American Sign Language to Text and Speech Glove

[EECS 149/249A Fall '17 Class Project]

Oscar Dorado
oscardorado1138@berkeley.edu

Aashyk Mohaiteen
aashyk@berkeley.edu

Juan Rodriguez
juanmrq95@berkeley.edu

Sunny Zhang
zzysunny@berkeley.edu

Department of Electrical Engineering and Computer Science
University of California, Berkeley
San Francisco State University

1. MOTIVATION

Deaf and hard-of-hearing communities often have a difficult time communicating with those who do not know American Sign Language (ASL). We aim to create a glove that can translate ASL letters into text and speech in real time. Although we are aware that this type of glove only allows unidirectional communication, we believe it is a good step towards the goal of having a bidirectional communication system.

2. PROJECT VISION

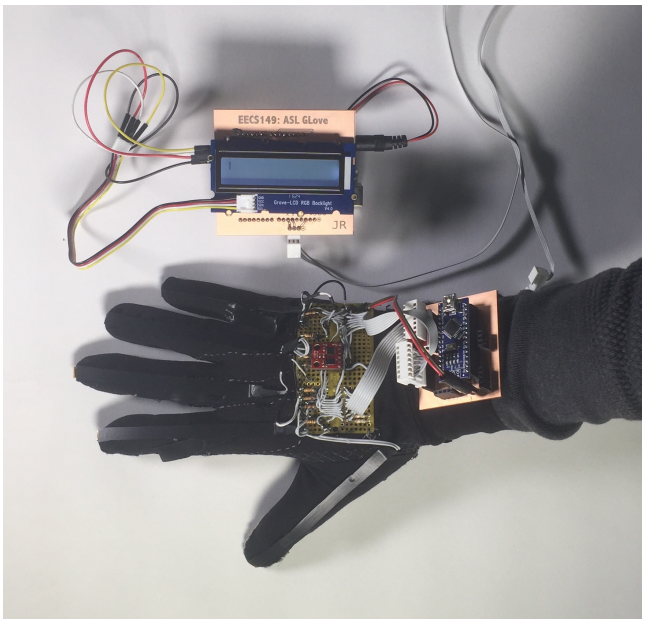

Figure 1: Complete Glove. Complete system with LCD and final glove version.

Our ASL glove will be a comfortable, easy to use glove that allows people to communicate through hand gestures. In order to reduce complexity for this project, we aim to only translate ASL letters into text and speech. This project models the physical inputs of various sensors embedded on a polyester glove, in order to identify letters of the ASL alphabet. The goal will be to accurately and reliably detect every letter in the ASL alphabet in real time and spell out any word through a LCD display and output speech though a small speaker placed on the user's chest.

3. MODELING SYSTEM DYNAMICS

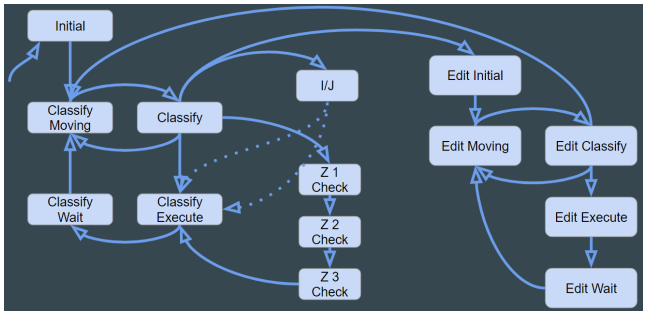

Figure 2: High-Level State Machine. High-level state machine without guards.

First, we looked at the kinds of gestures we are modeling. Since our letters are mostly static gestures (with the exception of J and Z), we decided to use a hierarchical state machine to implement the ASL glove. We created letter classification and edit classification state machines with similar functions, the only differences being the gestures they detect and the corresponding actions.

4. SYSTEM ARCHITECTURE

Since we are transforming hand gestures to quantitative data, we needed to figure out how to best represent each letter. We found that between a mixture of flex sensors to measure the bend on each finger, and digital touch sensors to determine if certain fingers were in contact, we would have enough inputs to accurately distinguish between letters.

The inputs to the ASL glove include data values from the flex sensors on each finger, the touch sensors on the tips

and between fingers, and values from a gyroscope and accelerometer. All inputs are processed by an Arduino Nano microcontroller and then fed to an Arduino Uno which listens for outputs from the Nano and updates the local LCD buffer.

The outputs to the ASL glove follow two distinct paths: one output is to an one-row LCD that will be on the shirt of the person wearing the ASL glove. The second output is to a text-to-speech board that will convert a stream of digital text from the microcontroller to natural sounding speech.

On top of the LCD output, our system also has editing functions such as moving the cursor left or right, deleting one letter, and clearing the LCD screen. To perform editing functions, we enter editing mode, and then perform the corresponding gesture that maps to the function we desire.

In figure 3, we can see the data flow between our various modules. We have a total of 14 sensor inputs: five flex sensors, seven touch sensors, and the accelerometer and gyroscope. All sensor outputs are processed and interpreted by the Arduino Nano, which then sends the results either directly to the text-to-speech board for voice output, or to the LCD through the Arduino Uno.

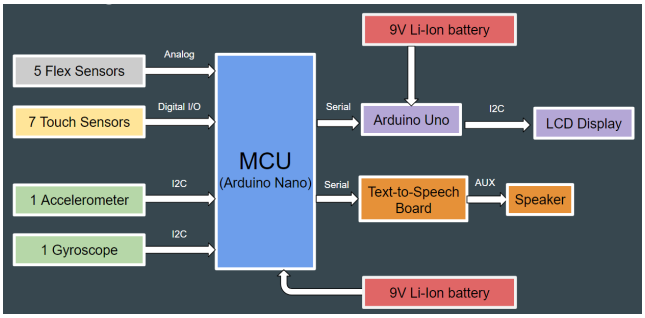

Figure 3: System Architecture. Block diagram that shows the inputs and outputs of our system and their interconnections.

5. HARDWARE COMPONENTS

This section examines all the hardware components we used to build the the glove. Table 1 below lists the main hardware components integrated into our glove.

Table 1: Hardware Components

	Components
Inputs	5 x Flex sensors 7 x Touch sensors SparkFun 6 DoF IMU
Outputs	1 x LCD display Speech
Microcontroller	1 x Arduino Nano 1 x Arduino Uno
Power	2 x 9V Li-Ion Battery
Other	Conductive fabric Conductive thread Glove 2 x PCBs 1 x Flexboard

5.1 Flex Sensors

The flex sensors are variable resistors that gain resistance as they bend. We attach long 4.5" flex sensors to the thumb, index, middle, and ring fingers, and attach a shorter 2.2" flex sensor to the pinky. Even though the thumb is not much longer than the pinky, we found that many of the letters require thumb movements, so we used a long sensor to give us a wider range of motion. We attached the flex sensor to a voltage divider circuit in order to convert the increasing resistance when bending the flex sensor to raw voltage. We calibrated the raw voltage values using the following affine model we learned in class:

$$f(x) = ax + b$$

$$x = \frac{f(x) - b}{a}$$

We found that for optimal resolution of voltage values, the fixed resistor in the voltage divider circuit should have a resistance of 39Kohm . The 2.2" flex sensor voltage divider circuit produced a linear voltage variation ranging between 2.2V and 3.3V. The 4.5" flex sensor voltage divider circuit produced a linear voltage variation ranging between 1.8V and 2.8V.

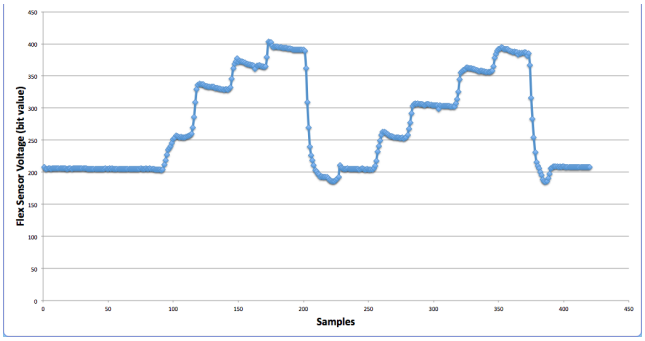

Figure 4: Flex Sensor Calibration Curve. Our flex sensors gave us clear steps that allowed us to easily quantify the amount of bend in each finger.

5.2 Touch Sensors

We created touch sensors by sewing conductive fabric to the fingertips of the thumb, and then knobs of conductive thread at the tips of the index and middle finger. We also sewed knobs of conductive thread in between all fingers except the thumb and index fingers to help us detect when certain fingers were touching. In our initial prototype, we used conductive fabric for all touch sensors and used the thread only to connect the touch sensors to the breadboards, but we found this to be too clunky. Sewing patches of conductive fabric in between fingers and at the finger tips was too prone to shorts, so we switched to using small knobs of conductive fabric to create contact points for the large patch of conductive fabric at the thumb. We powered the thumb patch as well as a knob of fabric on the side of the index finger (to detect if the index and middle fingers were touching) to 3.3V, and kept all other contact points at ground. We then directly wove the threads from each touch sensor into the glove to the flexboard, where we completed the circuit.

Figure 5: Thumb Touch Sensor. We can see clearly the conductive fabric sewn into the thumb, as well as the conductive thread that leads down the hand towards the flexboard.

5.3 Flexible Breadboard (Flexboard)

In our initial prototype, we used a breadboard as a bridge between the sensors and our microcontroller. For our final product, we wanted to replace the breadboard with something more flexible for better form factor. We used a flexboard and soldered the flex sensors' circuits and IMU directly onto it, and ran the conductive thread directly to the flexboard. Unfortunately, we found that not only were soldered connections on the flexboard prone to break, the flexboard connections themselves (within the same row) often broke. As a result, we were forced to constantly solder over existing connections to fix breaks.

5.4 Inertial Measurement Unit (IMU)

We attached an IMU to the flexboard on the back of the glove to detect hand movements and orientation. Letters such as G/Q, H/U, and K/P have the same sign, but are in different orientations, and letters such as J and Z require movement to sign. By attaching the IMU to the back of the hand, we are able to accurately distinguish between these letters. We calibrated the accelerometer using the same affine model as before, and we also added the following low pass filter to reduce the amount of noise:

$$y[n] = (1 - \alpha)y[n - 1] + \alpha x[n]$$

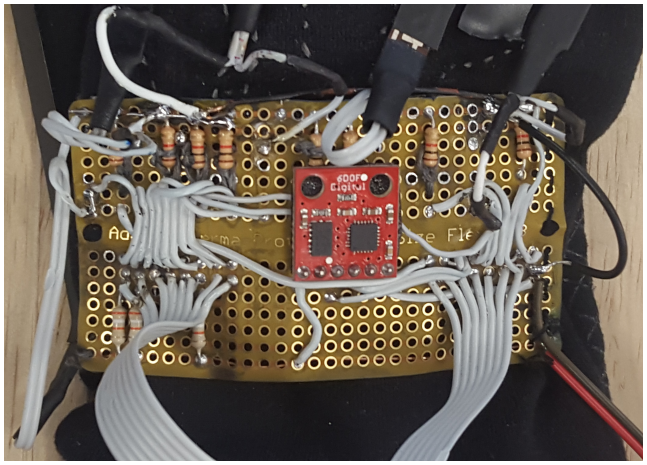

Figure 6: Flexboard. Our flexboard contains all the flex sensor's circuits as well as the IMU and is attached directly to the glove.

Through trial and error, we found that an alpha value of 0.5 was the perfect balance between noise reduction and IMU response. In our current implementation, we use the accelerometer values in g's to determine orientation and movement.

5.5 LCD and Speech

Our design outputs interpreted letters from the Arduino Nano to the Arduino Uno, which uses an I2C bus to communicate with the LCD. The Arduino Nano sends the results to a serial-to-speech board that will voice the signed word. The LCD primarily acts as a debugging screen, so that the user can verify that the output matches what they intended to sign. In the case that there is a discrepancy, the user can enter editing state and then backspace or clear the current word and sign the incorrect letter again. Upon completion of signing a word, the output is sent to the text-to-speech system and the word is voiced out loud.

5.6 Microcontrollers

We are currently using an Arduino Nano board to calibrate sensors and classify data. The Arduino Nano has eight analog I/O pins, and 22 digital I/O pins, more than enough for the five analog and seven digital pins that we need. We also use an Arduino Uno for the LCD module. The Uno listens for output from the Nano and then writes the received character to the LCD string buffer. Both Arduinos are powered by their own 9V Li-ion batteries, although the Uno's also powers the LCD, and the Nano's powers the rest of the glove system.

5.7 PCBs

Our design incorporates two PCBs to help slim down the form factor. The first PCB houses the Arduino Nano and interfaces between the sensors, power supply, voice, and the UART connection to the LCD module. The second PCB houses the Arduino Uno and interfaces to its own power supply, and also the UART communication back to the Arduino Nano. We encountered some issues soldering on the PCB and creating reliable connections for the various ribbon cables we had. We found that using male headers on

the PCBs often resulted in the individual pins breaking the PCB's copper layer when we removed ribbon cables, which we remedied by switching to female headers for the PCB and male headers for the cables. This put less pressure on the individual PCB connectors, and thus, prevented tears in the PCB's copper face.

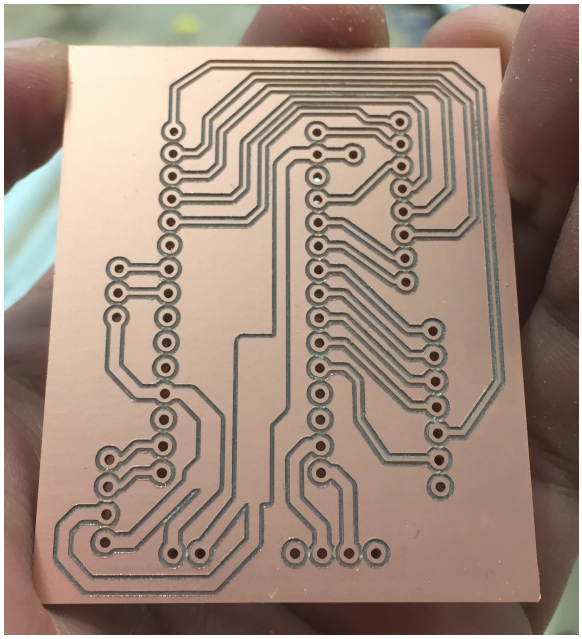

Figure 7: PCB. PCB used to house the Arduino Nano.

5.8 Power

Initially, we attempted to use a 1 cell lipo battery to power our system. However, we found that though the lipo was outputting 3.8V, the glove components were only receiving 2.8V after the Nano's voltage regulator. As a result, we switched to using standard 9V li-ion batteries to power our entire system. The maximum resistance of the flex sensor is twice its flat resistance of 10kOhms. From this, we calculate each flex sensor's power dissipation to be 16mA, which multiplied by five sensors gives us a power draw of 80mA. Our Arduino Nano consumes 30mA at peak, and our IMU consumes 7mA so from our 9V 500mAh li-ion battery, we can power our system for about four hours. Our LCD system should last at least 1.5 hours, since the Arduino Uno draws 45mA at peak, and the LCD draws 300mA at peak.

5.9 Glove

When looking for a glove, we searched for a thin, flexible, non-conductive glove that could act as the base for our sensors. Our goal was to create a glove with a slim form factor, so choosing a sufficiently thin glove was key.

6. SOFTWARE COMPONENTS

6.1 Classifier

In our design, we hard-coded values to classify various letters. We took the raw voltage values that were output by the flex sensors and split them into twenty increasingly

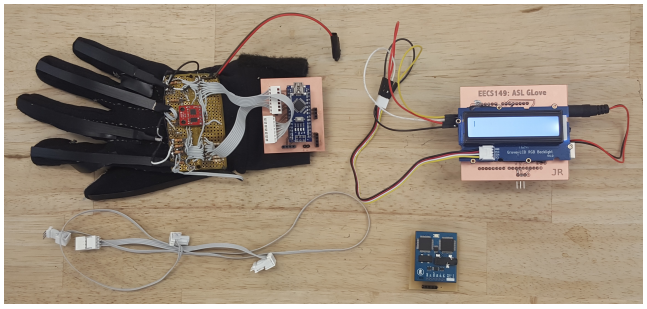

Figure 8: Complete System. Glove with LCD and speech modules.

bent buckets. This gave us enough resolution to distinguish between all the letters we needed. By pairing these hard-coded flex sensor inputs with the digital inputs from our touch sensors, we were able to accurately distinguish letters. We collected data from project members and then selected upper and lower thresholds for each letter and edit gesture. Unfortunately, this approach only works well for the user it is finely tuned for. For better results, we can use machine learning with the decision tree to train the glove to recognize letters with higher accuracy.

For the purposes of this project, we decided to manually find these thresholds rather than using machine learning due to the unreliable nature of the glove hardware. Since the flexboard that connects our microcontroller to our sensors often broke after repeated use, it would be nearly impossible to gather enough training data to train a decision tree. In addition, to create a diverse dataset, we would need to collect data from more than just team members, and given the time constraints, this was not plausible. In the future, given more time, we could pursue a machine learning approach, but for the purposes of this class, we did not deem it an effective use of our time.

7. RELIABLE REAL-TIME BEHAVIOUR

Because our glove is worn by a human being, we concluded that the system does not need a high polling rate to get accurate outputs from our sensors. In our current iteration, we have a 250ms hold time that ensures that we only classify a letter once we have reached a steady state, and we will not classify in between states. We tested various values for the hold time before settling on 250ms. Although the delay function in Arduino does not provide an accurate delay, our system is not a hard real-time system, and thus, we do not require an accurate delay.

8. FINITE STATE MACHINE

Figure 2 above shows our complete high-level FSM for our glove system. We have two high level states: classifying state, and editing state, shown in figures 9 and 10 respectively. Under the classifying state, there are four substates: classify_moving, classify, classify_wait, and classify_execute. We are in the classify_moving state when the flex sensor value changes, and we transition into classify after a 250ms hold-time and we stop moving. The hand is determined to be in motion only when the flex sensor values are changing; a change in IMU values is not considered as moving. The code enters a decision tree and picks which letter the cur-

rent gesture corresponds to. Classify_execute then outputs the letter from the decision tree to the Arduino Uno, where it is sent to the LCD, and we are left in classify_wait, where we remain until the hand moves away from the current gesture. We enter the editing state from the classify state when the classifier detects the editing state gesture. Within the editing state, we have another four substates: edit_moving, edit_classifying, edit_execute, and edit_wait. The functionality of these states correspond to the classify states, and classify_execute is equivalent to edit_execute. Instead of outputting a letter to the Arduino Uno, we perform one of our various editing functions.

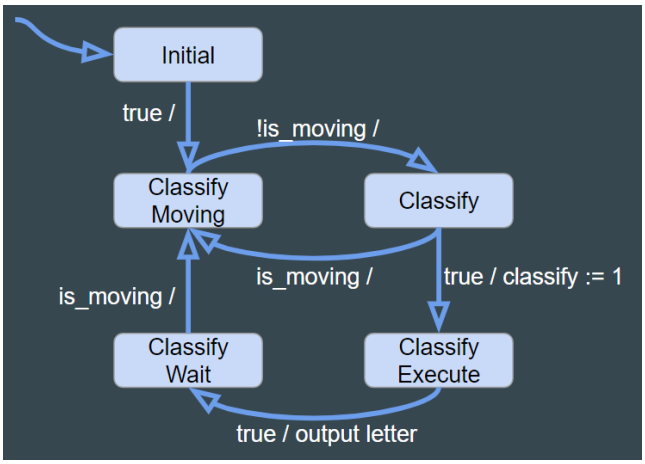

Figure 9: Classify FSM. Classify subsystem with complete transitions.

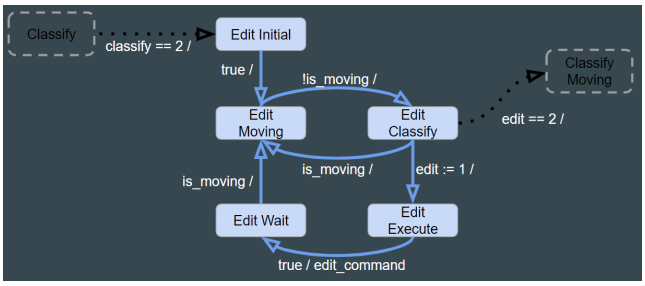

Figure 10: Edit FSM. Edit subsystem with complete transitions.

8.1 I and J FSM

Since I and J have the same gesture, only differing in that J requires additional motion, we created a special state machine to distinguish between these two letters. As we can see in figure 11, we have an I/J state that checks for accelerometer readings. If the accelerometer output matches J's expected accelerometer readings, we output J. If the flex sensor readings start to change without having matched J's expected accelerometer readings, the state machine selects I and moves on.

8.2 Z FSM

Because Z also requires movement, we need a special state machine. Figure 12 shows the special subsystem we created

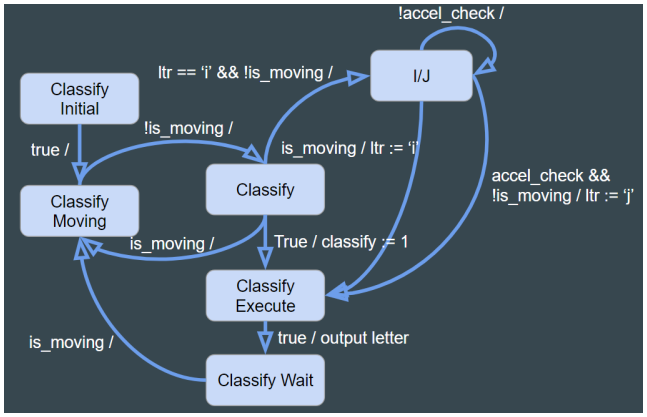

Figure 11: I/J FSM. I/J subsystem with complete transitions.

to solve this problem. The Z subsystem has three checks since the Z gesture is a zig-zag motion. We first check for horizontal motion to the right by checking to see if the accelerometer's y-value is decreasing from its current value. We then check for a downward-diagonal motion from right to left by checking if the accelerometer's y and z-values increase. The third check is a repeat of the first, and if all three checks are successful, we choose Z. While in any of these states, our system continues to check for horizontal and downward-diagonal motions as explained before, until the flex sensors change and we are no longer in the Z states.

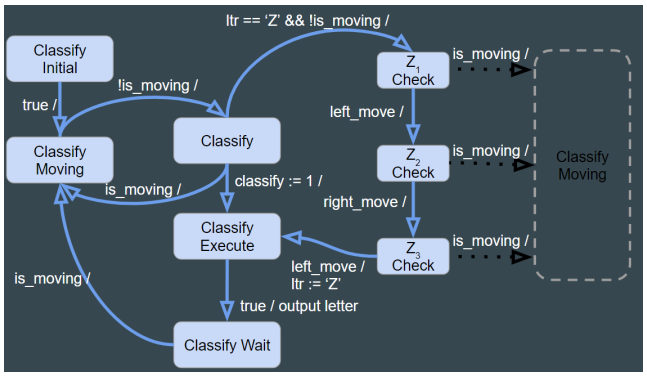

Figure 12: Z FSM. Z subsystem with complete transitions.

9. REFERENCES

1. O'Connor TF, Fach ME, Miller R, Root SE, Mercier PP, Lipomi DJ (2017) The Language of Glove: Wireless gesture decoder with low-power and stretchable hybrid electronics. PLoS ONE12(7): e0179766. https://doi.org/10.1371 / journal.pone.0179766

2. Adam. bildr Stable Orientation Digital IMU 6DOF Arduino. Retrieved November 21, 2017 from http:// bildr.org/2012/03/stable-orientation-digital-imu-6dof-arduino/

3. Steven. How would I get a full range voltage reading from a pressure sensor? Retrieved November 21, 2017 from https://electronics.stackexchange.com/questions/158/how - would-i-get-a-full-range-voltage-reading-from-a-pressure- sensor/3336233362

4. ADXL345 Datasheet. http://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf
5. ITG-3200 Datasheet. https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf
6. Euler Angles Formula. https://en.wikipedia.org/wiki/Euler_angles
7. Atmel SAM D21E Datasheet. https://cdn-shop.adafruit.com/product-files/2772/atmel-42181-sam-d21_datasheet.pdf

10. ACKNOWLEDGEMENTS

We would like to thank Professor Lee and Dutta for an amazing class, and special thanks to Meghan for her guidance throughout the semester on our project.