EchoSign

Authors: Ria Balli, Ricky Gao, Somya Pathak
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A pair of sensor-based gloves capable of identifying double handed ASL words in real time and translating them to speech. This product is designed to be portable, accurate, and easy to use.**

***Index Terms*—Flex sensor, Gesture Detection, IMU, Machine Learning**

1 INTRODUCTION

The goal of our product is to devise a pair of gloves that have the ability to translate double-handed American Sign Language (ASL) words to speech output. Its purpose is to help facilitate communication between the deaf and non-deaf population. The deaf/hard of hearing community primarily communicates through sign language, and our device will rely on training a machine learning model with sensor information of the gestures that make up sign language. Our hope is that our product is lightweight, easy to use, and has a sign-to-identification accuracy and speed of classification that is as high as possible. For some example words in our vocabulary, please refer to Figure 1.

2 USE-CASE REQUIREMENTS

Based on our goal of providing seamless communication from a deaf speaker to a non-deaf speaker, we focused our use-case requirements on both the performance of the system itself as well as its usability.

EchoSign should have an accuracy of 85% on the targeted vocabulary. This means that for every time a specific sign is performed, the predicted letter/word should match the sign at least 85% of the time. This aligns with our original goal to provide an avenue of communication between deaf to non-deaf speakers and accuracy is a cornerstone of that conversation. We settled on 85% because it was a nice middle-ground between professional research group performance (98.63% by UCLA [4]) and past capstone groups (75.86% by Gesture Glove [3]). This value is motivated by specific considerations into public health and safety. If this system becomes widely used, it needs to be as accurate as possible so accurate information can be conveyed which is especially important in emergency settings.

EchoSign should be able to output a prediction 500ms after the user signed. This means that after the user has signed, the non-deaf speaker should expect to hear the corresponding letter at latest 500ms from the signing. If nothing is announced after the 500ms, the recipient is expected to interpret that as a no signing gesture. The major concern that motivated this requirement is a social one. We wish to create a product that allows for a *seamless* conversation between deaf and non-deaf speakers. The rate at which conversation can occur is important for this and it will inherently be limited by our product. With this in mind, we seek to have only 500ms of delay because this reflects past research that identified that the average speed of signing is 2 signs per second [1]. With this goal, we can try to match the seamlessness of a signed conversation.

EchoSign should be able to classify a set of 10 gestures from the ASL vocabulary that span various parts of speech. A significant portion of the ASL vocabulary makes use of both hands to distinguish themselves from other words and signings. Therefore, the set represents a perfect vocabulary for us to represent. However, due to the vastness of the vocabulary, it would be infeasible to classify all possible words. Thus, our goal has shifted to represent words from each part-of-speech from the ASL vocabulary. Specifically, the set of words are: what, time, car, church, family, meet, live, big, more, but, and an additional "no-sign" state. These words were selected because they were present as some of the most used words [2] and represent various parts-of-speech like nouns, verbs, prepositions, etc.

EchoSign should be a durable product that can last multiple hours on a single battery and withstand repeated signings without need for repairs or replacements. Again, our reasoning relates to social and safety factors. We want to create a product that people can depend on with confidence for a significant period of time. With this in mind we want to ensure that the gloves can be powered by the selected battery for 2 hours at a time. We also want to ensure that our physical design of the glove allows it to be used for multiple charged sessions. The product shouldn't need to be fixed more often than it needs to be recharged or have the battery replaced.

EchoSign should be a comfortable product to use and light (100g). This is primarily motivated from a health perspective. As this is a product that is mean to be worn, we want to ensure that it is minimally impactful and easy to use. Comfort will be evaluated by user testing as we have people rate how easy it is to use. Weight will be measured by the different components that are added to the glove so we can evaluate their cumulative impact.

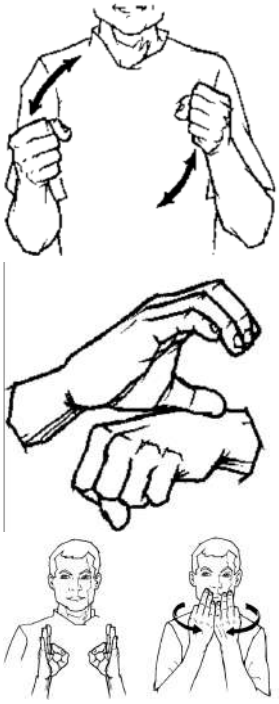


Figure 1: ASL Words (from top-to-bottom: car, church, family)

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

EchoSign's design can be partitioned into its software and hardware components. There are three main components in EchoSign's MVP structure: Glove 1, Glove 2, and the Main Compute Unit. The figure below outlines the various components that make up EchoSign's glove architecture.

Glove 1 comprises the 5 flex sensors from which we will read voltage output from using our on-hand compute board. This computer on the glove is labeled "Arduino" in the diagram. Trade studies on our board specifications can be seen in section 5. For the MVP, we are using the Arduino Nano 33 BLE Sense Rev2 for its compact design, on board IMU, and BLE capabilities. Notably, the IMU data will also be read by the Arduino and sent to the laptop as a datapoint alongside the flex sensor readings. Feedback is also controlled with this compute unit to the haptic vibration motor mounted on the hand. Our batteries, two 3.7V 550 mAh lithium polymer battery, will power this system. To turn on the glove system, we will have the user connect two cables into the designated power pins labeled on the side of the PCB. Once turned on, they will serve as Bluetooth clients and wait until they can connect to the laptop.

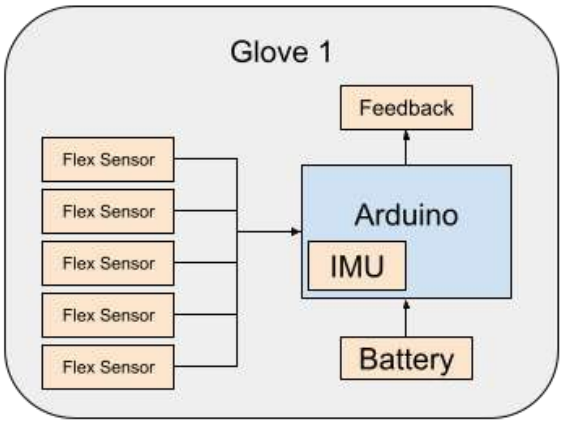
Glove 2 differs from Glove 1 solely by the fact that the haptic feedback component will not be included. But for operating purposes, it has the same principle of operation as the previous glove. Note that the main changes from the design report include the removal of the speaker system on the glove and the incorporation of a PCB on the glove.

The last component of our system contains primarily software implementation. The overall system starts when the user runs a specific python script for detection in a variety of modes. We currently have three modes: one model, two models, or time-series detection. For most purposes, the two models detection scheme will suffice. Afterwards, the laptop will attempt to connect to the Arduino via the Bluetooth connection. After connection has been established, the software will attempt to calibrate to the user's maximum and minimum range of finger flexion. Then, after the calibration has been set, the main loop of inference will commence. The software will receive data from the Arduino. This data will be passed to the ML model which will return a predicted word. Once the ML model predicts the same word multiple times in a row, the speaker will output the predicted word and a signal will be sent to the Arduino for the haptic feedback feature. 7. Note that the main changes here are based in the focus on neural networks and the ML classification algorithm. The other software components remain the same.

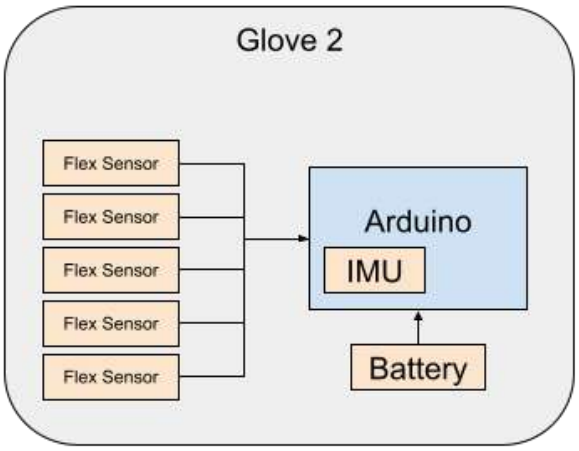
For engineering principles, we emphasized an iterative approach with out prototype scheduling. By developing different subsystems sequentially, we were able to verify the functionality of those subsystems before they were incorporated into the overall design. We also created a simpler product halfway into the semester for our interim demo before proceeding to our final MVP. This scheduling of development allowed us to set smaller goals during our system which helped to keep us motivated and gave us opportunities to verify our theoretical designs in real life.

For science principles, we emphasized some of the content we learned from our classes. For our circuit design, we drew from our knowledge of op-amps and how to amplify voltages so that we could increase the range of detection for our flex sensors. For the ML algorithm, we drew from past knowledge about collecting data, over-fitting issues, and how to create ensembles of models to improve performance.

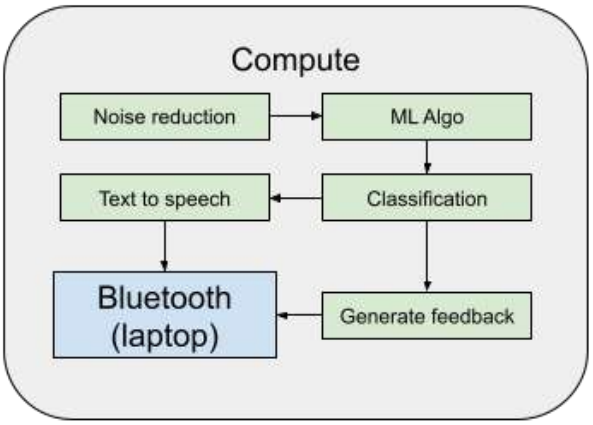
For math principles, we didn't use too much but we did use specific numbers to derive the resistors that we needed for our op-amp circuit design. This allowed us to circumvent the tedious process of testing various resistors and let us find the optimal resistance immediately.



(a)



(b)



(c)

Figure 2: System block diagram.

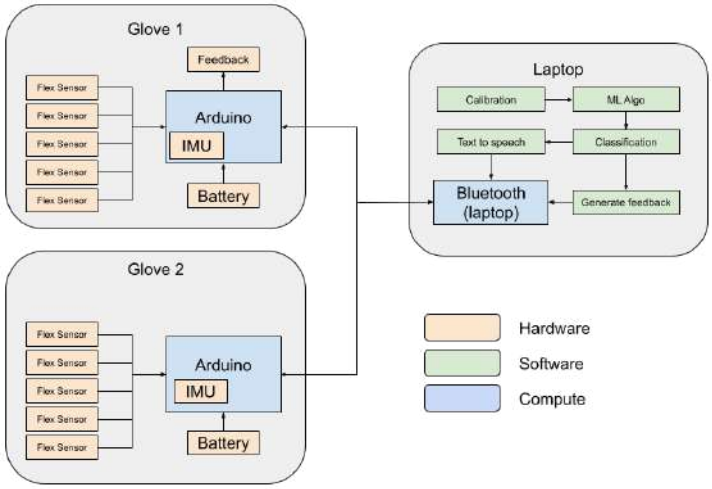


Figure 3: Full block diagram.

4 DESIGN REQUIREMENTS

To achieve our use-case requirements, we have set out the following list of design requirements. The actual requirement is marked in bold, while the explanation surround them.

An IMU was included in this list due to its ability to provide positional measurements. Most 6 axis and 9 axis IMUs include a triaxial accelerometer which will be integral to our data collection. **Since our design requires positional and gyroscopic data regarding hand placement, we found it optimal for space and weight considerations to use a board with an on board IMU.** There is a constant gravitational force pointed downward, and the x, y, and z components of acceleration of the glove at rest should add up to this download gravitational force. Given each component, we are presented with a unique orientation of the hand at some angle.

Further expanding on our sensor reading requirements, **we need at least 5 analog input pins on our board to connect five flex sensors to.** Typical flex sensors (and all of the ones we looked into) need a power source, a connection to ground, and a connection to an analog input pin on the board. This means that the board we choose must have at least 5 analog input pins. **We also need at least one digital out pin for our haptic motor**. In addition, we would like the range of voltage values outputted by the flex sensors to be as wide as possible. This is to ensure that the feature vector corresponding to one gesture is as distinct as possible from another. To accomplish this, we need to appropriately choose the pull-down resistor value that each flex sensor will be connected in series with.

The next feature we wanted to be built into our board was Bluetooth capability, preferable at least 4.x LE for its improved latency compared to older models. If we were to use a board without a built in Bluetooth controller, we would have to attach one to it and they are typically the length and width of an Arduino

Nano but much thicker than them. This would likely cause us to exceed our weight requirements and size use case requirements so we prioritized having Bluetooth built into the board.

Finally, we preferred to purchase a board with a sufficient processor that would be capable of transmitting data as fast as possible to our remote MCU and, in a later prototyping stage, running our ML model on board. We determined that it is approximately 32MHz. With 25 readings per second, this would leave us with approximately 5 million clock ticks in between readings. This leaves a lot of room for other computation that we may want to move onto the board later on. The board we decided on is the Arduino Nano 33 BLE Sense Rev2. The range of its IMU is ±16g. We decided to base our IMU selection on the board we choose (instead of first deciding which IMU meets our needs) because this project does not require a state of the art accelerometer and we felt that the one mounted on the Arduino Nano 33 BLE Sense Rev2 satisfied our sensitivity needs.

We then needed to determine latency targets for our data flow and transmission to get a better idea of how we will reach our target use case requirement of .5 seconds from sign to speech. For general device communication like sensor readings and based on BLE specifications, we can expect latencies of about 10 ms to 100 ms. Based on this data, **we will strive to achieve a sign to ML input latency of 100 ms**. We will achieve this by making the data we send as compact as possible when formatting sensor readings. **We also determined that we will allow 25 ms for our ML model to output a prediction** based on a past capstone project's results, and we will wait for 8 consecutive gestures that are the same to classify the predictions as a gesture with confidence. This gives us a total of 200 ms to classify a sign. **Then we will then strive for 200 ms for data to go back to the gloves for haptic feedback and speaker output.** Bluetooth latency for fully wireless headphones can range from 100 to 300 ms so we will set our requirement to 200 ms since we are not transmitting high quality audio data but want a conservative estimate. Adding these values together gives us 500 ms, our use case latency requirement. We will attempt to minimize latency in all cases and identify bottlenecks that can help us go below this number if possible. If we find that Bluetooth is a bottleneck, we will consider moving to a WiFi communication protocol with the Arduino Nano 33 IoT. See 5 for more information.

The ML model should be able to perform inference at a rate of less than 25ms per prediction. This metric is directly tied to the speed use-case requirement. We predicted that the bulk of the time delay will stem from the latency caused by the wireless transmission. With this in mind, we hoped that the classification section will only take up less than half of the allotted 500ms requirement from sign to prediction. Then, since our final classification module might require repeat ML model predictions, we divided that time by 8 to arrive at our design goal of 25ms.

The ML should be able to achieve a testing accuracy of 90% on excluded test data. This is directly tied with the accuracy use-case requirement. We plan on adding additional checks on the output of the ML model to make the entire system more robust to noise and interference. With that in mind, we at least wanted to start out with a solid baseline with a model that performs relatively well as a standalone. We arrived at 90% because we believe that we can correctly fix the small case of errors with our additional final classification paradigms we will present later.

Our software should be ready to collect around 2000 datapoints per letter with a variety of different signers. This design requirement is directly tied to our accuracy requirements and indirectly tied to our desired vocabulary size. The main hope with this requirement is to establish enough data to train all our ML models sufficiently while leaving enough wiggle room for additional testing or cross-validation. By seeking a variety of signers to collect our data, we hope to make the model more robust to achieve our use-case requirement in accuracy.

5 DESIGN TRADE STUDIES

Glove Sensors vs Computer Vision

When we came up with our initial goals to enable communication from deaf to non-deaf speakers, we were immediately faced with a decision to make: whether to use a glove-based sensor design or a camera with a computer vision backend.

A computer vision based approach would involve having a camera take images of someone else while they are signing. The resulting images would be passed into a CV/ML backend for classification. The pros of this method is that the high dimensionality of the input data allows for more minute movements to be detected. Even slight flexions in a particular finger might be detected due to the nuances of CV techniques. Additionally, we would most likely be able to make a less user-impactful product, meaning that a camera alone would most likely be easier to carry. However, the cons are that the CV setup requires a lot of training and a consistent background environment. Due to the high dimensionality of images, CV methods often require collecting a huge dataset to train the proper CNN architectures for performance. Collecting data can be very labor-intensive due to the time collecting the data as well as the additional effort to label each image. Additionally, there is a requirement to match the background and framing of the training data. If this requirement isn't matched, performance is very likely to suffer. This makes this product extremely hard to use in an uncontrolled environment.

A glove-based approach involves using flex sensors and IMUs to detect the positioning of the fingers and hand to predict the corresponding signed letter. The pros involve a quicker training process and a system that is more ro-

bust to varying environmental factors. The quicker training process stems from the smaller dimensionality of the data, namely only the flex sensors and the IMU. And since the readings will be relatively consistent regardless of varying conditions, we can expect a similar performance at all times. The cons are the sensitivity of the sensors. Being less information dense than an image, we are at the mercy of the sensors being able to convey enough information to make accurate predictions.

At the end of the day, based on previous groups' success, and the more robust design, we decided to choose our glove sensor to tackle the problem of deaf communication to non-deaf speakers.

New User Calibration / Adaptability

One of the initial considerations for the project was to develop an additional calibration mode for new users to adjust the ML model backend to fit the current user more closely. This was inspired by similar designs like Apple's face recognition or fingerprint identification. The premise is that the way that people vary from person to person and the classification might suffer if the variations differ from the training set too much.

Ultimately, we decided to put adaptability as a stretch goal of ours. This was due to our belief that the variability will be minimal enough to still maintain a relatively high performance even with outside testing input. If we have enough slack time, this would be a feature we would definitely implement to maximize accuracy across different users.

Choice of Sensors

We have decided to use Spectra Symbol's 'Original Flex Sensors' for a variety of reasons, the first being precedent—this type of sensor was most commonly used by similar projects. The second primary reason was cost. Since our design has two gloves, we need 12 sensors (ten for each finger, two for slack), so cost was important to keep in mind, especially if the goal of this project in the distant future is to be as accessible as possible. Spectra Symbol has since come out with the more lightweight and sensitive 'SpectraFlex Flex Sensors', but these were $12.5 more expensive than the sensors we went with. We were also looking into Bend Lab's flexible soft sensors, which were appealing due to their silicone-based material that seemed to offer higher sensitivity, but at $49.00 a sensor this was unfeasible in terms of cost. In addition, we considered other sensor options, like Hall effect sensors and conductive thread based sensors but opted against these as they were not as common and used by research groups with significantly more resources.

BLE Sense Rev2 Board vs IoT Board

As described in the design requirements, we are using the Arduino Nano 33 BLE Sense Rev2 board. This board has a 64MHz clock frequency, BMI270 and BMM150 IMU's on board (totalling to 9 degrees of freedom) giving us accelerometer, gyroscope, and magnetometer readings. It was stated that we are considering using the Arduino Nano 33 IoT in the near future due to its WiFi communication capability. WiFi is faster than BLE, which is why we will utilize it if we fail to meet latency use case requirements with Bluetooth. The reason we are not using WiFi right off the bat is because Bluetooth is simpler to use and the BLE Sense board has more computing power. Since we plan to move some computation to the board in our final prototype as described in system implementation, it was clear that our priorities meant that the BLE Sense board was more appropriate. However, the 48MHz clock speed on the IoT board is not trivial and we may be able to take some advantage of its compute power if the BLE does not allow us to meet latency requirements.

One other factor we considered is that the IoT board's IMU is the LSM6DS3 which gives us 6 degrees of freedom whereas the BMI270 and BMM150 on the Sense Rev2 gives us 9 (additional magnetometer). While we do not foresee using a magnetometer currently, it may be a datapoint of interest if we decide to change the kind of data we gather at any point. This may become useful if IMU and flex readings for two gloves do not give us the accuracy we are looking for causing us to want data about how far the hands are from each other using a magnet. This additional benefit of the BLE Sense further confirmed that we should start with that board.

Presence of a Remote MCU

Part of our MVP design is that we have a laptop running the classification model as well as noise reduction. Past projects have used a smartphone or laptop to show proof of concept speech output, however we believe the product will be more versatile and useful if there is no dependency on these devices. However, we are aware of the complexity that such a change could bring to our system. Therefore, our system implementation discusses how we will achieve wireless, independent functionality only after our MVP requirements have been met.

6 RAPID PROTOTYPES

We have adapted a rapid prototype approach to completing our product within the given class time.

Prototype 1

Our first prototype will be a one glove design with flex sensors and an Arduino mounted on the glove. One of the team members' laptops will be used to do computation and speaker output will be from the laptop. The "haptic feedback" will initially be an LED lighting up on the Arduino to visually inform the user that a sign was successfully converted to speech. Communication will be done via USB for

this prototype. Our design will be able to interpret 10 ASL letters. We chose this number because it is small and does not require an ample amount of data to be collected but it is large enough to give us some idea of our system accuracy.

Prototype 2

Our second prototype will have wireless communication (initially Bluetooth but optionally WiFi if we do not meet latency requirements). We will duplicate our working glove with one having a haptic vibration motor attached. This design will be trained against data representing the 10 words ASL vocabulary as specified by or use-case requirements. This data will need to be recollected as our hardware and vocabulary changes. We have also developed a PCB and outside case to hold the circuitry as part of this prototype. Complete wirelessness is also achieved via battery power. The overall physical design will be streamlined during this prototype to make the system as robust to damage as possible as well.

7 SYSTEM IMPLEMENTATION

Physical Design

A significant step in our architecture design was deciding how to mount everything on the user's hands. Below is a picture of our final glove design.

Figure 4: Final Glove Design

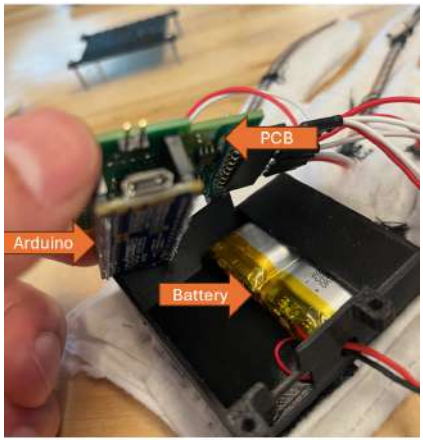


Figure 5: Case internals

The flex sensors are attached to the glove using a blend of stitching at the joints with super glue used at the extremities of the hand. The flex sensors are connected to the main PCB unit through the soldering of some wire connections. Then, the battery and the PCB are placed onto their carrier box, which was made via 3D printing. Once the two components are placed into the case, the case is covered with the help of screws and a bolt. Then, the case is secured onto the glove with the use of super glue and velcro straps. Lastly, the haptic motor on the left hand is attached with some glue. The glove system can be turned on wirelessly by having the user attach the battery connector wires to the extruding pins.

Flex Sensor Integration

Flex sensors are typically used in conjunction with a pull-down resistor to form a voltage divider circuit. This is because a flex sensor is a variable resistor that changes its resistance based on the degree of bending. As such, a voltage divider configuration allows for the conversion of the resistance variations into a voltage signal that can be easily read by the Arduino's analog-to-digital converter. In addition, the pull-down resistor ensures that there is a defined voltage range across the flex sensor—we want as large of a range as possible as this implies distinctiveness across the data readings for each sign. In terms of unit testing, we will test a series of resistors within a predetermined range with a few of the letters in our MVP letter set to see which produces the largest range of V_{out}. The flex sensors we chose have a resistance range from $\approx$ 10,000 Ohms (flat resistance value) to 20,000 Ohms (2x the flat resistance), and our power supply is 4.8V.

Below is how we determined what this range will be, given that we ideally want a voltage output range of 1V to 4V:

$$1 \leq 4.8V \cdot \frac{R_{pull}}{10,000 + R_{pull}} \leq 4 \quad (1)$$

Solving for R_{pull}:

$$10,000 + R_{pull} \leq \frac{4.8R_{pull}}{1} \leq 4(10,000 + R_{pull}) \quad (2)$$

Lower Bound: $10,000 + R_{pull} \leq 4.8R_{pull}$
Subtract R_{pull} from both sides:

$$10,000 \leq 3.8R_{pull} \quad (3)$$

Divide by 3.8:

$$R_{pull} \geq \frac{10,000}{3.8} \quad (4)$$

Upper Bound: $4.8R_{pull} \leq 4(10,000 + R_{pull})$
Distribute 4 on the right side:

$$4.8R_{pull} \leq 40,000 + 4R_{pull} \quad (5)$$

Subtract $4R_{pull}$ from both sides:

$$0.8R_{pull} \leq 40,000 \quad (6)$$

Divide by 0.8:

$$R_{pull} \leq \frac{40,000}{0.8} \quad (7)$$

Thus, by simplifying the inequality, the pull-down resistor range becomes $2631.51\Omega \leq R_{pull} \leq 50000\Omega$. The most readily available resistors within this range are $2.7k\Omega, 3.3k\Omega, 4.7k\Omega, 10k\Omega$ and $47k\Omega$, so we will test all of these. Below is a circuit schematic of our flex sensors' wiring.

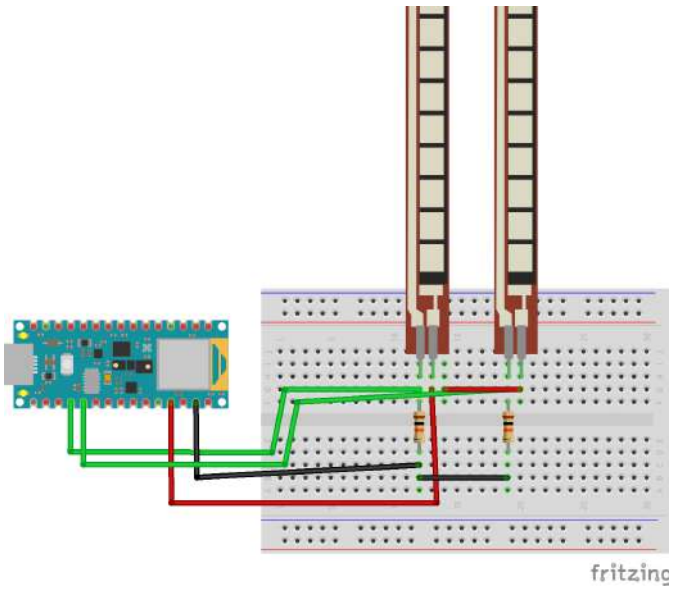


Figure 6: Flex sensor circuit diagram.

Initially, we planned on using a protoboard for our final design encompassing the operational amplifiers to widen the flexion voltage range as well as the haptic motor driver for the feedback system. However, we found that this system was too bulky and the attachment to the glove was not secure. This caused us to collect and read noisy data during detection which likely hurt our performance. Therefore, we decided to make a small 1.5"x 1.5" PCB that we can mount our Arduino onto and plug our sensors into.

Data Communication

Our final product will rely on Bluetooth communication between the Arduino and the python script on the laptop. On the Arduino side, Bluetooth is handled via the *ArduinoBLE.h* library. This essentially sets up the individual Arduinos as advertisers of a specific service as they continually transmit the data they read to all connected subscribers. On the laptop side, Bluetooth is handled via the *SimplePyBLE* package in Python. This package allows for the easy connection to BLE devices from a Python script. It follows a standard flow of selecting the adapter (laptop), the BLE device, and then the service it should subscribe to. Note that these steps can be automated if the Python script knows what MAC address the Arduinos use which is the setup we have for our project. Additionally, note that BLE communication must always be done in bytes so we have additional code on the Arduino side to convert the float values to bytes and the reverse operation within the Python script. Note that we attempted to use the *Bleak* package in Python to handle BLE communication initially but scrapped it due to its lack of compatability with Windows.

Data Collection

We will implement a rigorous number of scripts in Python to handle data collection from all possible avenues. Recall, though our system will be primarily implemented for Bluetooth (wireless) performance, we will have access to a USB connection to the micro-controller as well. Therefore, we plan on designing a suite of scripts that can handle data input from both a wired connection and a wireless connection. Within the system, the data from the flex sensors and IMU will be formatted in a pre-defined, consistent way to make sure there are minimal issues with data compatibility and reformatting. We will develop multiple scripts to handle collecting data one class at a time or a consecutive script that iterates through all classes for a full dataset collection in one shot.

The data will be processed using the calibration metrics for normalization. This is new as our original approach relied on the raw input readings from the Arduino. We believe that this approach will allow the product to be robust against a variable set of hand sizes. We also want to emphasize that for our purposes we primarily used the wired, manual data collection script. This method allows us to collect data at the fastest rate and allows us to adjust the

sensor system more easily if we spot an issue within the collected data.

Machine Learning Classification

The primary backbone of the software component will be focused on the classification algorithm. We had originally planned on testing four different types of models. But brief tests into accuracy and speed revealed that neural networks had the most accurate predictions with no sacrifice on inference speed. Therefore, we focused the majority of our efforts into refining the neural network architecture.

The training of models will be handled directly from the laptop's computing unit. The small scale of our dataset did not warrant the use of a GPU for training so all training was handled using our local laptop's CPU. The trained models will be saved in a separate directory, and can be loaded for additional training or test-time inference. Saving of models took leverage of the *joblib* and *pytorch* packages.

Within the training of models, there will be a heavy emphasis placed on cross-validation training. For neural networks, there are a large variety of hyperparamaters and design choices that can affect performance. The most relevant two that we looked into were the sizes of the hidden layers and the choice of activation function. We eventually settled on a size of 128 for our hidden layers as a balance between speed and accuracy. We also settled on the sigmoid function through testing as it seemed to have better performance than a ReLU activation function.

For inference, we will have a final script that will load in a pre-selected model that we have fully trained. This model will continuously receive inputs from the microcontroller and make predictions on the letter that is being signed. Then, it will pass the predictions to another module responsible for final classification.

Final Classification Module

To prevent potential issues from noise or transition states between different signings, we plan on implementing an additional module that will take in the output from the ML model and make a final prediction on the signing of a letter. This module will collect and store a number of the past ML model outputs and identify patterns to predict if a sign is actually being signed. We will primarily implement this as a repetition check. This means that we will like to see a specific number of repeats of the same classification prediction before we are confident in the ML model. Through testing, we determined the optimal number to be 4 repetitions to balance accuracy and speed of inference. This idea was inspired by past implementations from Gesture Glove [3] .

Feedback System

Once a classification is generated by our classification module, we will send a signal to each glove indicating the event that has occurred.

Haptic Feedback

When designing our system, we realized that there is a huge need for some form of feedback given to the deaf user to let them know that their word has been signed. We initially planned to have a complex system where specific haptic signals would denote specific meanings. For example, one beep for a successful sign, two beeps for an incorrect sign. We realized, however, that this leaves a lot of gaps in the deaf user's understanding of the system. How would the system know if the sign is incorrect? How will the deaf user know if the sign is actually successful? The solution we came up with for this was to have an LED screen on the glove that could tell the user what word/letter was signed. This was out of our scope for this project's timeline, so we decided to do a simple vibration to indicate that something was signed, without any relation to if it was what the signer intended to sign. We have also added a beep for before and after calibration to let the user know when they should be flexing their fingers to calibrate. This is just a first step in the process of incorporating reliable haptic feedback into the product.

Audio Output and Text to Speech

We used the *win32com* library to create audio signal data. We originally were using the *pyttsx3* library but that has a requirement of a single-threaded environment that conflicts with *SimplePyBLE*'s multi-threaded requirements. Obviously, this limits the audio capabilities to Windows devices.

Custom PCB

We created a custom PCB with amplification circuit and haptic motor driver I2C circuit on board. Users simply have to plug in the battery, haptic motor, flex sensors and the Arduino. This allows for a more compact design of the board.

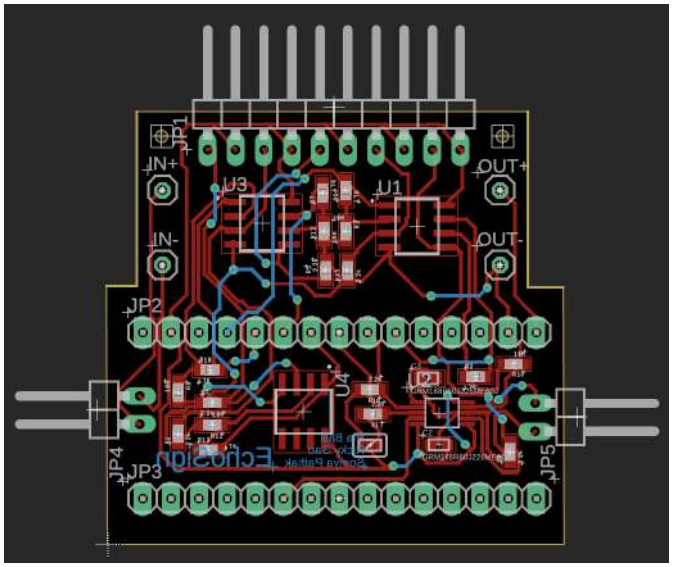


Figure 7: PCB diagram.

8 TEST & VALIDATION

Unit Tests

Unit Tests for Sensors

We want our sensors to have as high sensitivity as possible, as low sensor sensitivity is commonly listed as being a challenge past projects faced. As such, we tested a spectrum of pull-down resistors to be used in conjunction with the flex sensors that exist within a specific range (see System Implementation for the mathematical derivation of this range). We also tested each resistor value on a few signs that are distinct in shape, and whichever resistor yields the widest range in voltage output we will select for our final glove circuitry. In addition, since we still encountered low sensitivity issues, we incorporated op-amps into our design, as they can amplify the small voltage variations produced by the flex sensor to a more measurable and usable level.

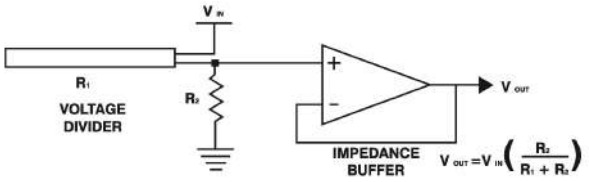


Figure 8: Circuit diagram of flex sensor, pull-down resistor, and LM358 op-amp

Furthermore, after the calibration phase prior to data collection, we manually inspected the flex sensor readings to ensure they were on par with what we expected given the handshapes. We also made sure to tug on the sensors after soldering to try and mitigate breakage mid-signing, as sometimes the sensors were prone to breakage especially at the joints of the fingers.

Unit Tests for PCB

Our PCB design was relatively simple. To test, I first plugged in the Arduino and powered it using the USB cable. Then I tested that all flex sensors had appropriate reading values. Next, I plugged in the haptic motor driver to the protoboard, tested a program on that, then moved it over to the PCB to test there. Once that was working as expected, I plugged our battery into the PCB and saw that the Arduino lit up appropriately and we could carry on with our Bluetooth connection as before. One deviation of plans we faced was that we were no longer using a step-up DC-DC converter that we considered at some point. This is because it draws too much current and drains the battery too fast due to the inductor on board. The PCB we fabricated came with mounting pins for the converter, we soldered two small wires onto the board to route the power rails and then plugging in the battery was successful.

Unit Tests for Arduino Board

In order to unit test the sensor input from the Arduino, we started by recording the maximum and minimum values of each flex sensor. We checked what output we would get if the sensor was totally detached or attached backwards. This helped us understand inconsistencies and errors that we got later down the road during integration. We then tested Bluetooth integration at various distances from the computer. Trivially, at closer distances we saw good results and lower latency than at further distances. We were able to see success at up to 30 feet away which we felt was a sufficient amount of distance to demonstrate from for our purposes. We did notice in some runs the connection would be finicky if there was a lot of Bluetooth Low Energy in the room nearby, however this is a problem we did not think was feasible to resolve with the given time constraints.

Tests for Data Evaluation

Because the data is extremely critical for the performance of the ML model, there were several steps we took to ensure it's quality. Testing of the data basically entails inspecting the flex sensor values and comparing them to our own assumptions for what the data should look like. This allows us to quickly check for hardware issues and potential outlier values. Below is an example visualization of the data from our dataset.

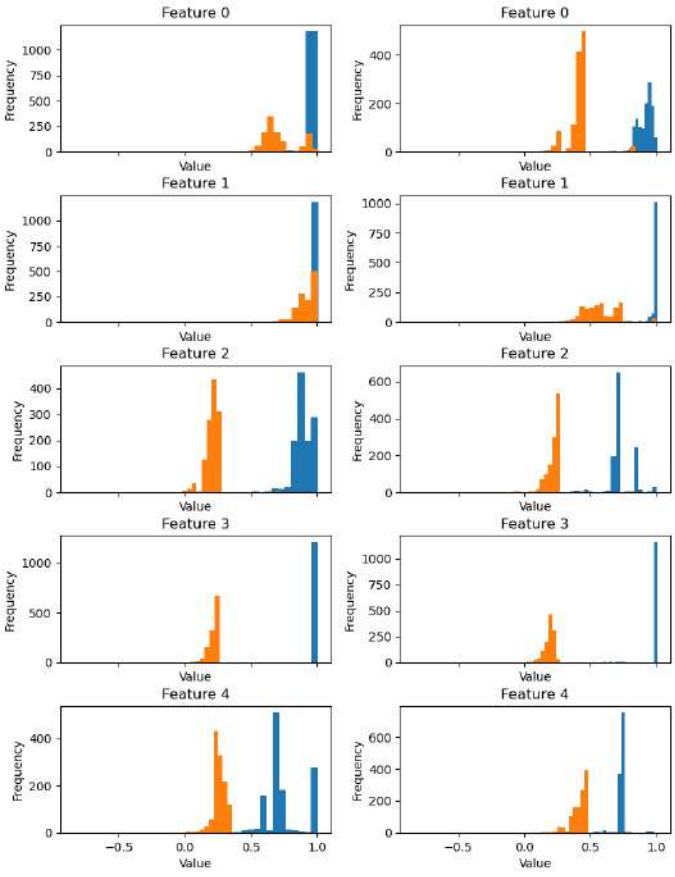


Figure 9: Sample data from the time word. The left side is data from Ricky, the right is data from Somya. The top plot represents values from the thumb, and the bottom represents values from the pinky, with the rest of the fingers in between. The blue values represent the right hand, the orange represent the left hand

By manually checking the data for each word, we were able to verify that the dataset was consistent with our expectation and free of outliers.

Tests for ML Model Speed

Latency of the ML model was calculated by using Python's timing module to evaluate the time for inference when data is passed into the model. When calculating the metric, we looked and averaged the latency over 50 signs. The time that they all took was infinitesimally small. Therefore, we concluded that the delay caused by the ML model is infinitesimally small. This met our original design requirement which stated that the latency should be at most 25ms.

Tests for ML Model Accuracy

For testing the accuracy of the ML model, we used a 80-20 split of our original dataset into training and test data. Then, after the model has converged over the training set, the accuracy performance of the ML model is evaluated on the test data. This metric was evaluated to be **97.4%**. This meets our design requirement which states that the ML model should have a test-time accuracy of at least 90%. Pictured below is the test accuracy over the epochs as the ML model trains.

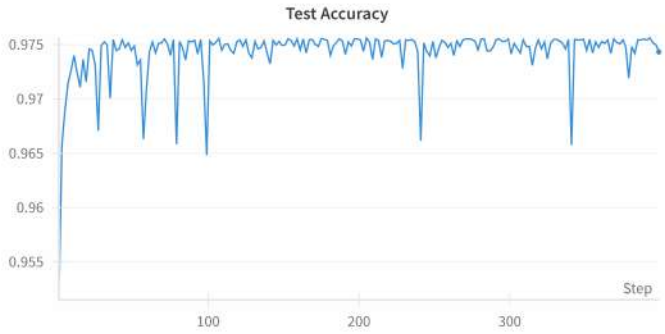


Figure 10: Test Accuracy as ML Model Trains

Use Case Tests

Battery Life

In order to test the battery, we attached it to the glove and made approximate measurements of how long they last. Based on our calculations, it should last at least 2 hours. Our battery is two 3.7V LiPo batteries in series each with 550 mAh. Our circuit's current draw is 75 mA, so our system should be up for 7 hours and 20 minutes which is well above 2 hours.

Weight

We also want to ensure that the weight of each glove is less than 100 g. Each component in our design added together is approximately 50 grams with some give or take. Adding wires and more slack gives us a 100g total that we will stay under at every prototype completion checkpoint. In order to minimize weight of our glove, we got rid of our breadboard and printed an enclosure and PCB for our design. Weighing all components, we calculated the total weight for our gloves to be 71.7g for the left and 70.8g for the right. This is well below our goal of 100g.

Testing Overall System Accuracy

These tests help us address our main use-case requirement of accuracy. To test this, we had one of us sign our entire vocabulary several times and recorded the corresponding prediction. Specifically, we signed each words approximately 12 times, in a random order, and recorded the prediction (from speaker output) of the glove system. This style of testing allowed us to find that the glove had a real-time accuracy of **89-90%**. This range was established from multiple runs of this testing. This result does meet our use-case requirement of $\geq 85\%$. Below is the confusion matrix from one of our runs of the tests.

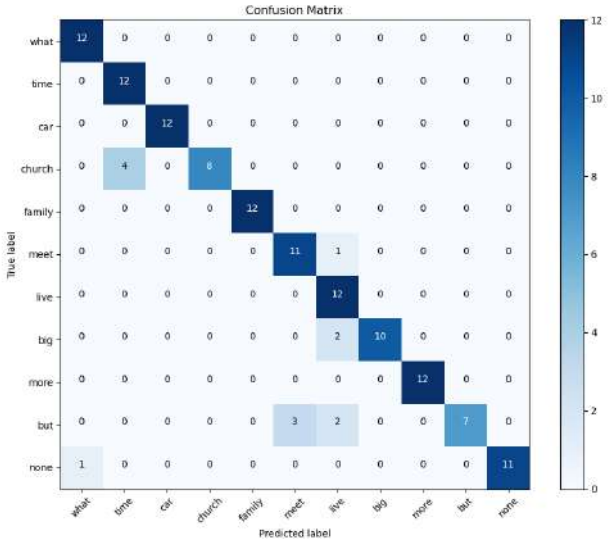


Figure 11: Confusion Matrix of Real-Time Testing

Notably, we have issues with the model distinguishing "time" from "church" and "but" from "meet". However, these errors are not significant enough to keep us from meeting our use-case requirement.

Testing Overall System Speed

Our tests for speed was done concurrently with our tests for accuracy. During each prediction, we take advantage of Python's timing module to evaluate the time it takes from the initial sensor input for the system to make a firm prediction and generate an audible response. By taking the time for all the predictions and averaging them, we settled on a value of **0.6-0.7s**. Unfortunately, this is slightly higher than our design requirement. However, we believe that given other trade-offs, this would be an acceptable outcome for our product.

9 PROJECT MANAGEMENT

Schedule

The schedule is shown in Fig. 13. We apologize for the small font. A better computer version is available on our website for reference.

The major changes involve the development cycle for prototype 1 extending past the anticipated time during the design report. This was due to the difficulty in integrating the flex sensors as we worked out the exact nuances of the glove design. Also, note that we used up all our slack time due to our change in MVP. Since we changed the vocabulary, this meant that we took a few extra weeks to collect data, process it, train a model, then benchmark performance. This ate up all of our slack time, so we finished just in time.

Team Member Responsibilities

Ricky primarily on the ML integration, software, and Bluetooth communication. Ria primarily worked on glove hardware design and implementation, including PCB design and robustifying the entire structure. Somya worked on sensor interfacing, haptic feedback design, and robustification of flex sensor design. We will collect data, test, and perform integration altogether.

Bill of Materials and Budget

The bill of materials is shown in Table 1.

Risk Mitigation

Ideally, our product would allow for complete wireless sign language detection with all compute on the glove as well as speaker output coming out of the glove. In order to do this, we drafted a circuit with a small 8-ohm 1-watt lightweight speaker that could amplify the sound signals coming from Arduino to convert to audio. We looked at various Arduino libraries that would allow us to easily convert text to speech, namely Talkey. After reading the documentation, we found that it only worked with specific architectures, and the Arduino BLE Sense was not listed as a compatible board. At this point, we had already been using Bluetooth comfortably to communicate data between the gloves and our machine, so we decided to scrap using an onboard speaker and stick to using our laptop speaker with a different library. We found it risky to switch boards entirely and iron out a new communication protocol and it would be really time-consuming to generate audible clean audio output from our small speaker. Therefore, our risk mitigation plan here was to eliminate this part of our final design.

A major issue we noticed when testing our gloves was that the fully flexed state for one person might be very different for another person. If we train our model on raw flex values, one person's fully flexed finger might get confused with another person's slightly flexed finger and this will make it difficult for the machine learning model to differentiate between the two. In order to mitigate this risk of bad performance, we decided to add a calibration phase for the glove. This would involve a user to fully flex and stretch out their fingers for five seconds before sign detection begins. The program will detect the maximum and minimum flexion of the fingers and map all subsequent readings to a value between 0 and 1 depending on where it was in the user's range of flexion. This normalization of data was necessary to minimize unwanted uncertainty and errors with the machine learning model we were using and it resulted in us seeing better performance in later testing.

One of the major risks that we encountered was the performance of the ML model. The first few times we trained a model and tested real-time performance, the accuracy of the model was around 50%. This was very concerning since the test-time accuracy was around 97% which sug-

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Flex Sensors	182	Adafruit	12	$12.95	$160.55
Arduino Nano 33 BLE Sense Rev2	ABX00069	Arduino	2	$39.24	$239.03
Pair of Cotton Gloves	N/A	COYAHO	1	$6.99	$246.02
USB to USB-C Adapter	N/A	JSAUX	2	$8.09	$262.20
USB cables	N/A	JSAUX	2	$8.09	$278.38
PCB Fabrication	N/A	JLCPCB	1	$42.89	$321.27
Wires (Male-Male)	N/A	JSAUX	1	$8.99	$330.26
Wires (Female-Female)	N/A	JSAUX	1	$8.99	$339.25
3.7V LiPo Batteries	N/A	Adafruit	10	$5.99	$399.15
					$399.15

gested that the model had converged on the optimal solution based on the training set. To mitigate this issue, we tried two approaches. We first identified the words that were commonly mistaken with each other and trained separate models on that subset of confused words. Then, we developed a cascade of model architecture in which if the output of the wider model was one of the commonly confused words, it would then again pass the same input to the smaller model for refined prediction. This allowed us to boost performance to the metrics stated earlier. We also played around with a time-series approach where we trained a model on the last 5 frames of data. Performance of this model was not as robust but served as an alternative to the naive initial approach.

When we were first testing performance with the BSL alphabet, we were immediately alerted by the poor performance of the system. Upon further investigation, we discovered that much of the alphabet was dependent on touch, which made our system inadequate for detection. With this in mind, we pivoted to the ASL word vocabulary to mitigate the risk of complete system failure due to sensor inadequacy.

There was also some concerns about the sensitivity of the flex sensors when we started the project as well. However, upon testing the system's performance with the flex sensors, we determined that the classification algorithm was robust enough and it was no longer a risk to consider.

10 ETHICAL ISSUES

Current society is progressing towards providing various disabled communities with tools that they can use to seamlessly navigate daily life. Our goal is to extend this mission to users who speak sign language wanting to communicate with people who don't. This is a step towards establishing better conversions between the deaf community and those who can hear. There are a few ethical issues that may arise with our product that we would want to improve upon and eliminate if given more time. When we conceptualized this idea, our goal was to bridge the communication gap between those with hearing difficulties and those without. Our final design only addresses one way communication but fails to bridge the gap between able users speaking to deaf users without sign language. In terms of safety and usability, our product could be really useful for the deaf to communicate in an emergency situation that could arise in any location where there may not be anyone that understands sign language. In such situations, timely communication is of utmost importance, and our product will be designed with this use case in mind. However, having emergency signals reach deaf people without the use of sign language can pose a threat to them and could put them in unsafe situations. Nonetheless, our product not only attempts to solve a complex problem, but it also raises awareness about the nuanced challenges that the deaf communities face every day. By learning more about their needs and struggles, we are aiming to spark conversations - pun intended - about how we can go even further. We hope that this project adds a few cobblestones to the road being paved towards a more inclusive society.

11 RELATED WORK

Sign-to-speech translation using machine-learning-assisted stretchable sensor arrays [4] This project was a collaboration between UCLA and Chongqing Normal University to develop a new wearable system that can detect a variety of signings. Their system consisted of a singular glove paired with additional sensors on the face to detect facial expressions. Their system primarily innovated on the development of their own yarn-based flex sensors to track the finger flexion. It also innovates on the use of additional facial sensors to convey more meaning than can be conveyed with solely the hands.

Although their research was a significant undertaking with many collaborators, we hope to innovate on their work by incorporating the use of the second glove. We believe that if we can demonstrate a system with two gloves, we can greatly improve the possibilities for communication between deaf and non-deaf speakers.

Gesture Glove [3] This was a past capstone group who also sought to develop a glove that could read in ASL sign language and convert it to speech. Their design uses flex sensors, IMU, and tactile sensors as input. These are

processed using ML to predict letters at output. Notably, their design requires a wired connection to their laptop for performance.

We hope to use their insights and experience as a springboard for us to achieve higher accuracy and speed metrics. We also hope to innovate from their past design by implementing the bluetooth (wireless) component so users can sign more freely.

12 SUMMARY

To summarize, we have provided in this report a comprehensive overview of how our gloves will be designed from a hardware, signal processing, and software perspective. We have provided justification for each critical design choice, namely our selection of sensors/computing units, as well as the machine learning training scheme we plan on using to enable accurate gesture identification. We have made these decisions keeping the user's experience in mind. We have also outlined specific metrics that our system was able to achieve. We were able to meet our requirements for vocabulary and accuracy but were slightly short of our latency goals. This speed issue could be addressed by using different forms of communication beyond Bluetooth, like WiFi. However, we are confident that our work serves a great first step into the possibilities of dual-glove deigns for ASL translation.

12.1 Future Works

For future works, there are many avenues to consider. The first is an increase vocabulary and the inclusion of additional sensors. One of the primary reasons why we could not incorporate a larger vocabulary was due to the various words in the ASL language that incorporate arm movement or touch. Both of these would not be able to be detected given our sensors. Future work can explore those avenues. Additionally, we hypothesize that a more rigorous assessment of different NN architectures could improve performance. We primarily stuck to a fully-connected architecture due to the scale of our project, but if one could build on our datasets, additional architecture testing could prove beneficial.

12.2 Lessons Learned

For lessons learned, one important thing is to always test components before assembly/integration into the system. We had many issues with our flex sensors at the beginning of the project and the PCB towards the later end of the project. We made the mistake of integrating those components and then having to replace them after complete system failure. This could be alleviated with more rigorous unit tests. Also, we recommend finalizing the physical design as soon as possible. This allows for software development to properly match the expected final design. We wasted a lot of time redeveloping software for a changing hardware which made life difficult. Lastly, be prepared to adapt. We changed our MVP and various stages of our prototypes to ensure we would be able to ccomplete on time. Being flexible will allow you the best chance for success.

Glossary of Acronyms

- ASL – American Serial Bus
- AWS – Amazon Web Services
- BLE – Bluetooth Low Energy
- BSL – British Sign Language
- CNN – Convolutional Neural Network
- CV – Computer Vision
- EDA – Exploratory Data Analysis
- GPU – Graphics Processing Unit
- IMU – Intertial Measurement Unit
- IoT – Internet of Things
- kNN – k Nearest Neighbors
- MCU – Main Computing Unit
- ML – Machine Learning
- MVP – Minimum Viable Product
- NN – Neural Network
- SVM – Support Vector Machine
- USB – Universal Serial Bus

References

[1] Ursula Bellugi and Susan Fischer. "A comparison of sign language and spoken language". In: *Cognition* 1 (1972), pp. 173–200. DOI: 10.1016/0010-0277(72)90018-2.

[2] Biyi Fang, Jillian Co, and Mi Zhang. "DeepASL: Enabling Ubiquitous and Non-Intrusive Word and Sentence-Level Sign Language Translation". In: *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. SenSys '17. ACM, Nov. 2017. DOI: 10.1145/3131672.3131693. URL: http://dx.doi.org/10.1145/3131672.3131693.

[3] Sophia Lau, Rachel Tang, and Stephanie Zhang. *18-500 Design Review Report - Oct 11, 2021*. Gesture Glove. 2021.

[4] Zhihao Zhou et al. "Sign-to-speech translation using machine-learning-assisted stretchable sensor arrays". In: *Nature Electronics* 3.9 (2020), pp. 571–578. URL: https://www.nature.com/natureelectronics.

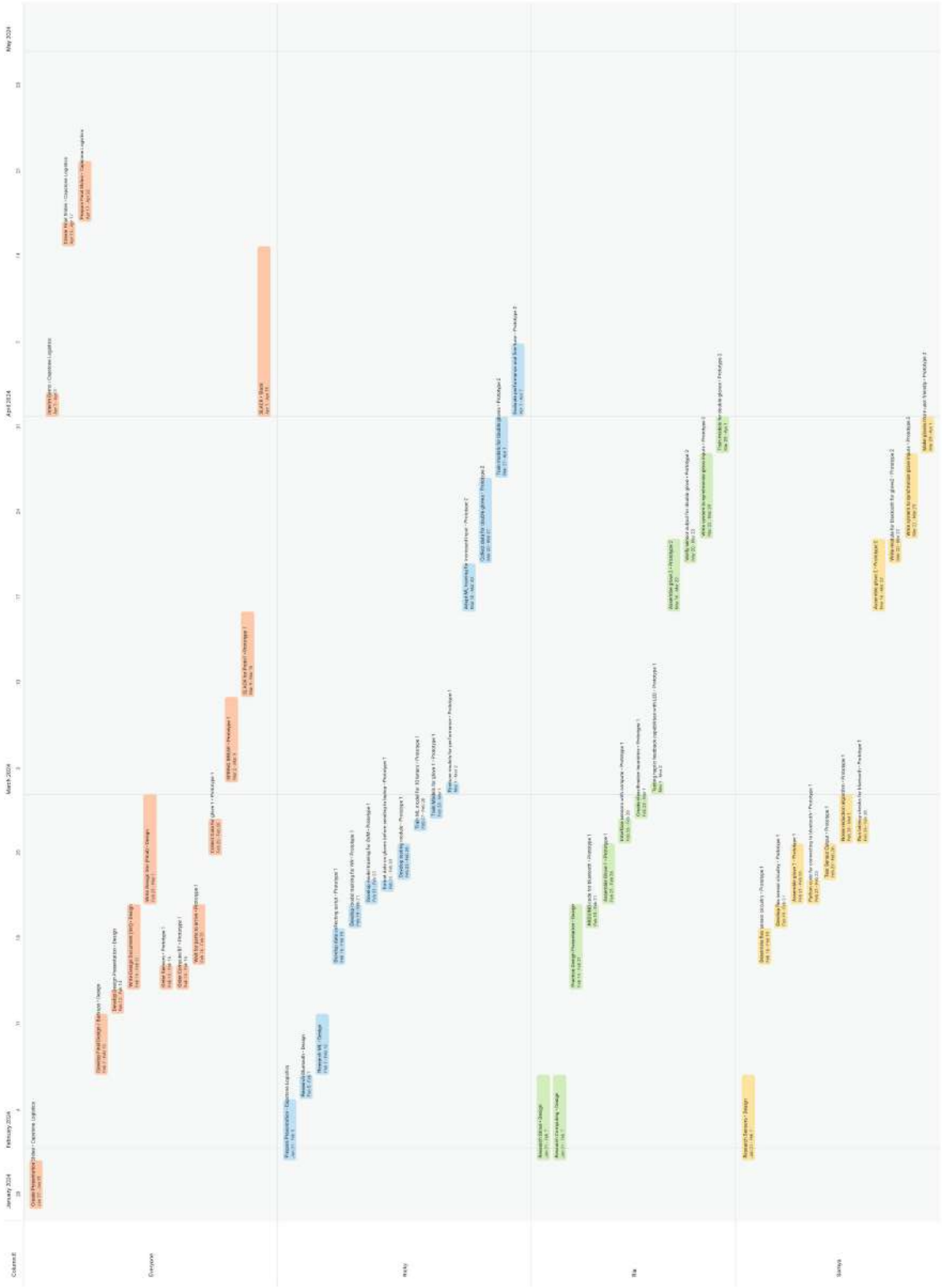


Figure 13: Gantt Chart