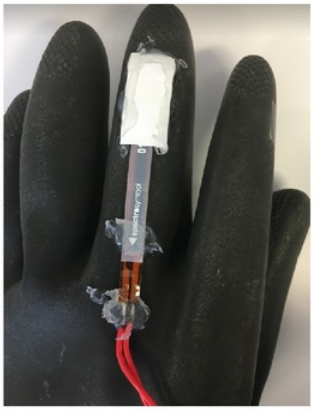

Figure 5.2: Flex Sensor With Housing

Figure 5.2 shows the flex sensor with the insulated connections, and second connection point. It also shows the housing on the tip of the sensor. This housing allows the sensor to bend along with the finger without being fixed to the glove.

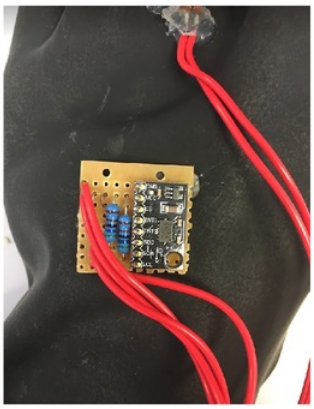

Figure 5.3: Accelerometer Circuit

Figure 5.3 shows the accelerometer circuit with pullup resistors and extension wires that connect back to the Arduino.

Figure 5.4 shows the circuit for ACC2, gyroscope and their connections back to the Arduino and Dynamixel shield.

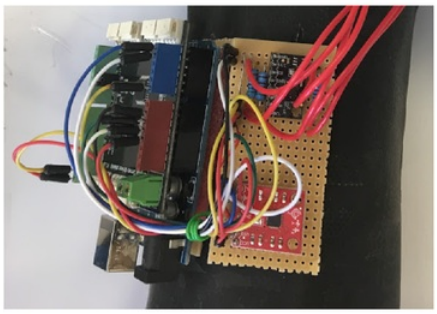

Figure 5.4: Accelerometer, Gyroscope, and Arduino on Glove

5.3 Arm and Glove Comparison

The following Figures 5.5, 5.6, and 5.7 show the robotic arm's repose to certain glove positions.

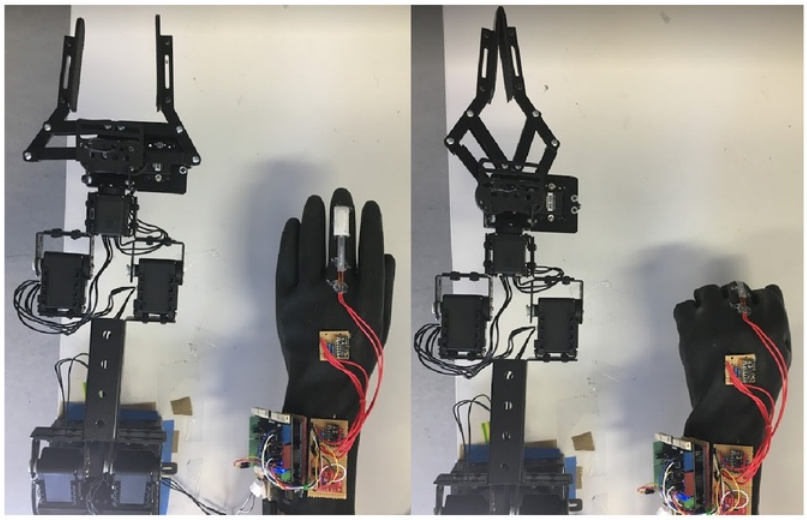

Figure 5.5: Open Hand vs Closed Hand Gripper Comparison

Figure 5.5 shows the response of the arm to the pronated hand being open and closed.

Figure 5.6 shows a side view of the hand at a roughly 30 degree angle in a neutral position and a pronated position. As you can see there is a slight difference in the angles between the hand and the arm. This will be discussed in section 5.4

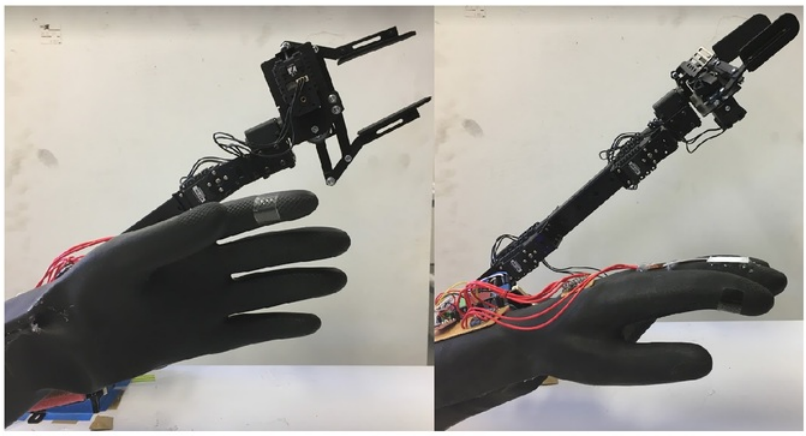

Figure 5.6: Neutral Hand vs Pronated Hand Comparison

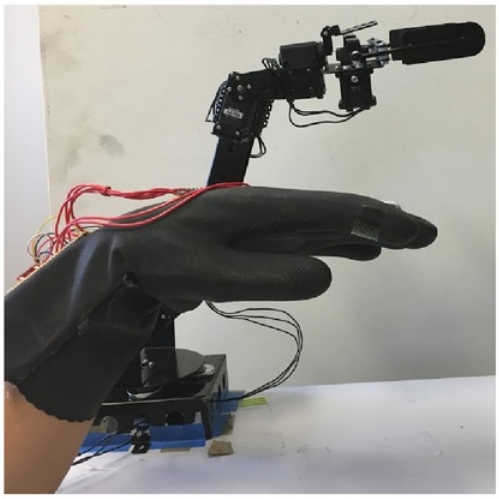

Figure 5.7: Bent Wrist, Robotic Arm Response

Finally in Figure 5.7, the hand is bent at the wrist to show the robotic arm position when ACC1 and ACC2 are at differing angles

5.4 Accuracy testing

The movement being tested is an upward extension, and then a return to neutral position of the wrist (Figure 5.8). This movement uses motors 4, and 5 of the robotic arm.

Figure 5.8: Wrist Extension Movement

Figure 5.9 shows the signal output from the glove accelerometer and the test accelerometer. Taking the data points and plotting them in Excel gives a better visualisation of the signals for us to compare. Figure 5.10 shows the comparison in an excel graph.

5.4.1 Total Settling time

There are a few things that can be discussed with this data. The delayed rise time to arrive to the desired angle is due to the low motor speed setting. The Dynamixel motors have a no load speed of 59rpm. Within Arduino, this speed is broken up into a range of 0-300 that can be sent to the motors with the myservo.write command. The current speed setting for the wrist and elbow movement is 20. This translates to a no load speed of 3.93rpm. If the motors are programmed to faster settings, the desired position is reached earlier however due to the characteristics of the robotic arm, the whole arm shakes dramatically leading to an increase in settling time of the robot. The total time for motors 4 and 5 to reach the extended position is 8.7 seconds and the total time to settle back to the neutral position is 10.2 seconds. Figure 5.12 Shows the test accelerometer data for a rotation movement from pronation to neutral as seen in Figure 5.11. The total time taken to reach the neutral position is 6.4 seconds, and the time to return to the pronated position is 7.3 seconds. This movement is more efficient than the wrist extension because there is less inertia involved in this rotation movement compared to the extension movement. Also, the wrist extension and flexion movements are controlled by dual motors. If there is any slight delay in signal between the motors, settling time will be increased as opposed to the single motor rotation where this possible problem is completely eliminated.

5.4.2 Overshoot

The sections of the data that overshoot the desired position can provide some interesting information. The Dynamixel motors themselves are quite accurate and have negligible

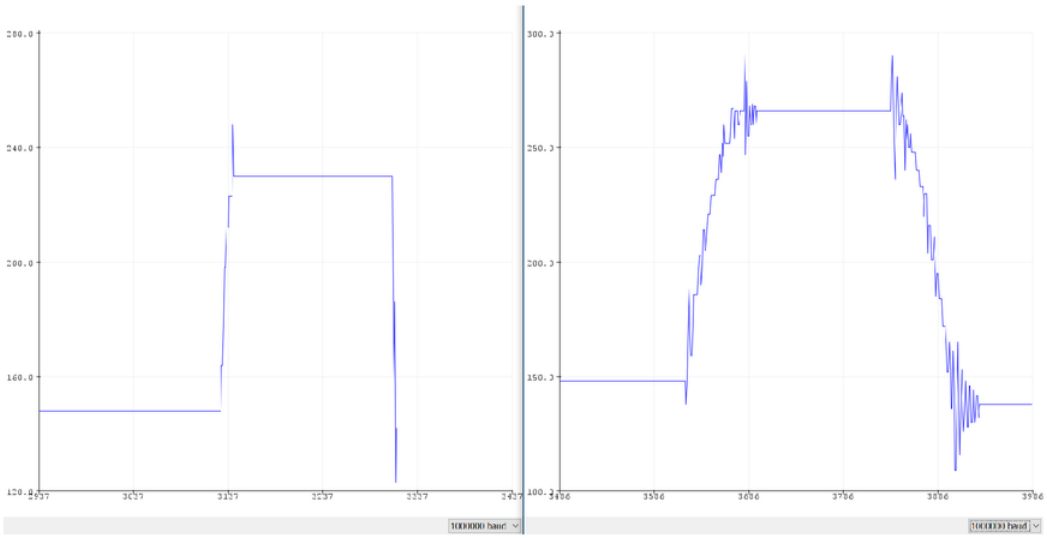

Figure 5.9: Glove Accelerometer (Left) and Test Accelerometer (Right) Showing Signals for Extension Movement (x-axis = miliseconds, y-axis = degrees)

overshoot. With this knowledge it can be deduced that any instability of the robotic arm is attributed to the looseness of some of the joints in the arm. The overshoot can be calculated for the tested movements. These overshoot errors are shown in Table 5.1

Movement	Max Overshoot Error (deg)	Final Steady State Error (deg)
Wrist Extension	23	7
Wrist Neutral	22	4
Wrist Pronation	25	19
Wrist Neutral	14	4

Table 5.1: Error Calculation

The final resting position of the arm after the extension movement shows that there is only a difference of 7 degrees between the glove and test sensor. Compared to the rotation movement on the arm, this is a fairly accurate response to the signal. Figure 5.12 shows the increased difference between glove sensor position and test sensor position. The reason there is such a large difference in steady state positions is due to accelerometer signal scaling. In future, these motor movements can be tweaked in Arduino to match each other more accurately.

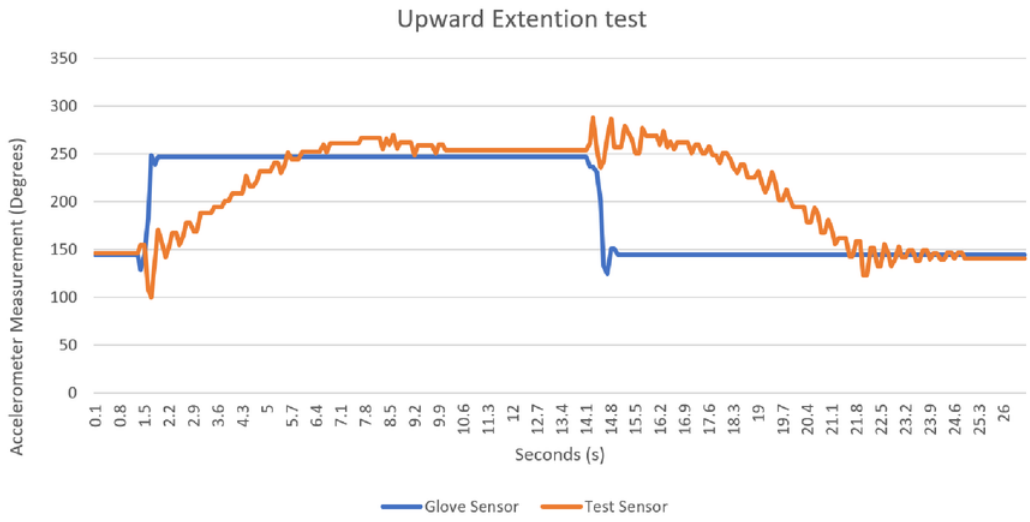

Figure 5.10: Improved Signal Comparison for Extension Movement

5.5 Base Rotation

Rotation for the base of the robot was a major challenge in the delivery of this project. Initially it was attempted to rotate the robot with only the accelerometers that have already implemented. Figure 5.13 shows the movement that will rotate the robot.

To use this motion to control the robot rotation, the possibility of using the y-axis of ACC1 to send acceleration information to the robots was explored. Figure 5.14 shows the signal for a right internal rotation and a left external rotation.

It can be seen that the acceleration signal data for an internal rotation dips first and then peaks before returning to a constant acceleration value. The reverse is seen for the external rotation. This signal is not suitable to be sent to the arm with the myservo.write command because as you can see, once the motor moves one direction, it will then move the opposite direction and return back to its original position. Even with the myservo.rotate command, the use of this method of control will not accomplish the desired result.

With the failure to control rotation with existing sensors, research was performed to find another method of control. This lead to the discovery of the TIG-3200 gyroscope. This gyroscope measures rotation at a point. Figure 5.15 shows the data signal for the same right internal and left external rotation that was performed in Figure 5.14. This signal is much cleaner and requires less signal processing to achieve the same result.

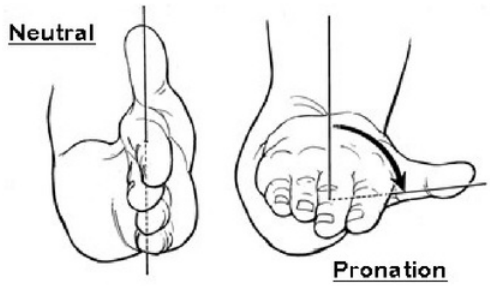

Figure 5.11: Pronation to Neutral Movement

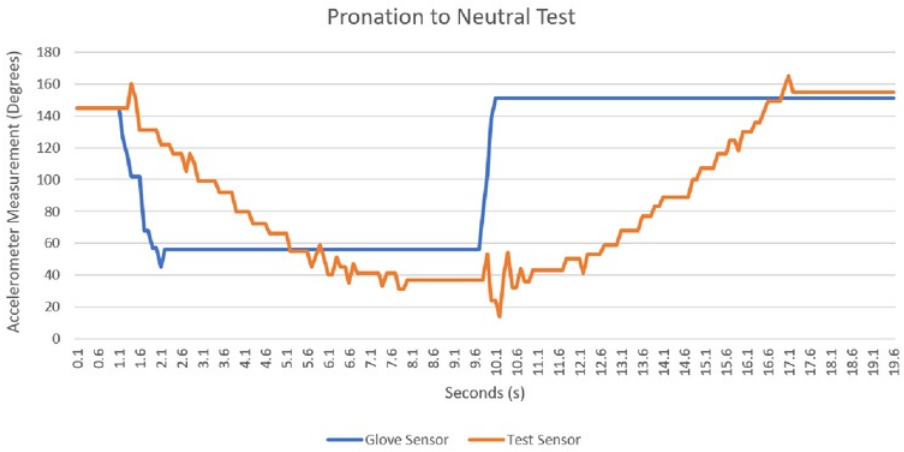

Figure 5.12: Signal Comparison for Pronation to Neutral Test

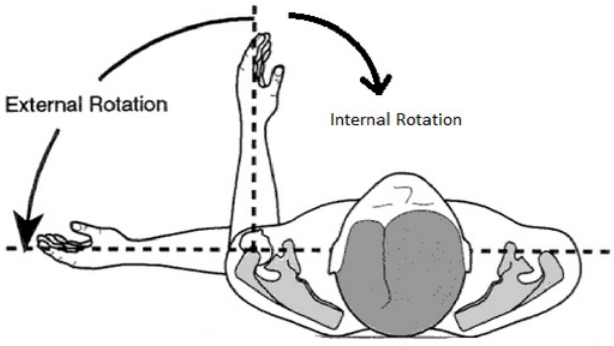

Figure 5.13: External Elbow Rotation

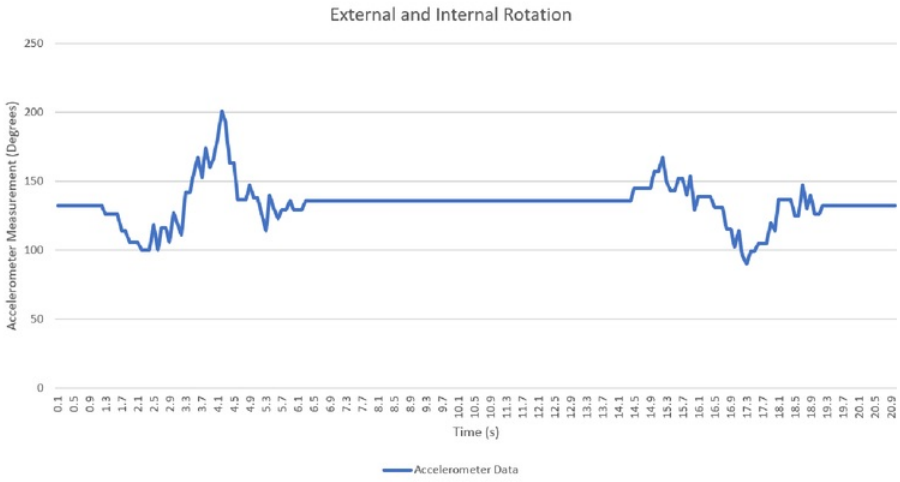

Figure 5.14: Accelerometer Data of Right Internal and Left External Rotation

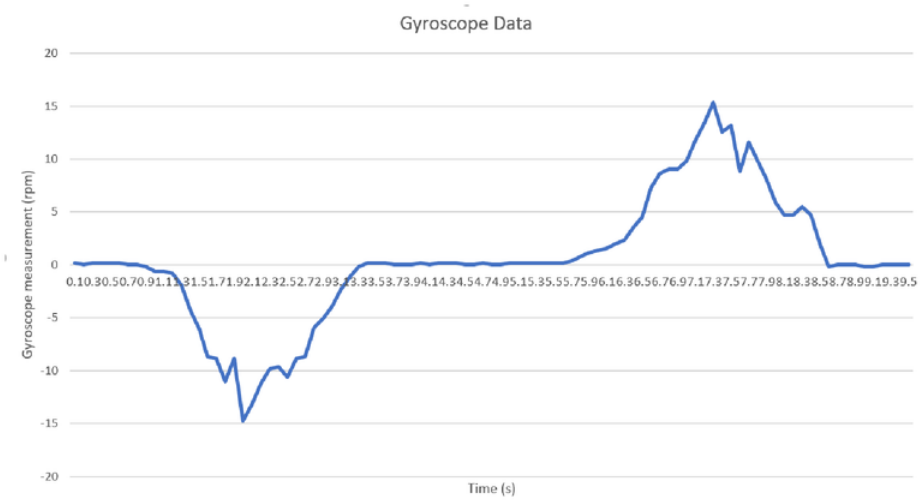

Figure 5.15: Gyroscope Data of Right Internal and Left External Rotation

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The aim of this project was to create a fully functional prototype to interpret hand gestures in such a way that a 5 degree of freedom robotic arm can be controlled. With a review of current literature, a knowledge base was created to start this project. The design stage commenced and prototype development produced the final product. This goal was achieved and is now set to be a starting platform for many future projects. With various improvements to the system, this project can be implemented into industry for many applications. The concepts of this project can be extrapolated to many areas such as medical, military, nuclear and others.

6.2 Future Work

Signals recieved from accelerometers are converted from raw signal data to a range between 0 and 300 degrees as this is the accpeted range of movement for the Dynamixel motors while in joint mode. Simply mapping raw data to the new range is not enough, the raw accelerometer data must be smoothed. Figure 6.1 shows the x and y axis signals from ACC1 while the accelerometer is stationary. (All figures referenced in this chapter have x-axis in miliseconds and y-axis in degrees)

As you can see, this signal has a small amount of noise which the motors respond to. This noise causes the motors to jitter and move sporadically, therefore causing the system to become unstable and decrease the accuracy of the robotic arm. Currently, this is mitigated by including a buffer of 10 degrees before sending the signal to the motor. Figure 6.2 shows the x and y axis signals of ACC1 with the buffer smoothing while the sensor is stationary.

The signal is completely smooth with this method of smoothing however when performing any type of movement, the the signal is still quite jerky and can sometimes be

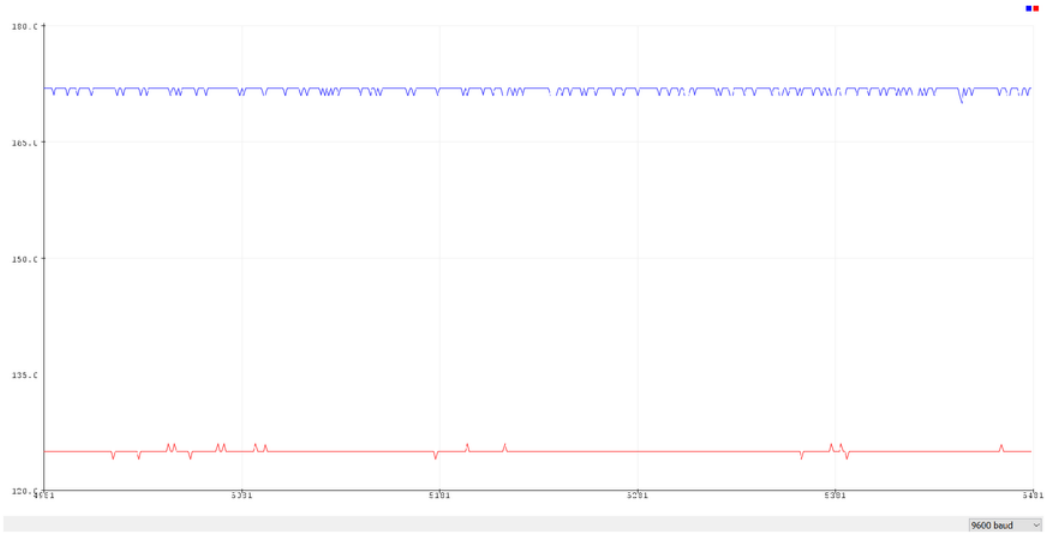

Figure 6.1: Signal of Raw Accelerometer Data While ACC1 is Static

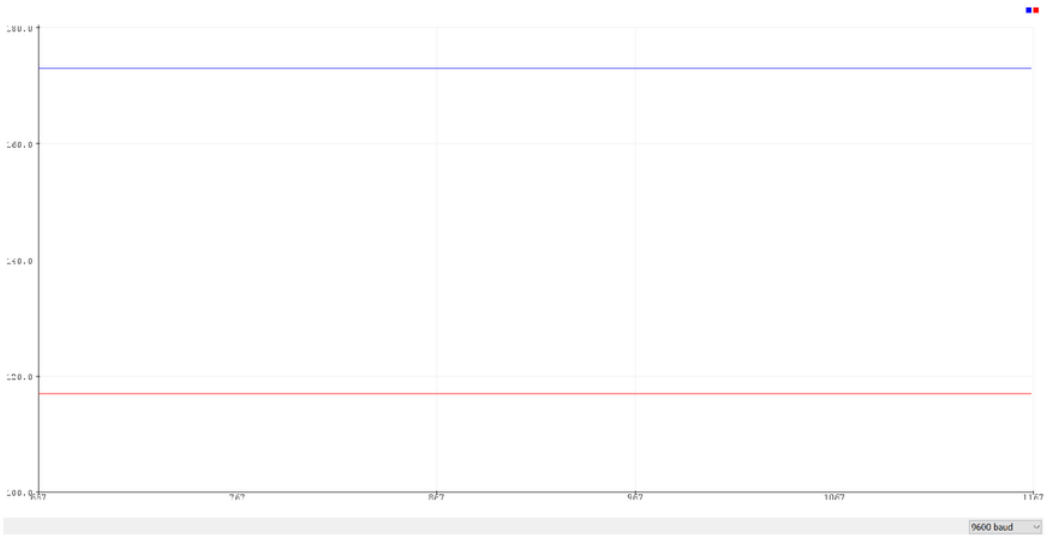

Figure 6.2: Signal of Accelerometer Data With Buffer Smoothing While ACC1 is Static

inaccurate due to the 10 degree buffer. Figure 6.3 shows the signals of a pronation-to-neutral and an extention movement of the left hand (after each movement, the hand returns to a resting position).

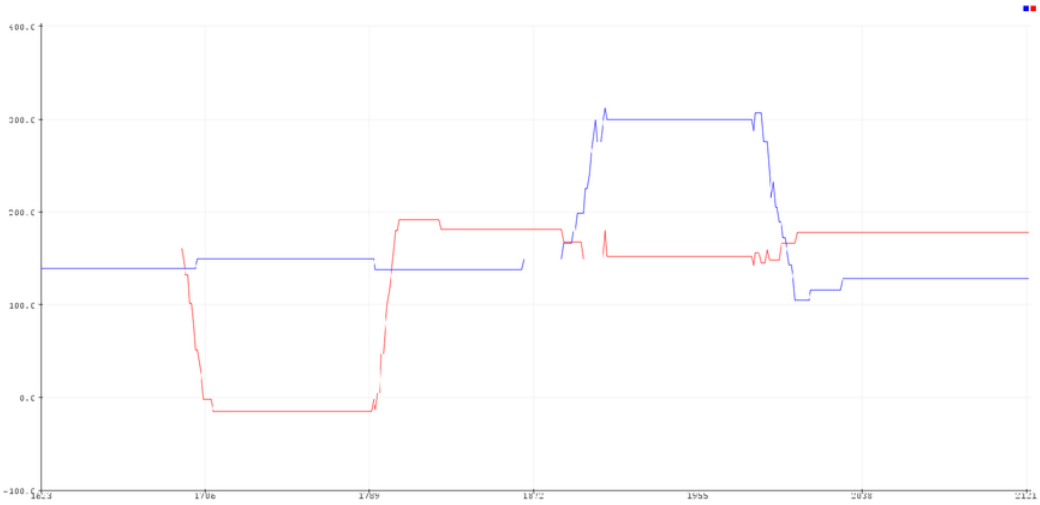

Figure 6.3: Signal of Accelerometer Data With Buffer Smoothing While ACC1 is Moving

The signal of dynamic movement with buffer smoothing is only slightly smoother than the raw signal shown in figure 6.4. It becomes apparent that it is necessary to smooth the signals further, therefore a method window averaging was developed. This method of signal processing reads the last 10 values of the input signal and averages them, therefore outputting a smooth signal. Its effects on static and dynamic signals are shown in figures 6.5 and 6.6.

The way this method works is by sending each variable through a smoothing function and using the returned integer as the signal for the motors. In theory this method of smoothing would greatly improve the stability of the system. However, due to the nature of the software required to use this method of smoothing, many problems were encountered when attempting to complete implementation. During testing, the arm was sluggish and it appeared variables being sent through the function were not being sent correctly.

This method would have been a great addition to the system however due to the many problems encountered, it will need to be implemented later in future work.

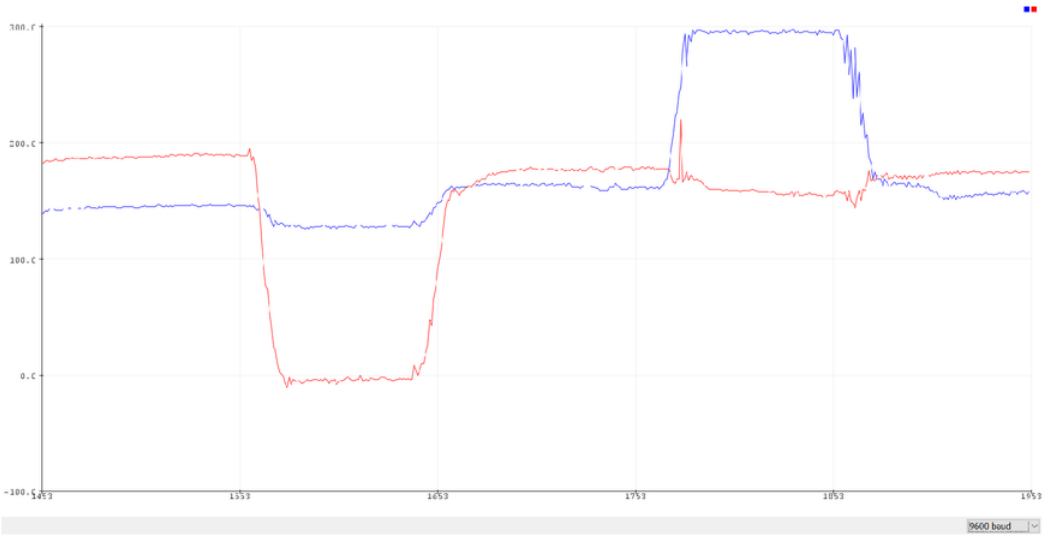

Figure 6.4: Signal of Raw Accelerometer Data While ACC1 is Moving

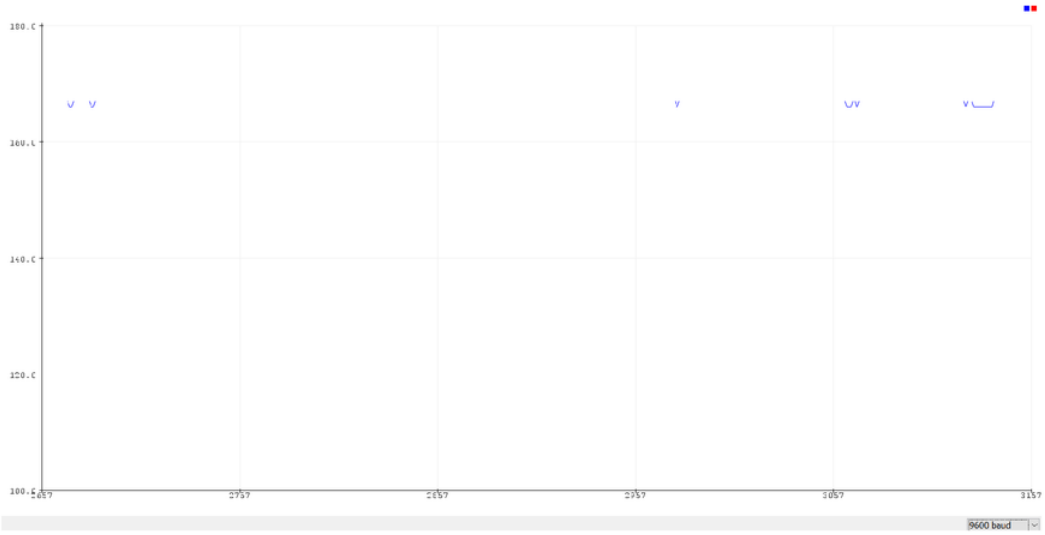

Figure 6.5: Signal of Accelerometer Data With Window Smoothing While ACC1 is Static

Figure 6.6: Signal of Raw Accelerometer Smoothing While ACC1 is Moving

Appendix A

Arduino Code

//accel include ¡Wire.h¿ define DEVICE-A (0x1D) //first ADXL345 device address define DEVICE-B (0x53) //second ADXL345 device address define TO-READ (6) //num of bytes we are going to read each time (two bytes for each axis)

byte buff[TO-READ] ; //6 bytes buffer for saving data read from the device char str[512]; //string buffer to transform data before sending it to the serial port char output[512];

void writeTo(int device, byte address, byte val) Wire.beginTransmission(device); //start transmission to device Wire.write(address); // send register address Wire.write(val); // send value to write Wire.endTransmission(); //end transmission

void readFrom(int device, byte address, int num, byte buff[]) Wire.beginTransmission(device); //start transmission to device Wire.write(address); //sends address to read from Wire.endTransmission(); //end transmission

Wire.beginTransmission(device); //start transmission to device Wire.requestFrom(device, num); // request 6 bytes from device

int i = 0; while (Wire.available()) //device may send less than requested (abnormal) buff[i] = Wire.read(); // receive a byte i++; Wire.endTransmission(); //end transmission int regAddress = 0x32; //first axis-acceleration-data register on the ADXL345 int xa = 0, ya = 0, za = 0; int xb = 0, yb = 0, zb = 0;

//dynamix include ¡SPI.h¿ include ¡ServoCds55.h¿ include "pinsarduino.h" ServoCds55 myservo; int baseRot = 200; int xbValP = 0; int xaValP = 0; int yaValP = 0;

////flex const int analogInPin = A0; // Analog input pin that the potentiometer is attached to

int sensorValue = 0; // value read from the pot int flexSensor = 0; // value output to the PWM (analog out) int flexP = 0;

//Gyro char WHO-AM-I = 0x00; //Address of sensor char SMPLRT-DIV = 0x15; char DLPF-FS = 0x16; char GYRO-XOUT-H = 0x1D; char GYRO-XOUT-L = 0x1E; char GYRO-YOUT-H = 0x1F; char GYRO-YOUT-L = 0x20; char GYRO-ZOUT-H = 0x21; char GYRO-ZOUT-L = 0x22;

char DLPF-CFG-0 = (1 ¡¡ 0); char DLPF-CFG-1 = (1 ¡¡ 1); char DLPF-CFG-2 = (1 ¡¡ 2); char DLPF-FS-SEL-0 = (1 ¡¡ 3); char DLPFFS-SEL-1 = (1 ¡¡ 4);

char itgAddress = 0x69; int readX(void) int

```
data = 0; data = itgRead(itgAddress, GYRO-XOUT-H) ¡¡ 8; data —= itgRead(itgAddress, GYRO-XOUT-L); return data;
int readY(void) int data = 0; data = itgRead(itgAddress, GYRO-YOUT-H) ¡¡ 8; data —= itgRead(itgAddress, GYRO-YOUT-L); return data;
int readZ(void) int data = 0; data = itgRead(itgAddress, GYRO-ZOUT-H) ¡¡ 8; data —= itgRead(itgAddress, GYRO-ZOUT-L); return data;
void setup() Wire.begin(); // join i2c bus (address optional for master) Serial.begin(1000000); // start serial for output
//accel //Turning on the both ADXL345s writeTo(DEVICE-A, 0x2D, 24); writeTo(DEVICE-B, 0x2D, 24);
//dynamix //Dynamixel.begin(1000000); digitalWrite(SS, HIGH); SPI.begin (); SPI.setClockDivider(SPI-CLOCK-DIV8);
//gyro char id = 0; id = itgRead(itgAddress, 0x00); //Serial.print("ID: "); //Serial.println(id, HEX); itgWrite(itgAddress, DLPF-FS, (DLPF-FS-SEL-0 — DLPF-FS-SEL-1 — DLPF-CFG-0)); itgWrite(itgAddress, SMPLRT-DIV, 9);
void loop()
//Gyro int xRate, yRate, zRate; //Read the x,y and z output rates from the gyroscope. xRate = readX(); yRate = readY(); zRate = readZ();
//Accel readFrom(DEVICE-A, regAddress, TO-READ, buff); //read the acceleration data from the ADXL345 //each axis reading comes in 10 bit resolution, ie 2 bytes. Least Significat Byte first!! //thus we are converting both bytes in to one int xa = (((int)buff[1]) ¡¡ 8) — buff[0]; ya = (((int)buff[3]) ¡¡ 8) — buff[2]; za = (((int)buff[5]) ¡¡ 8) — buff[4];
readFrom(DEVICE-B, regAddress, TO-READ, buff); //read the acceleration data from the second ADXL345 xb = (((int)buff[1]) ¡¡ 8) — buff[0]; yb = (((int)buff[3]) ¡¡ 8) — buff[2]; zb = (((int)buff[5]) ¡¡ 8) — buff[4];
//interval scaling int xgVal = map(xRate, -10000, 10000, -1000, 1000); int zgVal = map(zRate, -10000, 10000, -1000, 1000); int xbVal = 300 - map(xb, -255, 255, 80, 260); int ybVal = 300 - map(yb, -255, 255, 0, 300); int xaVal = 300 - map(xa, -255, 255, xbVal - 90, xbVal + 150); int yaVal = 300 - map(ya, -400, 400, 0, 300);
//Base rotation with Gyro //gyro switches values when glove is supinated if (yaVal ¡ 100 —— yaVal ¿ 220) myservo.rotate(1, xgVal); else myservo.rotate(1, zgVal);
myservo.setVelocity(20); if ((xbVal ¿ xbValP + 5) —— (xbVal ¡ xbValP - 5)) myservo.write(2, xbVal); myservo.write(3, xbVal); xbValP = xbVal;
if ((xaVal ¿ xaValP + 5) —— (xaVal ¡ xaValP - 5)) myservo.write(4, xaVal); myservo.write(5, xaVal); xaValP = xaVal;
if ((yaVal ¿ yaValP + 10) —— (yaVal ¡ yaValP - 10)) myservo.write(6, yaVal); yaValP = yaVal;
//flex sensor stuff sensorValue = analogRead(analogInPin); flexSensor = map(sensorValue, 900, 700, 150, 250);
myservo.setVelocity(30); if ((flexSensor ¿ flexP + 20) —— (flexSensor ¡ flexP - 20)) myservo.write(7, flexSensor); flexP = flexSensor;
void itgWrite(char address, char registerAddress, char data) //Initiate a communication sequence with the desired i2c device Wire.beginTransmission(address); //Tell the I2C address which register we are writing to Wire.write(registerAddress); //Send the value to write to the specified register Wire.write(data); //End
```

```
the communication sequence Wire.endTransmission();
    unsigned char itgRead(char address, char registerAddress)
    //This variable will hold the contents read from the i2c device. unsigned char data = 0;
    //Send the register address to be read. Wire.beginTransmission(address);
    //Send the Register Address Wire.write(registerAddress);
    //End the communication sequence. Wire.endTransmission();
    //Ask the I2C device for data Wire.beginTransmission(address);
    Wire.requestFrom(address, 1);
    //Wait for a response from the I2C device if (Wire.available()) //Save the data sent from the I2C device data = Wire.read();
```

Bibliography

[1] P. Girovskỳ and M. Kundrát, "Robotic arm based dynamixel actuators controlled by the data glove," in *International Journal of Engineering Research in Africa*, vol. 18. Trans Tech Publ, 2015, pp. 152–158.

[2] A. Sharma, K. Lewis, V. Ansari, and V. Noronha, "Design and implementation of anthropomorphic robotic arm," *International Journal of Engineering Research and Applications*, vol. 4, no. 1, pp. 73–79, 2014.

[3] N. X. Tran, H. Phan, V. V. Dinh, J. Ellen, B. Berg, J. Lum, E. Alcantara, M. Bruch, M. G. Ceruti, C. Kao *et al.*, "Wireless data glove for gesture-based robotic control," in *International Conference on Human-Computer Interaction*. Springer, 2009, pp. 271–280.

[4] "Robotic control." *Mechanical Engineering*, vol. 135, no. 5, pp. 24 – 25, 2013. [Online]. Available: http://simsrad.net.ocs.mq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=iih&AN=87340579&site=ehost-live

[5] M. Georgescu, A. Sacccomandi, B. Baudron, and P. L. Arbeille, "Remote sonography in routine clinical practice between two isolated medical centers and the university hospital using a robotic arm: a 1-year study," *Telemedicine and e-Health*, vol. 22, no. 4, pp. 276–281, 2016.

[6] Robotis, "Dynamixel ax12-a user manual," 2006.

[7]

[8] A. Devices. Adxl345 digital accelerometer datasheet. [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf

[9] Sparkfun. Flex sensor hook up guide. [Online]. Available: https://learn.sparkfun.com/tutorials/flex-sensor-hookup-guide

[10]