Kaitlyn Cason
-UID Removed-

Happy Hands

Introduction and Background

The project goal was to build a device useful for basic communication from ASL to English, while also exploring a variety of Electrical Engineering concepts in the process. Many Deaf and hard of hearing individuals are able to lip read to some degree, but they often have difficulty communicating to people who cannot understand sign language. The ultimate goal was to help people in the ASL community to "be heard". The foundation of ASL is the "fingerspelling" handshapes used in alphabet. As such, I set out to design a portable and independently powered system that could be connected to via Bluetooth, and trained the glove to recognize these letters of the ASL alphabet as a proof of concept.

The system would work much like communication using pen and paper, but would ease the process by skipping the writing medium by translating ASL letters straight to displayed text. As fingerspelling letters are inherently unique, the goal of the glove would be to correctly recognize and separate the varying letters through identification of hand positions, general shape, and contacts between fingers. If the glove is able to detect sign letters with a high level of accuracy -- as judged in correctness by myself -- the device can further be used as a fingerspelling teacher and trainer. For the design, I decided to mount all of the components and wiring to the backhand side of the glove to keep it completely portable. The discussion of the components and the designs reasons for picking them are further detailed in the basic theory.

Basic Theory

For the design, it was clear that I needed a portable microcontroller to mount onto the glove, especially since for testing, since I have particularly small hands. Anything too bulky (such as the Arduino UNO) could potentially hinder palm orientation and movement for the several signs that are not static and upright in orientation. Everything was initially tested with the ubiquitous Arduino UNO, but switched over to using the Teensy because it had a number of advantages, including having a higher maximum clocking frequency -- 48 MHz as opposed to the UNO's 16 MHz, larger storage, a much smaller form factor and increased I/O pins. The biggest downside was that the Teensy was only rated to handle input voltages between 3V to 5.5V, reducing the options in external power sources. The UNO, on the other hand, is voltage regulated and capable of handling input voltages up to 20 volts. Because of this, I used a rechargeable 3.7 V lithium polymer battery to power the circuit, given to us by our TA, which was secured to the back of the hand using velcro tape.

To capture all of the letters of the alphabet, I needed to collect different types of data from various sensors. This included data from flex sensors, which would tell us about the amount of bend in each finger, contact sensors to tell us which fingers are touching (for when the resolution of the flex sensors are too low to distinguish between handshapes) and an accelerometer-gyroscope for orientation and motion data. This data is read into the Teensy and sent over Bluetooth to be received and processed. Bluetooth is a short-range wireless communication that shares the same 2.4 GHz frequency as WiFi.[1, 2] the project uses the Sparkfun BlueSMIRF.

On the receiving end, a simple Python program was written, which implemented a rudimentary machine learning algorithm. Since the data came in different forms, including binary and continuous data, at the time, I was unable to find a way to treat all of the data equally or come up with weights to process this data using a simple algorithm like K-means clustering or K-nearest neighbors. Therefore, I settled on a naïve approach to quantize the flex sensor and accelerometer data into discrete states based on thresholding, so that the condition that state was met could be handled like binary data and each handshape therefore had a unique "signature".

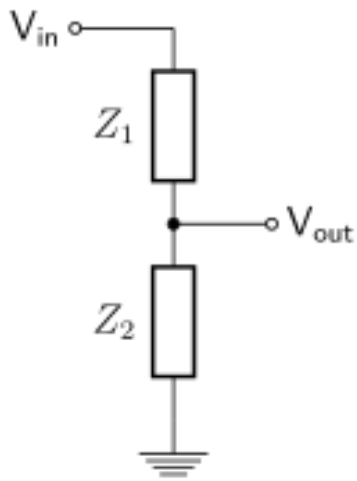

Figure 1: Voltage divider

Flex sensors for each finger were added in order to gather data on the general shape of the hand. Flex sensors can be thought of as variable resistors, where the resistance increases as the sensor is further bent. I recorded the data for these sensors using the analogRead range of 0 to 1023. Using an independent threshold for each finger, the numerical data was then condensed into at most three different states: straight, curled, and bent. Using the Teensy's internal pull-up resistor, I then created a voltage divider similar to that shown in Figure 1 with a 10 KΩ internal pull-up resistor as Z_1 and the flex sensor as the variable resistor Z_2. This allowed me to analogRead the voltage depending on the resistivity of the flex sensor as $V_{out} = 3.3 * \frac{Z_{flex}}{Z_{flex} + 10K\Omega}$, where the voltage value from 0 to 3.3V was mapped to the range of 0 to 1023.[5]

After consulting with others on how best to approach specific edge cases, we decided on adding 10 contact “pads” to capture enough data to account for edge cases, where two letters could not be distinguished from the other sensors but could be determined based on how fingers made contact with one another. The contact pads were 10 “digital” in/out ports set as input pins, and also made use of the approximately 10 KΩ internal pull-up resistor, this time to pull the voltage values to HIGH (3.3 V) when the contact pads were “floating”, or not connected to an output. A diagram of a pull-up resistor is shown in Figure 2. In order to determine the matrix of which contact was touching which, the program uses a nested loop that iterates through each of the contact pads, set its IO pin as output to DigitalWrite a LOW (0V) signal, ran an inner loop and then set that pin back to input. In the inner loop, we iterate through the other pins and used DigitalRead to determine their input values. By doing this, it allows one pin to write LOW (0 V with respect to the ground terminal) to the pins touching it, and create a mapping of all the contact pads in relation to one another. The pins that were making contact read LOW with respect to one another, and the pins that were not touching were “floating” and had a value of HIGH due to the pull-up resistor pulling their value up to 3.3V.[6]

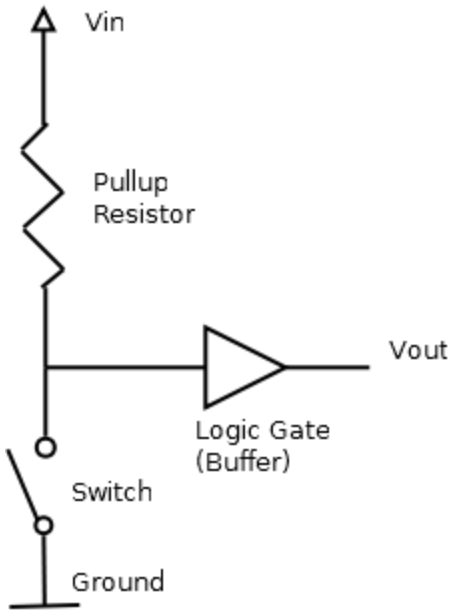

Figure 2: Pullup resistor

In order to measure motion and orientation of the hand, I used an MPU-6050, an accelerometer-gyroscope processor, on a SparkFun breakout board. The accelerometer and gyroscope measure static and dynamic forces by measuring the change in voltage due to movement on a microelectromechanical systems (MEMS) device.[3,4] By moving the sensor, similar to the one shown in Figure 3, the resistance and capacitance is changed, and the embedded sensor can measure the voltage value that corresponds to a movement.

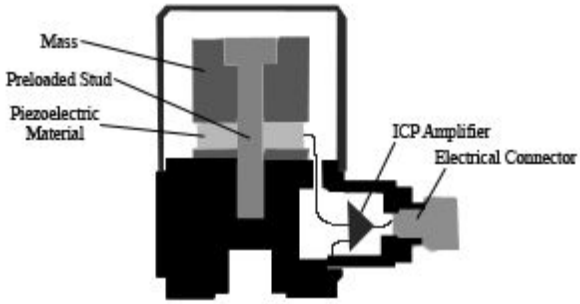

Figure 3: “The cross-section of a piezoelectric accelerometer”

Testing Methodology

Originally, the Arduino code was written in the Arduino IDE environment, but was eventually switched over to the Teensy microcontroller for the numerous reasons described previously. In switching to the Teensy, most of the code remained the same, but it used the Teensyduino driver to load Arduino code from the Arduino IDE onto the microcontroller. From there, I devised a basic model of what “black-box” components a fully functional glove would need to have in order to correctly translate fingerspelling into text (Figure 4). The Happy Hands device consists of three units in addition to the Bluetooth module to transmit the data: ASL glove, handshape data processing, and PC. When the user “fingerspells” a letter, the voltage levels detected by each flex sensor and the outputs of the contact pins are fed to the microcontroller for data processing. The Teensy LC takes the data and compresses it into discrete states for each set of data. The analog data from the flex sensors are converted to three states: bent, curled, and straight. The analog data from the accelerometer is also converted into states: neutral, upright, sideways, and downward. These states, along with a 10-by-10 matrix for the contact pads are then sent as one vector to the PC unit via Bluetooth. In total, 10 contact pads were needed after systematically testing for uniqueness of each character, as seen in the “truth table” in Table 1. On the computer, the vectors are analyzed by the python code that attempts to match the vector to previously mapped vectors in a dictionary created by the supervised learning algorithm. Once a signed letter is recognized and held for 125 instances, it is outputted to written text on the display screen of the PC. This system flow is illustrated simply in Figure 4 as a block diagram. Each of the components was connected to the Teensy like in the schematic diagram in Figure 5 and was independently verified to be operational using the Arduino test sketches attached.

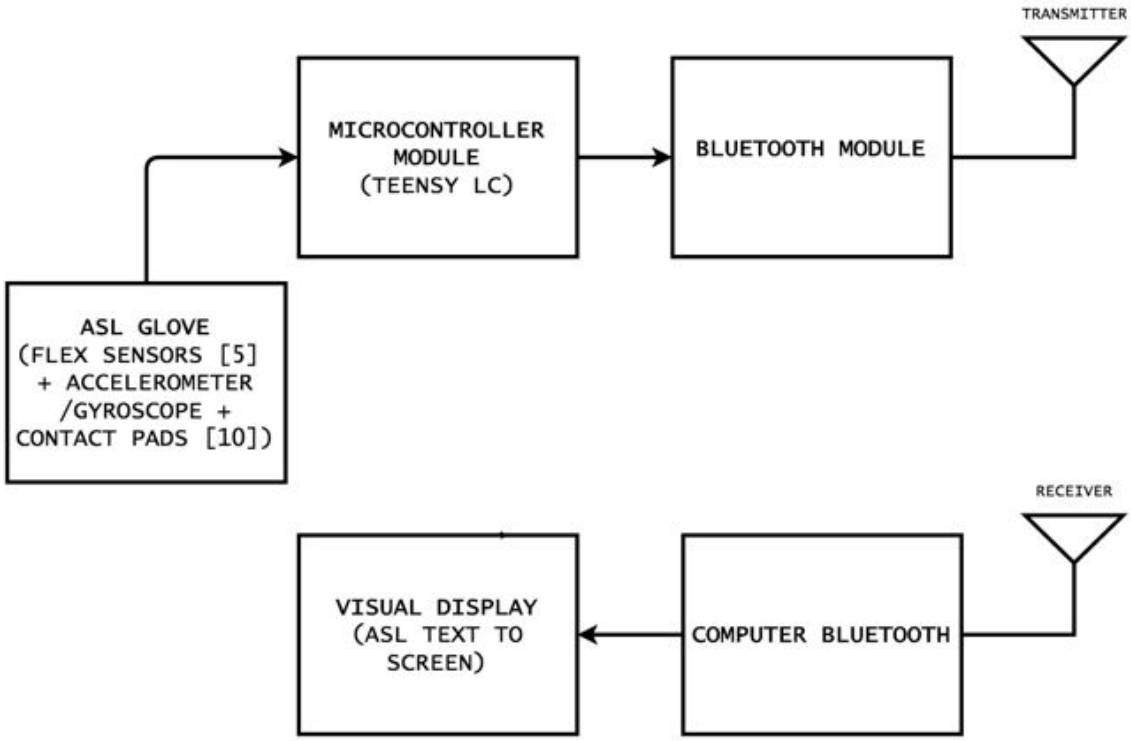

Figure 4: Block diagram of high level flow for the glove.

Testing flex sensors:
Instead of using external resistors in combination with the flex sensors, I relied on the internal pull-up resistors that were included in the Teensy. This allows for have less external hardware on the glove and have a more reliable final product. However, since the variable resistor was now on the bottom of the voltage divider instead of on top, changes in the resistance of the flex sensor amount to non-linear changes in the voltage read. Based on the flex sensor values outputted for each alphabet handshape, you can determine a suitable threshold to bin the values from values from 0 to 1023 to a maximum of three binary states (straight, bent and curled). The flex sensor values for each letter of the alphabet can be found in Table 2.

Testing contact pads:
In order to account for more indistinguishable letter shapes, I needed to create contact pins for certain parts of the hand. For example, the letters ‘A’, ‘E’, and ‘S’ have a similar range in the flex sensor values for each finger. To identify between those gestures, it was necessary to distinguish the positions of different fingers. Each pin is initially set as an input pin connected to an internal pull-up resistor. This allows each pin to read HIGH when not connected to anything. From this, we cycle through all contacts, setting one to DigitalWrite a LOW signal at each iteration. With one pin writing LOW, we read from all other pins to determine which

of the pins are connected. An additional small delay after setting any pin to DigitalWrite LOW to accounted for finite fall time due to discharge of load capacitance. Each pin can be modeled as having an equivalent R and C value; therefore it has a small RC time constant and takes finite time to switch states.

To capture and test the contact pins, I created a 10x10 matrix where each numbered column and row represented the corresponding numbered contact pin counting from the thumb to the pinky finger. When a pin reads a LOW signal, the corresponding two places in the matrix are marked as 0. For example, if contact pins numbered 1 and 9 were touching, then we should expect to see a 0 at locations (1,9) and (9,1) in the matrix. While cycling through the iterations, when the first pin is set to DigitalWrite LOW, the ninth pin will read a LOW signal. Similarly, once the ninth pin is set to DigitalWrite LOW, the first pin will read a LOW signal. This would give us the desired results.

If the fourth and fifth pins are making contact and the sixth, eighth and tenth pins are making contact, the matrix would look as follows:

	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	0	1	1	1	1	1
5	1	1	1	0	1	1	1	1	1	1
6	1	1	1	1	1	1	1	0	1	0
7	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	0	1	1	1	0
9	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	0	1	0	1	1

Testing Bluetooth:
Bluetooth testing involved using the Serial1 hardware serial pins on the Teensy to read and write from the Bluetooth module and printing the read contents to the serial monitor. A simple sketch was used to test two-way communication and also to periodically print a message to Bluetooth, which could be received on a computer that was connected to the Bluetooth module through the command line. Part of the code used and tested with came from Jim Lindblom's public domain example sketch for SparkFun electronics.

Testing Accel/Gyro (raw data):
Testing the accelerometer and gyroscope board first involved setting up a Wire transmission using the I^2C serial communication protocol, indicating the registers to read from and then storing or printing the 16-bit data from the Teensy. Part of the code used and tested with came from JohnChi's public domain example sketch for reading raw data for Arduino.

Testing Machine Learning:

The machine learning script was designed as a naïve supervised algorithm. The testing for the scripts was done iteratively, by running the script and making improvements to places that seemed to be lacking. One of the difficulties faced was how to handle the user input and also the vector data sent in from the glove simultaneously, since the serial data is sent into a buffer and does not get flushed until it is read, and manually flushing removes the useful data. the solution to this problem is elaborated in the end of the discussion section. The machine learning script was designed to wait for the user to input a letter of the alphabet or a command, and then read the input vector through Bluetooth. Since the different letters were determined to have unique input vectors, the script would ask to store the new mapping into a dictionary or tell the user if there was a conflict (the vector was already mapped to a different letter). The dictionary is then "pickled"-- saved as a serial byte stream file, so that the data can be quickly loaded into memory and used again in either the learning script as well as the display script.

The corresponding display script then loads the pickled data into memory and monitors the vectors entering through the serial port. It maintains a counter of the number of times a letter mapped to the inputted vector occurs in sequence, and then displays that letter when the count reaches 125. This number was determined by printing how many times a letter mapped to the input vector was seen when intentionally signing a letter as opposed to transitioning from one sign to another.

Table 1: ASL alphabet "truth table"

	T	1	2	3	4	1-KI	1-KO	2-KI	3-KI	4-KI	Accel		
A	1	x	x	x	x	1	1	1	0	0	x		
B	0	x	x	x	x	0	1	1	0	0	x		
C	0	x	x	x	x	0	1	1	0	0	x		
D	1	0	1	x	x	0	0	0	0	0	x		
E	0	x	x	x	x	0	1	1	0	0	x		
F	1	1	0	0	0	0	0	0	0	0	x		
G	0	0	x	x	x	x	x	0	0	0	1		
H	1	x	x	x	x	0	1	1	1	0	1		
I	1	x	x	x	0	0	1	1	0	0	x		~Legend~
J	1	x	x	x	0	0	1	1	0	0	1		T = Thumb
K	1	0	0	x	x	0	0	1	0	0	1		1 = Index
L	0	0	x	x	x	0	0	0	0	0	x		2 = Middle
M	1	x	x	x	x	0	1	1	0	1	x		3 = Ring

N	1	x	x	x	x	0	1	1	1	0	x		4 = Pinky
O	1	x	1	x	x	0	1	1	0	0	x		K = Knuckle
P	1	0	0	x	x	0	0	1	0	0	1		I = Inside (towards thumb)
Q	0	0	x	x	x	0	1	1	x	0	1		O = Outside (towards pinky)
R	x	1	1	x	x	0	0	0	x	0	x		
S	1	x	x	x	x	0	1	1	0	0	x		
T	1	x	x	x	x	0	0	1	0	0	x		
U	1	x	x	x	x	0	1	1	1	0	1		
V	1	0	0	x	x	0	0	0	1	0	x		
W	1	0	0	0	x	0	0	0	0	1	x		
X	1	0	x	x	x	0	0	x	0	0	1		
Y	0	x	x	x	0	0	1	1	0	0	x		
Z	1	x	x	x	x	0	0	1	0	0	1		
Slide/ Tap	x	x	x	x	x	x	x	x	x	x	1		

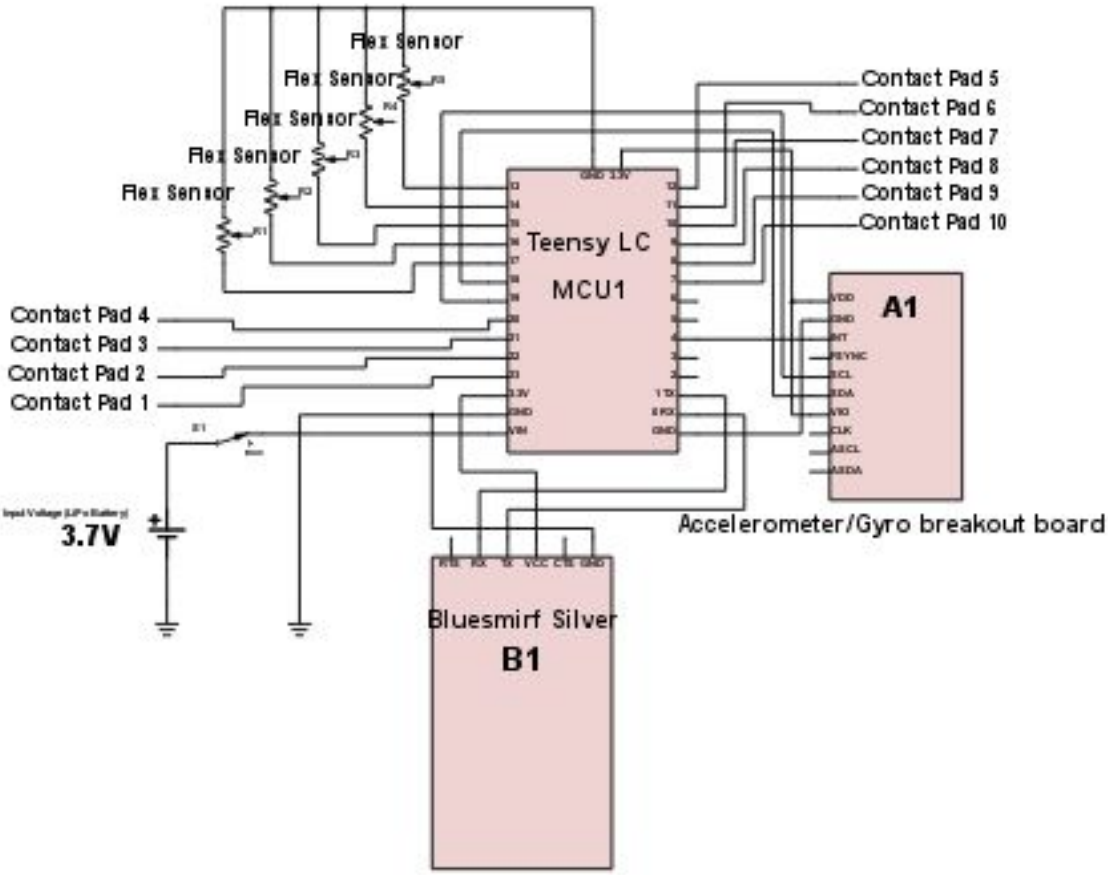

Figure 5: Schematic diagram of the various components and their connections

Results and Discussion

Table 2 shows the results of testing the flex sensors for each letter of the alphabet, where the column marked F5 shows the letters in a similar range being combined into a single state. Figure 6 shows the completed and sewed glove and includes the wiring, external power and modules involved.

During project design and testing, I ran into a number of hurdles which I attempted to overcome. Early on, I noticed that Bluetooth has an unstable connection that took time to set up, connect and reconnect, as opposed to USB, and if the program had to be interrupted on one end, the Bluetooth sometimes needed to be rebooted in order to reopen the serial port. Additionally when training the program, we only want access the most recent vector data from the glove when the user inputted a letter, but reading from the serial port buffer meant that the data values in the buffer were only removed when the buffer was read. This meant that the buffer was filling up while the user was inputting a character, and the program was only reading old values until the buffer filled up and the program crashed. I could not

reconnect serial connection for each user input because connection time was not predictable, and I could not rely on running the code sequentially and reading from a buffer of old data values. To solve this issue, the project implements asynchronous multithreading by creating a child thread which continuously read data and communicating this data to a shared global variable. After getting user input, the main thread could now just pull the value from this variable, which was now a vector snapshot of the user's fingerspelling hand shape at that time. This removed any difficulties with connecting and buffering data in Bluetooth.

On the accelerometer/gyroscope, I learned that it was hard to get any data other than raw values. Although we send motion data in the vector, it was hard to capture 'J' or 'Z' because they depended on time series data which the code as is did not have the ability to express, so I decided to omit them for the time being.

Early on in the project, I had to decide between solid wire, which was more sturdy but not flexible, and stranded wire, which was more flexible but could break more easily. Because it was for a wearable, I opted for the flexibility of stranded wire and therefore ended up having wires that were less durable. However, many of the wires ended up breaking from the stress of being repeatedly pulled due to extra movement when fingerspelling, and this led to frequent delays in testing to replace or lengthen existing connections. For some final testing, I ended up having to hot-glue the wire above the solder joints to the Teensy board to protect from the possibility of any more broken or disconnecting circuits.

The flex sensors outputted a small range of values, because of the 2.2" size of the flex sensors and the voltage range of the Teensy was smaller than that of the (0 to 3.3 V range as opposed to a 0 to 5 V range on Arduino). Taking this into account, it forced me to only map to 2-3 states for each flex sensor. Because the flex sensors act as variable resistors, making contact with the flex created unpredictable circuit behavior, where the Teensy did not know whether to read the contact values as LOW or HIGH. The contacts "shorting" against flex sensors therefore occasionally made the data unpredictable and non-deterministic, a flaw which when having smaller hands, was at times difficult to avoid. In the future, I hope to separate the flex sensors by a layer of fabric so that the contact pads do not have the possibility of shorting with the flex sensors, and also to have more predictable flex sensor values by limiting their ability to shift out of place. Although I originally thought it was an advantage, I used a pair of gloves that had conductive fabric sewn into the thumb and index finger to use with touchscreen technology. Thi was done to hopefully increase the surface of my contact pads. However, this was not the case. For future implementations, I will use a pair of gloves that do not have conductive fabric already in the thumb and finger, since the fabric surface sometimes caused an unwanted connection between the contacts and flex sensors. As discovered in testing, the frequency of shorting was different for different user's hands. Since my hands were much smaller, they did not fill in the gloves as much, and therefore caused more unwanted shorting between contacts and flex sensors on the same finger. Given in my

demonstration, it should be noted that the “shorting” was one of the bigger sources of inaccuracy.

After training and testing my model, I found that when signing each of the letters I was testing for, the program displayed the correct letter for approximately >60% of the time. I was unable to create accurate graphs or tables for this data because testing involved time series data and it was difficult to report quantitative data especially when transitioning between letters.

Table 2: Threshold table for flex sensors

	Flex 1	F2	F3	F4	F5	Contact #s
A	460-464	536-542	566-571	541-546	575-588	0/1, 3/4
B	455-461	408-412	414-416	424-427	411-413	3/4,
C	419-421	480-487	529-533	489-493	453-460	3/4,
D	405-409	406-410	488-498	486-495	507-509	0/5
E	460-466	526-533	549-560	523-530	549-553	
F	425-431	459-463	408-410	425-430	423-428	0/2
G	417-419	403-407	500-504	504-507	529-533	0/4
H	422-427	403-407	383-387	463-470	488-496	0/6, 3/4
I	440-443	516-524	539-543	503-508	414-418	0/1, 3/4
J						
K	381-383	402-404	410-412	479-484	547-551	0/4
L	351-354	407-409	543-547	539-544	549-553	
M	453-463	479-483	482-485	497-502	541-543	0/8
N	415-418	475-478	500-506	488-496	530-535	0/7, 3/4
O	386-388	479-483	497-500	469-472	468-473	0/5
P	400-402	394-403	403-408	465-468	504-506	0/4
Q	368-369	403-405	510-515	509-512	535-539	
R	444-447	398-403	413-415	510-514	535-540	
S	446-452	535-540	576-585	536-543	561-567	
T	415-418	477-482	519-525	510-514	549-555	0/4
U	446-450	402-405	405-407	509-512	541-545	3/4,
V	443-447	402-406	417-425	497-501	526-530	0/6?
W	455-465	405-408	403-405	436-439	534-540	0/9?
X	458-464	500-508	525-530	510-514	518-525	
Y	360-365	498-505	488-493	470-474	408-411	
Z	433-438	403-408	550-558	532-538	572-578	

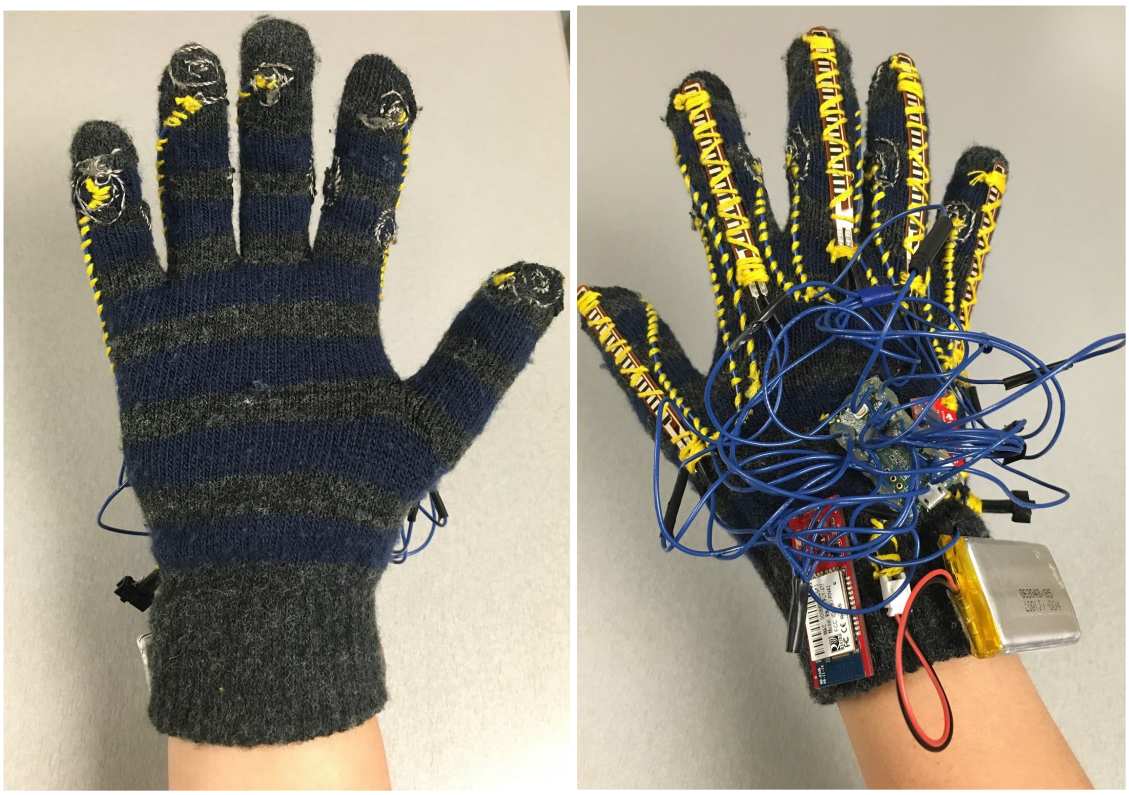

Figure 6: Pictures of completed construction of the Happy Hands glove. Left: Palm of glove includes contact pins on the each finger pad. Right: Back of hand includes all modules (Teensy LC, Bluetooth, Accelerometer/Gyroscope, external battery), 5 flex sensors, and contact pins along sides of fingers.

Conclusions and Future Work

Deaf people often find it difficult communicating with hearing people in a nonprofessional setting because of the extra lengths they must use to accurately portray their message. Using interpreters can be expensive and impractical in an everyday setting, so providing a product that would translate basic ASL would save time, effort and miscommunication that other workaround methods unfortunately see. With this, the motivation was to help Deaf people communicate more easily in settings where an interpreter would not be on hand, or where pen-and-paper communication would be cumbersome.

Overall, the glove met a significant number of my goals. Ultimately, I had hoped to create a portable prototype that translates ASL fingerspelling movements and handshapes into text efficiently and accurately. Out of the 26 letters, 18 of them were read and captured by the glove with significant accuracy. 6 of the letters (for a total of 24) were able to be captured, but were either commonly mistaken for another letter or less accurate. Because there was trouble capturing and using movement data with the use of the accelerometer-gyroscope, the letters ‘J’ and ‘Z’ were left untested. The glove correctly maps ASL letters to their text counterparts, and outputs them to a screen on a computer. Many letters, such as ‘B’, ‘K’, ‘L’, ‘P’, ‘Q’, etc., were distinct and easily detectable with little to no error, while several others

such as ‘A’ ‘O’ and ‘E’ handshapes were more easily mismatched. I had achieved portability through the use of an external battery as well as a bluetooth module to send the captured handshape data to the assisted machine learning algorithm. The hand glove is safe, operating at a low voltage, and lightweight due to the use of small modules and components.

Possible future innovations for the Happy Hands glove range from immediate improvements to broader goals. We can replace the Teensy LC microcontroller with an Adafruit Lilypad to make the device washing machine safe, and therefore more marketable. Other immediate goals include adding fabric coverings to the flex sensors to help prevent shorting the contact pins when touching the top of the hand for handshapes used in the letters ‘E’ ‘O’ and ‘R’, exploring more sophisticated types of machine learning, increasing portability with a phone application, and adding a text to voice feature. This text to voice feature could even enable fluid communication between Deaf and blind individuals, where the main source of communication currently is Tactile Sign Language, where the blind person has to have a working knowledge of sign language to be able to understand when feeling the shape, movement, and locations of the signs with their hands. For machine learning, I had originally considered the unsupervised K-Means algorithm (partitioning objects into k-clusters by the nearest mean distance) for grouping the data, but because of the large dimensionality in each of the vectors, implementing a decision tree type model worked better instead. Other projects have achieved success using a feed forward back propagation neural network.[7] The feed forward algorithm will be used to detect a certain output sequence, where the output will be 1 of 26 letters. The back propagation would handle the error and adjust the weights in the nodes for the learning network.

From a broader perspective, improvements might include allowing customization and calibration for individual users. All signers are different, and certain areas have different stylistic preferences throughout the country. Happy Hands also could be expanded to other subsets of ASL, including adding another glove--and possibly other motion sensors--to capture actual signs in addition to just fingerspelling. I also hope that a device like this could help non-signers improve their fingerspelling by storing commonly used, but incorrect shapes, and offering corrections to fix their handshapes.

Illustration Credits

Figure 1: “Impedance voltage divider” by Krishnavedala - own work, CC BY-SA 3.0, https://en.wikipedia.org/wiki/Voltage_divider#/media/File:Impedance_voltage_divider.svg.
Figure 2: “Pullup resistor” by C4VC3 - own work, CC BY-SA 3.0, https://commons.wikimedia.org/wiki/File:Pullup_Resistor.png#/media/File:Pullup_Resistor.png.
Figure 3: “The cross-section of a piezoelectric accelerometer” by Archiem - self-made, CC BY-SA 3.0, https://en.wikipedia.org/w/index.php?curid=15409574.

References

[1] https://learn.sparkfun.com/tutorials/bluetooth-basics
[2] https://www.sparkfun.com/products/11028
[3] https://learn.sparkfun.com/accelerometer-basics
[4] https://www.sparkfun.com/products/10264
[5] https://learn.sparkfun.com/tutorials/voltage-dividers
[6] https://learn.sparkfun.com/tutorials/pull-up-resistors
[7] https://agrosy.cs.uni-kl.de/fileadmin/Literatur/Khan02.pdf

Code

Arduino code (*.ino), Python code (*.py) and alphabet pickle attached in zip files