Infinity Control Gauntlet

Ashish Pabba (apabba2), Chris Schodde (schodde3) and Ramakrishna Kanungo (kanungo3)
Team 25
TA: Yifan Chen

Abstract

This report intends to inform the reader of our senior design project, a wearable glove input device that recognizes certain pre-configured gestures. Our glove is equipped with flex sensors and an IMU to obtain finger flexion and orientation data streams, which are processed to recognize and declare a gesture from our gesture set. Our final prototype satisfied all of our defined high-level requirements and was able to achieve a high degree of accuracy in recognizing all gestures with low latency, while having virtually zero false positive or false negative gesture recognition instances during our testing. However, there is room for improvement in terms of variations in accuracy between testers and overall functionality, which is also discussed in detail.

Table of Contents

1. Introduction

1.1 Objective

For certain applications, the issues with conventional input devices such as the mouse and keyboard are evident. **There are numerous such fields that lack a sufficiently intuitive and natural input method, including but not limited to VR technology, video games, CAD/3D modelling, and even basic menu navigation.** For example, in the context of 3D modeling and CAD applications, rotation, zooming, and translation are extremely inconvenient actions. Such a lack of intuitiveness can seriously hamper workflows and negatively affect efficiency and accuracy. Therefore, an advancement or breakthrough in this area would be analogous to the invention of the mouse as a point and click device.

Moreover, another source of inconvenience is that most input devices are external and secondary instruments as opposed to extensions of the body, the latter being overwhelmingly preferable. This is evident in the complete replacement of stylus-based touch screens by capacitive touch screens that can be used with fingers.

With these aspects of the problem space in mind, our proposed solution is to engineer a wearable glove that can be used to recognize the hand-based gestures of the wearer to serve as an input device. Such a solution would enable the user to translate specific and precise gestures to a directive/input command. As previously discussed, the applications for this method of input would be significant. **Therefore, our objective is to essentially develop a glove that can serve as an immersive and intuitive input instrument with applications in several fields.**

1.2 Background

Our goal is to attempt to develop a more intuitive, immersive, and natural method of input in the form of a glove that is capable of recognizing gestures. While there are similar products in development from companies such as HTC and Sony, such products are either nascent, niche, or prohibitively expensive. Furthermore, any viable glove input device is inextricably tied with proprietary hardware such as VR headsets or consoles and is therefore difficult to use as a general-purpose device. Consequently, our product is appealing as an open-source device for which drivers can be written and custom gesture mapping can be made. In order to distinguish itself, our glove solution will have to be both reliable and inexpensive. Another possible argument against the pursuit of the development of such a glove is the alternative possibility of using computer vision to track hand/finger movements. However, this solution can be easily dismissed when the field-of-vision requirements, camera costs, and necessary computer vision hardware and software are contrasted with the simplicity of the glove.

1.3 High-Level Requirements

• Sensor data should be transmitted to the microcontroller with precision and accuracy, with minimal noise.

• Control loop on microcontroller should be able to detect the move right, move left, zoom in, zoom out, and thumbs-up gestures while effectively reducing false positive detection. Gestures should be recognized every 1 seconds i.e., there is a downtime of 1 second between gestures that will be recognized.

• Detected gestures should be transmitted to a computer using a UART transmission and printed in a readable format.

2. Block Diagram

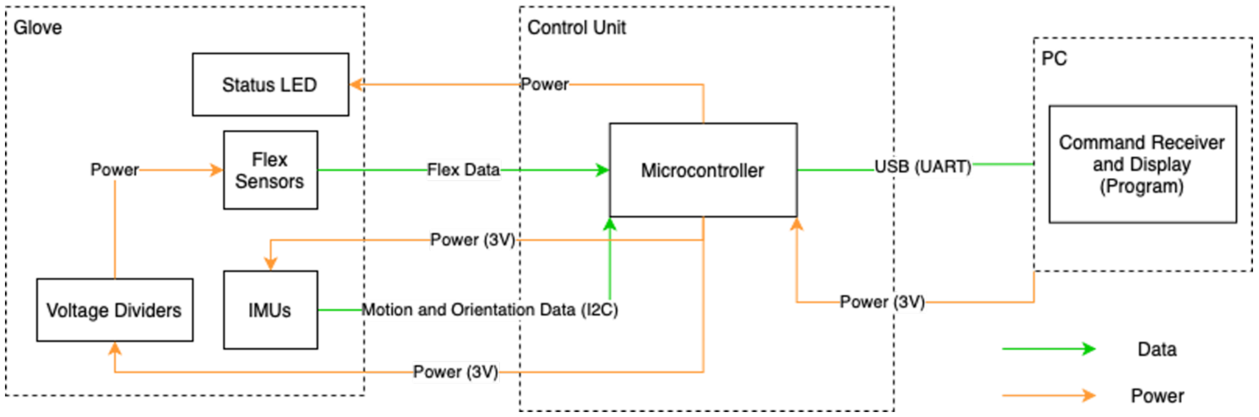


Figure 1:Block Diagram

Figure 1 above illustrates our final block design. In comparison to our initial block design, we decided that all our components could be powered by the computer through our UART module. This led to the elimination of the battery and linear regulator. This reduced the overall size and weight of the glove and was an improvement both aesthetically and functionally. However, if it becomes imperative to pivot to a wireless glove design in the future, we will have to reintroduce a similar power block to be placed onboard the glove.

3. Functional Overview

The ***glove subsystem*** encompasses the sensors mounted on the glove that are used to help detect gestures. And since a gesture is just a combination of different motions happening in parallel the aim of the glove subsystem is to provide the control unit, with the microcontroller, data from different types of sensor values that aim to detect a certain motion.

The ***control unit*** consists of the microcontroller and the analog to digital converter. The ADC is used to convert the voltage divider circuit value from the glove subsystem and send the digital output to the microcontroller. The microcontroller is used to interface with the various sensors and detect gestures, these include rotational motion, translational motion and hand contractions.

The ***PC and power supply unit*** is used to provide power to the various sensors and the microcontroller through the USB port and interact with the microcontroller and report the gestures sensed.

4. Design

4.1 Physical Design

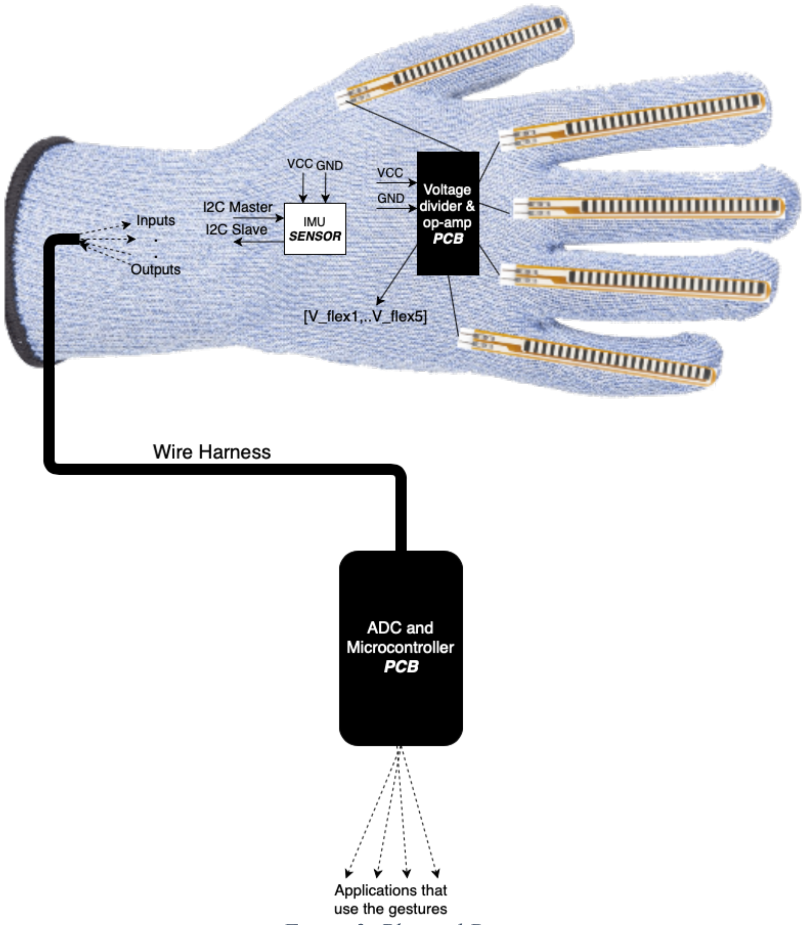
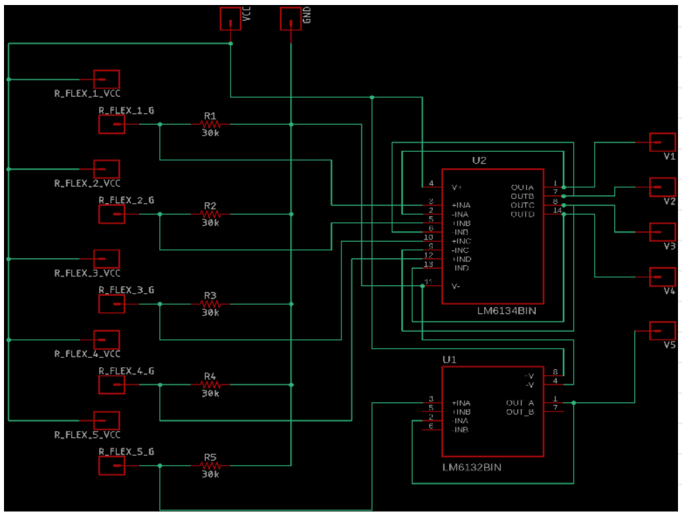

VCC
GND
VCC
GND
Voltage divider & op-amp PCB
I2C Master
I2C Slave
IMU SENSOR
Inputs
Outputs
[V_flex1,..V_flex5]
Wire Harness
ADC and Microcontroller PCB
Applications that use the gestures

Figure 2: Physical Design

4.2 Glove Subsystem

4.2.1 Flex Sensors & Voltage Divider PCB

The flex sensors [1] are placed to capture the contraction of each of the fingers. The flex sensors are connected to another set of resistors in series to form voltage divider circuits. The voltage drop is fed to a voltage follower op amp [2] circuit to avoid source impedance. Then this is routed to the control unit. The routing to the control unit is done using 20 AWG wires that have a low resistance of $10.5\Omega/1000\ feet$ [3]. This is done to avoid noise gained during transmission.



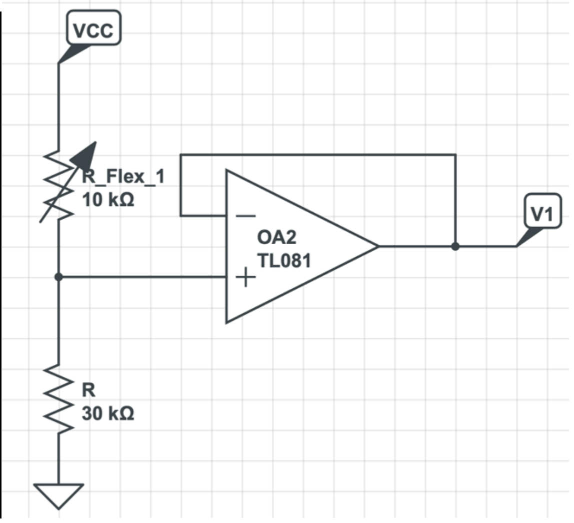
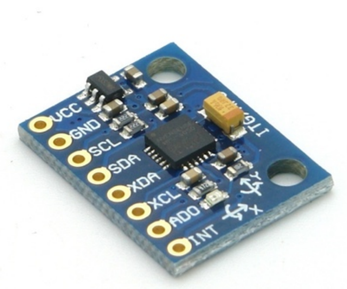


Figure 3: On hand PCB circuit *Figure 4:Individual flex sensor circuit*

4.2.2 IMU

Although we initially selected the Bosch BNO055 IMU, we later chose to use the MPU6050 for increased simplicity and reduced power consumption. The BNO055, however, has advanced sensor fusion and includes a magnetometer, which make it our IMU of choice in the future as we further develop our prototype.

The MPU6050, our chosen breakout-board-mounted IMU, is used to detect rotational and translational motion across six degrees of freedom. An onboard gyroscope and accelerometer are used to the acquisition of the aforementioned data. The IMU relays this sensor data stream by interfacing with the control unit's microprocessor over an I^{2}C line with $2.2\ k\Omega$ resistor pullups for the SDA and SCL lines. The MPU6050 is placed on the glove right below the base of the middle finger. It lays flat along the back of the hand, relaying orientation and acceleration data with the back of the hand as the "zero-plane". The MPU6050 breakout board and its position on the glove are illustrated below in Figures 5 and 6.



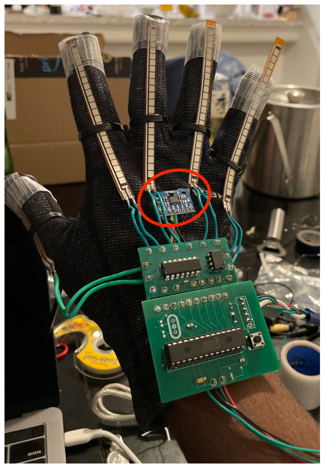


Figure 5: MPU 6050 Breakout Board   *Figure 6: MPU6050 Position on Glove*

More specifically, the MPU6050 provides the following relevant sensor data:

- Angular velocity across 3 axes
- Linear acceleration across 3 axes

Given this sensor data profile, we were able to compute linear acceleration and current orientation accurately recognition of our chosen gesture set.

4.3 Control Unit

4.3.1 Control Unit PCB

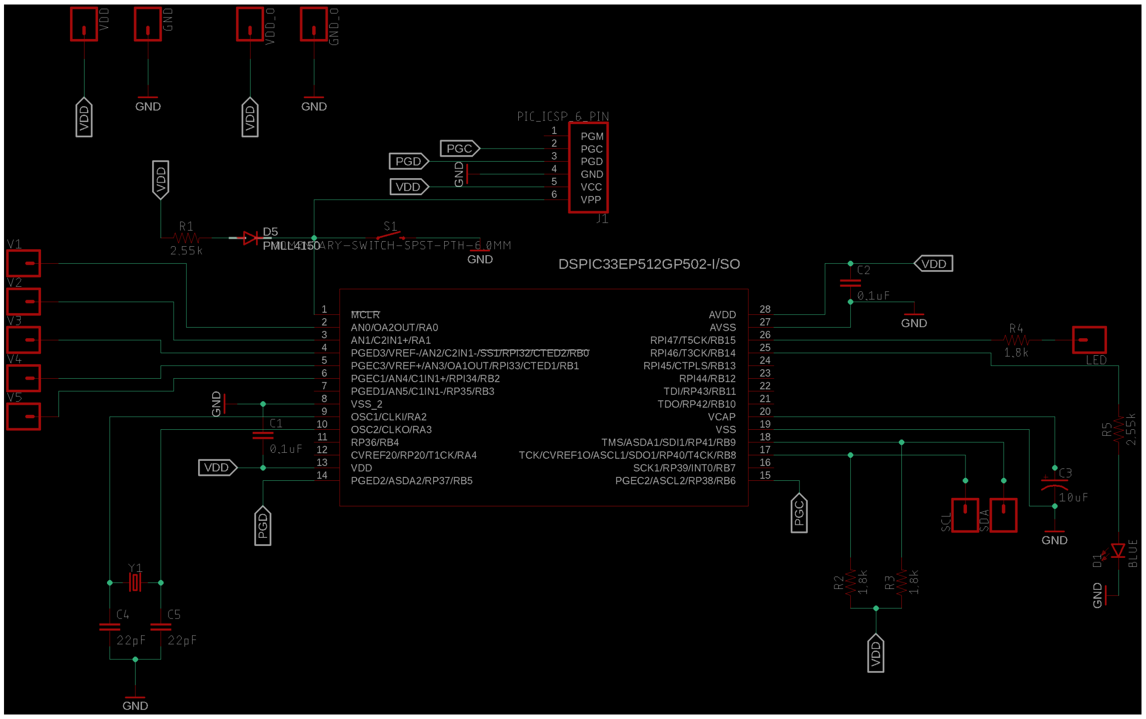


Figure 7: Control Unit Schematic

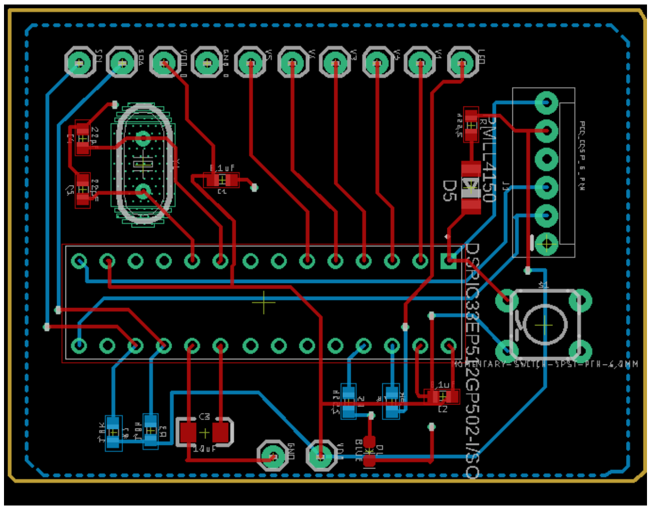


Figure 8: Control Unit Board

Figures 7 and 8 above display the design for the control unit. A header is added to connect the programmer to the microcontroller. The clear pin is directed to the $MCLR$ port. And this is coupled with a pull up resistor and diode. The $MCLR$ port is an active low port; hence it is fed to VCC. There is also a push button that is used to reset the board at will and perform calibration for sensors. The PGC and PGD pins are routed appropriately to program and write into the ROM of the controller.

The way the schematic is set up the board can be powered by either an external power source or by the programmer. All the power ports (VDD VSS) have denoising protective 0.1uF ceramic capacitors running in parallel with the ground and VCC lines. The microcontroller has an internal VRM and the external capacitors across VDD and VSS are for first degree denoising. Further, a tantalum capacitor is added in series with the $VCAP$ and GND nets. This is to assist the internal VRM with denoising and stability which is required for the high current transients seen by the microcontroller. Hence, we can use an unsophisticated DC power supply, and not have to worry about the high current transients. The $AVDD$ and $AVSS$ pins are connected to VCC and GND to provide a reference to the ADC, without which the internal ADC cannot function. There are pull up resistor on the SDA and SCL pin which are required for I^{2}C communication. There is a test LED, and all the analog pins are exposed as header to get analog data from the PCB on the hand. Further, the RX and TX ports are exposed for UART communication with the PC to transmit data when the gesture is detected.

4.3.2 Control Loop

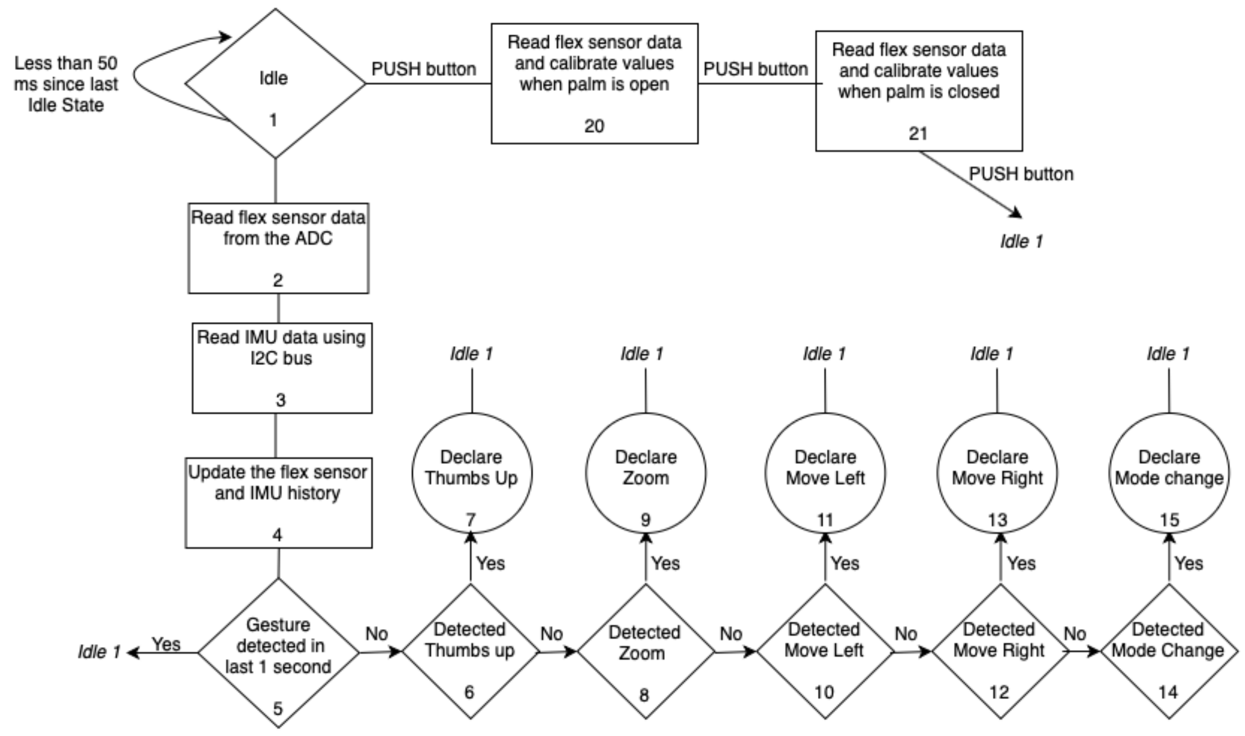

Less than 50 ms since last Idle State
Idle 1
PUSH button
Read flex sensor data and calibrate values when palm is open 20
PUSH button
Read flex sensor data and calibrate values when palm is closed 21
PUSH button
Idle 1
Read flex sensor data from the ADC 2
Read IMU data using I2C bus 3
Update the flex sensor and IMU history 4
Gesture detected in last 1 second 5
Yes
Idle 1
No
Detected Thumbs up 6
Detected Zoom 8
Detected Move Left 10
Detected Move Right 12
Detected Mode Change 14
Yes
Declare Thumbs Up 7
Declare Zoom 9
Declare Move Left 11
Declare Move Right 13
Declare Mode change 15
Idle 1

Figure 9: High level SW Control Loop

Figure 9 above illustrates the software flowchart for the control loop on the microcontroller. The idle state (state 1) is reached after every successful gesture recognition. The idle state has a condition where it checks whether it has been $50ms$ since the previous idle state. That ensures a total of 20 loops over 1 second. This is more than enough data to detect gestures.

A push button on the PCB with the microcontroller is used to move into the calibration state (state 20). The flex sensors have different values in resistance and hence need to be calibrated before use. Further, this enables custom calibration for each individual using the glove. When the button is pushed the user is expected to keep their fingers spread out and palm flat. The voltage value is noted. Then the push button is again clicked, and a state is entered where the flexed voltage sensor values are noted by making the user make a fist. Then another push on the button leads us back to the idle state where gesture detection begins again.

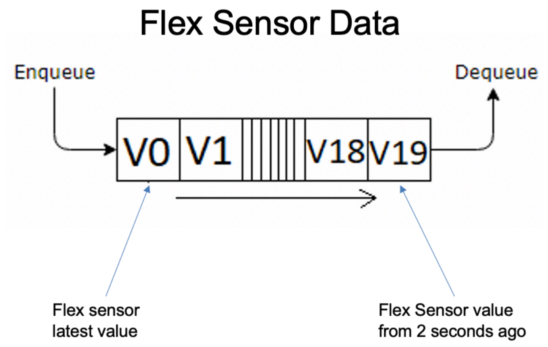


Figure 10: Sensor Data Storage Structure

After waiting in idle the control loop moves into both the read for flex sensors data state. Here in state 2, the microcontroller reads the various flex sensor voltage values through the on-chip ADC. The data is sampled and then loaded into a queue with 20 values. One such queue (as illustrated in Figure 10 above) exists for each of the flex sensors and since we sample every $50ms$, the 20 values in the queues correspond to the state fingers on the hand in the last one second. In state 3 we probe the IMU for data, and that data is stored using a similar data structure.

State 5 makes sure to avoid constant gesture detection. If a gesture has been detected in the last one second, we bypass the gesture detection code.

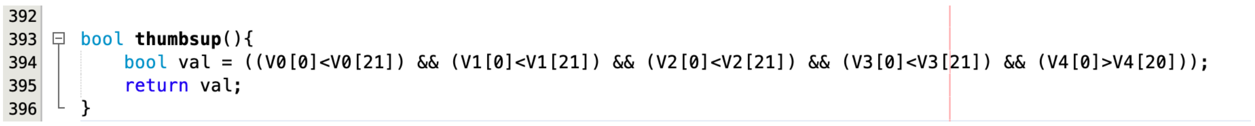
Thumbs up: This is the simplest of the gestures. If the thumb is not bent and all other fingers are bent a thumbs up is detected.

Move left & move right: If the palm of the person wearing the device is facing the left, which means the IMU is facing the right, and there is some translational acceleration detected towards the left, then we detect move to the right. Vice versa for move to the left. Also, the persons thumb needs to be kept bent at all times for this gesture to be detected. The bent thumb is detected using the flex sensor.

Mode change: The mode variable is toggled between one and two if the mode gesture is detected, i.e., if mode was one it is set to two when this gesture is detected. The gesture is detected if a fist is made quickly and released. If all the fingers are spread out and one's palm is flat, and then in quick succession a fist is made followed by spread out fingers and flat palm. This gesture is detected by processing the data in the history table which stores the data of all the flex sensors. The current mode is also displayed by lighting up the LED on the glove in mode 1 and not lighting it in mode 2.

Zoom in/ Zoom out: The flick gesture is used to declare a zoom in our out. A flick is detected if all the fingers are bent like a fist and then the pointer finger and thumb suddenly extend. It is similar to the action when a carrom striker is hit with the index finger while playing carrom. This is detected again by looking at the history data of all the sensors. Once a flick is detected, a zoom in or a zoom out is declared based on the current mode.

As seen in the control loops, states 6,8,10,12 & 14 are gesture detection states. They are function calls that look at the sensor data to determine if a gesture can be declared. In the declare gesture states, the data about the gesture detected is transmitted to the PC. The PC then prints the gesture on its screen.



```
bool thumbsup(){
    bool val = ((V0[0]<V0[21]) && (V1[0]<V1[21]) && (V2[0]<V2[21]) && (V3[0]<V3[21]) && (V4[0]>V4[20]));
    return val;
}
```

Figure 11:Thumbs up gesture

Figure 11 above displays an example of the simplest gesture, thumbs up. $V0, V1, \ldots . V4$, are queues with data for the five flex sensors. In this code snippet we check whether the thumb is not flexed, and the other four fingers are flexed. If this is true, then we declare a thumbs up.

4.4 PC and Power Supply Unit

This consists of a PC that supports serial communication through UART and a USB to TTL UART serial converter. The chip serves two purposes, it provides $3.3V$ power to the whole device. And it acts as a communication channel between the microcontroller and the PC.

5. Verification

5.1 Glove Subsystem

5.1.1 Flex Sensors & Voltage Divider PCB

Requirement	Verification
- Resistance needs to lie between $7k\Omega$ and $13\ k\Omega$ - Needs to output voltages between $[1.1, 1.7V]$ based on finger orientation - Consumes $\sim 2\ mA$ across sensors and op-amps	• Measure resistance of each resistor • Create resistor divider circuit on a breadboard and measure output of flex sensors using voltmeter and power supply • Recreate unity gain buffer circuit on op amp and check if requirements are met • Vary length of 20 AWG wire and confirm low resistance

Table 1: Requirements & Verification: Flex sensors

First each of the individual flex sensors were tested to see whether their resistance values fall within the expected range. Then a breakout board was used, and the op amp circuit was recreated on breadboard. After monitoring the current draw and verifying the functionality of the voltage divider circuit, the PCB was soldered. The PCB was tested for shorts across VCC and GND on multiple occasions. Finally, the PCB was connected to a power supply and the current limit was set to $200mA$. This is done to avoid destruction of components in case of a short. The integrity of the PCB was verified and the individual V_out ports for each flex sensor were tested using a voltmeter and bending the flex sensors.

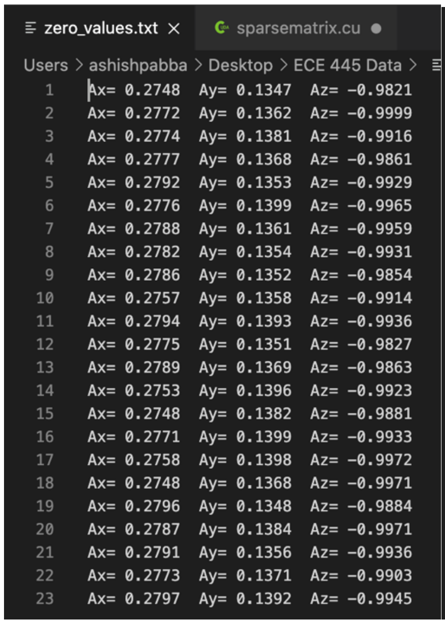
5.1.1 IMU

Requirement	Verification
- Stationary and level IMU should output zeros across output as expected. - Needs to be calibrated to $\pm\ 2g$ range (or $\pm\ 4g$ based on actual values obtained) for precision - Must consumes manufacturer specified amount of $\sim 12.3\ mA$ - Must communicate orientation and motion data reliably over I^{2}C	• Verify zero output while IMU is level and stationary • Measure operating current draw of sensor • Monitor output to microprocessor while in different positions and ranges of motion

Table 2: Requirements & Verification: MPU6050 IMU

The IMU was rigorously tested after consideration of its importance in our project. We mounted the IMU onto the glove and laid the glove flat on a table to measure the values in the level and stationary state. The Az (acceleration on the z-axis) value was close to $1g$, which is expected as the accelerometer would pick up acceleration due to gravity along the z axis. All the other values were expected to zero or close to zero, but instead displayed significant constant zero-error. Some values copied from our serial monitor are displayed below in Figure 12. The gyroscope values were not similarly affected, with the zero error being negligible enough to ignore. This observed zero error was unfixable, prompting us to add a calibration state in the beginning, where the user is prompted to lay their hand flat on the table. During this period, 100 samples are taken from the IMU to compute zero error and averaged to compute offset values, after which any accelerometer/gyroscope value is corrected accordingly by subtracting the corresponding average error value from it.

For orientation computation, we wrote our code to compute orientation, which uses values from the accelerometer and gyroscope and "fuses" them based on the IMU-specific Kalman filter, which allows for the negation of accelerometer inaccuracy and gyroscope drift. Having written code for this computation, we put on the glove and oriented in various positions along the axes as the roll, pitch, and yaw values at such orientations are known. After error correction, these values also matched up with expectations. Figure 13 below illustrates the computation involved in the "fusion" of the accelerometer and gyroscope data.

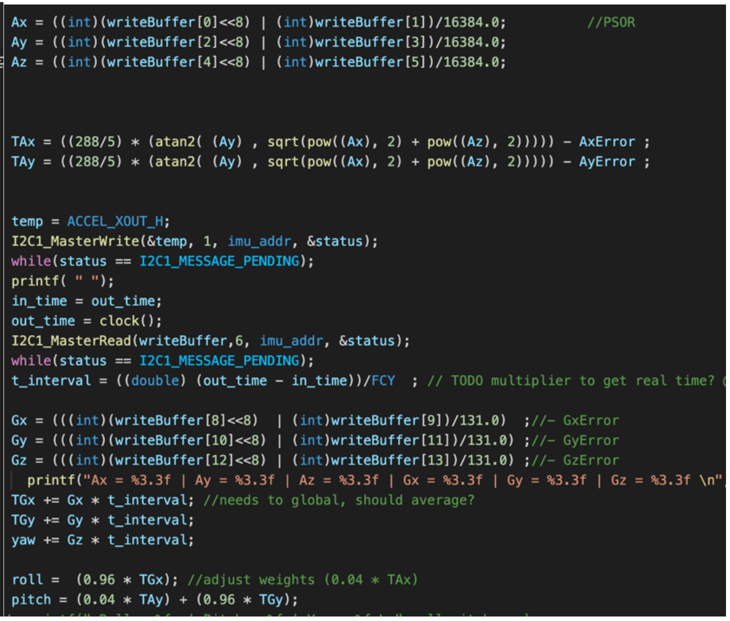


zero_values.txt × sparsematrix.cu ●

Users > ashishpabba > Desktop > ECE 445 Data >

```
Ax= 0.2748  Ay= 0.1347  Az= -0.9821
Ax= 0.2772  Ay= 0.1362  Az= -0.9999
Ax= 0.2774  Ay= 0.1381  Az= -0.9916
Ax= 0.2777  Ay= 0.1368  Az= -0.9861
Ax= 0.2792  Ay= 0.1353  Az= -0.9929
Ax= 0.2776  Ay= 0.1399  Az= -0.9965
Ax= 0.2788  Ay= 0.1361  Az= -0.9959
Ax= 0.2782  Ay= 0.1354  Az= -0.9931
Ax= 0.2786  Ay= 0.1352  Az= -0.9854
Ax= 0.2757  Ay= 0.1358  Az= -0.9914
Ax= 0.2794  Ay= 0.1393  Az= -0.9936
Ax= 0.2775  Ay= 0.1351  Az= -0.9827
Ax= 0.2789  Ay= 0.1369  Az= -0.9863
Ax= 0.2753  Ay= 0.1396  Az= -0.9923
Ax= 0.2748  Ay= 0.1382  Az= -0.9881
Ax= 0.2771  Ay= 0.1399  Az= -0.9933
Ax= 0.2758  Ay= 0.1398  Az= -0.9972
Ax= 0.2748  Ay= 0.1368  Az= -0.9971
Ax= 0.2796  Ay= 0.1348  Az= -0.9884
Ax= 0.2787  Ay= 0.1384  Az= -0.9971
Ax= 0.2791  Ay= 0.1356  Az= -0.9936
Ax= 0.2773  Ay= 0.1371  Az= -0.9903
Ax= 0.2797  Ay= 0.1392  Az= -0.9945
```

Figure 12: Zero values on IMU.



```c
Ax = ((int)(writeBuffer[0]<<8) | (int)writeBuffer[1])/16384.0;          //PSOR
Ay = ((int)(writeBuffer[2]<<8) | (int)writeBuffer[3])/16384.0;
Az = ((int)(writeBuffer[4]<<8) | (int)writeBuffer[5])/16384.0;



TAx = ((288/5) * (atan2( (Ay) , sqrt(pow((Ax), 2) + pow((Az), 2))))) - AxError ;
TAy = ((288/5) * (atan2( (Ay) , sqrt(pow((Ax), 2) + pow((Az), 2))))) - AyError ;


temp = ACCEL_XOUT_H;
I2C1_MasterWrite(&temp, 1, imu_addr, &status);
while(status == I2C1_MESSAGE_PENDING);
printf( " ");
in_time = out_time;
out_time = clock();
I2C1_MasterRead(writeBuffer,6, imu_addr, &status);
while(status == I2C1_MESSAGE_PENDING);
t_interval = ((double) (out_time - in_time))/FCY  ; // TODO multiplier to get real time?

Gx = (((int)(writeBuffer[8]<<8)  | (int)writeBuffer[9])/131.0)  ;//- GxError
Gy = (((int)(writeBuffer[10]<<8) | (int)writeBuffer[11])/131.0) ;//- GyError
Gz = (((int)(writeBuffer[12]<<8) | (int)writeBuffer[13])/131.0) ;//- GzError
  printf("Ax = %3.3f | Ay = %3.3f | Az = %3.3f | Gx = %3.3f | Gy = %3.3f | Gz = %3.3f \n"
TGx += Gx * t_interval; //needs to global, should average?
TGy += Gy * t_interval;
yaw += Gz * t_interval;

roll =  (0.96 * TGx); //adjust weights (0.04 * TAx)
pitch = (0.04 * TAy) + (0.96 * TGy);
```

Figure 13: Orientation computation.

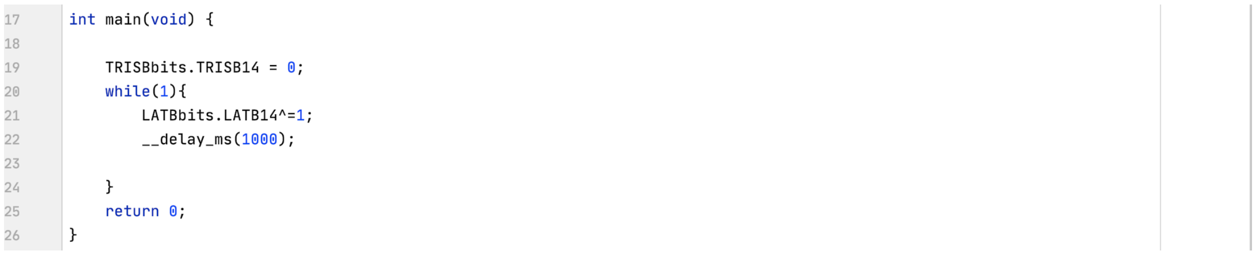
5.2 Control Unit

5.2.1 Control Unit PCB

Requirement	Verification
- Control unit board draws a maximum of $24mA$ at $3V$ - Able to program microcontroller with external programmer	• Prototype the microcontroller on the bread board with the push button and the clock generator. • Program board with blinking LED to test programmer • Program microcontroller to do Dijkstra algorithm, which is computationally intensive and check current draw on breadboard.

Table 3:Requirements & Verification: Control Unit

The schematic circuit for the control unit was recreated on a bread board with all the necessary components. Then a power supply with a current draw limit of 500mA was used to power the circuit. Once the integrity of the circuit was verified, a blinking LED code was deployed to verify the functionality. The code snippet is seen below. This helped in confirming that the microcontroller was programmable with the given set up.



```
int main(void) {

    TRISBbits.TRISB14 = 0;
    while(1){
        LATBbits.LATB14^=1;
        __delay_ms(1000);

    }
    return 0;
}
```

Figure 14: Blinking LED Code Snippet

Then the control unit board was soldered and tested for shorts. The same procedure as above was followed thereafter to test integrity and functionality of the board. Further, the Dijkstra algorithm was run, and the current supply was monitored on a power supply. The current draw was well below the 24mA limit accounted for.

5.2.2 Control Loop

Requirement	Verification
- Communicate with PC - Interface with the IMU using I^2C protocol and store this data - Interface with flex sensor unit, convert data using ADC and store the data. - Processing is fast enough to allow for at least 20 loops per second. - Able to detect the gestures reliably with minimal false positives.	• Verify the UART communication • Verify the ADC output • Verify the I^2C interface. • Write functions for various gestures and verify them individually. • Then construct the control loop to detect all the gestures

Table 4: Requirements & Verification: Control Loop

To verify the various components on of the control loop it was essential to print data from the microcontroller. Hence the first step was to set up the UART communication line using the RX and TX ports. All print statements in the code were redirected to the serial port. After the serial communication was verified by printing random text and making sure it was displayed on the PC, communication with the IMU and flex sensors was verified. Using I2C, the IMU was probed and the values were displayed on the PC. The image in the IMU section (Figure 10) shows the data received from the IMU after post processing. The ADC on chip was probed and the analog voltage values from each on the flex sensors were read and displayed as well to verify its functionality. Then a control loop was created with each individual gesture to verify whether gesture detection was successful and measure the computation time for each of them. Before integrating all the gestures into the final control loop it was vital to verify that the sum of computation time for each individual gesture was less than 50ms so that the control loop can function the way it was designed.

5.3 PC and Power Supply Unit

Requirement	Verification
- Output of 3.3V for microprocessor, IMU & flex sensor PCB (current draw of 24 mA) - Communication between PC and microcontroller enabled	• Probe output voltage to confirm 3.3V output • Check if the communication has been established by printing to UART terminal.

Table 5: Requirements & Verification: PC and Power Supply Unit

We were able to verify that the output voltage is a stable 3.3V by probing the output voltage using a multimeter. The establishment of UART communication was verified by writing code to print to the serial monitor on the PC.

6. Cost and Schedule

6.1 Cost Analysis

Description	Part Number	Quantity	Cost (USD)
Flex Sensor	FS-L-095-103-ST	2	21
Resistor 47 kΩ	CF14JT47K0	2	0.1
Microcontroller	dsPIC33CH64MP502	1	3.95
IMU	MPU6050	1	4.99
Op Amp	LM741CN/NOPB	2	0.88
Battery Holder	Rechargeable 3V battery	1	4.79
Gloves	Cevapro Running Gloves	1	14.99

Table 6: Cost breakdown

The total cost of the parts as listed in the parts table is $102.62. From UIUC's own data, the average salary of EE and CE majors is $88,000 per year [5]. Assuming 52 work weeks, 5 days a week, 8 hours a day, we arrive at an hourly rate of $42.31. Average partner contribution can be assumed at 10 hours per week. From the given formula for the cost of labor, we find that labor will cost

$$2.5 \times 30\ hours/week \times 7\ weeks \times 42.31\ \$/hour = \$22{,}212.75.$$

The **total cost** of both labor and parts is then:

$$\$72.62 + \$22{,}212.75 = \mathbf{\$22,285.37}$$

6.2 Schedule

Week	Ashish	Chris	Ramakrishna
10/5	Decide on and order parts for power unit, begin to test parts	Decide on and order parts for control unit, begin to test parts	Decide on and order parts for glove, begin to test parts
10/12	Start working on microcontroller programming based on sensor tests, help with PCB design	Work on PCB Eagle Schematic, submit PCB order	Work on PCB Eagle Schematic, submit PCB order
10/19	Program microcontroller and test motion and flex recognition	Program microcontroller and test motion and flex recognition	Program microcontroller and test motion and flex recognition
10/26	Test power units, explore computer interface and how to display recognized gesture	Test control unit in conjunction with glove, verify microcontroller program and modify as necessary.	Test glove unit, log sensor data for gestures and identify usable trends/patterns
11/2	Complete assembly, test gesture recognition with integrated prototype and modify microcontroller program as necessary	Complete assembly, test gesture recognition with integrated prototype and modify microcontroller program as necessary	Complete assembly, test gesture recognition with integrated prototype and modify microcontroller program as necessary
11/9	Prepare for mock/final demos, grace time in case of delays in earlier steps	Prepare for mock/final demos, grace time in case of delays in earlier steps	Prepare for mock/final demos, grace time in case of delays in earlier steps

Table 7: Schedule

7. Conclusion

7.1 Accomplishments

We were able to satisfy all of our high-level requirements and consider our project a success in its defined scope. In terms of gesture recognition, we were able to achieve a significant degree of accuracy across all gestures and were able to do so by successfully acquiring and processing of data streams coming from our glove-mounted sensors. Moreover, we were able to keep variations in accuracy for recognition across different gestures to a minimum despite the varying complexities of our chosen gesture list, with the standard deviation for recognition accuracies for different gestures being about 3.06 %. Figure 15 below displays the recognition accuracy for each gesture. These results were derived by averaging the number of successful gesture recognitions for 100 repetitions of the gesture by two different testers.

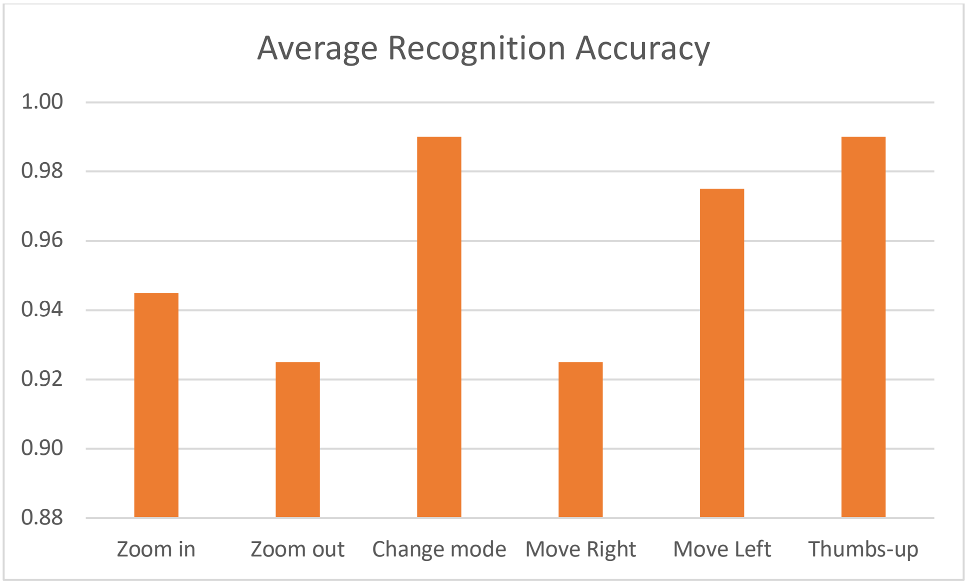


Figure 15: Average Gesture Recognition Accuracy Chart

7.2 Uncertainties and Shortcomings

While we were able to eliminate almost all uncertainties relating to sensor data accuracy through rigorous testing and troubleshooting, the primary uncertainty in our project was the source of variation in accuracy of recognition between testers for certain gesture. For this observed disparity, potential sources of error include:

- Variation in hand size of testers
- Variation in flex sensor alignment with fingers of different users
- Variation in the orientation of the IMU sensor with respect to different users' hands

We were able to decrease this disparity by implementing fixes such as:

- Dynamic calibration of flex sensor threshold values for flexed/unflexed fingers
- Dynamic calibration of offset values for the accelerometer and gyroscope
- Fastening zip ties around the flex sensors at the bases of fingers, creating flexible anchor points

The final disparities for a few gestures are illustrated below in Figure 16. There remains scope for improvement, as will be discussed in the Improvements section.

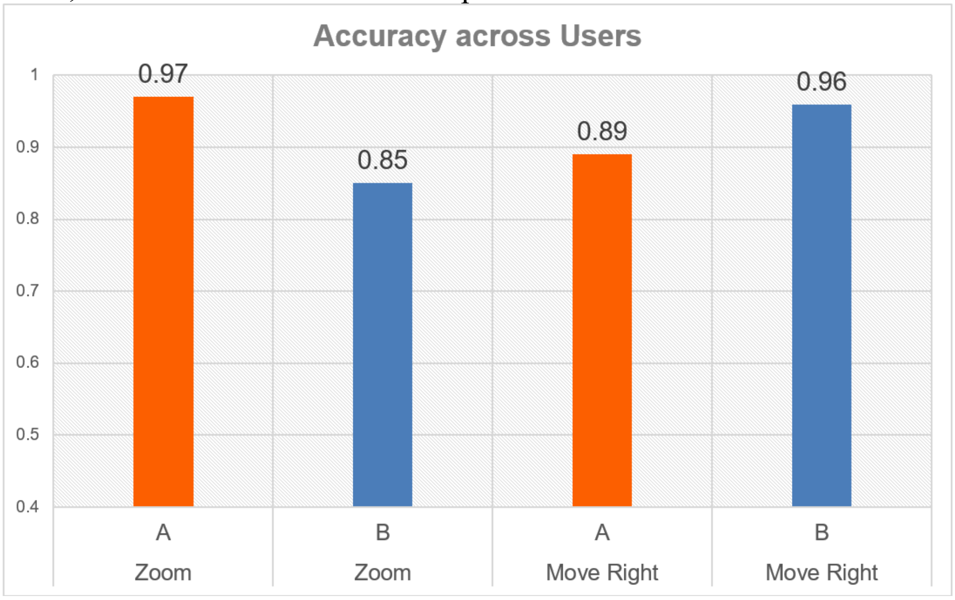


Figure 16: Disparity in recognition accuracy across testers

7.3 Future Works/ Alternatives

While we were mostly satisfied with the results, we look forward to making several future improvements that we have identified as possible and beneficial for our project. These improvements include:

- Add magnetometer: Using a magnetometer in conjunction with the accelerometer and the gyroscope on the IMU would significantly increase the accuracy of our orientation data stream, supporting our ability to implement and recognize more complex gestures.
- Improve build quality: Improving our build quality, especially better flex sensor/IMU anchoring to the glove, would improve accuracy and reduce accuracy disparities across testers.
- Custom gesture configuration: Adding the ability for users to "record" their own custom gestures and subsequently recognizing them when performed would greatly increase our project's perceived utility.
- Input driver: Writing a driver to allow the gestures to actually be used as an input device with computers as a plug-and-play device would elevate our product from a proof-of-concept/prototype to a usable product.
- Gesture granularity: Gesture granularity can be improved by further processing our sensor data streams. For example, the zoom gesture can be improved by also recognizing the speed at which the gesture was performed, and assigning a percentage value accordingly (150% zoom in, 200% zoom out etc.)

7.4 Discussion of Ethics and Safety

The weight and positioning of the sensors and boards may over time, if care is not taken, cause strain on the operator's hand or abrasions on the skin, potentially resulting in permanent injury. To comply with rule #1 of the IEEE Code of Ethics [6], that we may preserve the health and safety of our users, attention will be given to ergonomic placement of the said components on the hand and wrist, to achieve a balanced, comfortable experience.

Additionally, because the user is intended to wear a mounted electrical system, care will be taken to ensure proper insulation of the mounted electrical components and their associated wires, so that no part of the operator physically encounters these components.

For the battery, there must also be a consideration regarding the temperature, as there exists the possibility of overheating. However, since the battery is not mounted on the glove in our design, the risk of a burn is rather low.

Finally, the design of this system is open source. This is good ethical practice because it allows for adoption and modification of the system without additional expense by the community that utilizes the design.

8. Works Cited

[1] Spectrasymbol, "Sparkfun," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/FLEXSENSORREVA1.pdf.

[2] T. Instruments, "Texas Instruments," [Online]. Available: https://www.ti.com/lit/ds/symlink/lm6134.pdf?ts=1601536549483&ref_url=https%253A%252F%252Fwww.google.com%252F.

[3] Hyperphysics, "Hyperphysics," [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Tables/wirega.html.

[4] MicroChip, "MicroChip," [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/dsPIC33CH128MP508-Family-Data-Sheet-DS70005319D.pdf.

[5] UIUC, "courses.engr.illinois.edu," 2020. [Online]. Available: https://ece.illinois.edu/admissions/why-ece/salary-averages.

[6] IEEE, "ieee.org," 2020. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed 16 Sep 2020].

[7] Sparkfun. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/FLEXSENSORREVA1.pdf.