Summer 2019

Improving 3D Printed Prosthetics with Sensors and Motors

Rachel Zarin
rzarin@eagles.bridgewater.edu

Improving 3D Printed Prosthetics with Sensors and Motors

Rachel Zarin

Summer II 2019

Honors Project

Advisor: Dr. Jason Ybarra

A. Introduction

Artificial limbs, often called prosthetics, are unique devices designed to help regain the functions of missing limbs. Worldwide, there are millions of individuals who have gone through at least one amputation, which can be all or only part of a limb, who would greatly benefit from owning prosthetics ("Limb Loss," 2019). The cost of one prosthetic leg can range from $5,000 to $50,000, while an arm is $3,000 to $30,000 (Turner, 2018). However, even the best and most expensive only last three to five years of usage. This means they may have to be replaced several times during one's life. Prosthetics are not a one-time cost, so thousands of dollars are needed for each replacement (Mohney, 2013). More replacements or adjustments may be necessary for other reasons, such as weight change. Since children are continuously growing, they need more frequent replacements, usually every two years or less until the age of 18. This leads to a much higher cost, which can range from $21,000 to $300,000 if the prosthetic is replaced six times. Unfortunately, health insurance rarely covers the full cost of children's prosthetics because of a limit on how much is allowed to be spent on devices (Sharington, 2017).

In addition, they must be specially made for every patient, which contributes to the high price. Glenn Garrison, director of prosthetics and orthotics at Hospital for Special Surgery in New York pointed out that the price greatly varies depending on what components one chooses to include. He also commented that "[t]hey're probably in line with a cost of a car" (Mohney, 2013). As seen, it is often very difficult for individuals and families to afford prosthetics. Although, individuals can apply for financial assistance, including nonprofit organizations, children's and young adult services, and state and federally funded sources (Amputee Coalition, 2019).

Currently, prosthetics are in very high demand. According to The American Board for Certification in Orthotics and Prosthetics, over 1 million limb amputations take place every year worldwide. To put it in a different perspective, an amputation occurs somewhere in the world about every 30 seconds (Frost & Sullivan, 2018). In Virginia alone, almost 1.5 million veterans sought prosthetics-related care in only one year (Veterans Health Administration, 2009). Besides these individuals, there are many other causes of limb amputation, meaning the numbers of those seeking care are much greater. Common reasons are cancer, diabetes, or birth defects. Due to rising rates of obesity in the United States, more people are developing diabetes and atherosclerosis. Therefore, the current statistic of 0.5% Americans having amputations is likely to continue to increase and consequently so will the need of prosthetic limbs (Baird, 2015). In 2005, about 541,000 Americans experienced limb loss of the upper extremity. This number is predicted to double by 2050 (Cordella et al., 2016).

Prosthetics are needed in order for amputees to perform daily activities independently and comfortably (Baird, 2015). This encompases actions such as walking, dressing, feeding, or gripping and manipulating other objects. Hands and fingers may be considered the greatest necessity due to their role in sensing environmental conditions and performing sophisticated movements and precision tasks. In some cases, people feel they need prosthetics to improve their appearance and confidence as well (Cordella et al., 2016).

Moving forward, the need of low cost and limited function devices will continue to grow for the majority who cannot afford the normal cost ones mentioned above. Similarly, new materials and applications are helpful for amputees in developing countries due to their low prices and high availability. Meanwhile, advanced new technologies can be incorporated to help better mimic a real limb. However, the need and demand for these devices are much less since they are mostly used for very active individuals and are more costly. Similar techniques are eventually able to be added to moderate-cost prosthetics. Still, there is a significant need to discover ways to fund widespread innovations (Marks & Michael, 2001).

This project aims to improve 3D printed prosthetics with simple sensors and motors to keep the cost minimal. I plan to conduct research and design circuits that include the sensors and motors. I will construct a 3D printed hand model to demonstrate how a tilt sensor combined with digital circuitry and servo motors can be used at the wrist joint. Similarly, another servo motor will control the opening and closing of the fingers by pulling their artificial tendons. Another sensor that will be incorporated is a force sensitive resistor, which will allow the person to detect the amount of pressure they are applying as indicated by an LED. All of these components will be powered by a battery pack, and everything will be self-contained on the hand and arm. One goal is to discover how combining digital circuitry and computer programming may be incorporated into low cost 3D printed prosthetics while keeping the price low and availability high. All materials are “off the shelf,” meaning they are available for purchase at places like Amazon.com and hardware stores at affordable prices. Adding these elements would help fine-tune hand movements and responses, as well as provide individuals with information from their environment. Also, I hope to create useful feedback mechanisms for each circuit component. In other words, the information received from the sensors will be processed and cause a specific response for the device and its user. These functions would otherwise not be possible for 3D printed prosthetics without the addition of circuit elements.

Lastly, this project faces a few limitations such as the supplies and equipment available for use due to the price constraint and sophistication of other options. Also, due to the structure and size of the equipment, there are only a handful of ways to use them in the design of the model. Likewise, since the model has restricted space, there is a limited amount of equipment that can be attached. Next, the time period allotted to complete the research limits how much progress can be accomplished and how many details can be included. There are always uncertainties as well when using digital electronics in circuits. The devices, in this case sensors, motors, and the Arduino board, may not be one hundred percent accurate when taking in or using information or during other operations. As a result, outputs, calculations, and feedback responses cannot be as precise. Similarly, these components are limited in their performance capabilities. For example, the motors are only able to pull a certain amount of force and they likely do not move as smoothly as more expensive and complex motors. Another instance is that the Arduino boards are only able to be programmed with one code at a time, so multiple functions cannot happen simultaneously unless they are combined within a single code. Furthermore, resistors are essential in some circuits, but their resistance can vary from what they are supposed to be. This means there is a possibility that resistors will affect other circuit components and slightly alter outputs, calculations, and feedback responses again. In addition, the codes used might limit these computations and responses due to any rounding or approximations done throughout the equations or other parts of the code. If these calculations are done by hand, they would likewise face a similar uncertainty because of rounding.

B. Equipment

B.1 Arduino UNO

Arduino is a method to integrate hardware and software for making interactive projects. Its main advantages are the low cost and strong support. The Arduino boards use inputs from sensors to take in environmental information and then create responses accordingly. The Arduino software uses its own coding language to communicate with the board to tell it what to do, which is based on Java programming language (Arduino, 2019). For this project, the Arduino Uno board will be used as well as various codes. It contains a microcontroller chip, which has a microprocessor with several features such as timers, counters, interrupts, and more. Also, the board includes 14 digital input/output pins and 6 analog inputs for assembly to any circuit. There is a USB connection for receiving power from a computer, or it can be powered by other external sources. The USB connection also allows the Arduino board to be programmed by writing and compiling the code on a computer and then uploading it to the board. This connection allows for serial communication with the computer as well while the Arduino is running. Next, the Arduino Uno has an on-board voltage regulator and can provide 5V and 3.5V to other electrical components. There are numerous other pins on the board, including Vin for supplying voltage, reset to reset the program that is running, IOREF for providing reference voltage to the board, Serial Peripheral Interface (SPI) communication pins, Analog Reference (AREF) to provide reference voltage to the analog inputs, Two-wire Interface (TWI) communication pins, serial communication pins, and pins to provide external interrupts through a low or changing value (Aqeel, 2018).

B.2 MEMS Gyroscope-Accelerometer (MPU-6050)

This 6-axis MPU-6050 chip includes a 3-axis accelerometer and a 3-axis gyroscope, meaning they obtain values in the x, y, and z directions. The accelerometer is used to measure non-gravitational acceleration, or change in velocity. It is designed to respond to the vibrations associated with these movements. (Goodrich, 2018). The structure of an accelerometer includes plates to measure capacitance, which changes when acceleration is applied. This capacitance change corresponds to acceleration values (Hamza, 2019). Moreover, it has precise and accurate motion tracking ability, so it can be very beneficial in various applications, such as fitness monitors, laptops, cars, and earthquake detectors. These are possible due to its small size, low power consumption, high accuracy and shock tolerance, and ability to be programmed (InvenSense, 2013).

Next, the gyroscope measures or maintains orientation and angular velocity based on Earth's gravity. It is able to measure the rate of rotation around an axis. Gyroscopes are most often used to help with stability in navigation (Goodrich, 2018). Both the accelerometer and gyroscope use three 16-bit analog-to-digital converters (ADCs) for digitizing their outputs. To precisely track slow and fast motions, both can also be programmed on their full-scale range; the gyroscope's is ±250, ±500, ±1000, and ±2000°/sec (dps) and the accelerometer's is ±2g, ±4g, ±8g, and ±16g. These correspond to their sensitivity; lower range means higher sensitivity, which obtains measurements more easily than low sensitivity (InvenSense, 2013).

Additionally, the chip has a Digital Motion Processor (DMP) to process complex algorithms based on the information from the accelerometer, gyroscope, and other sensors. To accomplish this 9-axis MotionFusion output, it has an I^2C sensor bus that directly accepts inputs from other external sensors. Furthermore, the chip contains a 1024 Byte FIFO buffer that reduces power consumption through allowing the system processor to read data in bursts and run in a low-power mode while continuing to collect information. Similarly, the MPU chip permits decreased processing requirements for the system processor. With the MotionFusion output, the DMP is able to minimize the need for constantly measuring the motion sensor output. Other features are a temperature sensor and an oscillator with ±1% variation over the operating temperature range (InvenSense, 2013).

B.3 Servo Motors

Servo motors allow for precise control of rotating or pushing an object, such as by specific angles or distances. One motor is composed of a DC motor, a gearbox, a potentiometer, and a control circuit. The DC motor has low torque and high speed, but the gear box increases torque and decreases speed (Apoorve, 2015). Usually, these motors operate at 2.5kg/cm torque, meaning the motor can pull 2.5kg when it is suspended 1cm away (Components101, 2017).

When coupled with a sensor, the motor is provided with information on how to adjust its position through electrical pulses. This means it uses positive feedback to control its motion and final position, and the motor is a closed-loop system. The feedback signal is produced by comparing the reference output and input signals, and the feedback signal is the input that controls the device. For example, the difference between the potentiometer's output signal and a signal from another source are processed in this feedback mechanism at the control circuit. The output is in the form of an error signal, which is the motor's input signal as well. This causes the motor to rotate and consequently the potentiometer too. As the potentiometer moves, a signal is generated; this output signal corresponds to the change in angular position of the potentiometer. The motor stops rotating when there is no difference between the potentiometer's signal and the external applied signal (Apoorve, 2015).

In addition, there are 3 wires coming from the servo motor: power (red), ground (brown), and signal (orange). Power is connected to +5V, ground is connected to the ground of the system, and signal is connected to, for instance, a digital output so that it can be used to drive the DC motor (Components101, 2017). Servo motors are controlled by pulse width modulation (PWM), meaning the rotation angle is determined by the pulse duration that is applied to its control pin. The shaft can turn 90 degrees in both directions from its neutral position. Pulses occur approximately every 20 milliseconds, or at a frequency of 50 Hz. If the pulses last 0.5-1 millisecond, the shaft will rotate to 180 degrees, and if they are 2-2.5 milliseconds, it will rotate to 0 degrees (Apoorve, 2015).

B.4 Force Sensitive Resistor

A force sensitive resistor (FSRs) is a sensor that detects pressure, squeezing, and weight. It is considered a resistor that alters its resistance based on the amount of force applied to it. The sensor is made of a semiconductor and electrodes. As pressure increases, more electrodes touch the semiconductor, increasing the conductance and decreasing the resistance. With no pressure, the sensor resembles an infinite resistor, or open circuit (Adafruit Learning System, 2019). For this project, the ADA 166 (Interlink 402) FSR will be used.

B.5 Micro Limit Switch

Micro limit switches are smaller than other switches, allowing them to be used in small spaces. They have an actuating plunger (a long straight lever arm in this project) that move a small amount to activate the contact sequence. The contact positions alternate when a spring-loaded mechanism causes movable contacts to snap from one position to the other. These contacts either break or form an electrical connection. When in the active position, it regulates the circuit that controls a machine and its moving parts (Thomas Publishing Company). One advantage to using a micro limit switch is that it changes state in response to a specific actuator trip, whereas contacts in other switches move from any actuator movement. Also, they have a fast switching speed, which reduces damage done to contact surfaces. Lastly, micro limit switches have reliable and predictable switching due to the speed and movement of the contacts being independent to the speed and movement of the actuator (TEMCo Industrial).

C. Methods

C.1 Arduino Programming Language

Arduino boards have the ability to interface with other boards, microcontrollers, and computers. The boards use an Integrated Development Environment (IDE), which can be programmed using Java C and C++ languages. The microcontroller within the board has a bootloader that allows code to be uploaded to the board from a computer without needing an external hardware programmer. Also provided with the IDE software is the serial monitor, which allows data to be sent from the board to a computer (Aqeel, 2018). The monitor can be opened in a separate window to view the output. Since it is an IDE, the editor, compiler, and burner are all integrated in the same software (Mirza, 2019). In addition, the software has libraries for users to access that include premade codes for carrying out certain tasks and communications.

C.2 Calculation of Pitch and Roll Angles

Pitch and roll angles help describe the orientation of an object with respect to a coordinate system. Orientation is accomplished through rotating about the axes. The pitch angle is defined by the rotation around the y axis, and the roll angle is defined by the rotation around the x axis. In order to obtain these values, the acceleration in the x, y, and z directions must be known. An accelerometer is a simple method to measure this acceleration. When paired with an Arduino, the raw values for acceleration are displayed on the serial monitor. Therefore, prior to calculating the angles, the raw values must be converted to their respective desired values in grams based on the accelerometer's acceleration limit. These limits have corresponding sensitivity scale factors that are used in the raw value conversions. In this project, the acceleration limit is 2 g and the sensitivity is 16,384 LSB/g. The equation for this required value is:

$$G_p = \frac{raw\ value}{sensitivity}$$

(Kaulics, 2015).

Furthermore, the pitch and roll angles have their own formulas that are derived through using an initial position and rotation matrices for each direction. These matrices depend on the order in which the rotations of the different directions are applied. Most commonly, the order xyz or yxz is used to determine the individual formulas because this allows the angle in the z direction, yaw angle, to be eliminated. Normalizing the accelerometer reading G_p can be related to the matrices and then used to solve for the pitch and roll angles in both the xyz and yxz orders. The resulting equations are listed below, where pitch angle is denoted by θ and roll angle is denoted by ϕ.

$$\theta_{xyz} = arctan\frac{-G_{px}}{\sqrt{G_{py}^2+G_{pz}^2}}$$

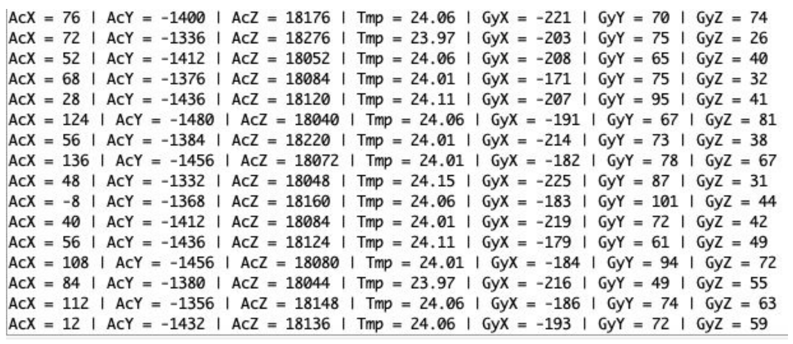
$$\phi_{xyz} = arctan(\frac{G_{py}}{G_{pz}})$$

$$\theta_{yxz} = arctan(\frac{-G_{px}}{G_{pz}})$$

$$\phi_{yxz} = arctan\frac{G_{py}}{\sqrt{G_{px}^2+G_{pz}^2}}$$

(Pedley, 2013).

There are also Arduino codes that calculate these angles and display them on the serial monitor after receiving data from an accelerometer chip. The one used in this project can be referenced to in the Appendix, and the equations within the code can be referenced to in section C.3. In the code I used for the final version of the model, the angles are calculated using information from the gyroscope. The equation for the angle of the motors combines both angle measurements from the accelerometer and gyroscope. The gyroscope angle is found using an update equation, which requires the starting angle, raw value, and elapsed time. The accelerometer angle is determined using the equations above.



```
AcX = 76 | AcY = -1400 | AcZ = 18176 | Tmp = 24.06 | GyX = -221 | GyY = 70 | GyZ = 74
AcX = 72 | AcY = -1336 | AcZ = 18276 | Tmp = 23.97 | GyX = -203 | GyY = 75 | GyZ = 26
AcX = 52 | AcY = -1412 | AcZ = 18052 | Tmp = 24.06 | GyX = -208 | GyY = 65 | GyZ = 40
AcX = 68 | AcY = -1376 | AcZ = 18084 | Tmp = 24.01 | GyX = -171 | GyY = 75 | GyZ = 32
AcX = 28 | AcY = -1436 | AcZ = 18120 | Tmp = 24.11 | GyX = -207 | GyY = 95 | GyZ = 41
AcX = 124 | AcY = -1480 | AcZ = 18040 | Tmp = 24.06 | GyX = -191 | GyY = 67 | GyZ = 81
AcX = 56 | AcY = -1384 | AcZ = 18220 | Tmp = 24.01 | GyX = -214 | GyY = 73 | GyZ = 38
AcX = 136 | AcY = -1456 | AcZ = 18072 | Tmp = 24.01 | GyX = -182 | GyY = 78 | GyZ = 67
AcX = 48 | AcY = -1332 | AcZ = 18048 | Tmp = 24.15 | GyX = -225 | GyY = 87 | GyZ = 31
AcX = -8 | AcY = -1368 | AcZ = 18160 | Tmp = 24.06 | GyX = -183 | GyY = 101 | GyZ = 44
AcX = 40 | AcY = -1412 | AcZ = 18084 | Tmp = 24.01 | GyX = -219 | GyY = 72 | GyZ = 42
AcX = 56 | AcY = -1436 | AcZ = 18124 | Tmp = 24.11 | GyX = -179 | GyY = 61 | GyZ = 49
AcX = 108 | AcY = -1456 | AcZ = 18080 | Tmp = 24.01 | GyX = -184 | GyY = 94 | GyZ = 72
AcX = 84 | AcY = -1380 | AcZ = 18044 | Tmp = 23.97 | GyX = -216 | GyY = 49 | GyZ = 55
AcX = 112 | AcY = -1356 | AcZ = 18148 | Tmp = 24.06 | GyX = -186 | GyY = 74 | GyZ = 63
AcX = 12 | AcY = -1432 | AcZ = 18136 | Tmp = 24.06 | GyX = -193 | GyY = 72 | GyZ = 59
```

MPU-6050 Raw Values from Serial Monitor

C.3 Arduino-Accelerometer Control

There are numerous ways in which the Arduino and MPU-6050 chip can be combined. For this project, codes focused on the raw values and roll and pitch angles will be utilized. Firstly, the accelerometer must be connected to the Arduino's power and ground pins. The SCL and SDA pins on the chip are also connected to analog inputs 5 and 4, respectively. This creates the interface between the two and allows them to use the I^2C Protocol. SDA is the data line and SCL is the clock line that synchronizes data transfers over the I^2C bus. Finally, the accelerometer's INT should be connected to digital pin 2. INT is the interrupt pin, which is used to tell the Arduino when the 1024 Byte FIFO buffer is full of data waiting to be read (Ravi, 2017).

Next, codes should be uploaded to the Arduino in order to take advantage of the MPU-6050's functions. One simple code used to test the accelerometer is one that gives raw values based on its different movements in addition to the temperature. The accelerometer provides acceleration information, and the gyroscope provides angular rate information. These values can then be used to calculate and determine more sophisticated data, such as roll and pitch angles mentioned above. It may be observed that the raw values fluctuate a lot; one reason is that the default sensitivity is high (Arduino, 2019b). Other useful codes make calculations from the raw values, especially those two angles, determined similar to how one would by hand.

After discovering this particular method does not work when the accelerometer chip is attached to the hand, an altered code had to be used. It is similar, but the angles are calculated using information from the gyroscope. The equation for the angle of the motors combines both angle measurements from the accelerometer and gyroscope. The gyroscope angle is found using an update equation to prevent gyroscope drift, which requires the starting angle, raw value, and elapsed time. The accelerometer angle is determined using the equations from section C.2.

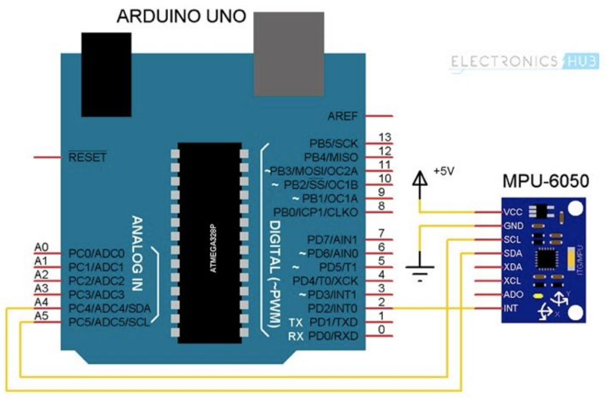
$$gyroXAngle = XAngle + gyroXrate * elapsedTime$$

$$gyroXrate = gx / 131$$

$$elapsedTime = (currentTime - previousTime) / 1000$$

$$accXAngle = (atan(ay / sqrt(pow(ax, 2) + pow(az, 2))) * 180 / PI)$$

$$XAngle = (ap * gyroXAngle) + ((1 - ap) * accXAngle)$$



MPU-6050 with Arduino Circuit Diagram

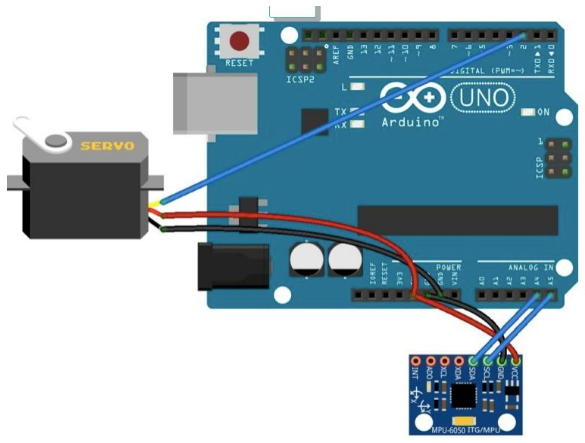
C.4 Arduino-Servo Motor Control

For the purpose of this project, the servo motors of the wrist joint will be controlled based on feedback from the accelerometer. Therefore, the accelerometer should be hooked up to the Arduino as specified in the previous section. The corresponding wires of the servo motors must be connected to power, ground, and a digital output. The Arduino does not supply enough power for multiple motors, so an external source must be used; I will be using a +7.4 V battery. The wrist motors must be connected to two different digital outputs. This allows one to move when the accelerometer is tilted in the x direction and the other to move when it is tilted in the y direction. Within the Arduino code, two motors must be defined as well as which pins they will be connected to. Similarly, the acceleration in the x and y directions need to be defined, mapped, and used separately in the code. Lastly, it is important to connect all the circuit components to the same ground, including the Arduino, accelerometer, and motors. This ensures the correct amount of power is distributed throughout the circuit, which prevents overload too.

In order to move, the motors use the accelerometer values (-17000 to 17000) and map them from 0 to 180. This means when the sensor is moved upward the sensor output is 180, and when it is moved downward the output is 0 (Hamza, 2019). For my model, I reversed the 0 and 180 so that the motors will move in the opposite direction of the accelerometer's tilt. Within the code, there should be a defined duration and frequency of the pulses generated for the digital output. It is also possible to specify how far the motor should rotate in terms of an angle from 0 to 180.

Furthermore, the tendon servo motor can be controlled by a push button that when pressed, will turn the motor's lever arm, pull the tendons, and close the fingers. When the button is pressed again, the motor, tendons, and fingers will return to the starting position. It is hooked up the same way as the other motors, except the signal wire is connected to the button, which is then connected to a digital output pin. The button is also connected to the Arduino's power, and a $1\text{k}\Omega$ resistor. The resistor is connected to ground and a PMW digital output pin (Ghosh, 2017).

An additional way to make the fingers open and close is through the use of a limit switch. There are three different metal bars, NC, NO, and COM. NC means normally closed, NO is normally open, and COM is the common. In the circuit, the COM is connected to ground, and the NO is connected to a resistor and then to a digital pin on the Arduino. When the plunger touches something, it causes the motor to move so the fingers close. When contact is made again, the motor returns to its starting position and the fingers open back up. Within the switch, the NO will be shorted with the COM and the NC will be open with respect to the COM (Albbg, 2014).

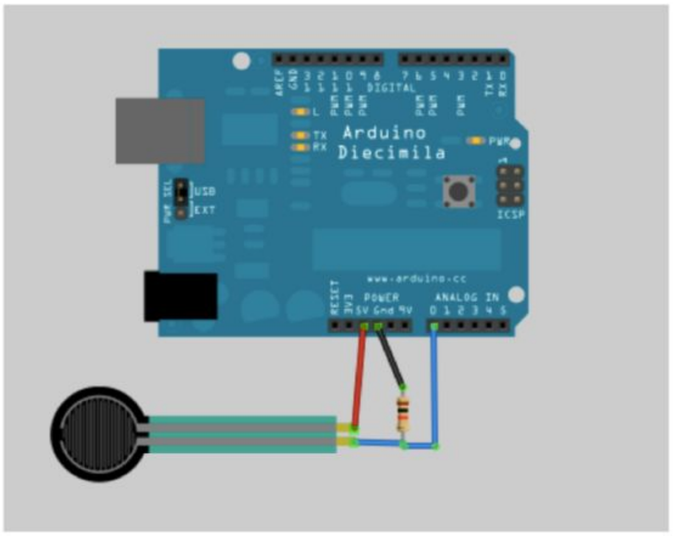


Servo Motor and MPU-6050 with Arduino Circuit Diagram

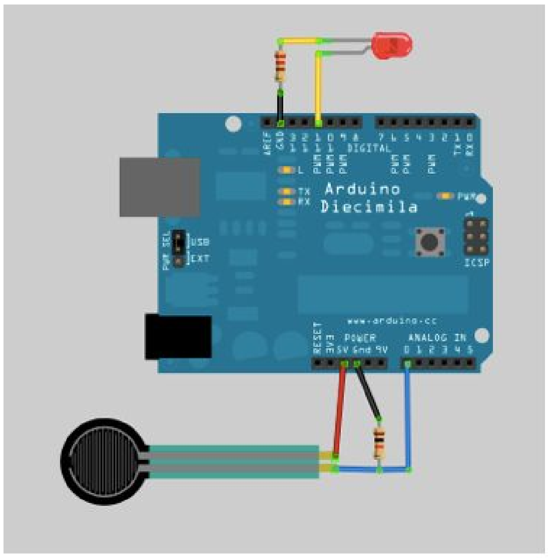
C.5 Arduino-FSR Control

When the FSR is used in a circuit with an Arduino, a 10kΩ pulldown resistor can be included, which contributes to the total resistance. The FSR is connected to power and an analog pin, and a 10kΩ resistor is connected to the same analog pin and ground. As the total resistance decreases, the current increases and subsequently the voltage across the 10kΩ resistor increases as well. This makes resistance close to linear but not voltage due to voltage being proportional to the inverse of the FSR resistance in the equation $V_O = V_{CC}\,(R / (R + FSR))$. This particular code uses the analog reading and converts it to voltage. Then, the voltage value can be converted to resistance (both the FSR resistance and conductance). Lastly, the resistance is used to calculate force in Newtons (Adafruit Learning System, 2019).

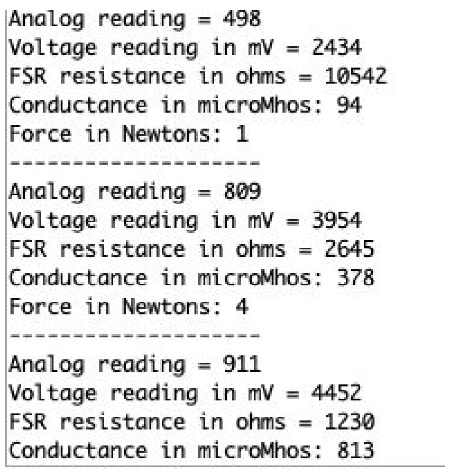
In addition, there is another Arduino code that uses an LED as an indicator of how much force is being applied. Greater pressure corresponds to a brighter LED, and less pressure makes it dimmer. A PWM digital pin is used for the LED so this can work properly. Besides the digital pin, the LED is connected to a 220Ω resistor, which is then connected to ground (Adafruit Learning System, 2019).



FSR with Arduino Circuit Diagram



FSR and LED with Arduino Circuit Diagram



```
Analog reading = 498
Voltage reading in mV = 2434
FSR resistance in ohms = 10542
Conductance in microMhos: 94
Force in Newtons: 1
--------------------
Analog reading = 809
Voltage reading in mV = 3954
FSR resistance in ohms = 2645
Conductance in microMhos: 378
Force in Newtons: 4
--------------------
Analog reading = 911
Voltage reading in mV = 4452
FSR resistance in ohms = 1230
Conductance in microMhos: 813
```

FSR Output from Serial Monitor

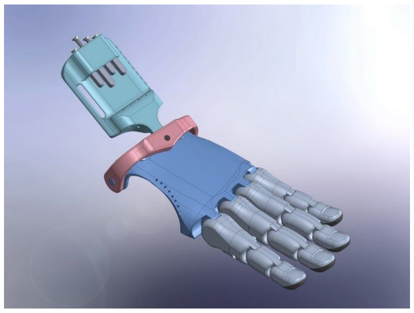
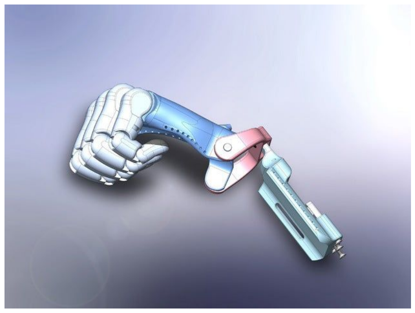
C.6 Hand Model and Installation

My own model for this project started with a pre-existing design for a 3D printed prosthetic hand. This one was created to utilize a person's mechanical movement in order to make the wrist bend. These bending motions cause artificial tendons to pull the fingers closed (Thomas, 2016). I used the palm of that hand and all the fingers, along with several additions. First off, I created a thumb from the middle finger joints and added a place for it on the palm, again using the middle finger post. I also had to make a hole in the palm for the thumb's tendon to go through. Next, I measured the motors' brackets and screw holes, and I used these measurements to design the 3D printed parts so they could be connected together. I then determined how to attach the two wrist motors so that their motions mimic the roll and pitch rotations.

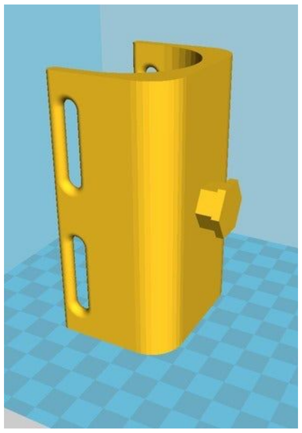
Acoustic guitar D strings are used for the artificial tendons, which begin at the fingertip, feed through the rest of the finger and palm, and end at a motor. At the fingertips, the guitar strings are held in place by cord end caps that act as clamps. These clamps are hooked onto eye pins, which must be heated so they can push through the fingertip filament. The pins and guitar strings can be reinforced by glue too. On top of the hand, elastic cord is attached to the fingertips and palm so that the fingers open back up from the closed position. The elastic is pulled by the motor, similar to the guitar strings, in order to obtain the necessary force to open them. In addition to the elastic, graphite lubricant may be used to make the individual finger joints looser and move more freely. Filing the plastic where it makes contact with the elastic helps reduce friction as well by smoothing the rough parts.

At the other end of the wrist motors is a 3D printed cuff that allows individuals to put it on their arm. I also modified this cuff from another design, which was originally made as a ski pole grip for a prosthetic arm (Wolf, 2019). I added an attachment site for the second wrist motor and screw holes for the tendon motor. Also, this piece was lengthened to have enough space for the circuit components, and I removed the hexagon peg from the original design.

In order to have a self-contained device, the other electronic components must be attached to the model as well. This required soldering the sensors and wires onto small circuit boards that were glued to the model. Once this was completed, the accelerometer chip was installed on the hand below the thumb and sitting vertically while the palm is facing down. It must be vertical since this is its horizontal position when the hand is facing thumb-up. If the force sensor was used, it would be located on the middle fingertip so one can determine how much pressure they are applying. The Arduino, resistors, LED, push button, and the other circuitry pieces are attached to the side of the arm cuff.

Original Hand and Fingers Design



Original Ski Pole Grip Design

D. Troubleshooting

D.1 Accelerometer with Arduino

When first learning how to use the Arduino with the MPU-6050 chip, it is necessary to see and understand the raw values it provides based on its motion. The code from Arduino's playground worked very well. There was one issue with the output from acceleration in the x direction. Instead of changing when the chip was tilted, the value remained constant at 32767, or 1.9999g. Running this code on a new accelerometer/gyroscope fixed this problem.

In addition, it is helpful to use a code for calculating pitch and roll values. The calculations can later be implemented in another code for moving servo motors based on these angles. The first library found (MPU6050_DMP6) provides yaw, pitch, and roll values. It requires the I2Cdev library, which came back with a 2Cdev error stating "undetermined #if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE." Within the code, it mentioned two websites for possible solutions to the common issue. Upon looking into this, neither resources, or others, were able to solve this problem (Rowberg, 2019). Therefore, similar codes were researched instead, and one was found that successfully provided pitch and roll values. The code also includes an attempt to determine the yaw angle, but states that it is incorrect. For the purpose of this project, yaw is not needed, so the yaw calculation part of the code can be ignored.

Next, how to hand-calculate the pitch and roll angles was researched. After finding the equations for pitch and roll, it was noticed that they use values in units of grams. Therefore, another equation for converting the accelerometer's raw acceleration numbers into grams had to be found. The methods for both equations proved to be accurate because the final answers matched the angles the code calculated. Before the new MPU-6050 chip was used, the pitch angle came out to be 66.75° and the roll was 0.4559° on a flat, horizontal surface. Both of these should have been zero, which was discovered to be true with the new chip.

D.2 Servo Motor and Accelerometer with Arduino

Once the accelerometer chip was learned how to use and understand the outputs, servo motors needed to be integrated with it and the Arduino. The library found that controls the motors based on the chip's movement did not work on the installed Arduino application on the computer due to an error compiling the board. Instead, the web version of the software had to be used. It is simple to specify how far the motor rotates by changing the angles within the code. Also, by decreasing the delay, the motors run smoother and quicker in response to the accelerometer's changing motions. This was successful until more than one motor was added to the circuit. The Arduino does not supply enough power to operate multiple motors, so they either would not move at all or would demonstrate a few jerky movements. To solve this issue, there was an attempt to use an adapter that plugs into the Arduino for power from a wall outlet. This did not help, so an external voltage box was hooked up to the circuit instead. It was able to provide the motors with higher voltage and connect all circuit elements to the same ground. Both motors moved smoothly and accurately with this additional power source.

D.3 FSR with Arduino

Another interesting sensor to use for the prosthetic is a force sensitive resistor. A couple ways to use and test it were found. Both were easy to setup and worked well with no problems. The serial monitor helped to determine this by showing the outputs. When more pressure was applied, the outputs signified this, for example, by showing a greater force in Newtons. The second circuit visually displayed this with an LED becoming brighter as pressure increased. In addition, two different force sensors were compared to see which worked the best. An ADA 166 (Interlink 402) and Tactilus 15mm 0-30 PSI were tested; the ADA 166 seemed to detect and measure the pressure more accurately and for a wider range.

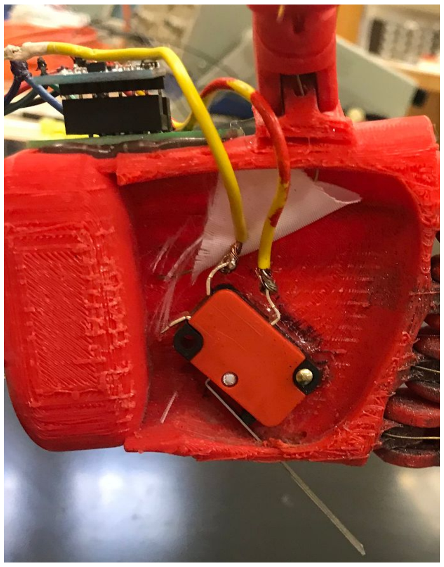
D.4 3D Printed Model

Ideas for the model came from already existing 3D printed prosthetics. The palm and fingers from the first design were useful, but modifications were necessary in order to meet the needs of this project. The arm piece needed a few adjustments as well. These changes can be seen in section C.6. It was difficult to visualize a design for the prosthetic that would work. At first, the idea was to attach the tendon motor close to or on top of the hand so the tendons could be short and avoid interfering with other parts of the arm. However, if the motor was on top of the hand, it would not allow the tendons the necessary space and leverage to pull the fingers closed. If it was connected to the hand, it would cause the arm to be too long or too wide. Therefore, it was decided to screw the tendon motor into the top of the arm cuff. It sits higher than the wrist motors, so there is less interference. Also, it is the first object on the arm cuff, so the tendons are not excessively long. In general, it took several designing and printing attempts to produce the correct sizes of each component. For instance, getting the screw holes to line up correctly took a few tries since it would not work if they were off by as little as a half millimeter. Similarly, adjusting the joint sizes required multiple trials as well for the same reason.

Several Trials of the Hand and Fingers

D.5 Servo and Fingers/Tendons

There needed to be a way to control when the fingers open and close, so options were explored. One idea was to implement a proximity sensor so that when the hand is close to an object, the motor will turn and cause the fingers to close around the object. However, the problem was that the sensor would still perceive the object when the fingers need to be opened. This means the object would not be released because the sensor must detect nothing close by in order for the fingers to open back up. Similarly, the proximity sensor may sense nearby objects that the user does not want to grab, creating further inconveniences. The next idea was to use a small button; the first press would turn the motor to close the fingers, and the second press would return the motor and fingers to their starting positions. This is a more reasonable approach because it allows the individual to determine when he or she wants the fingers to open and close. The circuit setup is straightforward, and the beginning and ending angles of the motor can easily be adapted for the needs of the fingers. Finally, another option was implemented for controlling the fingers: a limit switch. It was placed in the palm of the hand so that the metal bar can make contact with, for example, a table. The bar passes through the hand beneath the pinky, which can be accomplished by heating the bar and plastic so the plastic melts enough for the bar to be pushed through. In this instance, when it touches the table, the fingers can close around a cup in order to pick it up. Similarly, when the individual wishes to put the cup down again, the fingers release it by making contact with the table a second time.


Limit Switch Attached Inside the Palm

D.6 Fingers and Tendons

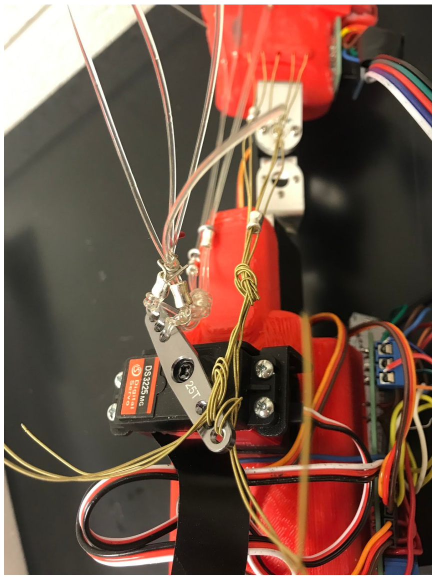
Once the push button method was decided upon and tested, the artificial tendons could be added to the fingers. The first step was to decide how the tendons should be hooked to the fingertips. The initial thought was to glue them since there was no attachment site, only slits on the underside of the fingertips where the end of the tendons sit. This may not secure them well enough and would make it difficult to test different types of materials for the tendons. The solution was to hook clamps to hold the strings onto pins. The easiest way to secure the pins in the fingertips was to heat them enough so that they pushed into the plastic.

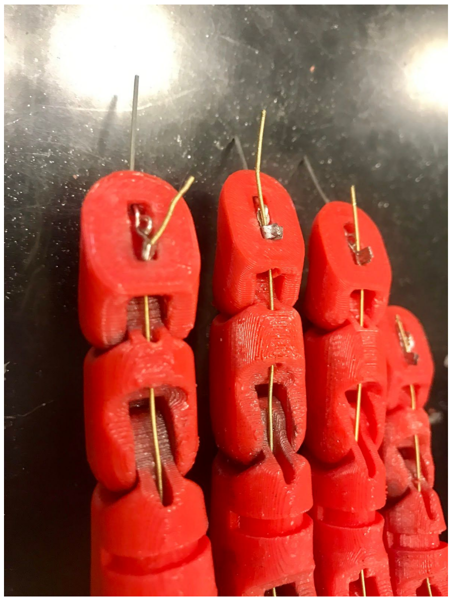
Next, different types of strings had to be tested to determine the best combination of elasticity and flexibility. These included silverwound guitar strings, clear nylon guitar strings, stiff wire, and acoustic guitar D strings. The silverwound and nylon ones were able to pull the fingers closed, but would not open them back up. There needed to be extra force to push them open, so strong wire was tried. It proved to not be flexible enough because it was unable to thread all the way through the fingers. Also, it was noticed that the joints seemed too tight, creating the need for more force to open and close them. One idea was to resize the joints by a millimeter or two so they could become looser. Rather than taking the time to reprint and continue to experiment with other materials, a different approach was taken. Elastic was glued on top of the fingers and hand to generate force for pulling the joints open. Again, both guitar strings were tested with the elastic; the fingers would not open with the sturdier nylon and opened halfway with the silverwound. There was still not enough force to allow the finger to return to the open position, so more than one piece of elastic was necessary. For most cases, two pieces was enough, but a couple fingers needed a third at a single joint. Another way to assist this problem was to decrease the friction, such as using graphite lubricant for stiff joints. Also, the top surface of the fingers was filed down because rough parts of plastic caused the elastic to get stuck and create additional friction.

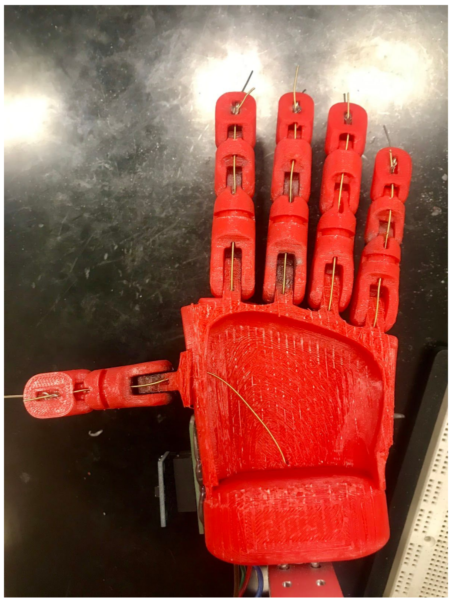
Now that all the fingers were able to open up on their own, the final guitar strings could be added. Acoustic guitar D strings were used, which have a stiffness that is between the silverwound and nylon ones. The guitar strings are tied to the motor's lever arm, but another issue is that they wrap around the base of the arm when it pulls them. Other parts of the arm model interfere with the strings too, specifically the strings rub against or get caught on the motors. Threading the strings through small metal loops and knotting them together guide the strings up away from the motors, prevent them from wrapping around the lever arm, and decrease the slack. Even with these improvements, the tendons still get pulled and rub against the wrist motors when the wrist motors move. The guitar strings may be too long, so moving the tendon motor closer to the hand or onto the hand could eliminate that problem.

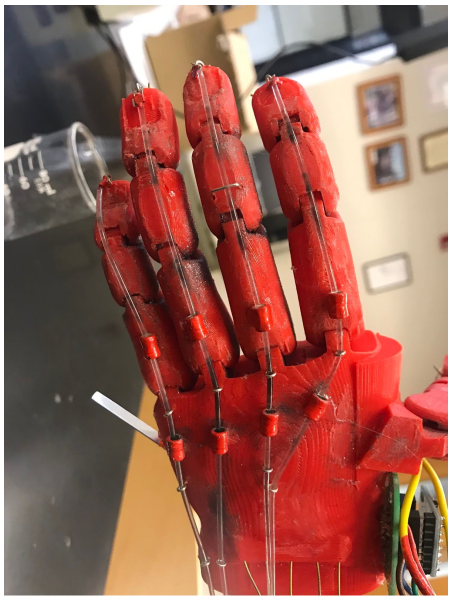
Although the fingers work well on their own, it is difficult for the motor to pull them all closed at the same time. The force required to close the fingers is now greater with elastic. It seems to be too much force for the motor to pull; the fingers will not close completely and parts of the arm get pulled out of alignment, such as the motors and at the screws. Increasing the angle that the motor's lever arm moves did not seem to help. Also, clamping the strings together then tying only one string to the motor made matters worse. Another possible contributing cause of this issue could be that the guitar strings are not tied tight. In other words, there may be too much slack at the starting, open fingers position. However, attempting to tie them tighter did not help either, but guiding them through loops and knotting them together as mentioned above decreased the slack and friction. One last potential problem could be that the finger joints were printed too tightly. Making them looser by resizing and reprinting would reduce the amount of elastic needed to pull them back open. Consequently, the force required to pull them closed would also decrease, which could possibly help this problem.

Still, the elastic would likely create too much force, so the approach of using thinner, stronger elastic pulled by the motor to open the fingers was attempted. This requires a different lever arm on the motor so that the elastic strings can be tied to the opposite side of the lever. They are attached to the top of the finger in the same way the guitar strings are attached. There is also the addition of pins with loops and other larger loops to help guide the elastic strings along the hand to the motor. This was not done for the thumb because the angle it is at does not aid in gripping objects. The angle creates friction that causes the string to not move smoothly too. Both of those problems might be fixed by redesigning and repositioning the thumb. Also, removing the original elastic on the hand lessened the amount of force the motor has to pull when closing the fingers.


Guitar Strings and Elastic Bands Tied to Motor


Guitar Strings Clamped at Fingertips

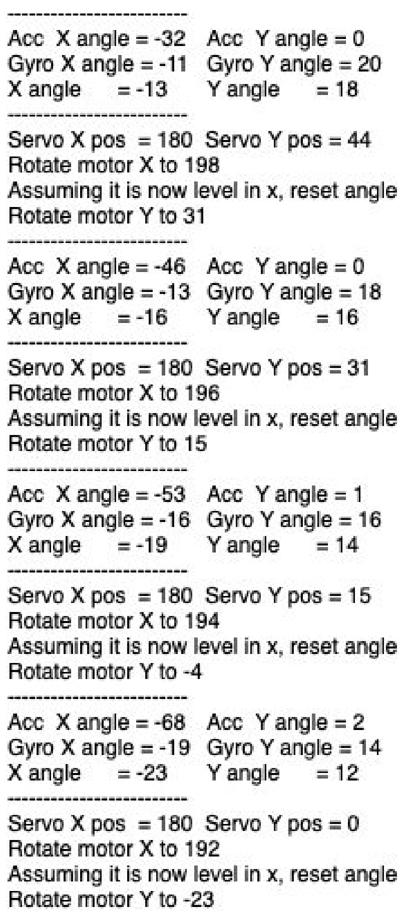

Guitar Strings Threaded Through Fingers and Hand


Elastic Bands for Reopening Fingers

D.7 Accelerometer Installation

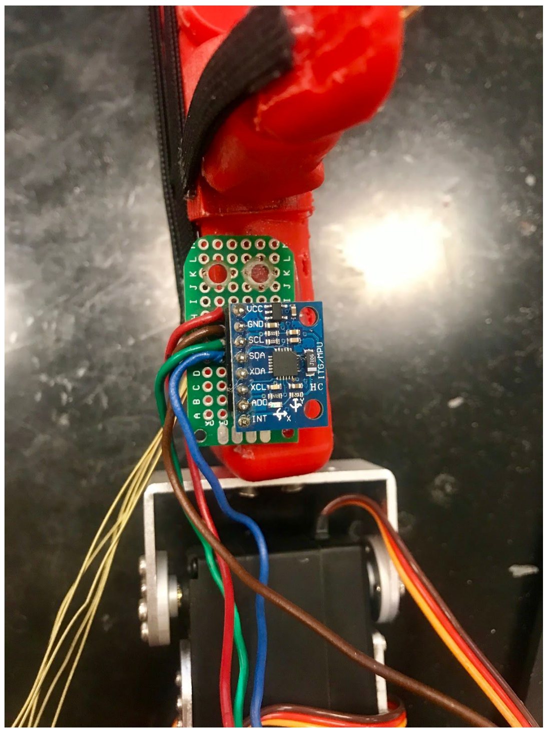
After attaching the accelerometer to the hand, there was trouble getting the wrist motors to work properly due to the accelerometer detecting acceleration when the hand moves. It tries to compensate for the acceleration it is creating, causing the motors to jerk in all directions. There are two possible solutions: control the speed of the servo motors so that they do not create significant acceleration when they move fast, or use gyroscope angles to detect and adjust the position so that acceleration other than gravity is not a factor. The Arduino website suggested using the servo.write() or delay() function for altering the servo motors speed, neither of which made a difference (Arduino, 2019c). Similarly, a library for controlling servo speed was found but did not work because of an error compiling the board (Hawk, 2017).

Moving onto the second option, a library for using the gyro to control the servos was tested. It required the GY6050.h library, which had an error compiling the board (Iqbal, 2019). Still, the ideal approach would be to alter the existing code so that the gyroscope values are used instead of the accelerometer values. The first try created a gyroscope drift, meaning the angles are slightly different than what they should be. To help fix this, angle update equations were added to the code that combined both the gyroscope and accelerometer angle measurements (Dejan, 2019). This code caused the motors to move in the same direction as the tilt instead of the opposite direction. The next attempt of changing the Xmove and Ymove equations made the motors move 90 degrees, and they did not move after that. Then, it was discovered that the variable ap was defined as an integer when it should have been a float. The motors still were not behaving correctly; they moved very slowly up and right when tilted upwards, up when tilted right, down when tilted left, and had no movement when tilted downward. Next, the code was altered to read the position of the servo and then make an offset. This caused the motors to turn 90 degrees up and right when tilted right, and 90 degrees down and left when tilted left. Finally, the Xmove and Ymove equations were fixed again, which allowed the motors to perform as desired. This final code calculates the angles the motors need to move to then resets its angles and time to zero. As a result, there is better accuracy in keeping the hand level due to decreased acceleration. Furthermore, the motors were moving too slowly and therefore adjusting to the altering orientation too slowly. They need to move immediately to keep up with quickly changing needs of the person. To attempt to fix this, the delays were reduced so that the motors adjust quicker to the changing position.



```
-------------------------
Acc  X angle = -32   Acc  Y angle = 0
Gyro X angle = -11   Gyro Y angle = 20
X angle      = -13   Y angle      = 18
-------------------------
Servo X pos  = 180  Servo Y pos = 44
Rotate motor X to 198
Assuming it is now level in x, reset angle
Rotate motor Y to 31
-------------------------
Acc  X angle = -46   Acc  Y angle = 0
Gyro X angle = -13   Gyro Y angle = 18
X angle      = -16   Y angle      = 16
-------------------------
Servo X pos  = 180  Servo Y pos = 31
Rotate motor X to 196
Assuming it is now level in x, reset angle
Rotate motor Y to 15
-------------------------
Acc  X angle = -53   Acc  Y angle = 1
Gyro X angle = -16   Gyro Y angle = 16
X angle      = -19   Y angle      = 14
-------------------------
Servo X pos  = 180  Servo Y pos = 15
Rotate motor X to 194
Assuming it is now level in x, reset angle
Rotate motor Y to -4
-------------------------
Acc  X angle = -68   Acc  Y angle = 2
Gyro X angle = -19   Gyro Y angle = 14
X angle      = -23   Y angle      = 12
-------------------------
Servo X pos  = 180  Servo Y pos = 0
Rotate motor X to 192
Assuming it is now level in x, reset angle
Rotate motor Y to -23
```

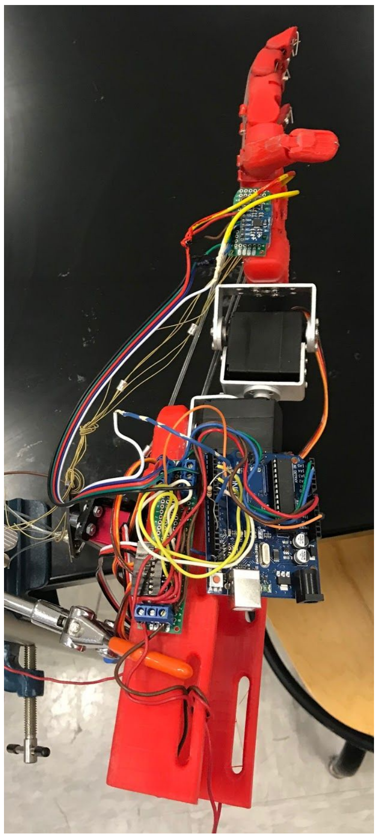
Serial Monitor Output from Final Code,
Tilting Downward in -x Direction

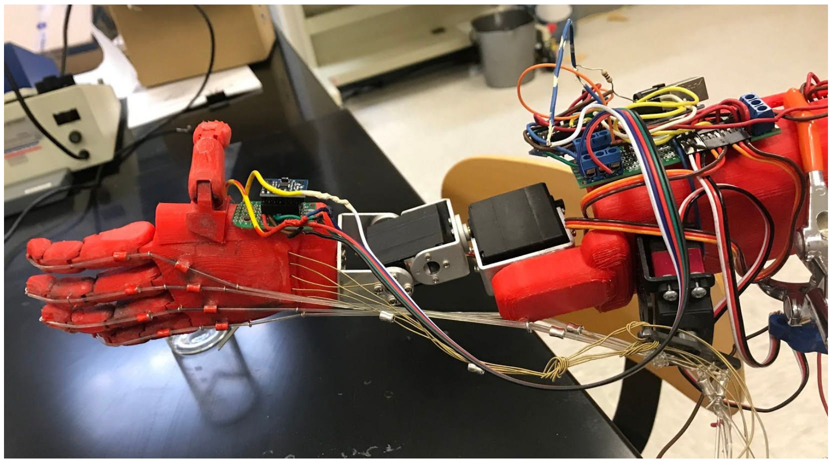


MPU-6050 Chip Installed on Hand

D.8 Installing all Components

When preparing for installation of the Arduino board, a single code had to be written to perform simultaneous actions since it can store and utilize only one code at a time. All three motors needed to be working at the same time so that the hand can remain level and the fingers can open and close. Combining these two separate codes was not overly challenging since they use separate control methods (gyroscope versus limit switch), and the motors are connected to different pins. The installation of all the other equipment onto the hand and arm was tedious when soldering, and the only problem presented was occasional loose wire connections, causing the circuits to become disconnected. Deciding where to place the equipment on the arm cuff was based on the most efficient use of space. The limit switch was attached inside the palm, where it would be most useful as well. All other circuit elements were placed on the thumb-side of the arm cuff. Although an external power source was used, a battery could take the place of it. It can be glued onto the top of the arm cuff or exist independently and function in the same manner. This would decrease the weight and bulk of the arm and increase versatility, which would allow for improved function. Lastly, the force sensor was not installed due to running out of time to attach all the components and figure out if it is possible to combine its code with the code for the three motors.





Final Design

E. Conclusion

Overall, this project was successful in improving 3D printed prosthetics while keeping the cost minimal. Combining digital circuitry and computer programing with preexisting 3D printed designs allowed this to be possible. In the beginning, methods for building different circuits for specific codes were tested and learned. Also accomplished was the design of special 3D printed parts based on the hardware measurements and needs of the model. Next, the calculations used for determining outputs and creating responses were all done within the Arduino codes. Analyzing the serial monitor outputs and comparing them to theoretical values ensured the MPU-6050 and FSR were operating correctly with the Arduino board and software. Computations used in codes could be checked too, as they were for the pitch and roll angles mentioned in section C.2. Similarly, observing how the motors responded to inputs, such as the pushbutton and accelerometer chip, determined if they were working.

All supplies and equipment used for the model can be found on Amazon.com or a hardware store at a low price, making it a much more affordable option than non-3D printed prosthetics. The general design of the model as a whole seemed to be an excellent option; space was utilized and not wasted and bulk and weight were as minimal as possible. Also, it greatly resembled a real hand and arm with multi-jointed fingers, a double-axis wrist motion, and an arm piece attachment. After numerous coding trials and errors, the accelerometer chip installed on the hand operated the wrist motors to keep the hand level, as desired. The fingers, artificial tendons, and elastic strips all worked very well on their own when pulled by the motor. Unfortunately, there was trouble pulling all five tendons at the same time. This seemed to be a result of too much force required to pull them open as well as tight joints and interference with the strings. It could be fixed with more time and alternative ideas and designs. In addition, the force sensor and its own circuit elements were not installed on the arm; there was not enough time to install them determine how to combine the code with the servos code. Although the project was mostly a success, further modifications are necessary and would be very beneficial to future users.

Costs of the Major Components

Arduino microcontroller	~$15
Accelerometer/gyroscope	~$5
Motor bracket	$6
High torque servo motor	$30

Servo motors (2 x $15)	$30
Limit switch	~$1
Force sensor	~$10
Total	~$100

F. Future Improvements

This preliminary prototype paves way for endless expansions in future research. With more time and resources, many components could be improved and added. Regarding the design, it would be beneficial to make it less bulky. For example, if possible, incorporating a single motor that allows both of the motions demonstrated would make it more compact and better resemble a real arm. Using a small battery (rather than the external power source) and smaller Arduino would reduce the weight and space taken up as well. Another improvement may include making the arm cuff piece more comfortable, such as through the use of padding. Similarly, this arm part could be redesigned to better fit the shape of the user's arm. For example, instead of it being a rectangular shape, it might need to be tapered in certain areas depending on the person's arm. Also to customize it, different sizes of the 3D printed parts could be made for individuals of various ages and sizes. For both these improvements, measurements of the person's body would need to be taken. Adjusting the fit and size is another way to increase comfort, compatibility, and usefulness.

In addition, there may be other printer filament to use besides PLA that is stronger and does not wear down quickly. This would ensure a longer window of use and therefore less money spent on replacements. Also, adding grips on the fingertips or using silicone-like printing material for the fingertips would help individuals grip objects more securely. Next, incorporating additional sensors would increase the amount of information gained by environmental stimuli. These could include temperature and proximity sensors, as well as the force sensor tested in this project. Finding an alternative way to program each component would be helpful since the Arduino can only upload and use one code at a time. All the pieces need to be able to function at the same time. A second option is to write a single code that incorporates all other components including the motors, assuming they would not interfere with one another and the board has enough memory. Another important improvement would be to create better wire connections. Currently, the wires sometimes come loose, which interrupts the circuits and makes them stop working.

Another interesting design idea would be to build an elbow joint for those who do not have one. There are many different options for this, but it is likely that another motor would be used for the joint. Additionally, the users would greatly benefit from a thumb with better range of motion. In this model, it is a hinge joint in the model, when in reality it should be a saddle joint to allow for the actions of a normal opposable thumb. This would be a tedious and complicated process, but it is doable. Another possibility is repositioning the thumb so that it can function more effectively and efficiently. For instance, moving it down and in towards the palm area would improve its ability to grip objects. This may also help the problem of not being able to pull the thumb open and closed as well as the others due to the strings not running parallel to the hand and arm.

Lastly, one of the most important and necessary changes that should be made is to the fingers and artificial tendons. The joints were too tight, causing the need for elastic to pull the fingers open from the closed position. After several different attempts, they still did not work as they should. Therefore, other approaches need to be tested in order to fix this issue. Resizing and reprinting the fingers to be looser would decrease the amount of force that the motor must pull. Another idea to incorporate may be to continue exploring other material options instead of the guitar strings and elastic bands, or considering a stronger motor. Also, determining a method in which the tendon motor is inside the palm may improve the opening and closing actions of the fingers. This is based on the fact that the wrist motors affect the length of the tendons, meaning that when the motors move, they pull the guitar strings away from their current proper position. Using this idea could also help the thumb's tendon be pulled more easily by decreasing the angle, thus decreasing the friction, it is pulled at with this model.

References

Adafruit Learning System. (2019, June 24). Force Sensitive Resistor (FSR). Retrieved from https://learn.adafruit.com/force-sensitive-resistor-fsr/overview

Ahsanraza. (2015, June 9). Re: YAW Calculation from MPU6050 [Web log comment]. Retrieved from https://forum.arduino.cc/index.php?topic=328765.0

Albbg. (2014, August 9). Retrieved from https://www.edaboard.com/showthread.php?320987-limit-switch-(NC-NO-COM)-working-principle

Amputee Coalition. (2019, April). Financial Assistance for Prosthetic Services, Durable Medical Equipment, and Other Assistive Devices. Retrieved from https://www.amputee-coalition.org/resources/financial-assistance-for-prosthetic-services/

Apoorve. (2015, August 1). Servo Motor: Basics, Theory, & Working Principle. Retrieved from https://circuitdigest.com/article/servo-motor-basics

Aqeel, A. (2018, June 21). Introduction to Arduino Uno. Retrieved from https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-uno.html

Arduino. (2019a). Arduino Uno Rev3. Retrieved from https://store.arduino.cc/usa/arduino-uno-rev3

Arduino. (2019b). MPU-6050 Accelerometer Gyro. Retrieved from https://playground.arduino.cc/Main/MPU-6050/#short

Arduino. (2019c). ServoWrite. Retrieved from https://www.arduino.cc/en/Reference/ServoWrite

Baird, J. (2015, October). Special Subjects: Limb Prosthetics: Overview of Limb Prosthetics. In *Merck Manual for the Consumer*. Kenilworth, NJ: Merck &. Retrieved from https://www.merckmanuals.com/home/special-subjects/limb-prosthetics/overview-of-limb-prosthetics.

Components101. (2017, September 18). Servo Motor SG-90. Retrieved from https://components101.com/servo-motor-basics-pinout-datasheet

Cordella, F., Ciancio, A. L., Sacchetti, R., Davalli, A., Cutti, A. G., Guglielmelli, E., & Zollo, L. (2016, May 12). Literature Review on Needs of Upper Limb Prosthesis Users. *Frontiers in Neuroscience,10*. doi:10.3389/fnins.2016.00209

Dejan. (2019, April 9). Arduino and MPU6050 Accelerometer and Gyroscope Tutorial. Retrieved from https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

Frost, & Sullivan. (2018). 3-D Printing to Lower Prosthetic Costs. Retrieved from https://aabme.asme.org/posts/3-d-printing-to-lower-prosthetic-costs

Goodrich, R. (2018, May 30). Accelerometer vs. Gyroscope: What's the Difference? Retrieved from https://www.livescience.com/40103-accelerometer-vs-gyroscope.html

Ghosh, A. (2017, May 9). Arduino Servo Motor Control With Pushbutton. Retrieved from https://thecustomizewindows.com/2017/05/arduino-servo-motor-control-servo-pushbutton/

Hamza, A. (2019, March 23). Learn how to control a servo motor using an MPU6050 sensor connected to an Arduino. Retrieved from https://maker.pro/arduino/tutorial/how-to-control-a-servo-with-an-arduino-and-mpu6050

Hawk, G. (2017, February 9). VarSpeedServo / VarSpeedServo.h [Web log post]. Retrieved from https://github.com/netlabtoolkit/VarSpeedServo/blob/master/VarSpeedServo.h

InvenSense Inc. (2013, August 19). MPU-6000 and MPU-6050 Product Specification Revision 3.4. Retrieved from https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

Iqbal, H. (2019, June 11). Controlling of Servo Motor with Arduino and MPU6050. Retrieved from https://create.arduino.cc/projecthub/hammadiqbal12/controlling-of-servo-motor-with-arduino-and-mpu6050-6375b9

Kaulics, B. (2015, June 22). Re: How to read a gyro/accelerometer [Web log comment]. Retrieved from https://electronics.stackexchange.com/questions/39714/how-to-read-a-gyro-accelerometer

Limb Loss. (2019, March 19). Retrieved from https://medlineplus.gov/limbloss.html.

Marks, L. J., & Michael, J. W. (2001, September 29). Artificial limbs. *BMJ,323*(7315). doi:10.1136/bmj.323.7315.0c

Mirza, K. (2019, April 7). Arduino UNO for Beginners. Retrieved from https://projectiot123.com/2019/04/07/arduino-uno-for-beginners/

Mohney, G. (2013, April 25). Health Care Costs for Boston Marathon Amputees Add Up Over Time. *Abc News*. Retrieved from https://abcnews.go.com/Health/health-care-costs-boston-marathon-amputees-add-time/story?id=19035114

Pedley, M. (2013, March). Tilt Sensing Using a Three-Axis Accelerometer [Web log post]. Retrieved from https://www.nxp.com/files-static/sensors/doc/app_note/AN3461.pdf

Ravi. (2017, November 21). Getting Started with Arduino and MPU6050. Retrieved from https://www.electronicshub.org/getting-started-arduino-mpu6050/

Rowberg, J. (2019, June). I2cdevlib / Arduino [Web log post]. Retrieved from https://github.com/jrowberg/i2cdevlib/tree/master/Arduino

Sharington, G. (2017, April 11). The True Cost of Prosthetic Limbs: What it Means For Families [Web log post]. Retrieved from https://jordanthomasfoundation.org/true-cost-prosthetic-limbs/

TEMCo Industrial. (n.d.). Micro Limit Switches. Retrieved from https://temcoindustrial.com/product-guides/switches-and-relays/micro-limit-switches

Thomas, A. (2016, November 9). Cathy's Lucky Fin - Prosthetic Hand - Bowden / Push-Pull Variant [Web log post]. Retrieved from https://www.thingiverse.com/thing:1880228

Thomas Publishing Company. (n.d.). Limit Switch Characteristics. Retrieved from https://www.thomasnet.com/articles/instruments-controls/limit-switches/

Turner, R. (2018, November 08). Prosthetics Costs: The High Price of Prosthetic Limbs. Retrieved from https://www.disabled-world.com/assistivedevices/prostheses/prosthetics-costs.php

Veterans Health Administration. (2009). *Prosthetics and Related Technology* [Brochure]. Washington, D.C.: Author. Retrieved from https://www.research.va.gov/resources/pubs/docs/prosthetics-brochure.pdf

Wolf, M. (2019, May 20). Ski pole grip for prosthetic arm [Web log post]. Retrieved from https://www.thingiverse.com/thing:3643701

Appendix

Appendix A: Accelerometer Raw Values Code

```
// MPU-6050 Short Example Sketch
// By Arduino User JohnChi
// August 17, 2014
// Public Domain
#include<Wire.h>
const int MPU_addr=0x68;  // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  Serial.begin(9600);
}
```

```
void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14
registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42
(TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48
(GYRO_ZOUT_L)
  Serial.print("AcX = "); Serial.print(AcX);
  Serial.print(" | AcY = "); Serial.print(AcY);
  Serial.print(" | AcZ = "); Serial.print(AcZ);
  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);
//equation for temperature in degrees C from datasheet
  Serial.print(" | GyX = "); Serial.print(GyX);
  Serial.print(" | GyY = "); Serial.print(GyY);
  Serial.print(" | GyZ = "); Serial.println(GyZ);
  delay(333);
}
```

(Arduino, 2019b).

Appendix B: Pitch and Roll Angles Code (-180° to +180°)

```
#include<Wire.h>

const int MPU=0x68;  // I2C address of the MPU-6050

int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
```

```
double compAngleX,compAngleY,/*gzangle,*/compAnglez,timer;
  double accXangle ,accYangle,acczangle ,gyroXrate
,gyroYrate,gyroZrate;
  double gyroXAngle, gyroYAngle, gyroZAngle;
  // float rgyro,w;
  int ap=0.955;
  void setup()
{
   Serial.begin(115200);
/////////////////////// SENSOR READING//////////
 Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  ///////////////////////////////////////
}
void loop()
{
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
```

```
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42
(TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48
(GYRO_ZOUT_L)
 accXangle = (atan2(AcY, AcZ) * RAD_TO_DEG);
 accYangle = (atan2(AcX, AcZ) * RAD_TO_DEG);
acczangle = (atan2(AcX,AcY) * RAD_TO_DEG);/* my attempt to calculate
yaw but not correct*/
 gyroXrate = GyX / 16.5;
 gyroYrate = GyY / 16.5;
 gyroZrate = GyZ/ 16.5;
 timer = millis();
 //angular position
 gyroXAngle += gyroXrate * (millis()-timer)/1000;
 gyroYAngle += gyroYrate * (millis()-timer)/1000;
 gyroZAngle += gyroZrate * (millis()-timer)/1000;/* my attempt to
calculate yaw but not correct*/
//---------------------------\\
 //COMPLIMENTRY FILTER STARTED\\
 //----------------------------\\
 compAngleX = ap * (compAngleX + gyroXAngle) + (1-ap) * accXangle;
 compAngleY = ap * (compAngleY + gyroYAngle) + (1-ap) * accYangle;
 compAnglez = ap * (compAnglez + gyroZAngle) + (1-ap) * acczangle;
/*yaw but not correct*/
 //----------------------------\\
 //COMPLIMENTRY FILTER ENDED  \\
 //----------------------------\\
   Serial.print("ypr\t");
            Serial.print(compAnglez);
            Serial.print("\t");
            Serial.print(compAngleY);
            Serial.print("\t");
            Serial.println(compAngleX);
}
```

(Ahsanraza, 2015).

Appendix C: Servo Motor and Accelerometer Code: Motion in X-Direction

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>
Servo sg90;
int servo_pin = 2;
MPU6050 sensor ;
int16_t ax, ay, az ;
int16_t gx, gy, gz ;
```

```
void setup ( )

{

sg90.attach ( servo_pin );

Wire.begin ( );

Serial.begin  (9600);

Serial.println  ( "Initializing the sensor" );

sensor.initialize ( );

Serial.println (sensor.testConnection ( ) ? "Successfully Connected"
: "Connection failed");

delay (1000);

Serial.println ( "Taking Values from the sensor" );

delay (1000);

}

void loop ( )

{

sensor.getMotion6 (&ax, &ay, &az, &gx, &gy, &gz);

ax = map (ax, -17000, 17000, 0, 180) ;

Serial.println (ax);

sg90.write (ax);

delay (200);

}
```

(Hamza, 2019).

Appendix D: Servo Motor and Accelerometer Code: Motion in Y-Direction

```
#include <Wire.h>

#include <MPU6050.h>
```

```
#include <Servo.h>

Servo sg90;

int servo_pin = 2;

MPU6050 sensor ;

int16_t ax, ay, az ;

int16_t gx, gy, gz ;

void setup ( )

{

sg90.attach ( servo_pin );

Wire.begin ( );

Serial.begin  (9600);

Serial.println  ( "Initializing the sensor" );

sensor.initialize ( );

Serial.println (sensor.testConnection ( ) ? "Successfully Connected"
: "Connection failed");

delay (1000);

Serial.println ( "Taking Values from the sensor" );

delay (1000);

}

void loop ( )

{

sensor.getMotion6 (&ax, &ay, &az, &gx, &gy, &gz);

ay = map (ay, -17000, 17000, 0, 180) ;

Serial.println (ay);

sg90.write (ay);

delay (200);
}
```

(Hamza, 2019).

Appendix E: FSR Advanced Information Code

```
/* FSR testing sketch.
Connect one end of FSR to power, the other end to Analog 0.
Then connect one end of a 10K resistor from Analog 0 to ground
For more information see www.ladyada.net/learn/sensors/fsr.html */
int fsrPin = 0;     // the FSR and 10K pulldown are connected to a0
int fsrReading;     // the analog reading from the FSR resistor
divider
int fsrVoltage;     // the analog reading converted to voltage
unsigned long fsrResistance;  // The voltage converted to resistance,
can be very big so make "long"
unsigned long fsrConductance;
long fsrForce;       // Finally, the resistance converted to force
void setup(void) {
  Serial.begin(9600);   // We'll send debugging information via the
Serial monitor
}
void loop(void) {
  fsrReading = analogRead(fsrPin);
  Serial.print("Analog reading = ");
  Serial.println(fsrReading);
  // analog voltage reading ranges from about 0 to 1023 which maps to
0V to 5V (= 5000mV)
  fsrVoltage = map(fsrReading, 0, 1023, 0, 5000);
  Serial.print("Voltage reading in mV = ");
  Serial.println(fsrVoltage);
  if (fsrVoltage == 0) {
    Serial.println("No pressure");
  } else {
    // The voltage = Vcc * R / (R + FSR) where R = 10K and Vcc = 5V
    // so FSR = ((Vcc - V) * R) / V        yay math!
    fsrResistance = 5000 - fsrVoltage;     // fsrVoltage is in
millivolts so 5V = 5000mV
    fsrResistance *= 10000;                // 10K resistor
    fsrResistance /= fsrVoltage;
    Serial.print("FSR resistance in ohms = ");
    Serial.println(fsrResistance);
    fsrConductance = 1000000;           // we measure in micromhos so
    fsrConductance /= fsrResistance;
    Serial.print("Conductance in microMhos: ");
    Serial.println(fsrConductance);
    // Use the two FSR guide graphs to approximate the force
    if (fsrConductance <= 1000) {
      fsrForce = fsrConductance / 80;
      Serial.print("Force in Newtons: ");
      Serial.println(fsrForce);
    } else {
      fsrForce = fsrConductance - 1000;
      fsrForce /= 30;
      Serial.print("Force in Newtons: ");
      Serial.println(fsrForce);
    }
  }
  Serial.println("--------------------");
  delay(1000);
}
```

(Adafruit Learning System, 2018).

Appendix F: FSR with LED Code

```
/* FSR testing sketch.
Connect one end of FSR to 5V, the other end to Analog 0.
Then connect one end of a 10K resistor from Analog 0 to ground
Connect LED from pin 11 through a resistor to ground
For more information see www.ladyada.net/learn/sensors/fsr.html */
int fsrAnalogPin = 0; // FSR is connected to analog 0
int LEDpin = 11;      // connect Red LED to pin 11 (PWM pin)
int fsrReading;      // the analog reading from the FSR resistor
divider
int LEDbrightness;
void setup(void) {
  Serial.begin(9600);   // We'll send debugging information via the
Serial monitor
  pinMode(LEDpin, OUTPUT);
}
void loop(void) {
  fsrReading = analogRead(fsrAnalogPin);
  Serial.print("Analog reading = ");
  Serial.println(fsrReading);
  // we'll need to change the range from the analog reading (0-1023)
down to the range
  // used by analogWrite (0-255) with map!
  LEDbrightness = map(fsrReading, 0, 1023, 0, 255);
  // LED gets brighter the harder you press
  analogWrite(LEDpin, LEDbrightness);
  delay(100);
}
```

(Adafruit Learning System, 2018).

Appendix G: Final Motor and Gyroscope Code: X- and Y-directions

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>
Servo sg90x;
int servo_pinx = 3;
Servo sg90y;
int servo_piny = 5;
MPU6050 sensor ;
int16_t ax, ay, az, accXAngle, accYAngle ;
int16_t gx, gy, gz ;
int16_t gyroXrate, gyroYrate, timer, gyroXAngle, gyroYAngle,
compAngleX, compAngleY, accXangle, accYangle ;
int16_t Xmove, Ymove ;
```

```
int16_t XAngle, YAngle ;

int16_t previousTime, currentTime, elapsedTime ;

float ap=0.90 ;

int accYOffset = -2 ;

void setup ( )

{

sg90x.attach ( servo_pinx );

sg90y.attach ( servo_piny );

Wire.begin ( );

Serial.begin  (9600);

Serial.println  ( "Initializing the sensor" );

sensor.initialize ( );

Serial.println (sensor.testConnection ( ) ? "Successfully Connected"
: "Connection failed");

delay (1000);

Serial.println ( "Taking Values from the sensor" );

delay (1000);

}

void loop ( )

{

previousTime = currentTime;        // Previous time is stored before
the actual time read

currentTime = millis();            // Current time actual time read

elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000
to get seconds

sensor.getMotion6 (&ax, &ay, &az, &gx, &gy, &gz);
```

```
// mx = map (ax, -17000, 17000, 180, 0) ;

// my = map (ay, -17000, 17000, 180, 0) ;

accXAngle = (atan(ay / sqrt(pow(ax, 2) + pow(az, 2))) * 180 / PI) ;

accYAngle = (atan(-1 * ax / sqrt(pow(ay, 2) + pow(az, 2))) * 180 /
PI) + accYOffset ;

Serial.print ("Acc  X angle = ") ;

Serial.print (accXAngle) ;

Serial.print ("\t Acc  Y angle = ") ;

Serial.println (accYAngle) ;

gyroXrate = gx / 131 ;

gyroYrate = gy / 131 ;

gyroXAngle = XAngle + gyroXrate*elapsedTime ;

gyroYAngle = YAngle + gyroYrate*elapsedTime ;

Serial.print ("Gyro X angle = ") ;

Serial.print (gyroXAngle) ;

Serial.print ("\t Gyro Y angle = ") ;

Serial.println (gyroYAngle) ;

XAngle = (ap * gyroXAngle) + ((1-ap) * accXAngle) ;  // combine both
angle measurements

YAngle = (ap * gyroYAngle) + ((1-ap) * accYAngle) ;

Serial.print ("X angle      = ") ;

Serial.print (XAngle) ;

Serial.print ("\t Y angle      = ") ;

Serial.println (YAngle) ;

Serial.println ("-------------------------") ;

Serial.print ("Servo X pos  = ") ;
Serial.print (sg90x.read()) ;
Serial.print ("  Servo Y pos = ");
Serial.println (sg90y.read()) ;
Xmove = sg90x.read() + YAngle ;
Ymove = sg90y.read() + XAngle ;
if (Xmove > 91) {
  sg90x.write (Xmove) ;
  Serial.print ("Rotate motor X to ");
  Serial.println (Xmove);
  delay (50);
  Serial.println ("Assuming it is now level in x, reset angle") ;
}
else if (Xmove < 89) {
  sg90x.write (Xmove) ;
  Serial.print ("Rotate motor X to ") ;
  Serial.println (Xmove);
  delay (50) ;
} ;
if (Ymove > 91) {
  sg90y.write (Ymove) ;
  Serial.print ("Rotate motor Y to ");
  Serial.println (Ymove);
  delay (50);
}
else if (Ymove < 89) {
  sg90y.write (Ymove) ;
  Serial.print ("Rotate motor Y to ") ;
  Serial.println (Ymove);
  delay (50) ;
} ;
delay (50);
Serial.println ("-------------------------") ;
}
```

(Hamza, 2019).

Appendix H: Servo Motor and Push Button Code

```
#include <Servo.h>
 const int servoPin = 8;  // Servo pin
 const int buttonPin = 9;  // Pushbutton pin
 int buttonState = 0;
 int directionState = 0;
 Servo myservo;
 int pos = 0;
void setup() {
   myservo.attach(8);
   pinMode(buttonPin, INPUT);
 }
 void loop(){
   buttonState = digitalRead(buttonPin);
   if (directionState == 0){
     if (buttonState == HIGH) {
       directionState = 1;
```

```
        for(pos = 0; pos < 90; pos += 1)
        {
          myservo.write(pos);
          delay(15);  // waits 15ms to reach the position
        }
      }
    } else if (directionState == 1) {
      if (buttonState == HIGH) {
        directionState = 0;
       for (pos = 90; pos>=1; pos -=1)
        {
          myservo.write(pos);
          delay(15);
        }
      }
    }
  }
```

(Ghosh, 2017).

Appendix I: Combined Code for all Motors

```
#include <Wire.h>
#include <MPU6050.h>
#include <Servo.h>
Servo sg90x;
int servo_pinx = 5;
Servo sg90y;
```

```
int servo_piny = 3;

Servo handservo;

const int handservoPin = 7;

const int buttonPin = 9 ;

int buttonState = 0 ;

int handState = 0 ; // 0 = open;  1 = close

int buttonDelay = 0 ; //

MPU6050 sensor ;

int16_t ax, ay, az, accXAngle, accYAngle ;

int16_t gx, gy, gz ;

int16_t gyroXrate, gyroYrate, timer, gyroXAngle, gyroYAngle,
compAngleX, compAngleY, accXangle, accYangle ;

int16_t Xmove, Ymove ;

int16_t XAngle, YAngle ;

int16_t previousTime, currentTime, elapsedTime ;

float ap=0.80 ;

int accYOffset = 0 ;

int accXOffset = -10 ;

void setup ( )

{

sg90x.attach ( servo_pinx );

sg90y.attach ( servo_piny );

handservo.attach ( handservoPin ) ;

sg90x.write(90) ;

sg90y.write(90) ;

handservo.write(135) ;
```

```
pinMode(buttonPin, INPUT) ;
Wire.begin ( );
Serial.begin  (9600);
Serial.println  ( "Initializing the sensor" );
sensor.initialize ( );
Serial.println (sensor.testConnection ( ) ? "Successfully Connected"
: "Connection failed");
delay (1000);
Serial.println ( "Taking Values from the sensor" );
delay (1000);
}
void loop ( )
{
buttonState = digitalRead(buttonPin); // read the button state
previousTime = currentTime;        // Previous time is stored before
the actual time read
currentTime = millis();            // Current time actual time read
elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000
to get seconds
sensor.getMotion6 (&ax, &ay, &az, &gx, &gy, &gz);
// mx = map (ax, -17000, 17000, 180, 0) ;
// my = map (ay, -17000, 17000, 180, 0) ;
accXAngle = (atan(ay / sqrt(pow(ax, 2) + pow(az, 2))) * 180 / PI) +
accXOffset ;
accYAngle = (atan(-1 * ax / sqrt(pow(ay, 2) + pow(az, 2))) * 180 /
PI) + accYOffset ;
Serial.print ("Acc  X angle = ") ;
Serial.print (accXAngle) ;

Serial.print ("\t Acc  Y angle = ") ;

Serial.println (accYAngle) ;

gyroXrate = gx / 131 ;

gyroYrate = gy / 131 ;

gyroXAngle = XAngle + gyroXrate*elapsedTime ;

gyroYAngle = YAngle + gyroYrate*elapsedTime ;

Serial.print ("Gyro X angle = ") ;

Serial.print (gyroXAngle) ;

Serial.print ("\t Gyro Y angle = ") ;

Serial.println (gyroYAngle) ;

XAngle = (ap * gyroXAngle) + ((1-ap) * accXAngle) ;  // combine both
angle measurements

YAngle = (ap * gyroYAngle) + ((1-ap) * accYAngle) ;

Serial.print ("X angle      = ") ;

Serial.print (XAngle) ;

Serial.print ("\t Y angle      = ") ;

Serial.println (YAngle) ;

Serial.println ("------------------------") ;

Serial.print ("Servo X pos  = ") ;

Serial.print (sg90x.read()) ;

Serial.print ("  Servo Y pos = ");

Serial.println (sg90y.read()) ;

Ymove = sg90y.read() - YAngle ;

Xmove = sg90x.read() + XAngle ;

if (abs(XAngle) > 1) {
  sg90x.write (Xmove) ;
  Serial.print ("Rotate motor X to ");
  Serial.println (Xmove);
  delay (50);
  Serial.println ("Reset angle") ;
  XAngle = 0;
  currentTime = millis();
}
else {
  delay (10) ;
} ;
if ((abs(YAngle) > 1) && (abs(sg90y.read()-90) <= 35)) {
  sg90y.write (Ymove) ;
  Serial.print ("Rotate motor Y to ");
  Serial.println (Ymove);
  delay (50);
  Serial.println ("Reset angle") ;
  YAngle = 0;
  currentTime = millis();
}
else if ((sg90y.read() > 125) && (YAngle > 0)) {
  sg90y.write (Ymove) ;
  Serial.print ("Rotate motor Y to ");
  Serial.println (Ymove);
  delay (50);
  Serial.println ("Reset angle") ;
  YAngle = YAngle / 4;
  currentTime = millis();
}
else if ((sg90y.read() < 65) && (YAngle < 0)) {
  sg90y.write (Ymove) ;
  Serial.print ("Rotate motor Y to ");
  Serial.println (Ymove);
  delay (50);
  Serial.println ("Reset angle") ;
  YAngle = YAngle / 4;
  //currentTime = millis();
};
if ((buttonDelay <= 0) && (buttonState == HIGH)) {
  if (handState == 0) {
    handState = 1 ;
  } else if (handState == 1) {
    handState = 0 ;
  }
  Serial.println ("--> Button Pressed") ;
  buttonDelay = 3 ;
} else if (buttonState == LOW) {
  buttonDelay = buttonDelay - 1 ;
}
if ((handState == 1) && (handservo.read() > 60)) {
  handservo.write(handservo.read() - 15) ;
} else if ((handState == 0) && (handservo.read() < 135)) {
  handservo.write(handservo.read() + 15) ;
}
}
```