Gesture Glove

Authors: Sophia Lau, Rachel Tang, Stephanie Zhang: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— This project explores the use of sensors in sign language detection. We built a glove fitted with sensors to recognize and translate the 26 letters of the American Sign Language (ASL) alphabet. Our goal for this glove is to help facilitate smoother communication between ASL speakers and non-ASL speakers by displaying the letters an ASL user signs as audio output for others to understand. Many gloves similar to ours have been made to classify the letters of the ASL alphabet. We draw inspiration from and improve upon these products by attempting to increase classification accuracy, generalizing to all users, as well as adding the audio output as a way to increase practicality of the product. We quantify the effectiveness of our gesture glove by analyzing accuracy as well as measuring latency and frequency of recognition.**

***Index Terms*— American Sign Language (ASL), ASL recognition, Fingerspelling, Flex Sensors, IMUs**

1 INTRODUCTION

The motivation behind our project is to help people who sign in American Sign Language (ASL) communicate with those who do not understand it. Thus, our target users are people who use ASL as their primary form of communication, such as those who are hard of hearing or are not able to speak. To achieve this goal, we are creating a system containing a glove and a computer that will recognize gestures and display them as audio output through speakers. From a user standpoint, we would ideally have a gesture classification accuracy of 100% and an undetectable latency (less than 15 ms).

We have chosen to create a system using sensors as opposed to taking a computer vision approach. An advantage of using sensors over computer vision is that sensors allow the product to be more portable since it would not require the user to have a camera facing them at all times in order to detect gestures. Additionally, sensors are less variant to external factors such as lighting. We also use a Random Forest Classifier to determine what is being signed by the user based on the measurements we receive from the glove.

2 DESIGN REQUIREMENTS

As the Gesture Glove will be a worn device, it's crucial to provide a comfortable user experience and accurate sign recognition. Based on the user requirements we have previously mentioned in the introduction, we have established the following design specifications regarding the product's accuracy, latency, gesture frequency, and craftsmanship.

The Gesture Glove should have a classification accuracy of at least 90% when classifying all 26 letters in the alphabet. This accuracy requirement means that whenever the user makes an ASL sign, the classification model should output the correct letter corresponding to the sign 90% of the time. Ideally, we want to achieve 100% accuracy to satisfy the user requirements and to ensure top-quality experience. While it may be quite difficult to reach such an accuracy with any type of classification models, having a recognition rate of 90% or more will not greatly affect user experience. This requirement is based off the average typing accuracy of roughly 90% [**1**]. Since typing and signing are common forms of communication, it's reasonable to have approximately the same accuracy. One thing to recognize is that with auto-correct as well as the backspace key, we can actually reach 100% accuracy with typing. However, without the implementation of an auto-correct algorithm, we aim to reach as high of an accuracy as possible with 90% simply as a baseline.

The accuracy of our glove can vary based on how long a user holds a sign. The longer the user holds a sign, the gesture they are holding will become more static and the most accurate measurements for classification will be captured since most of the ASL letters are static. To get the most representative accuracy, it is important that we capture the accuracy of both statically captured data as well as data involving some movement– we test this by segmenting our collected data into training and testing data as well as testing the glove in real-time (real-time usage of the glove will inherently have more movement to the gestures).

In terms of testing, we will be performing accuracy tests with various users and making adjustments to the classification model as needed if the product does not reach a satisfying recognition rate. Along with accuracy tests, we will perform an experience survey to see user satisfactions with our product's accuracy.

The Gesture Glove should have a latency of at most 100ms. Besides having a high accuracy, the Gesture Glove should also output the results of the sign recognition in a short amount of time. Our goal aligns with the user's requirement: to have an undetectable latency for the Gesture Glove. To determine an ideal latency, we start by looking through various research papers. We find that while users can detect latency as low as 33ms, the user experience do not suffer significant losses until latency is over 100ms[**2**]. Latencies over 100ms are much more noticeable to the users and significantly degrade user experience. We have determined that having a latency of less than 100ms is ideal. To ensure we achieve this latency, we will set up timers in our code to check for how long the communications will take

between the glove sensors and the computer and how long the classification model takes to recognize a sign.

The Gesture Glove should recognize two signs per second. Upon researching the rate of ASL signing, we find that mean signs per second is around 2.5 signs, not taking into account of pauses in between signs[**3**]. Considering this information, we have determined that an average person likely makes two signs per second, including pauses between signs. Since we want the users to feel natural when signing (not too fast and not too slow), we decide to use the average signing rate as a baseline for the frequency.

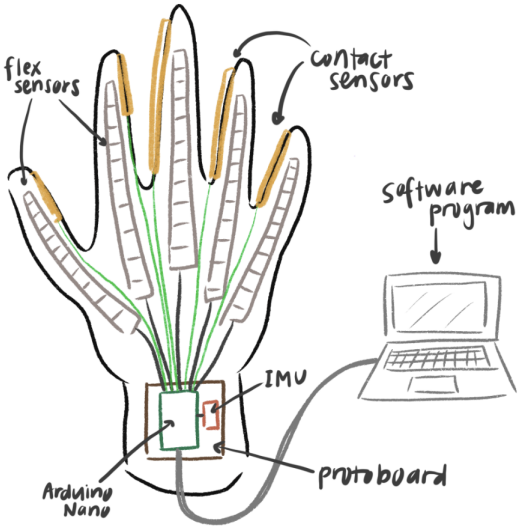
The Gesture Glove should have good craftsmanship, Our product will be a wearable device and its weight and stiffness should not affect the user experience, especially in making signs. After the glove is fabricated, the total weight of the glove should not exceed 200 grams so that the users can feel comfortable while wearing. The completed glove should also be flexible. Bending of fingers should be easily achieved without users feeling impeded when signing.

We also got feedback from actual ASL speakers on the practicality of the glove and ways we could improve our project.

3 ARCHITECTURE OVERVIEW

3.1 System Sketch

Our solution approach entails installing flex sensors along the length of each finger on a glove, installing touch sensors in between each of the fingers and on the pad of the thumb, and placing an IMU and Arduino Nano on the back of the glove near the wrist. The placement of the components is visualized in Figure 1.



(a)

Figure 1: Overall system sketch

The flex sensors collect information about the bend of each finger. The touch sensors detect which fingers the thumb is in between and whether the index and middle fingers are in contact. The IMU collects information about the orientation of the hand such as if the palm is facing up, down, left or right. The Arduino Nano samples the data from the sensors and formats the values. The Arduino Nano then sends the values over a serial connection to the computer which analyzes the data and runs a classification algorithm to determine the letter made by the pose of the hand.

3.2 Block Diagram

Our block diagram divides our system into two subsystems: the glove and the computer. As shown in Figure 2, the glove part of the system consists of the flex sensors, touch sensors, IMU, and an Arduino Nano. The Arduino Nano is connected to the computer with a USB cord. Through the USB connection, the computer will power the Arduino Nano which will then power the IMU, touch sensor circuits and flex sensors circuits. The Arduino Nano communicates with the IMU using I2C protocol which requires the SDA and SCL lines. The IMU outputs nine data points, three measurements in the x, y, and z dimensions for acceleration, rotation and magnetic field. The Arduino Nano's analog pins read the voltage fluctuations caused by the flex sensors in the five respective voltage divider circuits. The Arduino Nano's digital pins will read either a 1

or 0 if the touch sensors on the middle, ring and pinky fingers are engaged. The Arduino Nano, IMU, touch sensors and flex sensors are connected with a PCB in order to keep the circuitry compact. The flex sensors are not mounted on the PCB but are connected to some pinouts on the PCB for ease of maintenance and iteration. The Arduino Nano runs a script to continuously sample the voltage of the circuits made with the flex sensors, the circuits made with the touch sensors and the values outputted by the IMU. Then the Arduino Nano formats the seventeen values (five voltages from the flex sensor circuits, three from the touch sensors, and nine from the IMU) into seventeen by one vectors to send them serially to the computer.

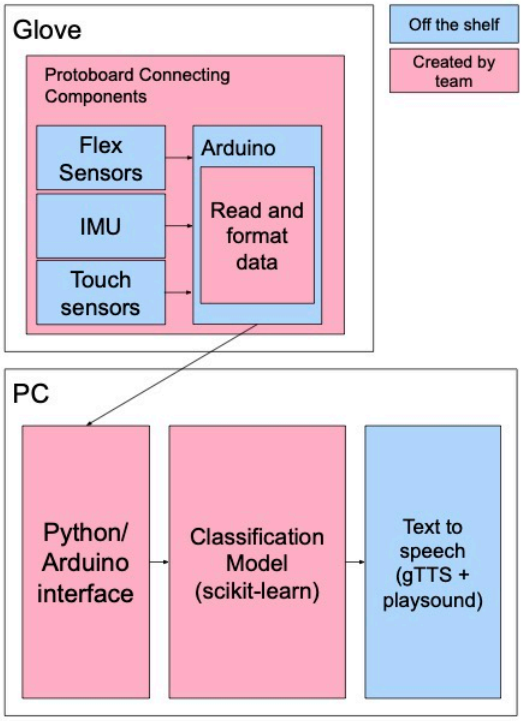


Figure 2: Overall system block diagram

The computer runs a Python script to read from the port the Arduino Nano is connected to and does any necessary parsing to manipulate the values in the seventeen by one vectors. This parsed information is then sent into a classification model which determines the ASL letter being formed by the glove wearer. The computer then speaks the letter by playing the audiofile of the corresponding letter.

The hardware in this project consists of the flex sensors, touch sensors, IMU, Arduino Nano and connective circuitry. The software in this project consists of the Arduino sketch that will read, format and send the data to the computer, the Python script which will receive the data from the Arduino and the classification model which will determine the shape of the glove wearer's hand and corresponding ASL letter.

4 DESIGN TRADE STUDIES

4.1 Sensors vs. Computer Vision

The largest implementation decision we had to make was whether we would use computer vision or a combination of several sensors to identify the gestures the user is making. Both have their benefits and flaws, but we ultimately went with the sensor approach.

Computer vision has the benefit of potentially providing more data that our machine learning model can train on since the machine learning model has all the pixels in a scene (and potentially several frames) to work with to classify the gesture. Additionally, computer vision would be able to capture more minute changes in position that sensors would likely not pick up. However, with computer vision, the accuracy of our system would heavily depend on environmental factors such as lighting. Using computer vision would also make the system less portable since it would require a camera setup.

Sensors are better than computer vision in several aspects. Unlike computer vision the sensor's performance is not affected by environmental lighting, allowing more consistent data collection, which would result in more accurate classification. Having sensors attached to a glove also makes our system more portable and easier to use since it would not require a camera setup and as a result, would also be a lighter system. On the other hand, sensors are slightly less sensitive to minute changes in gesture and will give similar measurements for different gestures. However, we decided that the trade-off between sensitivity to difference in poses and stability under different environmental settings was worth it.

4.2 Accuracy vs. Number of Gestures

When designing our system, we were faced with deciding how many gestures our glove would recognize. While the glove would ideally be able to identify all of ASL, it is not realistic for our system to do so. The biggest trade-off here is between the number of gestures our glove can identify and the accuracy of classification. Fewer gestures would result in a higher accuracy since the model would allow more range of motion. The machine learning model will output a class no matter what, based on the inputted data, so even if a user makes a gesture that deviates quite a bit from the "standard" pose for that gesture, it will likely still get classified correctly. Conversely, if we allow for a large amount of gestures, there would need to be minimal deviation from the "standard" pose that the glove is trained on in order for the system to classify the gesture correctly. This would both decrease classification accuracy and difficult for signers to use since everyone makes each gesture slightly differently due to difference in hand-sizes, flexibility of fingers, how they were taught, and such other

factors.

While higher classification accuracy is desired, we cannot reduce the number of classes to an amount that makes the glove impractical for users. For example, a glove that can only recognize a couple gestures would not be helpful in aiding ASL users to communicate with others since communication requires much more than a couple of gestures. Eventually, we settled on having our glove classify the 26 letters of the alphabet because this allows for full communication without compromising accuracy.

4.3 Comparison of Machine Learning Models

We considered 5 different machine learning models for our system: Support Vector Machines (SVM), Nerual Networks, Perceptron, K Nearest Neighbors (KNN), and Random Forest Classifiers. Of the 5, SVMs, neural networks, and perceptrons are classifiers that we have seen in past projects similar to ours use, and KNN and random forest are two other common classification algorithms in the machine learning world.

The perceptron algorithm trains the system to find a set of hyperplanes that separate the data into their respective classes. If our data is linearly separable, meaning that a set of hyperplanes exists to correctly classify all of our data, this algorithm would work well and be fast to use in our system since all the model has to do to classify a new data point is decide which side of the hyperplanes the new point lies on. However, due to variance in the data we will collect (since we are collecting from several people), the noise from the sensors, and the similarity in some of the gesture we are attempting to classify, our data is not likely to be linearly separable without data mapping. To determine whether our data is linearly separable, we can check if the perceptron algorithm has reached convergence by calculating its loss. The loss function for perceptron is called 0-1 loss, which means a loss of 1 is incurred when an incorrect prediction is made. Preliminary testing showed high loss and low accuracy, which signals that our data is not linearly separable and perceptron would not be an ideal algorithm to use.

Support vector machines (SVM) are similar to perceptrons in that they also look for a set of hyperplanes that linearly separate the data. However, SVMs improve upon this concept by using kernel methods to transform the data into a vector space that makes them linearly separable. SVMs also finds the hyperplanes that are maximally far away from each data point in each of the classes, meaning that new data points will be more likely to fall on the correct side of the hyperplanes for accurate classification. Due to these additional methods, we expect support vector machines to perform better than perceptrons. In terms of latency for this classification method, there is the added step of kernelling the data, but that transformation should not make the latency significantly greater than that of the perceptron algorithm.

Neural networks are also similar to perceptrons since neural networks are essentially several perceptrons working in sequence. Rather than the output of a perceptron being the output of the system, it gets fed into another perceptron, and this process continues for as many layers as is specified by the model designer. Because there are several layers in a neural network, the model will be able to extract information and transform the data for more accurate classification of the data. However, more layers also implies higher latency and more storage than both the perceptron algorithm and SVMs.

K Nearest Neighbors (KNN) is a model that takes a new data point and gives it the same classification as the majority class of the K number of nearest training points. This requires no training, just remembering the training data and their classifications. Assuming that our training data can generalize to new data, this algorithm should perform quite well. The downside to this algorithm is that the latency is high since it takes a lot of computation to figure out the K nearest data points and also requires a lot of space to store all the training data. If we were to move away from using a computer to do the classification computations, the amount of memory required to store this information would need to be considered.

Random Forest is an algorithm that utilizes multiple decision trees that all operate on a subset of the data inputted. Each of these many decision trees will output a class for the data point and the class that appears the most among these decision trees is outputted as the final class. Because the decision trees within this random forest are uncorrelated with each other and use subsets of the data, the algorithm is more resistant to outliers and are less likely to overfit to training data. The disadvantage to this method, similar to KNN, is that it requires more memory to store each of the individual decision trees as well as needs more computational power to run all of the decision trees. After testing on various datasets, We found that the random forest and the neural network had the best accuracies. We did consider KNN but it can be computationally expensive when the training dataset gets larger and may add delays to prediction times. The neural network was only able to reach such a high accuracy after cross-validation and its real-time performance was not as good as the random forest's, hence eventually we have chosen the random forest classifier.

5 SYSTEM DESCRIPTION

5.1 Subsystem A - Glove

The glove subsystem consists of five flex sensors, an IMU, five touch sensors, Arduino Nano and PCB.

5.1.1 Flex Sensors

When deciding what sensors to use to collect information on the shape of the hand, we considered placing IMUs on each of the joints on each of the fingers. If we decided

on this strategy, we would need small enough IMU modules in a way which would not constrain the movements of the glove wearer. Although IMU chips are very small, making connections to them would be incredibly difficult without mounting them on a PCB. If the chips were mounted on a PCB, we felt that it would make the glove more bulky and less flexible. Thus we decided to use flex sensors. They would be easier to place on the glove and less bulky. Initially we had some concerns that the data from the flex sensors would not be enough to determine hand shape, but research into similar projects showed that flex sensors can be sufficient.

The two main flex sensors on the market are built by BendLabs and Spectra Symbol. We decided to choose the SpectraSymbol flex sensors over BendLab's for various reasons. The BendLabs flex sensor is very expensive at $50 per sensor for a 1-D sensor and $129 per sensor for a 2-D sensor. On the other hand, the SpectraSymbol flex sensors were only $13 per sensor. We have a budget of $600 and we needed at least five of these sensors, so we could not even purchase the 2-D BendLabs sensors. Additionally, the BendLabs flex sensors required six connections total to read data while the SpectraSymbol would only require three connections. Lastly, the BendLabs sensors had little documentation; there was only one tutorial about how to interface with the Sparkfun Pro mini. The SpectraSymbol sensors had been used in projects similar to ours and proved to be sufficient and had a lot more associated documentation on how to interface with different microcontrollers. To summarize, we decided on the SpectraSymbol sensors because (1) they were cheaper and we could buy multiple of them in case any of the five we needed got damaged (2) purchasing the minimum five sensors and three extras would still leave us a lot money left in our budget (3) they required fewer connections and we wanted to limit the bulkiness of our glove and (4) they had a lot more associated documentation.

5.1.2 IMU

There are a few ASL letters in which the finger poses are similar, but the orientation of the fingers make them different which is why we needed a way to determine the orientation of the hand. We decided to use an IMU to determine the orientation of the hand.

When researching IMUs to use, we found we could choose between purchasing a 6-DoF or 9-Dof IMU. A 6-DoF IMU has an accelerometer and gyroscope while a 9-DoF IMU has an accelerometer, gyroscope and magnetometer. The 6-DoF IMUs were cheaper and there was even one that we found which had some built in gesture recognition. However, we chose to go with the 9-DoF IMU since we found one that was marginally more expensive. Our research also revealed that 9-DoFs were more accurate since the addition of the magnetometer offset drift in the gyroscope. Furthermore, since we're feeding all of this data into a machine learning model, more data (the extra three data points from the magnetometer in a 9-DoF IMU) would be more beneficial. We chose the cheapest 9-DoF IMU we could find which was the Adafruit TDK InvenSense ICM-20948 9-DoF IMU. We had the option to just purchase just an IMU chip, however, we felt we did not have the skills or time to learn the tools to place it on a PCB. The product that we purchased included a breakout board with a 1.8V voltage regulator as well as level shifting circuitry to allow interface with 5V microcontrollers such as Arduino and Raspberry Pi.

5.1.3 Microcontroller

After selecting our sensors, we needed to figure out how to collect, read and parse the data so that it could be fed into a machine learning model to classify the shape of the hand. At first we were not sure if we wanted a microcontroller to perform ML, or if we wanted a device to act as a gate reading from the sensors and feeding in the data. We decided we would use our laptops to run the models and use a small microcontroller to format and forward the data because we found that the microcontrollers recommended for running ML models were quite bulky and would be difficult to comfortably install on the glove.

There were various requirements for the device that would act as our gate and forward the sensor readings to the computer. The device needed to be capable of I2C or SPI to communicate with the IMU. It needed to have at least five analog pins to read from the five flex sensors. It also needed to be small. The Arduino Nano and RPi Zero were two devices which fit all these requirements, however, the Arduino Nano is smaller with a size of 45x18mm while the RPi Zero is 66x30.5mm. Additionally, Arduino is something that all of our team members were familiar with, so we chose to use the Arduino Nano.

5.2 Touch Sensors

After analyzing the performance of our glove with only flex sensors and IMU information, we found that similar shaped letters would not be reliably classified correctly. These letters included m, n, and t, which are fists with the thumb in between different fingers, and also u and v, which are two fingers extended touching together and apart respectively. Since a way to differentiate between these letters is to see whether certain fingers are in contact with each other, we decided to add contact sensors.

The contact sensors are made of conductive tape. This was a design choice made in the final weeks of the project, and this was a material immediately accessible to us, so we could start testing sooner. However, the tape is easily broken. After continued strain, the conductive tape often breaks. A more robust touch sensor might be made of conductive fabric.

5.2.1 Protoboard Design

In order to keep the circuitry compact, we at first planned to design and manufacture a PCB to make all the