Haptic Feedback Glove

Ben Tures
Nicholas Minton

SUBSYSTEM REPORT

REVISION – 0
23 November 2020

SUBSYSTEM REPORT
FOR
Haptic Feedback Glove

PREPARED BY:

Team 64	11/23/2020
Author	Date

APPROVED BY:

Nicholas Minton	11/23/2020
Project Leader	Date
John Lusher, P.E.	Date
Andrew Miller	Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	11/23/20	Nicholas Minton	NM BT	Initial Release

Table of Contents

List of Tables

List of Figures

1. Introduction

The haptic feedback glove is a practical tool to emulate real-space object interaction, allowing for greater immersion in virtual environments. The system is broken down into 3 Subsystems: glove assembly, hand tracking, and the virtual environment. While each subsystem has been designed, our plan for integration has been delayed because of unexpected results from the electrostatic brakes.

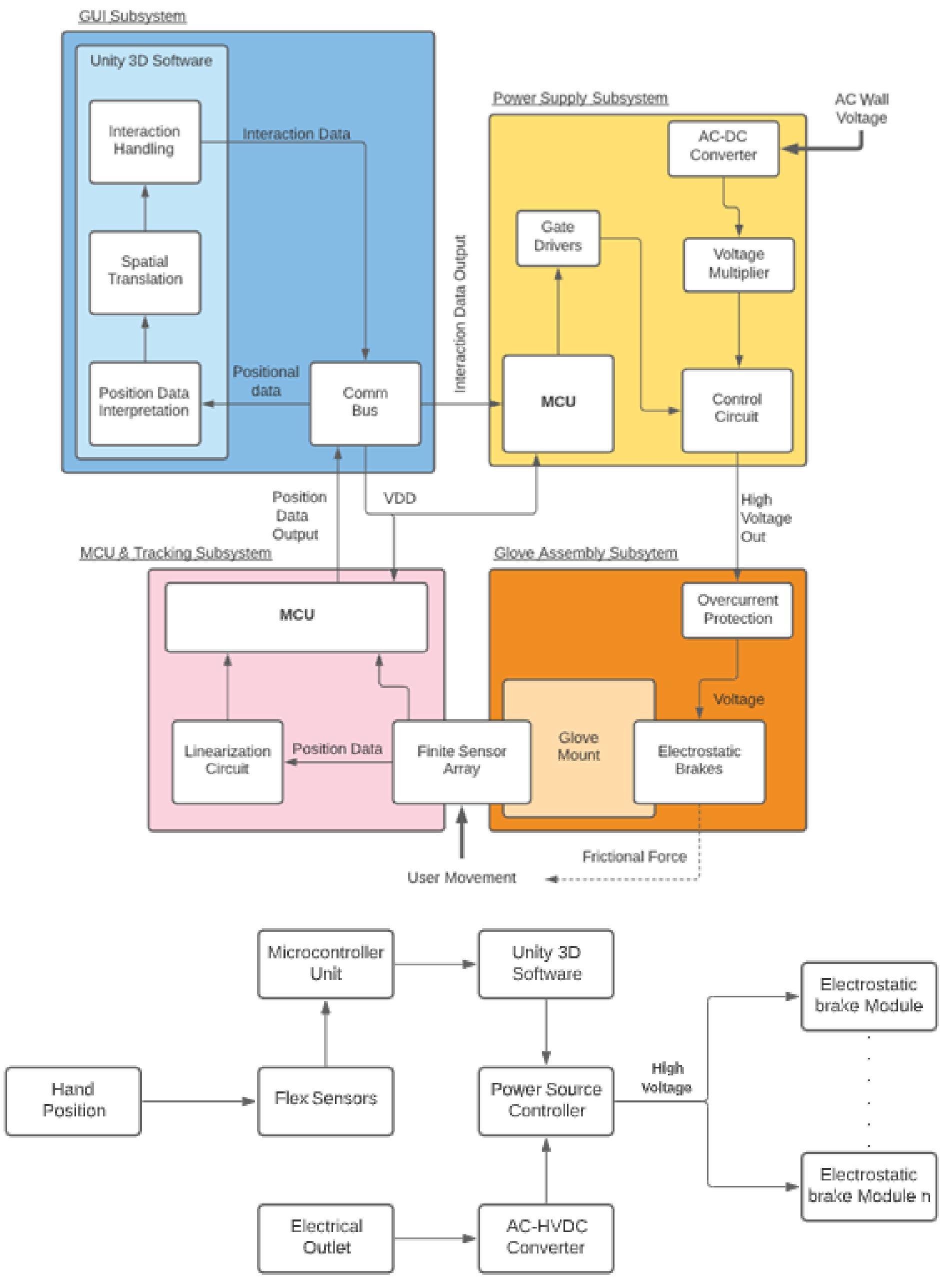


Figure 1. Haptic Glove Core System Interconnect Diagram

2. Hand Tracking Subsystem Report

2.1. Subsystem Introduction

The hand tracking subsystem is designed to provide hand and finger location data to the virtual environment. The subsystem is meant to track finger rotation as well as x, y, and z movement of the hand. Testing has confirmed that these positions can be tracked accurately.

2.2. Subsystem Details

The hand tracking subsystem is made of an array of sensors, an MCU, and voltage divider circuits for each resistor. The sensors include two bend resistors for each finger and a single IMU. The IMU is a spark fun MPU6050. The bend resistors measure rotation of each finger at the proximal and middle joints. The IMU tracks the orientation of the hand and is attached at the base of the palm.

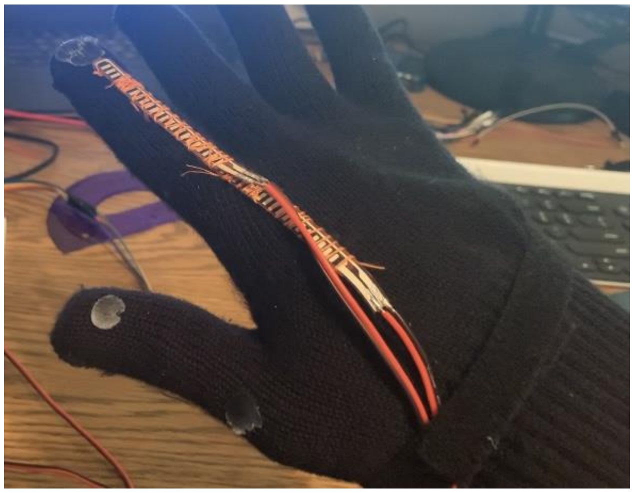


Figure 2: Hand Tracking Setup

2.3. Subsystem Validation

2.3.1. Bend Resistors

The bend resistors were tested on hand for rotation about the proximal and middle joint of the index finger. The measured rotation data was taken with the resistors on hand and read from the serial port my MCU is writing to. During this test, I lined each joint of my finger up with a protractor to assure an accurate actual rotated angle was accomplished.

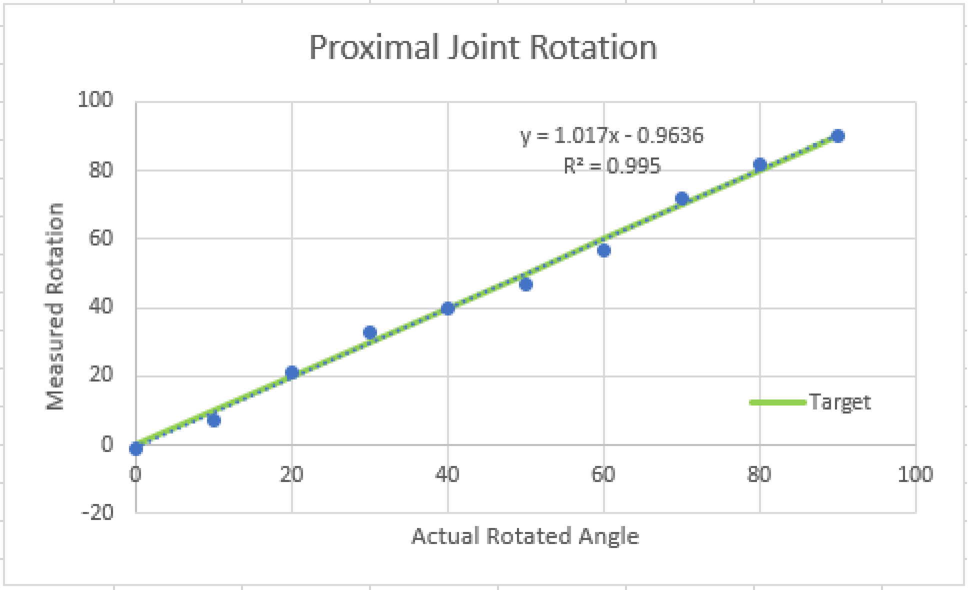


Figure 3: Proximal Joint Rotation

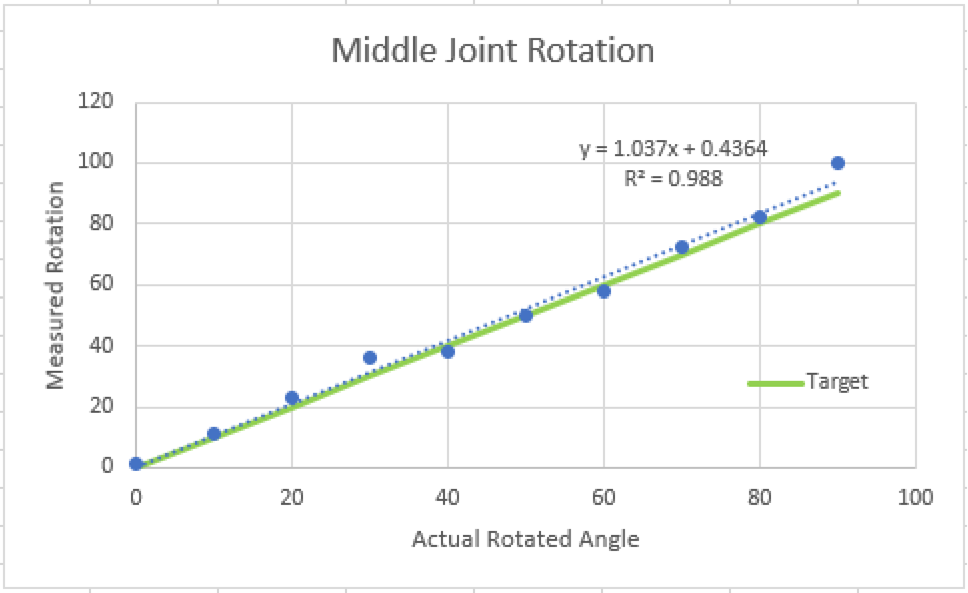
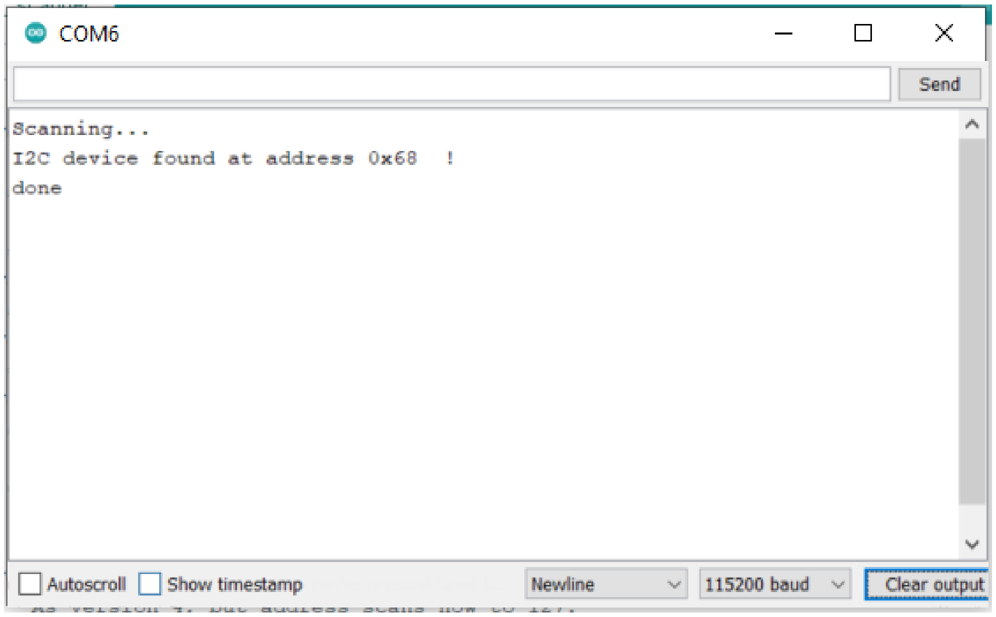


Figure 4: Middle Joint Rotation

2.3.2. Inertial Measurement Unit (IMU)

The first item to validate for the IMU was establishing the I2C protocol connection. This is done by connecting the SCL and SDA pins of the IMU and MCU. Then running a simple script that checks for an I2C device at each possible byte address (See appendix B).



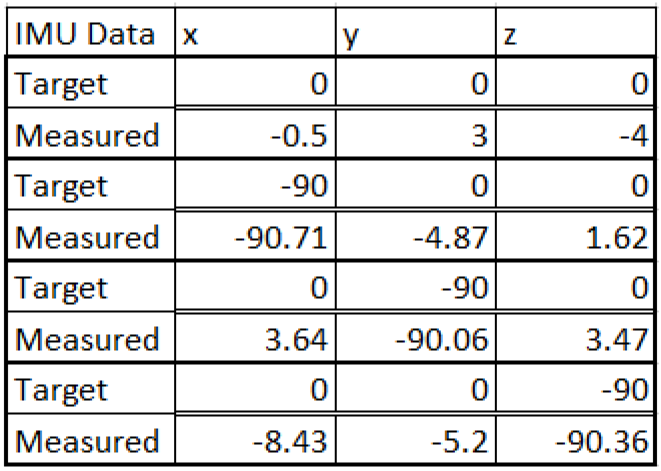
```
Scanning...
I2C device found at address 0x68  !
done
```

Figure 5: I2C Device Connection

The next step for the IMU was calculating values for rotation in pitch, roll, and yaw (y, x, z). To get these values, we utilize data from the IMU's gyroscope and accelerometer. The gyroscope measurement for change in degrees per second is multiplied by elapsed time to get the current angle in each direction. IMU gyroscope have errors in each direction. This Error could be found and corrected for by taking note of the returned values while the IMU is stationary. Our device had a gyroscope error of -2, -1.95, -0.6 in the x, y, and z directions respectively. The accelerometer's measurement of acceleration due to gravity in each direction is then converted into angles in the x, y and z direction.

$$\theta_{x,accel} = tan^{-1}\left(\frac{Accel_y}{\sqrt{{Accel_x}^2 + {Accel_z}^2}}\right) * \frac{180}{\pi}$$

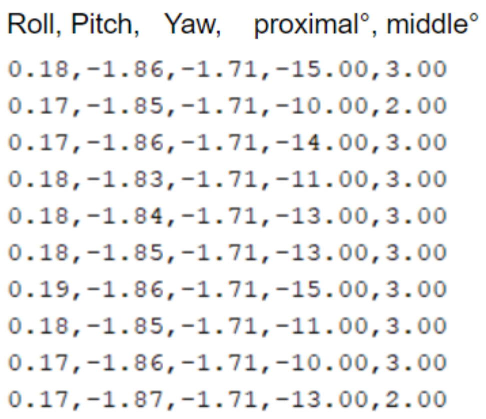
A similar calculation can be done for the y direction's angle. Roll and pitch is then calculated using a weighted average of the accelerometer and gyroscope measured angle. Yaw's calculation relies exclusively on the gyroscope calculation. Therefore, it is incredibly important to accurately account for the gyroscope error in the z direction. The code detailing this calculation can be found in the appendix B. Results from IMU angle calculations can be seen in the figure below. I account the slight deviation in the measured values to be from the difficulty of accurately being able to rotate my hand to the precise orientation.



IMU Data	x	y	z
Target	0	0	0
Measured	-0.5	3	-4
Target	-90	0	0
Measured	-90.71	-4.87	1.62
Target	0	-90	0
Measured	3.64	-90.06	3.47
Target	0	0	-90
Measured	-8.43	-5.2	-90.36

Figure 6: IMU Results

Last, the pitch, roll, and yaw data is combined with the bend resistor data and written to the serial port as detailed in the ICD. The formatted output is shown below.



```
Roll, Pitch,   Yaw,    proximal°, middle°
0.18,-1.86,-1.71,-15.00,3.00
0.17,-1.85,-1.71,-10.00,2.00
0.17,-1.86,-1.71,-14.00,3.00
0.18,-1.83,-1.71,-11.00,3.00
0.18,-1.84,-1.71,-13.00,3.00
0.18,-1.85,-1.71,-13.00,3.00
0.19,-1.86,-1.71,-15.00,3.00
0.18,-1.85,-1.71,-11.00,3.00
0.17,-1.86,-1.71,-10.00,3.00
0.17,-1.87,-1.71,-13.00,2.00
```

Figure 7: Hand Tracking MCU Output

2.4. Subsystem Conclusion

The hand tracking subsystem met the requirements detailed in the FSR. Once the values were passed into the virtual environment, it was easy to reconfirm that values were being calculated accurately by passing the eye test. The next steps for this subsystem rely in continued integration with the virtual environment. The HTC Vive tracker is the final component in hand tracking subsystem. This tracker is designed to interface with unity and should be relatively easy to plug and play. I anticipate integration with the force feedback and scaling up to all five fingers will reveal areas of need for fine tuning the tracking. An alternative that could be worth looking into is motion tracking through video detection.

3. Virtual Environment Subsystem Report

3.1. Subsystem Introduction

The virtual environment subsystem has been designed in Unity. Inputs from the hand tracking subsystem are operated on to allow the user to interact with objects in the virtual space. IN our environment, there are two types of objects: rigid and soft. The physics engine and coded calculation translate information to an output that will control the power supply and further the electrostatic brakes.

3.2. Subsystem Details

Where possible, portions of this subsystem have been borrowed from the 2019 – 2020 senior design team. In order to be effective at making changes to the code, time was spent understanding the interconnects and structure of last year's code. The key job of this subsystem is providing a calculation for force to be used for feedback to the user's hand. Another important piece is rendering a realistic human hand. The second goal was unable to be accomplished, so visuals in this report will be that of the highlighted hand game object (the green outline in the below figures). The virtual environment consists of a hand and a table with a few items for the user to interact with on it.

3.3. Subsystem Validation

The first item to validate for the virtual environment was the physics of the hand game object.

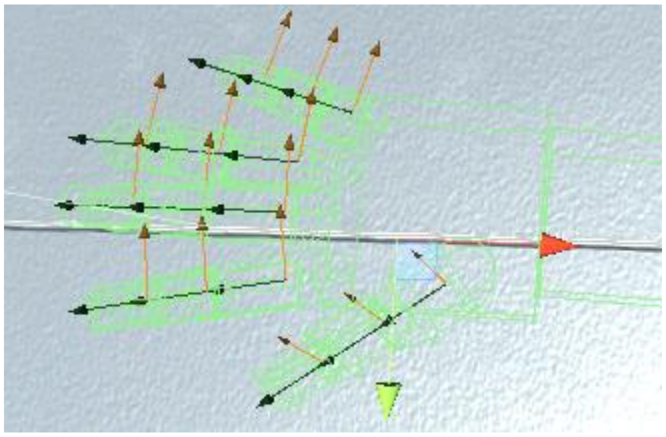
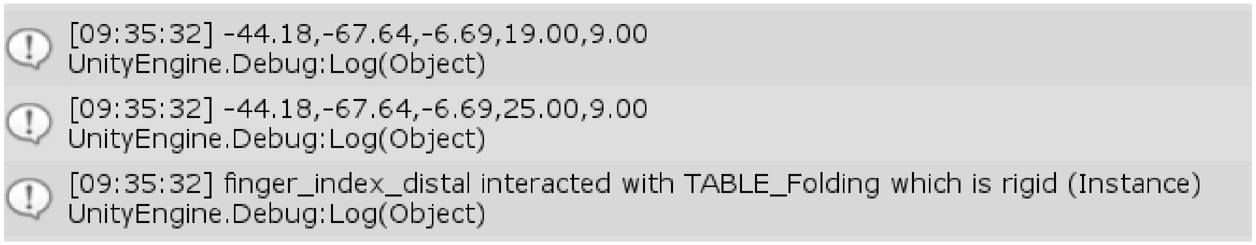


Figure 8: Hand Game Object

The hand can be operated with sliders manipulating the pitch, roll, and yaw of the overall hand or individual finger joints. When a portion of the finger intersects (or tries to intersect) with another game object, a force is returned that is split between the joints of the finger and hypothetical arm. As of now, this is simply being split up evenly between the finger's middle joint, proximal joint and the arm. An edge test case for validating the hands correct physical properties was forcing the hand to rotate into a rigid object. This causes maximum force to be applied on all finger joints. When the hand is rotated too far, it unrealistically snaps back into place. This behavior is acceptable because feedback would indicate to the user that they should stop trying to push through a rigid object.

Another Item to test was pushing a finger into a soft object that was placed on top of a rigid object. An interesting behavior was that quickly after pushing into the soft object, the force became the same as the rigid object underneath it. This models well real-world conditions when soft objects are atop rigid ones well. Lastly, half force was detected when the finger intersected a soft object on its own.

The next portion was to verify was proper handling of inputs from the hand tracking subsystem. Below you can see the unity regurgitating the hand tracking data and printing it to the console.



```
[09:35:32] -44.18,-67.64,-6.69,19.00,9.00
UnityEngine.Debug:Log(Object)
[09:35:32] -44.18,-67.64,-6.69,25.00,9.00
UnityEngine.Debug:Log(Object)
[09:35:32] finger_index_distal interacted with TABLE_Folding which is rigid (Instance)
UnityEngine.Debug:Log(Object)
```

Figure 9: Unity Console Output

The last line of the console output also demonstrates the integrated unity/hand tracking system identifying when the index finger interacts with the table. Below are visuals which visually show the virtual hand reacting to hand tracking subsystem input.

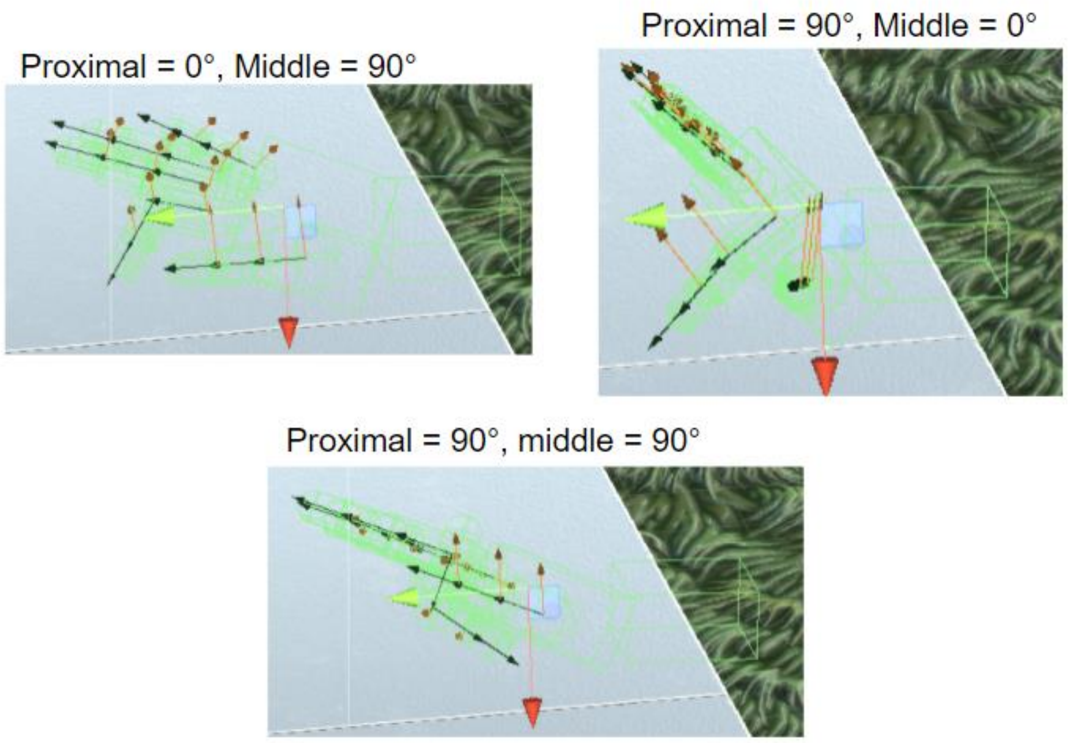
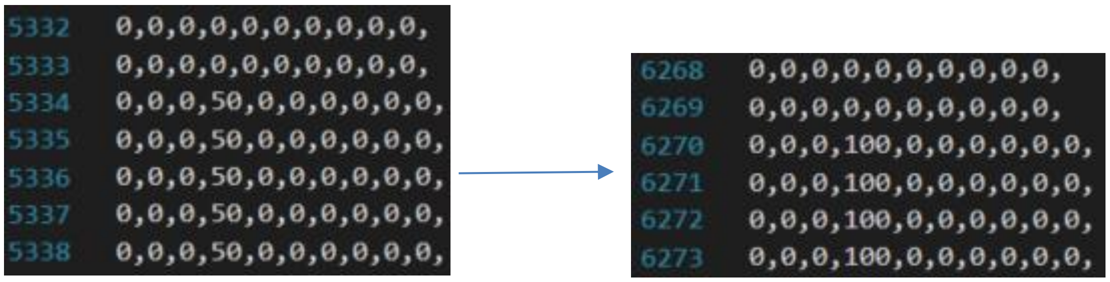

Proximal = 0°, Middle = 90°
Proximal = 90°, Middle = 0°
Proximal = 90°, middle = 90°

Figure 10: Unity with Hand Tracking Visual

The last point of validation was to show the output to be given to the power control MCU. As seen in the demo video, the index finger was depressed into a soft object, brought back up and then the process repeated for a rigid object. The below results were taken from a text file that was written to by unity as opposed to the serial port the MCU will read from.



```
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,50,0,0,0,0,0,0,
0,0,0,50,0,0,0,0,0,0,
0,0,0,50,0,0,0,0,0,0,
0,0,0,50,0,0,0,0,0,0,
0,0,0,50,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,
0,0,0,100,0,0,0,0,0,0,
0,0,0,100,0,0,0,0,0,0,
0,0,0,100,0,0,0,0,0,0,
0,0,0,100,0,0,0,0,0,0,
```

Figure 11: Output for Power Control

3.4. Subsystem Conclusion

The above validation proves unity subsystem is in an operational state. The major piece of work left on this subsystem is rendering a realistic looking hand. Though we hoped to have this complete by the end of this semester, it should be noted that interfaces with the hand tracking and power supply MCU's are complete. The last step of work before the hand tracking and virtual environment subsystems are fully integrated is interfacing the HTC Vive tracker. Therefore, we can say that the virtual environment subsystem is ready for integration.

4. Glove Assembly/Electrostatic Brakes Subsystem Report

4.1. Subsystem Introduction

The Electrostatic Braking subsystem is designed to provide resistive haptic feedback for the Haptic Glove. This subsystem is comprised of a high voltage DC power supply and the electrodes that will generate the feedback force. The DC power supply supplies large DC voltages to the electrodes which will become attracted to one another by electrostatic forces.

4.2. Power Supply Details

The power supply is comprised of a 4 stage Cockroft-Walton voltage multiplier and an output voltage control circuit for each finger. A schematic of the multiplier circuit is shown below. The multiplier is able to generate the large DC voltage required to create electrostatic force between the electrodes. Safety is a concern with the high voltage generated by the power supply as the electrodes will be mounted very near the user's hands. The Cockroft-Walton multiplier works well for this application, because the impedance of the pump capacitors limits the output current. A challenge with integrating the multiplier circuit is the potential for a short circuit fault. During early testing of the brake assembly the dielectric between the electrodes tore and shorted the high voltage DC output to ground. This short damaged the power supply

thus limiting the output voltage significantly. Minimizing the chance of a short circuit fault would be a key consideration when designing the electrodes for the braking system.
The output voltage is managed by a control circuit and an MSP432 microcontroller. The microcontroller outputs a control that can alter the output voltage of the control circuit by modifying its duty cycle. Being able to select from a range of different voltages at the output allows for different levels of forces to be simulated leading to a more immersive experience for the user.

4.3. Electrostatic Brakes Details

The electrostatic brakes themselves are comprised of two electrodes separated by a thin dielectric. The power supply supplies a DC voltage to one electrode, while the other is tied to the common ground. This voltage differential should be large enough that the plates are then pulled together. Once actuated, the plated remain separated by a thin dielectric material, in this case a polyimide tape was selected. Previous iterations of this project believed that wax paper would be ideal for this dielectric as it possessed more favorable mechanical properties, however to maximize the possible force generated polyimide was selected for its more favorable electrical properties (relative permittivity, breakdown voltage, etc.). The electrostatic force pulling the parallel plated toward one another multiplied by the coefficient of friction of the dielectric would translate to the maximum shear force that would be experienced if one were to move the plates.
To integrate the electrostatic brakes a 3D-Printed structure was designed. This structure would allow the shear generated by the brakes to be translated to the user's hand. The structure provides points for the electrodes to be mounted on top of the user's hand so to not impede regular movement. To maximize brake actuation time, the structure was designed to maintain minimal unpowered distance between the two electrodes of the braking subsystem. When fully integrated this structure would likely be fixed outside of a protective glove for the safety of the user in the event of an unexpected fault in the braking subsystem.

4.4. Subsystem Validation

The power supply used was already validated by the previous iteration of the project, so the focus was placed on validation of the braking subsystem.
To design an appropriate electrode the following formula for the electrostatic compression force was considered.

$$F_{\mathrm{N}} = \frac{\varepsilon_0 A}{2} \frac{\varepsilon_r V^2}{d^2} = \frac{\varepsilon_0 A}{2} \varepsilon_r E^2$$

Figure 11: Electrostatic Compression Force Equation

Firstly, the constraints of the system were considered. The maximum output of the power supply (V) was measured to be near 1350 VDC setting the best-case scenario for the force generated. The next constraint was the size of the electrostatic brake electrodes (A). For our planned design using small enough electrodes to fit over each finger on the user's hand would allow the mounting mechanism to be lightweight and unimposing for the user. An electrode overlap area of 19.5 cm^2 was then selected as the initial test value. The parameter with the most area for experimentation was the selection of the dielectric material. The relative permittivity of the dielectric material would decide the value for epsilon-r in the above equation. The thickness of the selected dielectric material would also set the minimum distance between the plates when in operation (d). Minimizing this distance was critically in the design process as the compression force was inversely related to the square of this distance.

With the listed parameters considered the first validation test was prepared with an overlap area of 19.5 cm^2 at a voltage of 1305 VDC and double layer of polyimide for the dielectric to prevent shorting (50 µm separation). During this test no shear force was able to be measured. Another test was run with the same configuration, but with only one layer of polyimide (25 µm separation). Again, no force was able to be measured as they were too small. These results were then compared with the following calculated values.

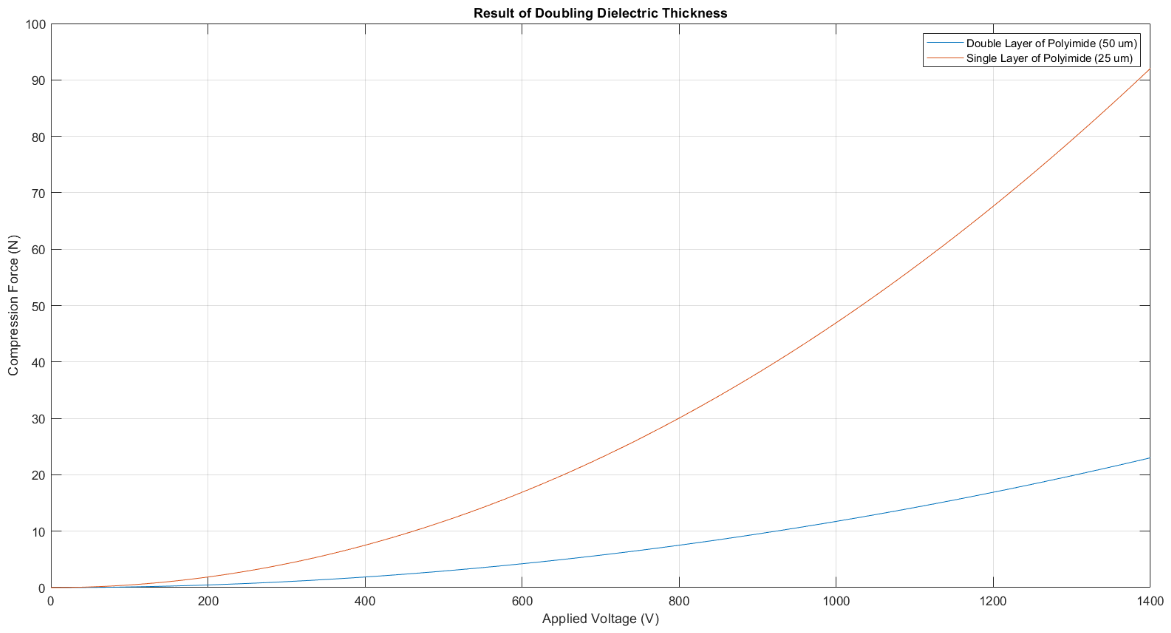


Figure 12: Theoretical Result of Doubling Dielectric Thickness

The calculated values were much higher than what was being observed with the testing setup. The initial method to adhere the polyimide to the electrode was to wrap non-adhesive polyimide film around the electrode and then to tape it. This was identified as potentially detrimental to the electrostatic force by leaving an air gap between the electrode and dielectric. To rectify this issue Kapton tape was used to adhere the dielectric material directly to the electrodes thus minimizing the air gap/distance when the electrodes were in operation.

The tests were repeated with this new dielectric, but the results were the same: no generated force.

In an attempt to generate a measurable force, the overlap area of the electrodes was considered. While increasing the size of the electrodes would modify the overall structure of the glove it was deemed a necessary tradeoff to consider in order to achieve the defined operation of the glove. It can be seen in the following figure that increasing the overlap area of the electrodes should lead to an increase in the compression force.

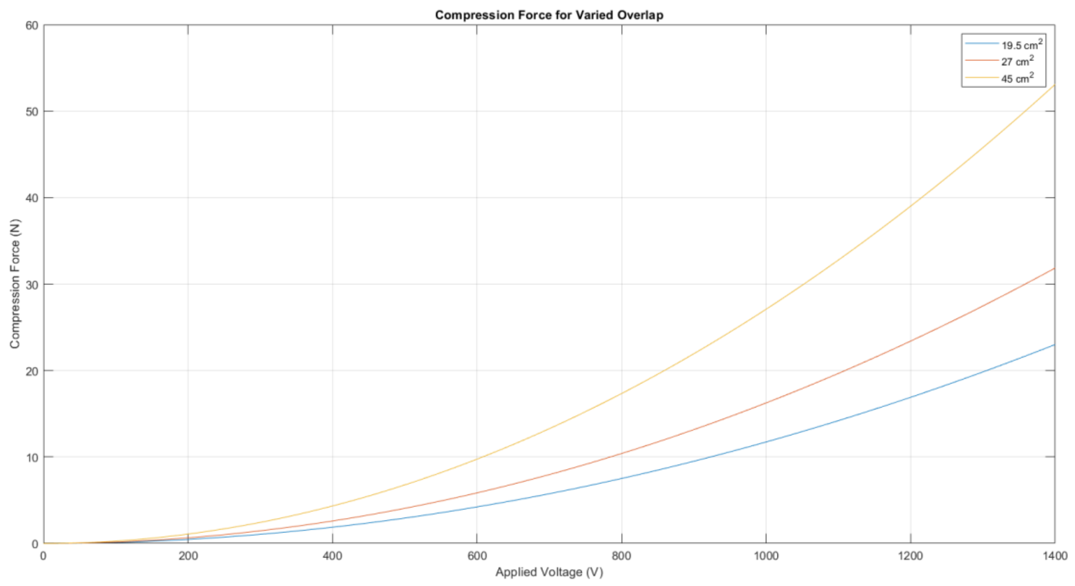


Figure 13: Voltage Response of Varied Electrode Overlap Areas

The three overlap sizes in the above figure were tested empirically. The overlap of 27 cm^2 saw the same results of the 19.5 cm^2 overlap. The 45 cm^2 overlap could be seen to flex 0.5 cm when the power was applied but the compression force was not enough to hold the plates together and so force was insignificant for the application. The electrodes will need further development to create an appropriate level of force for the application.

The mounting assembly designed for the ideal plate size was also tested. The 3D-Printed assembly allowed for free movement of the hand with minimal resistance. The maximum distance measured between the plates was 0.003 meters. The mounting system for one finger can be seen in the figure below.

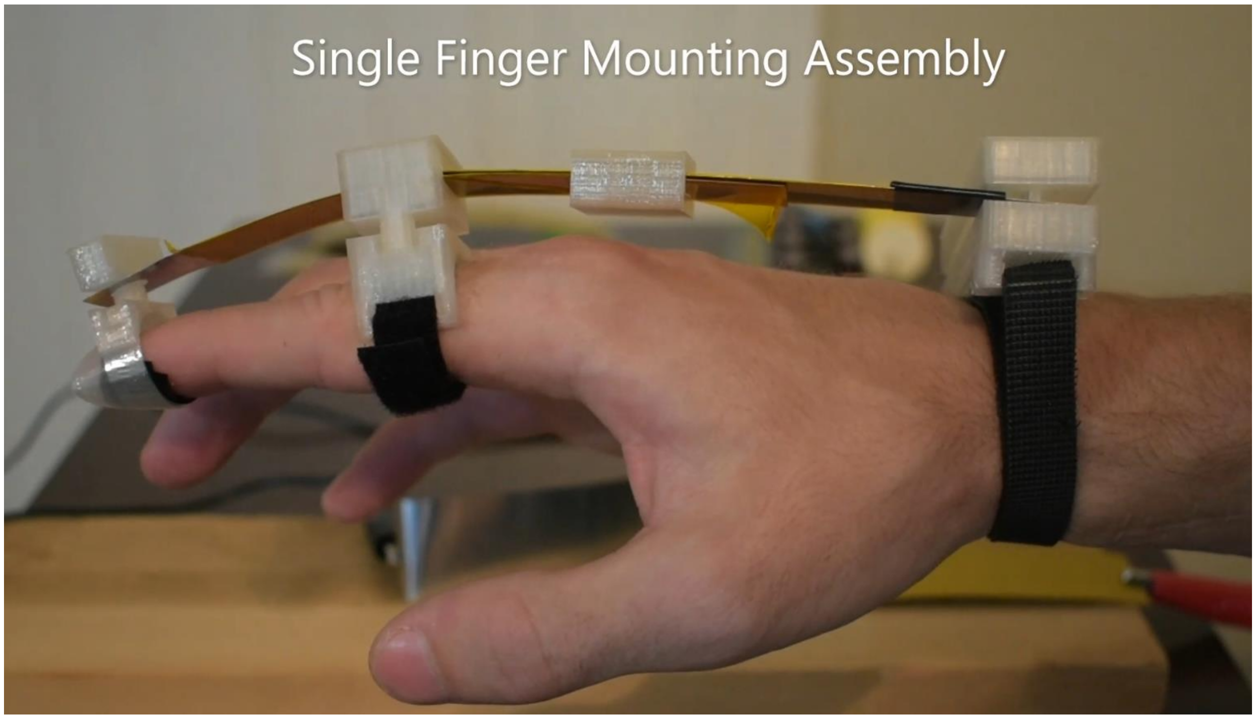


Figure 14: Electrode Mounting Assembly

4.5. Subsystem Conclusion

The power supply and mounting assemblies were shown to work correctly. The electrostatic brakes still need to be significantly improved before they are integrated into the full haptic glove. The operation of the electrostatic brakes is critical to the operation of the haptic glove and has been given full priority in further development of this project. Future modifications to be considered for the glove assembly would be to select a dielectric coating rather than an adhesive film so as to increase the surface area that comes in contact with the electrode. Modifying the output of the power supply is also in consideration, but will require further research. It has been proven by other studies that generation of large electrostatic forces is possible with a small electrode overlap area, so further refinement of the haptic glove is possible.

Appendix A: Acronyms and Abbreviations

MCU	Microcontroller Unit
UI	User Interface
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
ICD	Interface Control Document
MCP	Metacarpophalangeal
DIP	Distal Interphalangeal
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
VR	Virtual Reality
mA	Milliamp
μA	Microamp
mW	Milliwatt
V	Volts
N	Newton
ft	Feet
cm	Centimeter
lbs	Pounds
μm	Micrometer
Hz	Hertz

Appendix B: Code

I: I2C Address finder

```
// This sketch tests the standard 7-bit addresses
// Devices with higher bit address might not be seen properly.
//

#include <Wire.h>

void setup() {
  Wire.begin();

  Serial.begin(9600);
  while (!Serial); // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}

void loop() {
  int nDevices = 0;

  Serial.println("Scanning...");

  for (byte address = 1; address < 127; ++address) {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    // Serial.println("\ncheckpoint1");
    byte error = Wire.endTransmission();
    // Serial.println("\ncheckpoint2");
    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address < 16) {
        Serial.print("0");
      }
      Serial.print(address, HEX);
      Serial.println("  !");

      ++nDevices;
    } else if (error == 4) {
      Serial.print("Unknown error at address 0x");
      if (address < 16) {
        Serial.print("0");
      }
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  } else {
    Serial.println("done\n");
  }
  delay(5000); // Wait 5 seconds for next scan
}
```

II: Hand Tracking code

```
/*
   Combined and edited by Nick Minton
   Sources:
   Arduino and MPU6050 Accelerometer and Gyroscope Sensor Tutorial
   by Dejan, https://howtomechatronics.com


*/
#include <Wire.h>
/******************************************************************************
Create a voltage divider circuit combining a flex sensor with a 47k resistor.
- The resistor should connect from A0 to GND.
- The flex sensor should connect from A0 to 3.3V
As the resistance of the flex sensor increases (meaning it's being bent), the
voltage at A0 should decrease.

Development environment specifics:
Arduino 1.6.7
******************************************************************************/
//const bool TESTING = true;
const int sensorPin[2] = {A0, A1}; // Pins connected to voltage divider output
const int n_sensors = sizeof(sensorPin)/sizeof(sensorPin[0]);
float bend_angles[n_sensors];

// Measure the voltage at 5V and the actual resistance of your
// 47k resistor, and enter them below:
const float VCC = 4.98; // Measured voltage of Ardunio 5V line
const float R_DIV = 47500.0; // Measured resistance of 47k resistor

// Upload the code, then try to adjust these values to more
// accurately calculate bend degree.
const float STRAIGHT_RESISTANCE[2] = {32500.0, 23500.0}; // resistance when straight
const float BEND_RESISTANCE[2] = {48000.0, 56000.0}; // resistance at 90 deg

const int MPU = 0x68; // MPU6050 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;
float temp = 0;
void setup() {
  Serial.begin(19200);
  for(int i = 0; i < n_sensors; i++){
    pinMode(sensorPin[i], INPUT);
  }
  Wire.begin();                      // Initialize comunication
  Wire.beginTransmission(MPU);       // Start communication with MPU6050 // MPU=0x68
  Wire.write(0x6B);                  // Talk to the register 6B
  Wire.write(0x00);                  // Make reset - place a 0 into the 6B register
  Wire.endTransmission(true);        //end the transmission
  /*
  // Configure Accelerometer Sensitivity - Full Scale Range (default +/- 2g)
  Wire.beginTransmission(MPU);
  Wire.write(0x1C);                  //Talk to the ACCEL_CONFIG register (1C hex)
  Wire.write(0x10);                  //Set the register bits as 00010000 (+/- 8g full scale range)
  Wire.endTransmission(true);
  // Configure Gyro Sensitivity - Full Scale Range (default +/- 250deg/s)
  Wire.beginTransmission(MPU);
  Wire.write(0x1B);                   // Talk to the GYRO_CONFIG register (1B hex)
  Wire.write(0x10);                   // Set the register bits as 00010000 (1000deg/s full scale)
  Wire.endTransmission(true);
  delay(20);
  */
  // Call this function if you need to get the IMU error values for your module
  calculate_IMU_error();
  delay(20);
}
void loop() {
  // === Read acceleromter data === //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2
registers
  //For a range of +-2g, we need to divide the raw values by 16384, according to the datasheet
  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
  // Calculating Roll and Pitch from the accelerometer data
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - 0.58; // AccErrorX
~(0.58) See the calculate_IMU_error()custom function for more details
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) + 1.58; //
AccErrorY ~(-1.58)
  // === Read gyroscope data === //
  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
  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Gyro data first register address 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis value is stored in 2 
registers
  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range we have to divide 
first the raw value by 131.0, according to the datasheet
  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
  // Correct the outputs with the calculated error values
  GyroX = GyroX + 2; // GyroErrorX ~(-2)
  GyroY = GyroY + 1.95; // GyroErrorY ~(-1.95)
  GyroZ = GyroZ + 0.6; // GyroErrorZ ~ (-0.6)
  // Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by 
sendonds (s) to get the angle in degrees
  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  yaw =  yaw + GyroZ * elapsedTime;
  // Complementary filter - combine acceleromter and gyro angle values
  roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
  pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;

  //Bend resistor Calcs
  for (int i = 0; i < n_sensors; i++) {
     bend_angles[i] = calculate_bend(i);
  }
  // Adjust valuses to be sent to unity
//  temp = pitch;
//  pitch = yaw;
//  yaw = temp;
  // Print the values on the serial monitor
  Serial.print(roll);
  Serial.print(",");
  Serial.print(pitch);
  Serial.print(",");
  Serial.print(yaw);
  for (int i = 0; i < n_sensors; i++) {
     Serial.print(",");
     Serial.print(bend_angles[i]);
  }
  Serial.println("");

  //calculate_IMU_error();
}
void calculate_IMU_error() {
  // We can call this funtion in the setup section to calculate the accelerometer and gyro data error. From here we will get the error values used in the above equations printed on the Serial Monitor.
  // Note that we should place the IMU flat in order to get the proper values, so that we then can the correct values
  // Read accelerometer values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    // Sum all readings
    AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
    AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 / PI));
    c++;
  }
  //Divide the sum by 200 to get the error value
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  c = 0;
  // Read gyro values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    GyroX = Wire.read() << 8 | Wire.read();
    GyroY = Wire.read() << 8 | Wire.read();
    GyroZ = Wire.read() << 8 | Wire.read();
    // Sum all readings
    GyroErrorX = GyroErrorX + (GyroX / 131.0);
    GyroErrorY = GyroErrorY + (GyroY / 131.0);
    GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
    c++;
  }
  //Divide the sum by 200 to get the error value
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  GyroErrorZ = GyroErrorZ / 200;
  // Print the error values on the Serial Monitor
  Serial.print("AccErrorX: ");
  Serial.println(AccErrorX);
  Serial.print("AccErrorY: ");
  Serial.println(AccErrorY);
  Serial.print("GyroErrorX: ");
  Serial.println(GyroErrorX);
  Serial.print("GyroErrorY: ");
  Serial.println(GyroErrorY);
  Serial.print("GyroErrorZ: ");
  Serial.println(GyroErrorZ);
}
float calculate_bend(int pin_n) {
  // Read the ADC, and calculate voltage and resistance from it
  int flexADC = analogRead(sensorPin[pin_n]);
  float flexV = flexADC * VCC / 1023.0;
  float flexR = R_DIV * (VCC / flexV - 1.0);
  //Serial.println("Resistance: " + String(flexR) + " ohms");

  // Use the calculated resistance to estimate the sensor's
  // bend angle:
  float angle = map(flexR, STRAIGHT_RESISTANCE[pin_n], BEND_RESISTANCE[pin_n],
                   0, 90.0);
  //Serial.println("Bend: " + String(angle) + " degrees");
  //Serial.println();
  return(angle);
}
```