Spring 2017

ECPE 196 - Senior Project II

Course Instructor: Dr. Rahim Khoie

American Sign Language Interpreter

Final Report

Sponsor: Dr. Vivek Pallipuram

Team #2

Taryn Aranador, Ryan Chen, Bryan Ching, Liane Young

Date of Project Completion: 05/02/2017

Date Hard Copy of the Report was Turned in: 05/02/2017

Date Electronic Copy of Report was Turned in: 05/02/2017

Total Number of Hours Worked:

Taryn Aranador: 190 hours

Ryan Chen: 153 hours

Bryan Ching: 183 hours

Liane Young: 220 hours

Table of Contents

List of Figures

List of Tables

0.0 Abstract

Communication can be difficult between a mute person and an individual with limited to no knowledge of American Sign Language, one of the primary forms of communication for someone with the inability to communicate vocally. The design for the American Sign Language Interpreter is an effort to bridge this communication gap and enable greater accessibility in conversation between these two populations. This device is a cost-effective, portable, durable, user-friendly solution to this problem, and senses the user's input of American Sign Language alphabet letter gestures to produce audible interpretations of these hand signs. The final deliverable is a fully-functional device utilizing flexible and motion sensors that has wireless capabilities and the ability to recognize the full alphabet with an accuracy of 89.45%. Future improvements will focus on enabling the device to be trainable for multiple users.

Chapter 1: Timetable and Distribution of Tasks

1.1 Timetable

Week	Task	Team Member
1	Progress Meeting to check status	All
1	Start building	All
1	Work on signal processing of sensor signals	All
1	Find and order Bluetooth module	Bryan, Ryan
1	Design PCB	Liane, Taryn
1	Choose glove (make sure its sewn loose enough for finger)	Ryan, Taryn
1	Edit Final Report from last semester	All
1	Download compiler	All
2	Work on signal processing of sensor signals	All
2	Work on Bluetooth Protocol	Bryan, Ryan
2	Finish designing PCB	Liane, Taryn
2	Calibrate MPU-6050	Liane, Ryan
3	Characterize signals from sensor to determine signal values for each letter - Test on multiple people	All
3	Work on Bluetooth Protocol	Bryan, Ryan
3	Work on Presentation	All
4	Presentation (Tuesday)	All
4	Work on Bluetooth Protocol	Bryan, Ryan
4	Work on signal characterization	Liane, Taryn
4	Implement entire sensor system on the board	Liane, Ryan, Taryn
4	Edit PCB design	Liane, Taryn
5	Test algorithm to match signal to correct letter	Liane, Bryan
5	Test algorithm using signals from glove	Taryn, Ryan

5	Work on Bluetooth Protocol	Bryan, Ryan
5	Edit PCB design	Liane, Taryn
6	Work on building glove module	All
6	Work on Bluetooth Protocol	Bryan, Ryan
6	Design wrist strap	Liane, Taryn
6	Finish gathering sensor data	All
7	Work on Bluetooth Protocol	Bryan, Ryan
7	Send in final PCB design for printing	Liane, Taryn
7	Program Algorithm in C	All
7	Work on Presentation	All
8	Work on Bluetooth Protocol	Bryan, Ryan
8	Solder components onto PCB	Liane, Taryn
8	Fix PCB design	Liane, Taryn
8	Collect more sensor data	All
8	Characterize motion sensor data in state machine	Liane
8	Test sign-to-letter algorithm accuracy	Liane, Taryn, Bryan
8	Place final parts order	All
8	Presentation (Tuesday)	All
9	Build glove module	All
9	Create wrist strap	Liane, Taryn
9	Work on Bluetooth Protocol	Bryan, Ryan
9	Modify sign-to-letter algorithm, verify	All
9	Create speaker-amplifier module	All
10	Finalize Bluetooth component	Bryan, Ryan
10	Work on sign-to-letter algorithm	Liane, Taryn
10	Work on speaker amplifier	Bryan, Ryan
10	Design speaker module PCB	Liane, Taryn

11	Work on sign-to-letter algorithm accuracy	Liane, Taryn
11	Test glove module	Taryn, Liane
11	Test Bluetooth connectivity between glove and speaker	Bryan, Ryan
12	Clean up device	All
12	Test system accuracy	All
12	Work on final report/presentation	All
13	Full system testing and debugging	All
13	Algorithm accuracy improvement	Taryn, Liane
13	Work on final report/presentation	All
14	Full system testing and debugging	All
14	Work on final report/presentation	All
14	Create Poster for Project Day	All

1.2 Distribution of Tasks

Our project will be split into the following sub-sections, with the following people taking lead for the component and the rest of the team supporting:

- Liane: Sensors Lead, Characterization co-Lead, Programming Assist
- Taryn: Build Lead, Characterization co-Lead, Sensor Assist
- Bryan: Bluetooth Lead, Programming Assist, Sensor Assist, Build Assist
- Ryan: Parts Lead, Programming Lead, Bluetooth Assist, Sensor Assist, Build Assist

Chapter 2

2.1 Finalized Description of the Project

The American Sign Language Interpreter is an electronic glove that will capture the finger and hand movements of its user. The glove will be equipped with the appropriate sensors that will detect and recognize different hand gestures. Based on these processed hand signals, a corresponding letter will be selected and transmitted to a speaker module attached to a shirt. The speaker module will contain all the sound files in the alphabet along with a speaker to output the sound. The glove module will be responsible for communicating and transmitting the correct alphabet letter, while the speaker module will need to choose the correct sound file from memory and output the sound to the speaker. Communication between the glove and speaker module will be wireless. By the completion of the Senior Project course, only the English alphabet will be detected by our interpreter.

2.2 Finalized Project Requirements and Specifications

There will be two components for the American Sign Language Interpreter project: the glove and the speaker modules. The gloves should detect letters A through Z in sign language with an accuracy of 90%, and should have the ability to recognize sign language words if time permits. The speaker module will select the corresponding sound file and output it through a speaker which will be audible and clear. Communication between the glove and the speaker module will be wireless, and the two components should be lightweight, ergonomic, and portable. The weight of the glove should not exceed 0.125 kilograms which includes one microcontroller, Bluetooth device, power sources, and sensors. The American Sign Language glove should require minimal power and cost less than $200 to produce. The time duration of operation for the device should be at least 50 minutes.

Chapter 3: Finalized System Design

3.1 Finalized Behavioral Description

The Sign Language Interpreter will be used to convert hand signs in the American Sign Language alphabet into audible sounds of the letter represented. The user wears the glove and performs the hand sign, which is detected using flexible sensors attached to the glove. The signals from these sensors are processed and matched to the correct letter in the alphabet based on their values by the microcontroller. The correct letter is then sent to the Bluetooth module on the glove to transmit wirelessly to the Bluetooth module on the shirt. The microcontroller on the speaker chooses the correct sound file from memory based on letter signed by the user and outputs the sound through the speaker, which is wired to the microcontroller. This process then continues with the next sign. A neutral hand gesture will be created to signal that the user has completed his or her hand sign.

3.2 Finalized Top-Down Design

This is a top-down design of our Sign Language Interpreter depicting all of its different modules and how each module will interact with one another. Our project can be divided into four main sections, which includes Gesture Recognition, Sound, Control and Communications, and Power. Each one of these four sections can be divided into the aforementioned tasks from Chapter 1.2 with the different team leads.

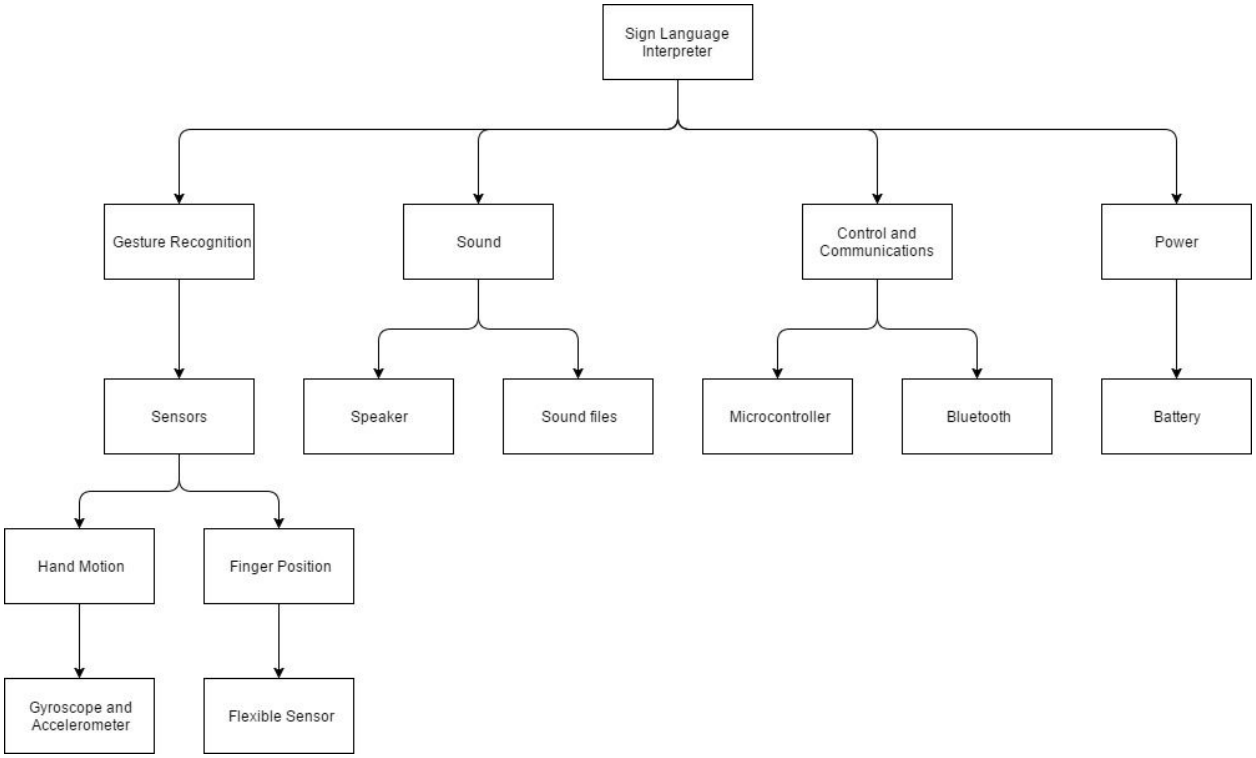


Figure 1. Top Down Design of Sign Language Interpreter

3.3 Finalized Functional Design

The next few diagrams depict our project's functional design. Level 0 in Figure 2 is our project's most fundamentally basic design. The inputs going into our system are the hand gestures and finger movements that correspond to a particular hand sign. The outputs from our Sign Language Interpreter are the corresponding alphabet sound files playing from the speaker.

Figure 2. Level 0 Block Diagram

Our Level 1 Block Diagram in Figure 3 depicts the five major components that sum up our entire project. Each one of these components and their overall functionalities are required for the success of our American Sign Language Interpreter. Because of the importance and criticality of these components, we have team leads overseeing the progress of each one of them.

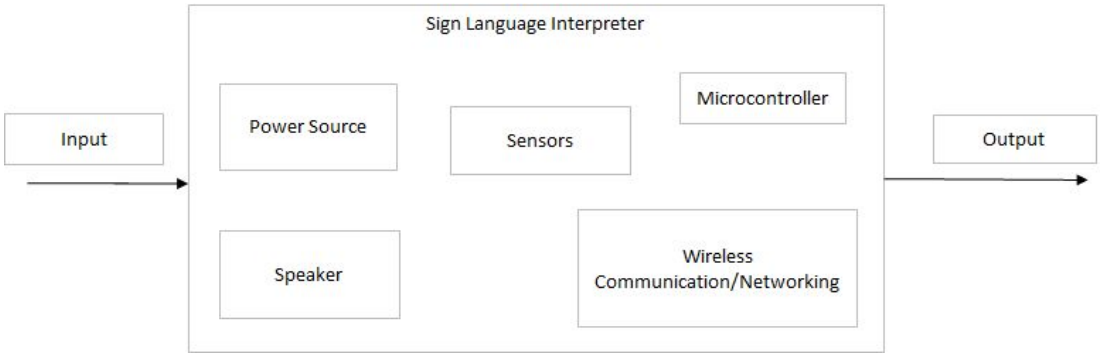


Figure 3. Level 1 Block Diagram

Our Level 2 Block Diagram in Figure 4 depicts our American Sign Language Interpreter broken up into two modules: the glove and the speaker modules. This diagram also shows how each individual component connects together to form our final project design. The inputs of our system are the hand gestures and finger movements, while our outputs are the audio playbacks of the corresponding sound files for the letters of the alphabet. The inputs feed into the flexible and motion sensors, which then sends the data to the microcontroller. The microcontroller performs signal processing on the raw sensor data and determines the correct letter. The letter is then transmitted via Bluetooth on the glove to the Bluetooth on the shirt. The microcontroller on the speaker chooses the correct sound file from memory and outputs the audio through a wired speaker.

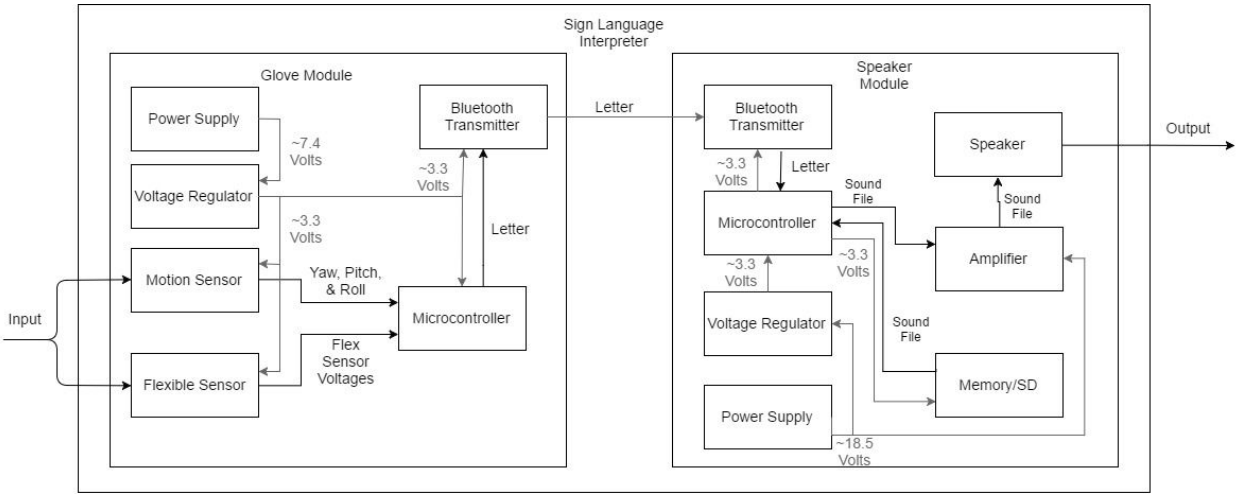


Figure 4. Level 2 Block Diagram

Chapter 4: Research on Different Hardware/Software Designs

4.1 Hardware

In this section, the various hardware components incorporated in the design for the Sign Language Interpreter are analyzed and reported. The research for each component is examined thoroughly.

4.1.1 Sensors

During the last weeks of the semester, most of the sensor research and development was focused on prototyping and testing each of the sensors. Each sensor was tested and characterized to determine the expected output signal.

The first sensor tested was the EMG Myoware Sensor. The sensor was tested using one set of electrodes to determine how clean the signal would look. Each member of the group was tested, with multiple readings being taken at different locations of the arm in order to determine optimal placement of the sensor. The sensor was tested using an Arduino Uno board because the microcontroller finalized for the design had not come in at the time the tests were performed. Output voltage values were taken using a multimeter with fingers both in the straight and bent position.

One observation while performing tests on the EMG sensor was the difficulty of finding a balance between maintaining consistency from user to user, and finding the best output for each user. Because each person tested had different arm lengths, consistency in similar placement was attempted but not necessarily the same for each person tested. This inconsistency indicates a need to customize the design to the user. A challenge going forward with this sensor would be determining a way to standardize the placement of forearm electrodes. Additionally, the signal resulting from the EMG sensor was noisier, and the voltage difference between the straight and bent finger positions was small.

The flexible resistor sensor was also tested to characterize the output signal of the sensor. This sensor was tested for each finger individually, with the finger in four different positions: straight, slightly bent, half bent, and full bent. The resulting output voltages show distinct differences in voltages that can also be summarized into voltage threshold values.

	Model	Sensor Type	Interface	Power	Cost
	MyoWare Muscle Sensor	EMG Sensor	Analog (AC Voltage)	3.3V or 5V, 9mA	$37.99
	Spectra Symbol Flex Sensor	Flexible Resistor	Analog	3.3V	$7.95/sensor

Table 1. Finger Position Sensor Options

The preliminary research found that both an accelerometer and gyroscope would be useful to read hand movement and position. Accelerometers measure how fast something is speeding up or slowing down, and can determine position in relation to Earth's gravity. The hand sign for 'Z' relies on the motion of the hand in a zig-zag formation, while hand signs for 'K' and 'P' use similar hand positions with the the fingers oriented in different directions. Gyroscopes measure angular velocity, which can help sense the hand sign for 'J', which is a sweeping, rotating motion of the pinky and fist. Due to the variety of hand movements and positions, both a gyroscope and accelerometer should be used to read hand motion and position.

The next phase of research on motion sensors tried to identify the best sensor for this project. Though initial research looked at both accelerometer chips and gyroscope chips, it was quickly determined that the most cost-effective and space-conservative solution was to find a single chip that incorporated both. Having the two types of motion sensor separate required either a printed circuit board if the chips were by themselves, separate breakout boards, or a larger breakout board to accommodate both chips.

The most common sensor on the market that incorporates both a gyroscope and an accelerometer is the MPU-6050. The MPU-6050 has a 3-axis gyroscope and accelerometer. Utilizing either SPI or I2C interfaces, it runs at a voltage of 2.375V-3.46V. The gyroscope has a programmable full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps). The accelerometer has a full-scale range of ±2g, ±4g, ±8g, and ±16g. The solitary MPU-6050 chip was priced at

$5.45. The other motion sensor chip investigated was the MPU-9250. This chip had similar specs to the MPU-6050, with one main difference. In addition to the gyroscope and accelerometer, the MPU-9250 has a magnetometer, which measures magnetic fields. The MPU-9250 was also priced slightly higher at $6.62. For the stand alone sensor chips, we considered buying the chip on its own and creating a shield for it to sit on with its connections and any additional circuitry. Since the chips have surface-mount pins, a PCB would need to be created for the sensor.

Most motion sensors with breakout boards discovered that integrated both an accelerometer and gyroscope used the MPU-6050 chip. A selection of the different motion sensors researched are compared in Table 3 below. One such sensor was the Me 3-Axis Accelerometer and Gyro Sensor. Priced at $19.99, this sensor uses either I2C or RJ25 interfaces. It is powered using a 5V source, and has the typical range of the MPU-6050 sensor.

A MPU-6050 sensor with a breakout board was also available on SparkFun. This sensor uses an I2C interface to connect with the microcontroller. It is powered using a 2.3-3.4V source, and is priced at $39.95.

The final MPU-6050 sensor with board researched was the GY-521. Priced at $4.28, it utilizes an I2C interface. When using 3.3V to the Vcc, the resulting voltage for the sensor can be too low for a good working I2C bus, so it is recommended to apply 5V to the Vcc pin.

Additionally, the GY-9250 sensor board was also investigated. This sensor utilizes the MPU-9250 chip, thus incorporating a magnetometer in conjunction with the accelerometer and gyroscope. It utilizes I2C or SPI interfaces, and has a price point of around $8.00. It utilizes a 3-5V voltage source.

	Model	Sensor Type	Interface	Power	Cost
	Me 3-Axis Accelerometer and Gyroscope	Gyroscope, Accelerometer	I2C, RJ25	2.3V - 3.4V	$19.99
	Sparkfun MPU-6050 w/ Breakout Board	Gyroscope, Accelerometer	I2C	5V	$39.95
	GY-521 MPU-6050	Gyroscope, Accelerometer	I2C	3.3V - 5V	$4.28
	GY-9250 MPU-9250	Gyroscope, Accelerometer, Magnetometer	I2C or SPI	3V - 5V	$7.94

Table 2. Hand Movement Sensor Options

4.1.2 Microcontroller

The microcontroller in this project is responsible for the computations routing of the system. After the sensor signals are passed to the microcontroller, the controller processes the signal and eliminates noise. Using an algorithm, the processor matches the corresponding sound files to the sensor signals and send them wirelessly via Bluetooth communication to a speaker.

Originally, the BlueDuino was the microcontroller of choice for the project as it had built in Bluetooth module. However, after further discourse with Dr. Hughes, the team decided that the BlueDuino did not have the processing power to perform signal processing, sign matching, and Bluetooth protocol simultaneously. The BlueDuino had 32 kB of flash memory and 2.5 kB of SRAM. Upon further research, we discovered that in order to perform signal processing efficiently, the microcontroller would need to be able to process a minimum of 16 bits due to the need of having long int numbers available for calculations. It would, however, be optimal to have a microcontroller that could process 32 bits. As a result of the aforementioned reasons, we reconsidered microcontrollers that we had previously researched that fit our new criteria better than the BlueDuino.

The three microcontrollers that most closely matched the new set of specifications were the MK20DX128VLH5, MK20DX256VLH7, and the STM32 family. These microcontrollers are shown below in Table 3. All of these microcontrollers are readily available in small development board form, which allows faster development as there is no need to design, order, and wait for development of a PCB. Though the STM32 family microcontroller had all of the connections necessary as well as having better processing power than the BlueDuino with up to 1 MB of flash memory and 4 kB of SRAM, had connectivity issues with the chosen Bluetooth, the Texas Instruments CC2540, and not considered further. The MK20DX128VLH5 microcontroller is manufactured on the Teensy 3.0 development board. It has 16 kB of SRAM and 128 kB of flash memory. This controller is a 16 bit processor which may optimize the performance of the digital signal processing that will take place. The newer version of this microcontroller comes in the form of the MK20DX256VLH7 microcontroller, and can be purchased on the Teensy 3.2 development board. This microcontroller has more flash memory and more SRAM than its predecessor at 256 kB and 64 kB respectively. Its clocking speed is also significantly faster than its predecessor at 72 MHz compared to 48 MHz. The MK20DX256VLH7 is also a 32 bit processor.

	Microcontroller	Development Board	Clock Speed	Processor	Flash Memory	SRAM	Price
	ATmega32U4	BlueDuino	16 MHz	8 bit	32 kB	2.5 kB	$14.00
	STM32F3	NUCLEO-F303K8	26 MHz	32 bit	1 MB	4 kB	$10.99
	MK20DX128VLH4	Teensy 3.0	48 MHz	16 bit	128 kB	16 kB	$17.95

	MK20DX256 VLH7	Teensy 3.2	72 MHz	32 bit	256 kB	64 kB	$19.80

Table 3. Microcontroller Options

4.1.3 Wireless Communication

The team researched multiple Bluetooth modules that would be viable possibilities for our project design. There are many aspects that we compared between the Bluetooth devices. These aspects include Bluetooth protocols, data rate, serial interfaces, supply voltage, receive and transmit current, output power, dimensions, and price. Initially, there were six Bluetooth modules that were heavily researched and considered. Throughout the project design, the decision on which Bluetooth module will be used has been altered on multiple occasions. Ultimately, the overall purpose of the wireless communication had changed from the original specifications. With this change in design, the final Bluetooth module for the project is the HM-11 module.

The next six Bluetooth devices under discussion are the modules that were under consideration in the initial research stages of the project. The first module is the SaBLE-x, which is a Bluetooth Smart device developed by Laird Technologies. Because it is a Bluetooth Smart device, it delivers data over a greater range than previous generations and operates with much less power consumption. Its supply voltage ranges from 1.8V to 3.8V. Its receive current ranges from 7mA to 11.8mA while its transmit current ranges from 7.9mA to 13.6mA. The SaBLE-x module's output power is 5.3 dBm. It has UART as its only type of serial interface for connecting with development boards. Lastly, the SaBLE-x is priced at $16.52, which is pricier than the other Bluetooth modules that were researched.

The second module is the BT900, which is another Bluetooth Low Energy Smart device developed by Laird Technologies. This device uses Laird Technologies own smartBasic programming language for system design. The BT900 module's supply voltage ranges from 1.8V to 3.6V. Its receive and transmit current is 85mA. The module has an output power of 8dBm, which is the highest output power out of all our researched Bluetooth modules. The BT900 module supports the I2C, SPI, and UART serial interfaces. This particular Bluetooth

module has an incredibly high data rate of 3 megabits per second, which probably contributes to its high cost of $19, making it the most costly Bluetooth device in our research.

The last Bluetooth module that we researched is the BL652, which is Laird Technologies' newest module that uses the most uptodate Bluetooth v4.2 protocol. This module also uses the smartBasic language for programming. Its voltage, current, and power is among the lowest out of all the modules and system on a chip devices. The supply voltage ranges from 1.7V to 3.6V. The receive and transmit currents are about 5.3mA while the output power is 4dBm. The BL652 module supports I2C, SPI, and UART serial interfaces. Each one of these modules cost $7.45.

The first Bluetooth module with a breakout board that was researched is the SparkFun Bluetooth Mate Silver. This device uses a RN-42 Class 2 Bluetooth v2.0 radio. The RN-42 module has a small form factor and operating range of 10 meters. The entire chip's supply voltage ranges from 3.3V to 6V. The transmit current ranges anywhere from 0.026mA to 40mA depending on what types of operations the RN-42 module is performing. RN-42 modules are capable of reaching data rates of 3 megabits and uses the UART serial interface. The overall size of the board is still fairly small with dimensions of 1.75 x 0.65 inches. The total price of this system on a chip is $24.95.

The second breakout board Bluetooth module is the SparkFun BlueSMiRF Silver, which is an alternative to the Mate Silver. According to SparkFun, the BlueSMiRF Silver and Mate Silver are nearly identical in all aspects except for the order of the pinouts on the boards. The Mate Silver is designed to attach directly with the serial header of certain development boards, such as an Arduino Pro. The Mate Silver uses a RN-42 module as well, so everything from the supply voltage to the data rates are identical between the two devices. The BlueSMiRF also costs $24.95.

The third breakout board Bluetooth module is the SparkFun BLE Mate 2, which uses a BC118 module. This particular module uses the Bluetooth v4.0 protocol and connects exclusively to other Bluetooth v4.0 devices. This module is also a Bluetooth Low Energy device, meaning power consumption is relatively lower. Its supply voltage ranges from 3.3V to 4.7V, and its transmit current is 16mA. The output power of the BC118 module is 7.5dBm. This module uses the UART serial interface to connect with development boards. The BLE Mate 2's

price is the highest among the three Bluetooth modules that have breakout boards with a price of $29.95. This Bluetooth breakout board was the last of the initially researched devices from the first semester of senior project research.

At the time of our second report from the previous semester, we had chosen the BlueDuino with built-in Bluetooth as our microcontroller of choice. We have, however, switched from the BlueDuino in favor of the microcontroller on the Teensy 3.2 development board. As a result of switching microcontrollers, we needed to purchase a separate Bluetooth module. The integrated circuits (IC) chip that comes with the BlueDuino is the Texas Instruments CC2540. Although we switched to the Teensy 3.2, we decided that any Bluetooth module that we will purchase will need to use the CC2540 IC chip.

Instead of completely designing and building a Bluetooth antenna from scratch for data transmission, a complete Bluetooth module with an antenna built-in was purchased. Two Bluetooth v4.0 HM-11 modules, which use CC2541 IC chips, were among the first Bluetooth modules procured. The CC2541 chip is slightly newer than the CC2540 one, but the two chips share similar hardware specifications. The main differences between these two chips is the optimized transfer power consumption. The transfer power of the CC2541 is 18.2 mA at 0 dBm as opposed to the 27 mA at 0 dBm power consumption of the CC2540. Since there were not many major alterations between these two chips, our progress was not hindered by the difference in chips.

Through the testing phase, the Bluetooth design has changed considerably. Thus, the final Bluetooth module changed from the BC127 to the HM-11 modules. Originally, the speaker module design was simply a Bluetooth speaker attached to a shirt. There was to be only one Bluetooth module, which would have been located on the glove module. The BC127 along with the RN-52 modules were chosen for the sole purpose of streaming audio from the glove module to the speaker module. The RN-52 proved to be challenging to interface with and was scrapped for its difficulty of understanding its protocols. The BC127, which had its own Arduino library developed by SparkFun, was much easier to interface with but had troubling communication protocols. With the redesign of the speaker module, the HM-11 modules were chosen for its overall ease of use while still completing its necessary functionality for the project. Table 4

represents all the Bluetooth modules that were considered throughout the entire senior project designing and testing phases. The table gives a visual representation of the different features that were compared, such as power output, serial interfaces, and prices to name a few.

	Bluetooth Module	Supply Voltage (V)	Tx Current (mA)	Output Power (dBm)	Dimensions (mm)	Serial Interfaces	Price
	BL652	1.7 ~ 3.6	5.3	4	14 x 10 x 2.1	I2C, SPI, UART	$7.45
	BT900	1.8 ~ 3.6	85	8	2.5 x 12.5 x 19	I2C, SPI, UART	$19.00
	SaBLE-x	1.8 ~ 3.8	7.9 ~ 13.6	5.3	11.6 x 17.9 x 2.3	JTAG, UART	$16.52
	Bluetooth Mate Silver	3.3 ~ 6	0.026 ~ 50	~4	Board: 44.45 x 16.51 (1.75 x 0.65 inches)	UART	$24.95
	BlueSMiRF Silver	3.3 ~ 6	0.026 ~ 50	~4	Similar to Mate Silver	UART	$24.95
	BLE Mate 2	3.3 ~ 4.7	16	7.5	Similar to Mate Silver	UART	$29.95
	CC2540	2 ~ 3.6	21.1 ~ 31.6	4	--	UART	$5.85
	HM-11	3.3	50	--	13.5 x 18.5 x 2.3	UART	$12.95
	BC127	3.3 ~ 4.7	15	10	11.8 x 18 x 3.2	UART	$26.95
	RN-52	3.0 ~ 3.6	30	4	26 x 13.5 x 2.7	UART	$25.00

Tables 4. Bluetooth Options

4.1.4 Speaker

For the speaker module, the initial design consisted of only a Bluetooth speaker that handled wireless communication. The final project design, however, only required a regular, wired speaker connected to a microcontroller. The purpose of the initial Bluetooth speaker design was to keep the entire speaker module practical and lightweight as well as to have the least amount of hardware components as possible. These goals were theoretically achievable considering that the Bluetooth module on the glove only needed to send raw audio packets to the Bluetooth speaker, which would have been the only hardware component on the shirt. The Bluetooth speakers that were being researched at the time were ones that had compatible Bluetooth versions with the Bluetooth modules being tested, which is Bluetooth v4.0. The BC127 Bluetooth module supports version 4.0, while the RN-52 Bluetooth module supports version 3.0. Both of these modules have the capability of sending and streaming audio packets. Because there were two different Bluetooth modules with two different versions being tested, Bluetooth v3.0 and v4.0 speakers were researched. These speakers include:

	Speaker	Bluetooth Version	Price
	MiniX3 Wireless Bluetooth 4.0 Speaker	4.0	$ 6.69
	Portable Mini Waterproof Shockproof Wireless Bluetooth 4.0 Stereo Speaker	4.0	$ 9.95
	Portable Waterproof Shockproof Wireless Bluetooth Speaker For Smartphone TSUS	4.0	$ 10.25

	Original Xiaomi Mini Bluetooth 4.0 Speaker	4.0	$ 15.97
	HD Water Resistant Bluetooth 3.0 Shower Speaker	3.0	$ 10.95

Table 5. Bluetooth Speaker Options

With the redesign of the speaker module, new wired speakers needed to be researched. Characteristics that were being considered during the decision-making process include size, weight, and sound quality. In order for a regular, wired speaker to play the sounds both loudly and clearly, there would need to be an amplifier to increase the voltage coming out of the microcontroller's DAC. Sound quality of the speaker, especially at higher volumes, is also important as the sound produced by the speaker is a human voice, which needs to be clear for people to easily understand.

A speaker that was researched for our final design was the 4 Ohm 3 Watt speaker. This speaker weighs 50.48 g, which makes it practical, lightweight, and portable. Additionally, this 3W speaker is capable of producing sounds that are loud enough to be heard clearly, especially in noisy environments. Another speaker that was researched was the 0.25W Thin Speaker, which is a much smaller speaker at only 40mm in diameter. Because this is a 0.25W speaker, its volume and clearness are not optimal for the purposes of this project. Similarly, the Round Frame Mini Speaker is small with a 2" diameter, but the volume output is once again not optimal for this project at 1W of power. Lastly, the Peerless speaker has the most power at 40W, which means it is capable of producing the loudest, most audible sound. However, due to its overall weight and bulky form factor, it would not be practical to use as part of the speaker module.

	Speaker	Price	Specifications
	Speaker - 3" Diameter - 4 Ohm 3 Watt	$1.95	4 Ohm 3 Watt 3“ Diameter 50.48 g
	0.25W Thin Speaker	$0.95	8 Ohm 0.25 Watt 40mm Diameter 3 g
	Peerless by Tymphany TC10FG00-04 4" Full-Range Line Array Driver 4 Ohm	$13.23	4 Ohm 40 Watt 4” Diameter 1.45 lbs
	2” Round Frame Mini Speaker 16 Ohm	$2.49	16 Ohm 1 Watt 2” Diameter 0.135 lbs

Table 6. Speaker Options

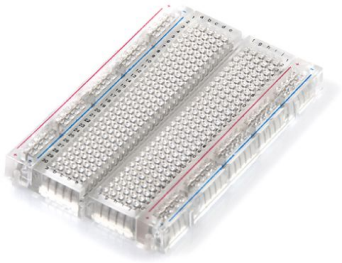
4.1.5 Printed Circuit Board (PCB)

In order to incorporate all the hardware components on the glove and speaker module, a board will be needed to connect everything. There are three different options to wire all the components, a breadboard, perfboard, and printed circuit board.

A breadboard is used to prototype circuits without the need to solder devices together. Breadboards are offered in different sizes which corresponds to the pricing. Using this type of platform allows testing flexibility for smaller scale circuits. This would not be ideal for high frequency, voltage, and current circuits.

Perfboards are another option in wiring all our hardware components. This board also varies in price in correspondence to the size of the perfboard. The main difference between a breadboard and perfboard is that perfboards require soldering components to the pads.

Another platform for hardware connection is a printed circuit board (PCB). In order to use a PCB, the designer needs to use a program called Eagle. Eagle allows free limited use of designing PCB layouts which can be sent to a manufacturer. PCBs allow for a compact product of the design which is helpful if a project has a restriction of space. The con to using a PCB is the time allotted for designing and manufacturing the board. If there is an error, the layout must be redesigned and submitted again.

	Type of Board	Price
	Breadboard	*Varies depending on size EX. 3.29 x 2.15 x 0.33" = $4.95

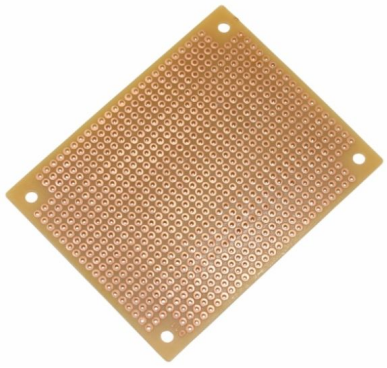
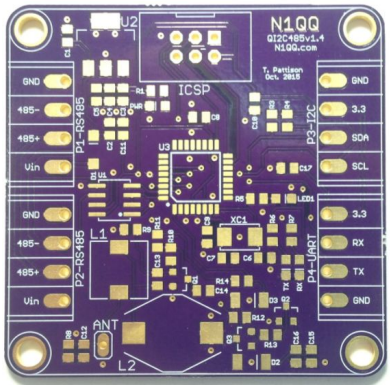
	Perfboard	*Varies depending on size EX. 2-1/2 x 3-1/8" = $1.35
	Printed Circuit Board	$5.00/square inch

Table 7. Board Options

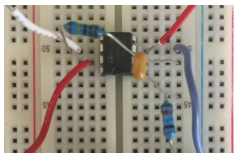
4.1.6 Amplifier

The speaker needed an amplifier to drive it to produce an audible sound from the microcontroller's DAC. The amplifier would have to meet a couple of requirements to be usable. One requirement for the amplifier is that it would have to be small and light, allowing it to be portable and easily wearable on a shirt. Another requirement that the amplifier would need to have is that it would have to be within our power budget.

One option that was researched for amplification of the signal coming out of the microcontroller's DAC was building our own amplifier to fit these requirements. This amplifier would be a high pass filter that would filter out all of the frequencies too high and too low for humans to hear. The amplifier that was designed did this and had an input voltage of 12V peak to peak. The designed amplifier would have to be integrated onto a PCB designed by the group. It

would also need purchase of surface mounted components. This would come out to a cost of over 25 dollars.

Another option that was researched for this was to buy a commercial audio amplifier. An amplifier that was researched to meet the requirements was the 12V Mini Hi-Fi PAM8610 2X10W Audio Stereo Amplifier Board Dual Channel. This amplifier costs 5.99 dollars and meets power requirements with a supply voltage of 12V. This audio amplifier also comes with a volume control.Another amplifier that was also considered through the research process was the NanoFree Audio Amplifier. This amplifier would need much less voltage to run, at 2.5V - 5V. However, the price of this amplifier at 49.53 dollars was extremely high compared to the other amplifier options. Another amplifier that was small and compact was the Geree audio amplifier. This amplifier would be pushing our power budget as it needed a voltage supply of 12V - 32V. It has a reasonable price at 9.54 dollars. The specifications of the researched amplifiers are shown in table 8 below.

	Model	Voltage Supply	Price
	LM741 Op-amp High Pass Filter	12 V	>$25
	12V Mini Hi-Fi PAM8610 2X10W Audio Stereo Amplifier Board Dual Channel	12 V	$5.99
	TPA2100P1 NanoFree™ 1-Channel (Mono) Output Class D Audio Amplifier Evaluation Board	~2.5 V - 5.5 V	$49.38

	GEREE AC 12V-32V 100W TDA7293 Digital Audio Amplifier AMP Board Mono Single Channel	12 V - 32 V	$9.54

Table 8. Amplifier Options

4.1.7 Power

With the hardware selections finalized, the amount of voltage and current needed to operate each device was noted. There are many options for power sources that can be considered for the American Sign Language Glove. However, size, weight, cost, and energy density for the devices will narrow down the options.

One option that has become popular over the years is photovoltaic cells. There has been a great advancement in the development of miniature inorganic solar cells. These miniature cells are made with crystalline silicon which uses an ultrathin silicon film as an active photovoltaic layer. This allows for a more flexible integration to a design as these cells are lightweight as well as environmentally friendly with the abundance harvesting resources. With more research, these photovoltaic cells can be wired with battery to charge and provide power for the system. However, the prices for photovoltaic cells cost more that the lithium batteries when looking for a thin flexible cell.

Another option is the ultra-thin flexible screen printed rechargeable polymer battery. There has been research done that shows that these batteries are a good option to consider. The lithium polymer battery can be as thin as a credit card and be designed into different shapes while still holding a relative good battery life. However, the cost of manufacturing is pricy. The rechargeable batteries are also lightweight, flexible, and environmentally friendly. These are ideal traits to add to a wearable device.

Similar to the lithium polymer battery, the lithium ion battery is thin and lightweight, just packaged differently with a different chemical composition. The lithium ion battery has a higher energy density compared to the polymer battery and cost less to fabricate. The recharge life is

approximately 300-400 cycles. The dimensions of both the lithium polymer and lithium ion battery vary depending on the power requirements of the components.

Comparing all the different power sources, Table 9, it is concluded that the lithium ion battery would be a viable power source, two 3.7V 150mAh cells were tested. Each of the hardware components have an operating voltage range which 3.3V will suffice. Note that there are two cells because the battery will dissipate over time. There needs to be enough clearance in providing enough voltage to the devices. There will also need to be a voltage regulator to step down the two cells combined to 3.3V. With our initial microcontroller selection of the Bluedino, there was a voltage regulator embedded with it. Now that our selection has changed to the Teensy 3.2, a voltage regulator will need to be circuited.

Battery	Voltage (V)	Current (mAh)	Price ($)	Charge Cycles	Seller	Power (mW)	Operating Hours for Glove (Hr.)	Weight (g)
Lithium Ion	3.7	80	12.09	500	Mouser Electronics	296	~1.34	2.6
Lithium Ion	3.7	105	5.95	500	Adafruit	388.5	~1.75	3
Lithium Ion ✓	3.7	150	5.95	300	Adafruit	555	~2.51	4.65
AA Alkaline	1.5	1800	1.00	0	Office Depot	2700	~30.07	23
Solar	1	200mA*	5.85	N/A	VellemanStor	200	N/A	–

Table 9: Power Source Options

4.2 Software

Four different algorithms are implemented in the final design of the American Sign Language Interpreter project. Each one is coded in C using Arduino libraries and was tested individually before being integrated with the overall design.

4.2.1 Sign-to-Letter Algorithm

The main purpose of this sensor algorithm is to collect data from the five flexible sensors and the motion sensor from the glove module. Once the sensor data is read in, the algorithm uses threshold values determined through data collection and testing, and converts the values measured from the sensors into corresponding letters. The threshold values determine how much

bene each finger has, while the accelerometer and gyroscope in the motion sensor return the yaw, pitch, roll, and acceleration of the hand, which allow the algorithm to determine how the hand is moving and is oriented. The algorithm creates a five-digit ID from the signals from the flexible sensors. The ID and the data from the motion sensor are then used to characterize which letter is being read using if-statements to match the appropriate letter based on hand orientation, hand movement, and the five-digit ID number. The values used to threshold the signal were determined using the testing methods described in Chapter 5.

4.2.2 Bluetooth Algorithm

The Bluetooth algorithm is open source and can be located on the github repository [21]. On the repository, it mentions that the code was developed for the legacy HM-10 module. However, both the HM-10 and HM-11 modules use the same AT command set. AT commands are instructions that allow the user to modify characteristics of the Bluetooth module, such as baud rate, master or slave roles, and device pairing.

The open source code uses the Arduino *SoftwareSerial* library to set up communication between the HM-11 module and the microcontroller on the Teensy 3.2. The first step of the Bluetooth algorithm is to check if the HM-11 module is powered on and detectable. It does so by attempting to ping the module with different baud rates. The HM-11 is operating at the 115200 baud rate. If the module is detected, then it responds with "OK+CONN". Depending on which module the Bluetooth device is attached to, the Bluetooth device's role will be set differently. The one on the glove module will be set to the role of *master* because it is transmitting data. The Bluetooth device on the speaker module will be set to the role of *slave* because it is receiving the data. The *slave* Bluetooth device is constantly listening and awaiting data from the glove module.

4.2.3 Sound File Algorithm

The sound file algorithm picks up right where the Bluetooth algorithm leaves off. This algorithm is solely implemented on the speaker module. As soon as the *slave* Bluetooth module receives the signed character from the glove module, it parses through the microSD card from

the corresponding sound file. The sound file is sent to the DAC on the microcontroller to output the sound through the wired speaker. The sound file algorithm uses open source code from the developers of the Teensy microcontroller development board, more specifically the *WavFilePlayer.ino* [20], which uses the Arduino *Audio* library.

Chapter 5: Results of Hardware and Software Testing

5.1 Hardware

With senior design coming to an end, all hardware components for the American Sign Language Interpreter project have been procured and implemented. Hardware testing has been done on each separate component of the project before integrating each of them into the overall design. The results of our hardware testing are described in the following sections.

5.1.1 Flex Sensor

Two sizes of flexible resistive sensors are being used in the design; 2 two inch sensors will be used to measure bend in the pinky and the thumb, while 3 four inch sensors will be used to measure bend in the index, middle, and ring fingers. All of these sensors, including backups have been purchased. These sensors have been implemented on a glove, and have been used to collect data for different hand signs in order to characterize the signals to create the sign-to-letter algorithm. The sensors use op-amp filtering and feed an analog signal to the controller. The data collected for each hand sign has also been displayed using Matlab in an effort to characterize the sensor data and determine if we can differentiate between the different signs. The graphical representation of the data for the letter "A" is shown in the figure below.

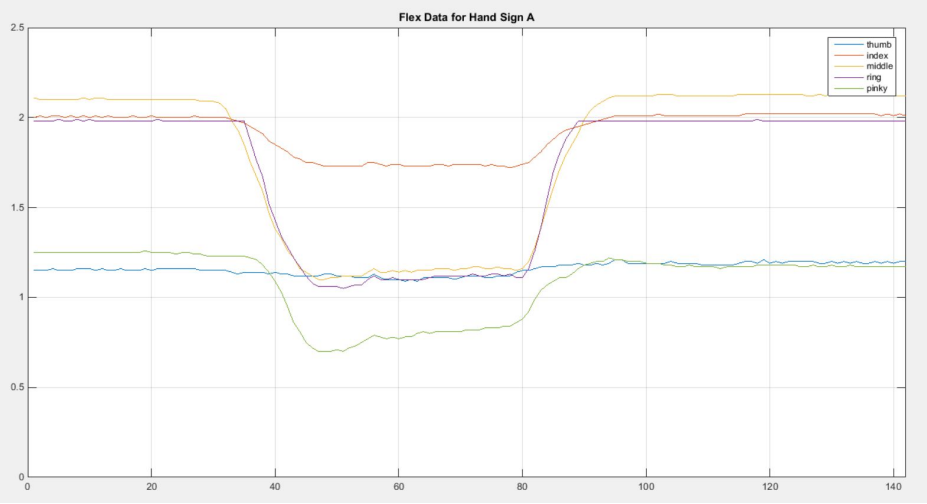


Figure 5. Plot of data collected from 5 flexible sensors for letter "A"

5.1.2 Motion Sensor

The MPU-6050 motion sensor has been obtained and implemented using the controller. This sensor has been tested using a variety of hand signs that incorporate different hand orientations or hand motions that will be used. A baseline value was also created by having the sensor measure a straight, horizontal hand with no movement. After calibrating the sensor, we were able to determine that there is a minimal amount of noise seen in the readings of the yaw, pitch, and roll axes, which can be filtered out using simple signal processing of averaging groups of around five data samples together. Furthermore, strings of hand signs were also tested to see if we could discriminate between each individual sign in the string, as shown in Figure 6 below.

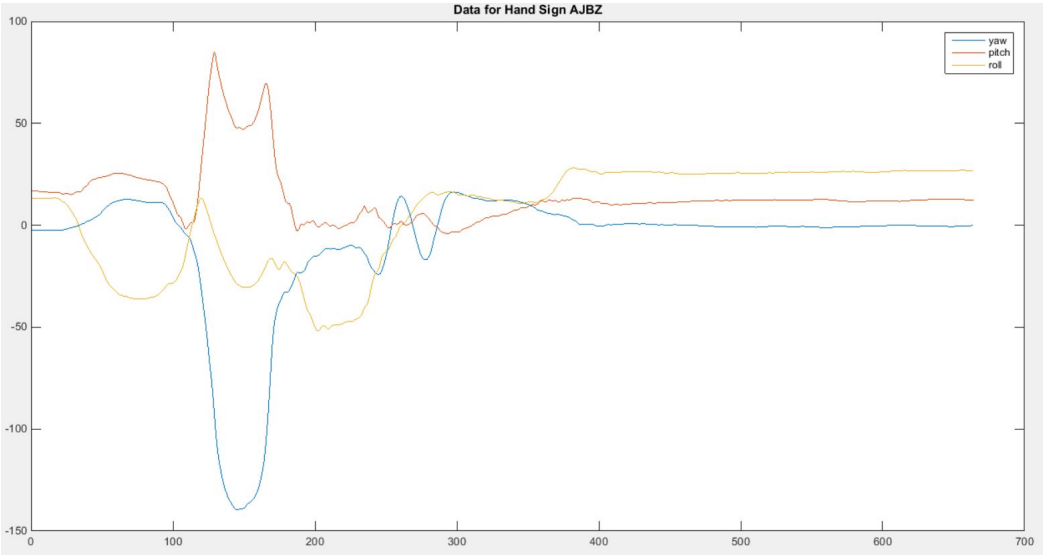


Figure 6. Plot of data collected from MPU-6050 sensor showing a string of hand signs

5.1.3 Teensy 3.2 (Microcontroller)

We have used the Teensy 3.2 Microcontroller to test and record sensor values and other hardware components that we plan on using. These hardware components include the flexible sensors, motion sensor, microSD Shield, speaker, and Bluetooth. These tests were done by coding in the Arduino IDE and uploaded to the board via the Teensyduino. The data of these tests were stored via microSD card. This data was then visualized using Matlab to create graphs to make the data more readable. The results of the sensor testing are described and shown in

Chapter 5.1.1 Flex Sensor and Chapter 5.1.2 Motion Sensor. The results of the Bluetooth testing are explained in Chapter 5.1.4 Bluetooth.

5.1.4 Bluetooth

Two HM-11 Bluetooth modules are being used for the wireless communication between the glove and speaker modules. The main task for the two Bluetooth modules is to transmit and receive the character of whatever sign the user is making. The glove module's Bluetooth is configured as the *master* Bluetooth device because it is the one transmitting the characters, while the speaker module's Bluetooth is set as a *slave* because it is the one receiving the characters. The *master* Bluetooth device is programmed to always pair and connect to the *slave* device during the algorithm setup phase.

Through the initial testing phase, one HM-11 module was connected to a Teensy 3.2 development board while the second module was wired to an Arduino Uno development board. The two Bluetooth devices were able to pair and communicate with each other. Characters were able to be sent between the two. However, the speed at which the two modules received and transmitted characters differed greatly. Arbitrarily speaking, the Bluetooth module connected to the Teensy 3.2 development board transmitted five times faster than the one wired to the Arduino Uno. This can most likely be attributed to the differing clock speeds between the two microcontrollers. After receiving a second Teensy 3.2 development board and performing tests with this new board, the HM-11 modules were capable of sending and receiving characters that the user was signing without much issue. The only problem with the wireless communication at this point is that the Bluetooth devices may occasionally unpair and disconnect from each other, which cannot be controlled.

5.1.5 Speaker and Amplifier

Tests were performed on the speaker and amplifier circuit to examine the volume and clearness of the sound. The way that the speaker was tested was that the microcontroller would pull a sound file stored on an sd card and would play the sound through the DAC. The DAC would be hooked up to an amplifier and the amplifier to the speaker. An unassisted hearing test

would be used to check if the sound of the speaker was acceptable. The setup is shown below in Figure 7.

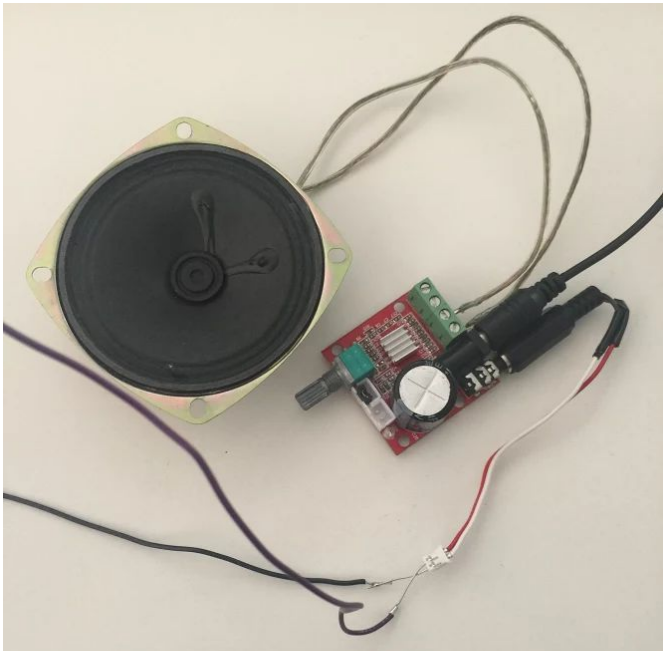


Figure 7. Set up of Commercial Amplifier System for Testing

As mentioned earlier, the speaker needed for the module needed an amplifier to drive it. This is because the peak to peak voltage coming out of the microcontroller's DAC was 0.125 V. To produce a good volume, the amplifier needed to amplify the peak to peak voltage to around 5 V for the 8 Ohm 3 Watt speaker that was chosen. This voltage was calculated from the equation: Voltage = sqrt(Power * Impedance). A circuit that was tested was a high pass filter shown below in Figure 8. The resistor and capacitor values for the circuit were calculated so that the cutoff frequencies would be the frequencies in the range of human hearing. However, this circuit, tested with a variety of different resistor and capacitor values that would give the same cutoff frequencies and gain, when connected with any of the speakers, would give an audible humming noise in the background along with the audiofile played.

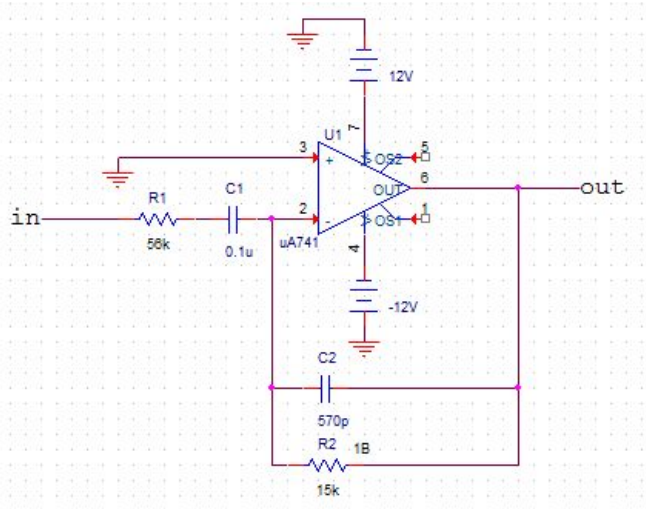


Figure 8. High Pass Filter Circuit Schematic

Due to the humming noise generated by the designed amplifier, it was decided to use a commercial amplifier to amplify the signal from the DAC to power the speaker. To use the amplifier, an AUX connection was needed. This connection would be the input into the amplifier, or the DAC. Due to the fact that the DAC off of the microcontroller does not come with this connection, we needed to make our own by stripping an existing AUX cord to get to the components that we could connect the DAC to. This is shown in Figure 9 below. The speaker was connected to the amplifier and it was powered on via a wall outlet to produce the sound. The sound from the speaker, when well grounded, was clear and loud. Exactly what we were looking for in our speaker for the speaker module.

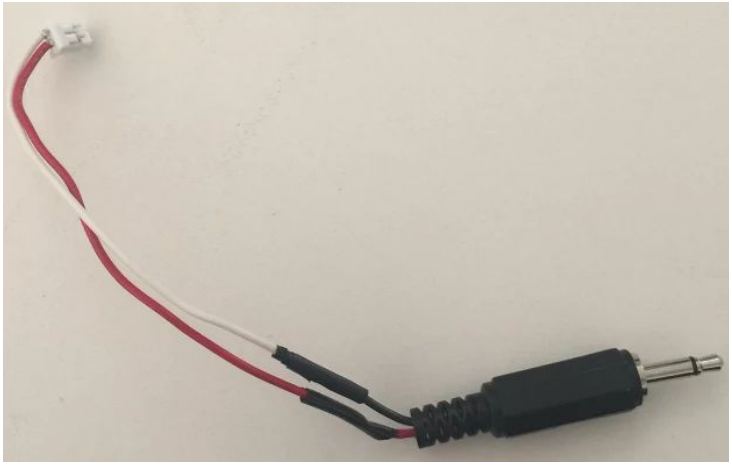


Figure 9. Dissected AUX cord

5.1.6 Printed Circuit Board (PCB)

After finalizing our sensor choices, we discovered that we needed some type of sensor shield to connect the sensors to the microcontrollers, as well as contain the circuitry needed for the flexible sensors. Additionally, the circuit for the voltage regulator is contained on this sensor shield.

There will be two PCBs: one for the microcontroller, sensors, and batteries (Figure 10); one for the Bluetooth and SD card shield (Figure 12). Both PCBs have been ordered on February 22nd from OSH Park, and we are currently waiting for the shipment so we can proceed with testing.

With most of the hardware finalized, we designed a PCB to connect the microcontroller, flexible sensors, motion sensor, voltage source, and the Bluetooth. This will allow us to consolidate all the hardware on the back side of the hand.

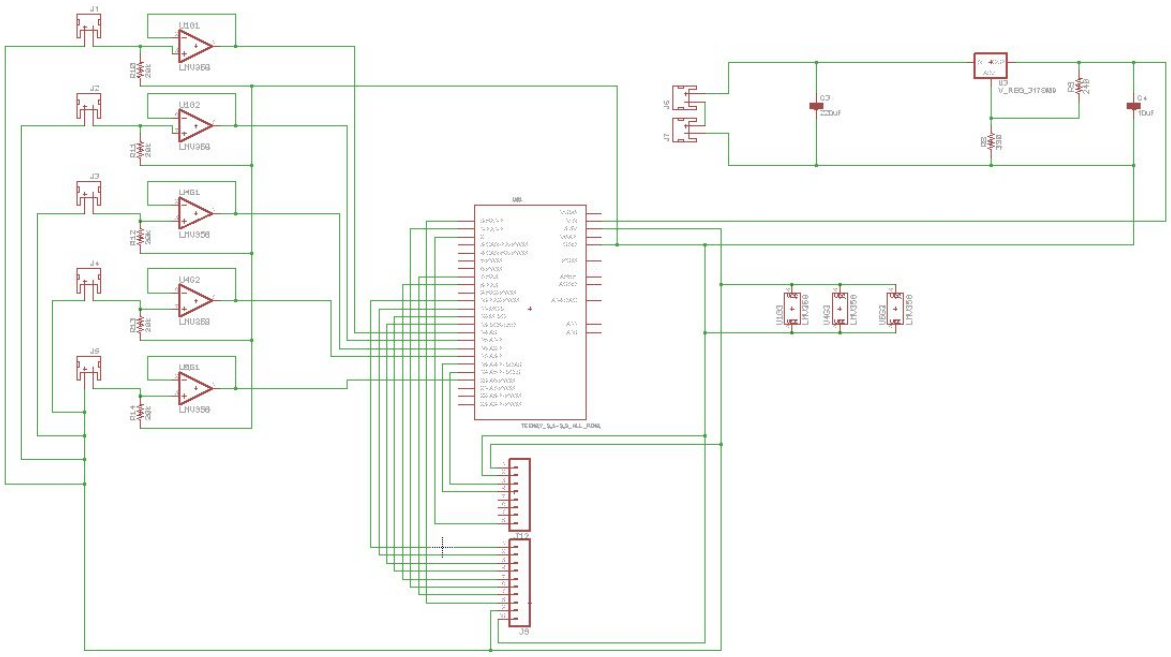


Figure 10. Eagle Schematic of Sensor Shield

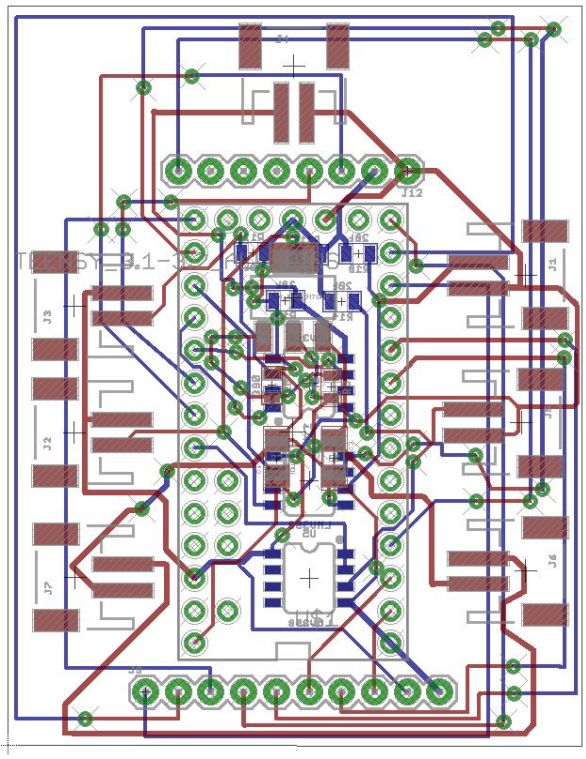


Figure 11. Eagle Board Layout for Sensor Shield

Figures 10 and 11 show the schematic and board layout of all the devices placed on the PCB for our sensor shield. This PCB will be placed on the glove module. There are two layers to the PCB, the top (red) and bottom (blue). The bottom layer contains the resistors and op amps which will keep the board flushed against the hand while the Teensy 3.2 microcontroller and JST connectors lay on top. The JST connectors is used for battery and flex sensor connections.

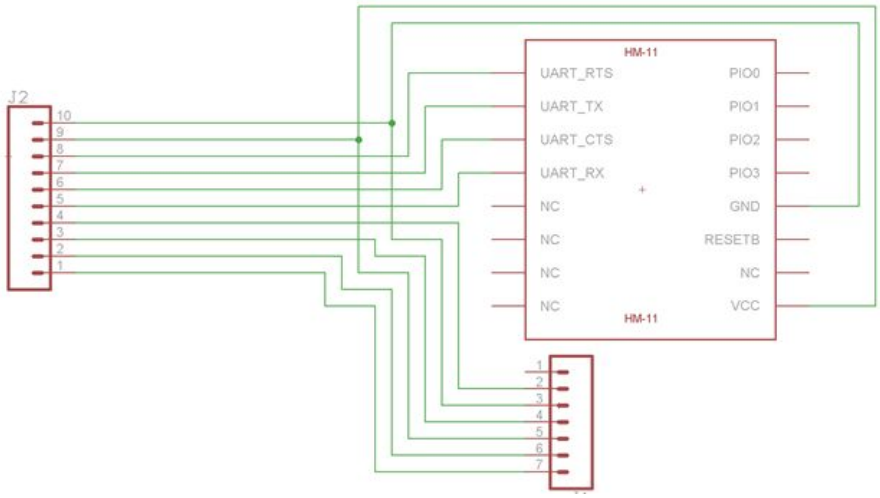


Figure 12. Eagle Schematic of Bluetooth Shield

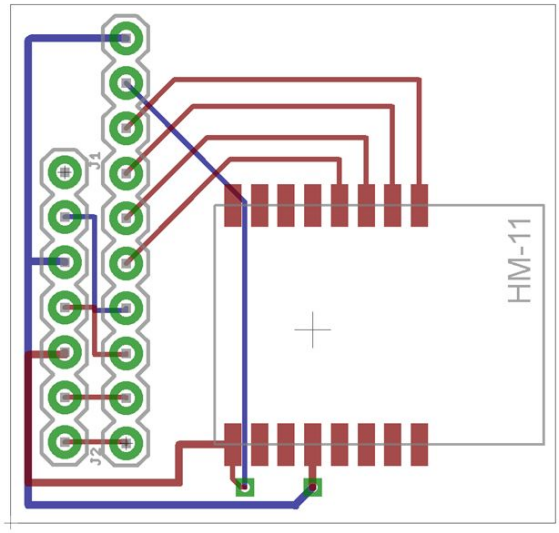


Figure 13. Eagle Board Layout for Bluetooth Shield

Figures 12 and 13 show the second PCB schematic and board layout that will also be on the glove module. The Bluetooth shield PCB is designed to connect the Bluetooth module from the wrist strap to the Teensy 3.2 on the glove.

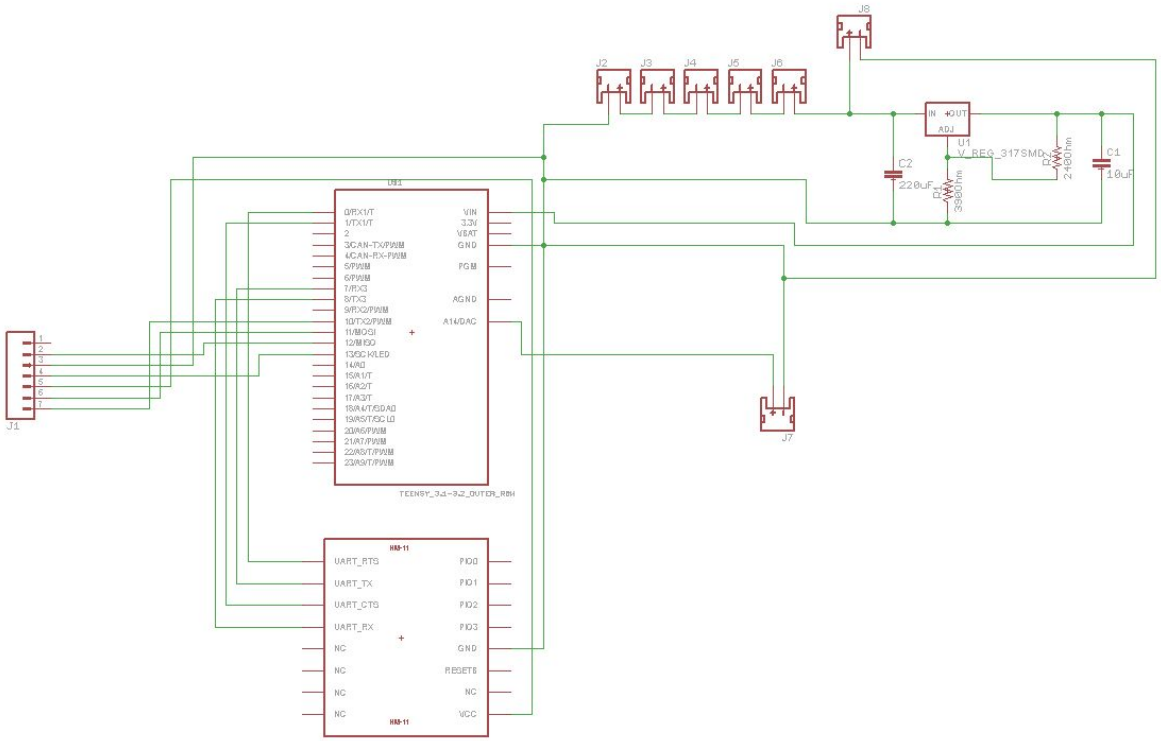


Figure 14. Eagle Schematic for Speaker Module

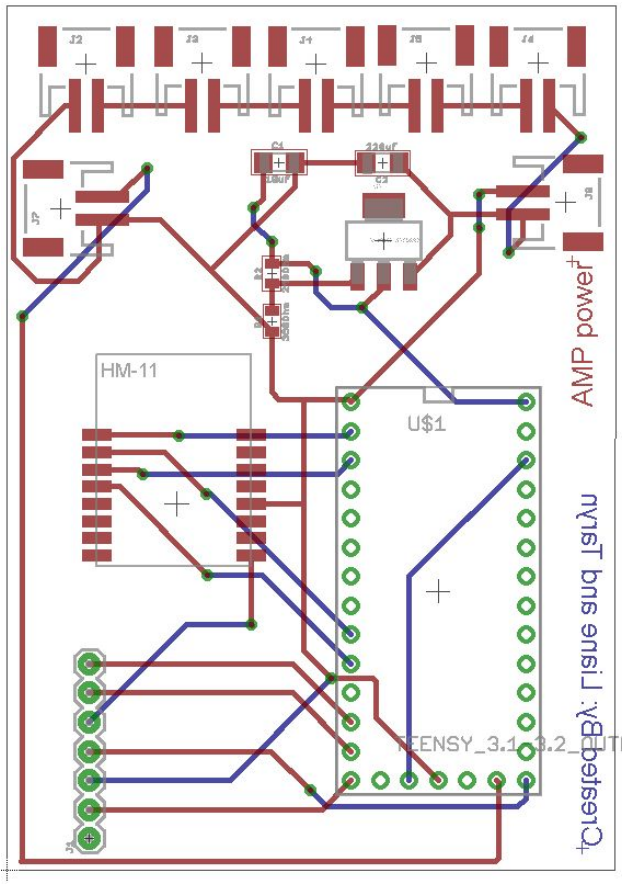


Figure 15. Eagle Board Layout for the Speaker Shield

Figures 14 and 15 depict the speaker shield schematic and board layout for the PCB . This PCB will be attached to a shirt on which an amplifier and speaker will be connected. The speaker shield consists of another Teensy 3.2 and Bluetooth module to allow communication between the glove and speaker.

5.1.7 Battery

The power source for the components of our design will be two 3.3V 150mAh lithium ion batteries. Two batteries will be required to allow clearance for battery voltage dissipation over time. This will provide about 2.7 hours worth of operation when all hardware devices are consuming maximum current. These batteries can be recharged more than 300 cycles with a compatible USB charger, minimizing the amount of waste for our project. The lithium ion batteries are from Adafruit at a cost of $5.95 each. The weight of each cell is 4.56g with

dimensions of 19.75mm x 26.02mm x 3.8mm, which is about the size of a quarter. These characteristics meet our project specifications.

Simulation and testing have been conducted to verify that the batteries will provide enough power for the entire system on the glove.

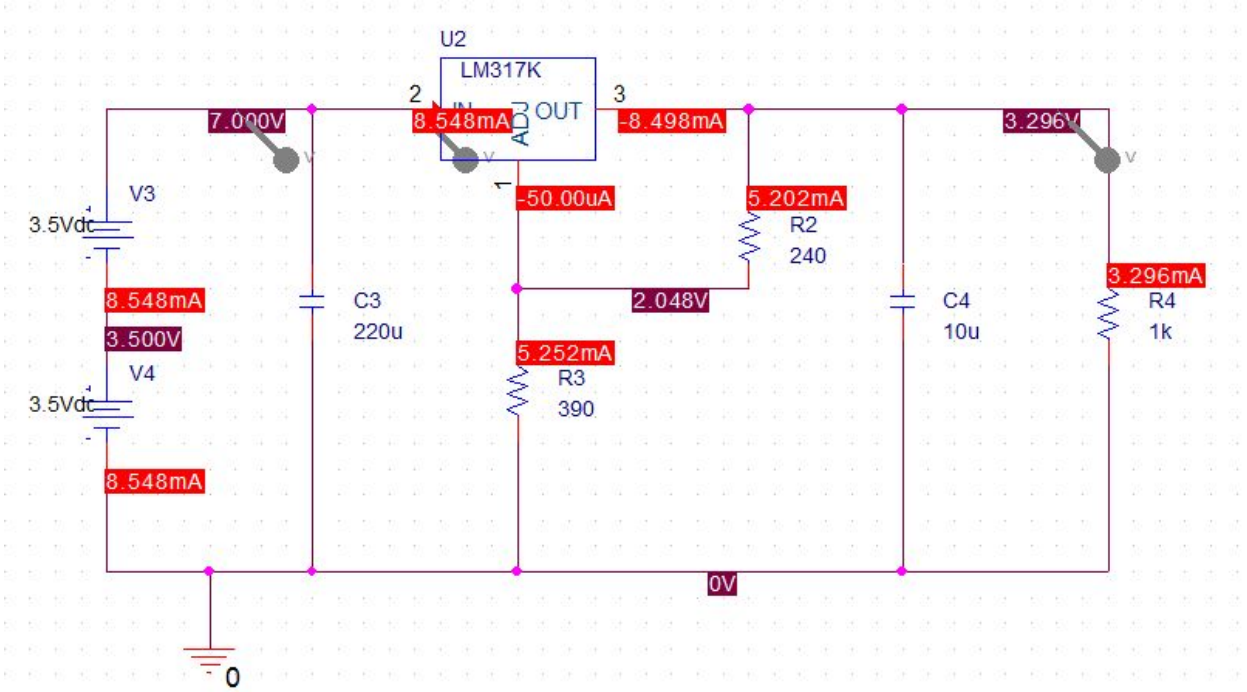


Figure 16. PSpice Simulation Schematic of Voltage Regulator

The simulation results of the step down voltage regulator in Figure 16 show that the LM317- regulator can step down input voltages to 3.3 volts allowing a maximum of 1.5A. When the input voltage is greater than 5V, the LM317- regulator outputs 3.296V. Verifying the simulation results with the output goal, testing was conducted on the voltage regulator hardware, LM317T.

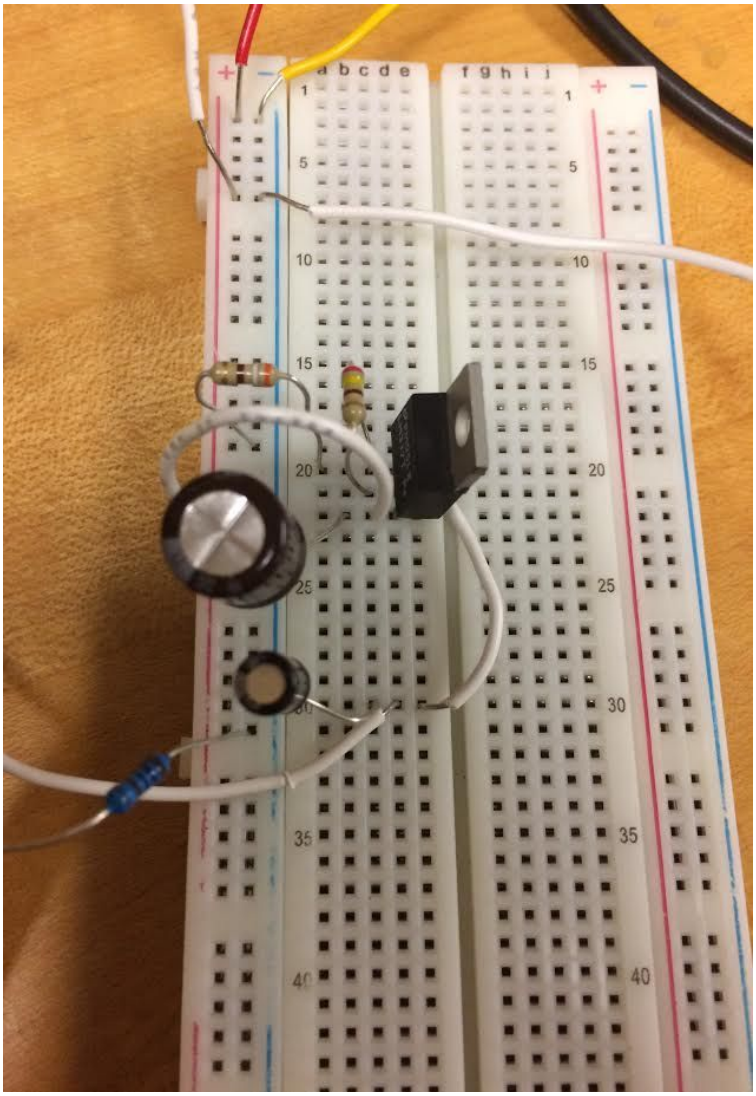


Figure 17. LM317T Voltage Regulator Testing

Input Voltage (Volts)	Output Voltage (Volts)
4.997	3.266
5.996	3.266
8.4	3.266

Table 10: Voltage Regulator Results

Figure 17 is the circuit implementation of the PSpice simulation in Figure 16. Using a power generator and multimeter, the LM317T regulator was tested. Different input voltages greater than 5V were tested and the output voltages were measured. Table 10 shows the results of the voltage regulator circuit.

5.2 Software

All software testing for the American Sign Language Interpreter project uses Arduino code and the Arduino IDE. Matlab has also been used to provide visual representations of the sensor data.

5.2.1 Sign-to-Letter Algorithm

Prior to the implementation of the signal-to-letter algorithm, preliminary data was collected from each of the sensors and plotted the flexible sensor data in Excel and motion sensor data in Matlab. These preliminary graphs were used to check sensor calibration, determine how much noise was produced by each sensor, and to provide a starting point of what to expect when characterizing the sensor data. Figures 18 and 19 shows the plots created for the data collected for the letter G.

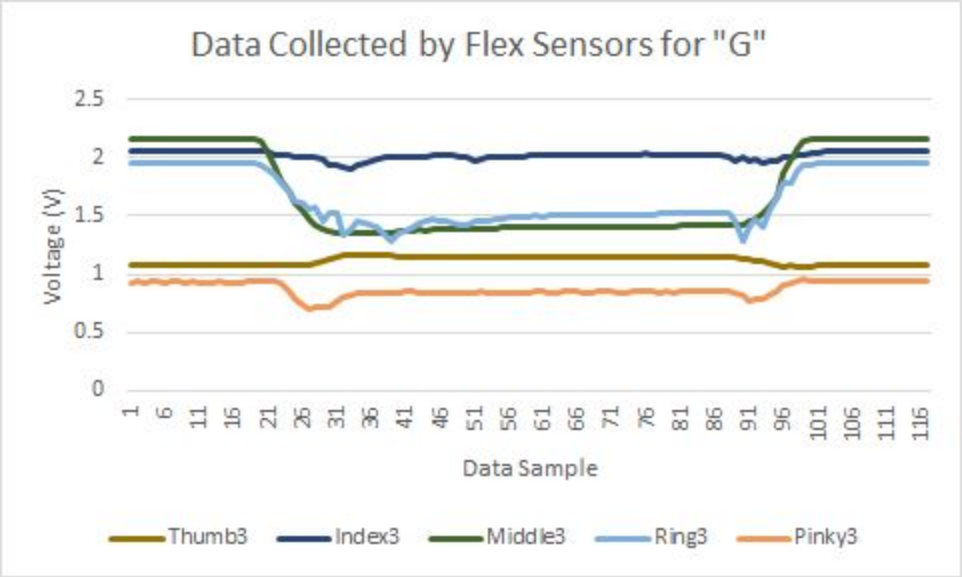


Figure 18. Data collected from the flex sensors for the letter "G"

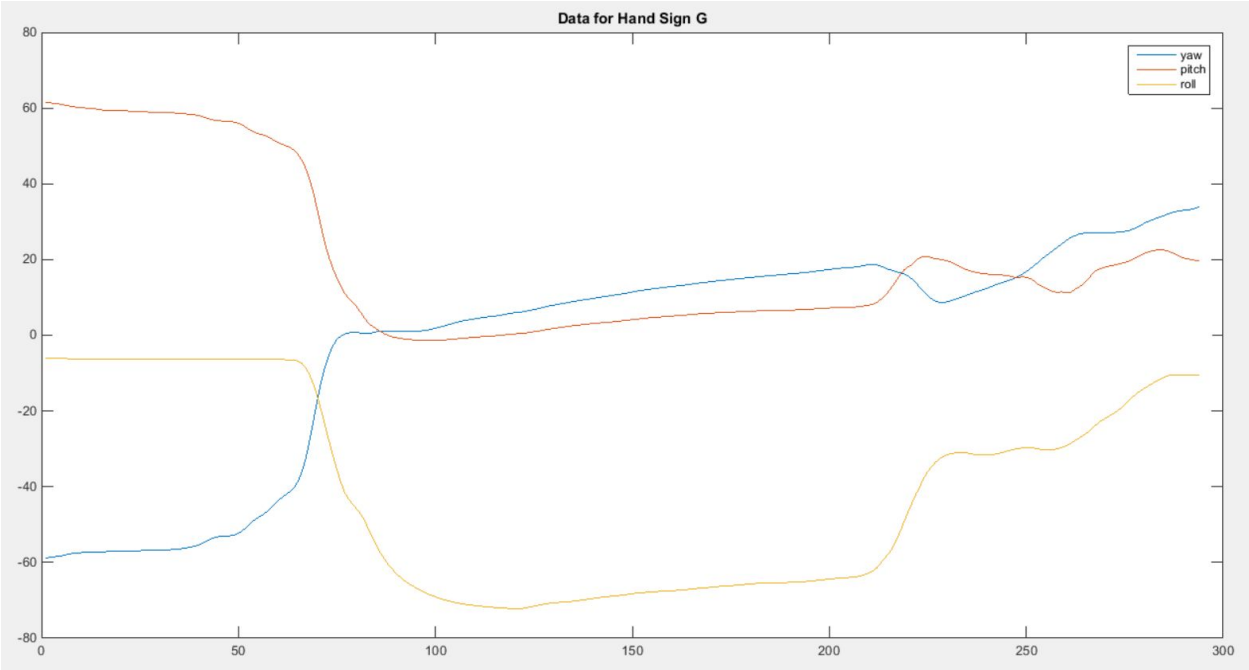


Figure 19. Data collected from the motion sensor for the letter "G"

The initial sign-to-letter algorithm used threshold values determined after sample data collection and analysis from each of the sensors for each letter in the alphabet. The data was plotted to ensure validity, and then used to extract values for each sensor with each letter. These values were then used to create threshold values to characterize four levels of bentness for each finger. By characterizing each letter to be equivalent to a five digit ID number, the algorithm chooses the corresponding letter with the correct ID.

Data Set	Number Correct	Number Incorrect	Total Trials	Percent Accurate
1	6	45	51	11.76
2	18	20	38	47.37
3	3	23	26	11.54
4	7	44	51	13.73
Total	34	132	119	28.57

Table 11: Initial testing results using multiple user data

In the second iteration of the algorithm, motion sensor data was incorporated into the algorithm. The motion sensor data takes into account hand orientation to discriminate between letters that have similar finger positions. To characterize hand motion, the algorithm implements if-statements to look at orientation, acceleration, and finger position to determine the corresponding letter. With the addition of the use of motion sensor data, the number of users used to test the algorithm was decreased to one, with the designated tester role assigned to Taryn Aranador. The testing procedure was modified to include five rounds of testing each letter of the alphabet, with a wait time of 3 seconds to let the algorithm to characterize the letter before the trial was determined incorrect. The results of the trial were recorded and are shown in Table 12 below. With these changes, the accuracy of the system increased to 79.26%.

Letter	Correct	Incorrect	Total	% Accuracy
A	4	1	5	80
B	5	0	5	100
C	5	0	5	100
D	5	0	5	100
E	4	1	5	80
F	5	0	5	100
G	5	0	5	100
H	5	0	5	100
I	4	1	5	80
J	5	0	5	100
K	3	2	5	60

L	5	0	5	100
M	2	3	5	40
N	0	5	5	0
O	5	0	5	100
P	4	1	5	80
Q	4	1	5	80
R	4	1	5	80
S	5	0	5	100
T	3	2	5	60
U	3	2	5	60
V	4	1	5	80
W	5	0	5	100
X	3	2	5	60
Y	5	0	5	100
Z	0	5	5	0
Neutral	5	0	5	100
Total	107		135	79.25925926

Table 12: Results of testing the second iteration of the sign-to-letter algorithm

Though these trials showed vast improvement due to the change in expectations regarding the versatility of the device, the system accuracy was still ten percent below the target accuracy described in the design requirements, and thus needed further improvement. The characteristics for the letter "Z" were redefined to look at the presence of acceleration measured by the accelerometer. Further adjustments were made to the thresholds and additional possible finger position IDs were added to each letter to improve the likelihood of the algorithm correctly

selecting the letter. The third round of testing included ten total trials of each of the letters. Similarly to the previous round of system accuracy testing, these trials involved forming a letter while wearing the glove and giving three seconds for the algorithm to guess the correct letter as its initial guess.

Letter	Correct	Incorrect	Total	% Accuracy
A	10	0	10	100
B	10	0	10	100
C	10	0	10	100
D	10	0	10	100
E	10	0	10	100
F	10	0	10	100
G	10	0	10	100
H	10	0	10	100
I	7	3	10	70
J	9	1	10	90
K	9	1	10	90
L	10	0	10	100
M	6	4	10	60
N	5	5	10	50
O	9	1	10	90
P	9	1	10	90
Q	8	2	10	80
R	7	3	10	70

S	10	0	10	100
T	6	4	10	60
U	7	3	10	70
V	8	2	10	80
W	10	0	10	100
X	8	2	10	80
Y	10	0	10	100
Z	10	0	10	100
Neutral	10	0	10	100
Total	228		260	87.69230769

Table 13: Results of testing third iteration of the algorithm. Ten trials were performed, with an increase of 8.44%

Letter	Correct	Incorrect	Total	% Accuracy
A	12		12	100
B	12		12	100
C	12		12	100
D	12		12	100
E	12	1	13	92.30769231
F	12		12	100
G	9	3	12	75
H	12		12	100
I	12		12	100
J	12		12	100

K	11	1	12	91.66666667
L	12		12	100
M	8	4	12	66.66666667
N	5	7	12	41.66666667
O	9	3	12	75
P	12		12	100
Q	12		12	100
R	12		12	100
S	12		12	100
T	10	2	12	83.33333333
U	8	4	12	66.66666667
V	12		12	100
W	12		12	100
X	9	3	12	75
Y	12		12	100
Z	7	5	12	58.33333333
Neutral	12		12	100
Total	**280**		**313**	**89.44773176**

Table 14: Final testing results. The final system accuracy was 0.55% below our target accuracy.

Additionally, another algorithm was designed in parallel to the aforementioned algorithm. This algorithm was created with the help of Tristan Watts-Willis, a graduate student whose thesis involves work on a meta-classification algorithm. He assisted in finding features and characteristics in the sensor data that would allow the algorithm to be accurate for multiple users.

The first data sets provided to him included six letters with over 40 data points, while the second data set provided the full alphabet, but only around 15 data points for each letter. The first decision trees Tristan was able to procure from the initial data sets were over 90% accurate, but did not include most of the alphabet, and did not use motion sensor data in the characterization method. The second decision tree for the full alphabet had a 70% accuracy, so in the interest of time, the efforts were switched to getting the full alphabet characterized as accurately as possible for one person. However, future work can further explore and improve this approach.

5.2.2 Bluetooth Algorithm

The HM-11 Bluetooth modules have the capability of transmitting and receiving characters from each other. The *master* Bluetooth device is programmed to constantly be on alert for the next letter being signed by the user. Likewise, the *slave* Bluetooth device is programmed to be constantly awaiting a character being transmitted by the *master* Bluetooth module. Originally, there was no delay implemented in the algorithm when transmitting characters between the *master* and *slave* devices, which resulted in the *slave* device receiving dozens of characters simultaneously. With the algorithm now adjusted appropriately, the two modules communicate at a much steadier and understandable pace. The final result is that the *master* Bluetooth device accurately transmits the letter signed by the user to the *slave* Bluetooth device.

5.2.3 Sound File Algorithm

The sound file algorithm has the simplest function of parsing the microSD card for the appropriate sound file and playing it through a wired speaker. Based on the character received by the Bluetooth device on the speaker module, the corresponding sound file was selected. The algorithm makes use of conditional statements for each letter of the alphabet to select sound files. The files are then transmitted through the on-board DAC of the Teensy 3.2 development board to be played through the wired speaker. This algorithm is fully functional with the overall American Sign Language Interpreter system.

The main issues that was encountered for this software component was the microSD card failing to initialize on multiple early and mid testing occasions. The microSD card initialization failure is most likely attributed to the fact that the microSD card itself was reaching its end-of-life. The pins on the microSD card or some internal hardware component prevented the card itself from being recognized by the interpreter system or by any computer system for that matter. This issue was easily remedied by procuring and formatting another microSD card.

Chapter 6: Justifications for Choices of Hardware and Software

6.1 Hardware

6.1.1 Finger Position Sensor

As previously mentioned, two types of sensors are needed to fully characterize the ASL alphabet: a sensor to measure finger position or bend, and a sensor to sense hand orientation and motion. For the final design, a flexible sensor will be incorporated to determine the level of bend in each finger. Additionally, two different lengths of flexible sensors will be used to best fit each individual finger. A 2.2'' sensor will be used on the pinky and thumb, while the middle three fingers will be measured with a 4.5" sensor to ensure the most accurate reading for each individual finger.

The tests performed on the prototypes of each sensor design showed that the flexible sensor yielded a greater voltage difference between each of the four finger positions, leading to the conclusion that the flexible sensor's threshold values between finger positions will be easier to anticipate. The EMG sensor output smaller voltage differences between the outputs for the straight and bent finger positions, and thus would require further signal processing and amplification. With the sensor already showing significant noise without the use of the needed multiplexer and amplifiers, there were concerns about how much noise would be introduced to the EMG sensor's signal after implementing a multiplexer.

Additionally, the need to customize the sensor placement to the user is greater with the EMG sensor, which could lead to a greater chance of user error due to incorrect placement of electrodes. The flexible sensor, on the other hand, could be incorporated straight on the glove, requiring little user input in determining the placement of the finger position sensors and thus a lesser chance of user error.

In the end, both the EMG sensor and the flexible sensor would accomplish finger position detection. While the EMG sensor would be more durable in the long run, the limited variety of EMG sensors currently on the market for a reasonable price makes it difficult to find an EMG sensor that is both cheap, user-friendly, and high-performance. For the purpose of this project, the flexible sensors are the more viable option with their cost-effective and user-friendly nature.

6.1.2 Hand Motion and Orientation Sensor

The research on the motion sensor focused primarily on two different chips: the MPU-6050 and the MPU-9250. The chips incorporate both an accelerometer and gyroscope. However, the MPU-9250 also includes a magnetometer, which is an unnecessary feature, making the MPU-6050 the ideal choice. Another consideration for the motion sensor was whether to buy the solitary chip or choose a chip already on a board. Though the standalone chip was cheaper in most cases, a PCB would need to be designed and created because the chip is a surface mount device, and needs a shield to connect it to the microcontroller. With this additional cost and the fact that a PCB may take several weeks to arrive, the chip on a board was the more suitable choice. The final decision for the motion sensor was which sensor to choose. In the end, the GY-521 breakout board with the MPU-6050 chip was selected because it fit all of the design specifications and had the lowest price point.

6.1.3 Microcontroller

The microcontroller that we ended up choosing was the MK20DX256VLH7, which comes on the Teensy 3.2 development board. The specifications that we were looking for in our microcontroller was that it would have to be able to perform signal processing. A microcontroller that would be able to do this would need at least a 16 bit processor (32 bit prefered). The Teensy 3.2 meets these specifications with its 32 bit processor. The board also has enough pins to incorporate all of the sensors and peripherals needed for the completion of the project. The Teensy 3.2 was also the most powerful of all of the microcontrollers researched with 64kB SRAM compared to the next most powerful microcontroller which had only 16kB SRAM. Although it was the most expensive microcontroller, the cost was not significantly different from its next closest competitor, the Teensy 3.1. Also, while the Teensy 3.1 was cheaper than the TEESY 3.2, the Teensy 3.2 had far most processing power and memory than the Teensy 3.1. The Nucleo Development Board was the cheapest by far, however it would not be compatible with the Bluetooth that we had chosen, as well as the fact that it's processing power was lacking at 4kB SRAM. The Nucleo Development board did have more flash memory, but the amount of code that was going to be used was not significant enough for flash memory to be an issue. That

is why the Teensy 3.2 was our microcontroller of choice. It was the most powerful, had enough flash memory for the amount of code planned, had enough pins to run all of the necessary peripherals, and was compatible with the chosen Bluetooth.

6.1.4 Bluetooth

The original project design specified that a Bluetooth device on the glove module would transmit audio packets wirelessly to the speaker module, which consisted of just a Bluetooth speaker at the time. With the redesign of the speaker module, a new method of implementing Bluetooth was designed. This new method called for the Bluetooth to transmit only single characters at a time instead of entire audio packets. The main reason for this shift in design can be attributed to the overall complex nature of Bluetooth itself with its various profiles and protocols. Most Bluetooth modules bought for design projects, such as this American Sign Language Interpreter project, do not use the Bluetooth device as a source for transmitting audio packets but rather as a sink for receiving packets. Though it is certainly possible to configure a Bluetooth module, such as the BC127, to transmit audio, the sheer lack of documentation alone was enough to warrant a redesign of the speaker module. In addition, most documentation and sample projects regarding audio implementation required additional proprietary hardware, which would greatly inflate the overall budget. In the end, it was determined that the ability to stream audio wirelessly did not outweigh the amount of time it would take to learn complex audio protocols or the inflation of extra costs to the overall budget.

Due to this redesign of Bluetooth implementation, the module that was selected for the final product was the HM-11, which is perfect for transmitting characters. The HM-11 module were among the first Bluetooth devices that were purchased and tested for this project. From the beginning, it was verified that two HM-11 modules could easily communication with one another. From these initial testings, the team had confidence that switching to these modules would not have been an issue in the grand scheme of the project. Additionally, these modules matched both the power and the cost budget perfectly. The HM-11 modules require input voltages of 3.3 V, which the Teensy 3.2 provides. Each module is relatively inexpensive at

about $13 per device. In terms of size and weight, these modules are incredibly compact and lightweight, which does not compromise the weight specifications of the project.

6.1.5 Speaker

The Bluetooth speakers that were researched on were not considered for the final design of the project as the specifications had changed to only need a normal speaker. The speaker that we ended up choosing was the 4 Ohm 3 Watt speaker. It was light a portable, which allows for it to easily be integrated onto the speaker module. Through the testing of the other speakers that would be integratable onto the speaker module, the 4 Ohm 3 Watt speaker was the loudest and clearest. Although the Peerless speaker was louder and clearer than the 4 Ohm 3 Watt speaker, its size and weight made it impossible to integrate on the speaker module. The 0.25W speaker had a constant humming noise in the background and did not produce a loud enough sound so that it was usable in a noisy environment. The price of the 4 Ohm 3 Watt speaker was also significantly lower than the 1 Watt speaker which was the speaker that was the most comparable of the speakers that would be integratable onto the speaker module.

6.1.6 Printed Circuit Board (PCB)

We decided to design a PCB instead of using through hole components on a perfboard. The PCB design, depicted in section 5.1.6, utilizes surface mount components, which are smaller than through hole components. This allows the shield to be smaller and fit more comfortably on the hand. The sensors will be connected to the shield using easy to use connectors, such as JST connectors, so the user can easily replace failing sensors without having to replace the entire device.

6.1.7 Battery

With all the prior research and testing, the lithium ion battery was the finalized choice. Table 12 shows the electrical requirements for each hardware device to operate. 3.3V will be supplied as it falls in the operating range for all the components. The total current consumed is 55.26 mA and the total power needed is 182.36 mW.

	# of device(s)	Device(s)	Voltage Range (V)	Voltage (Typical V)	Current (mA)	Power (mW)
	1	Motion sensor- accelerometer/ gyroscope	2.375-3.46	3.3	3.9	12.87
	1	Microcontroller Teensy 3.2	1.71-3.6	3.3	34	112.2
	5	Flex Sensor	–	3.3	2.36	7.788
	1	Bluetooth BLE HM-11	-0.3-3.9	3.3	15	49.5

Table 15. Hardware Component Electrical Specifications

Using the voltage and current rating of each components, a cell type was selected. The lithium ion cell is the most feasible option for this project. Referring to the project's specifications of weight limit of 125g, price budget of $200, and comfortable, ergonomic glove. The lithium ion battery has a higher energy density compared to other rechargeable batteries considering the size and weight of these types of batteries. The weight of each cell is 4.56g with dimensions of 19.75mm x 26.02mm x 3.8mm, which is about the size of a quarter. To supply the microcontroller and sensors, two 3.7V 150mAh cells will be wired in series to generate an input voltage of approximately 7.4V. This input voltage is higher than the voltage that will be supplied to the components because there needs to be clearance as the battery will be dissipating power. However, the voltage is still too high to be supplied to the loads.. With the Teensy 3.2, a voltage regulator will need to be added separately.

After the research was verified, testing was done on the voltage regulator using the LM317K. Different voltage inputs were examined with the voltage regulator: 5V, 6V, and 8V. The results

Another detail about the lithium ion battery is its capability to be recharged. The selected power source has about 500 cycles worth or charge life. Also, the device will last about 2.7 hours to minimize the number of charges. The lithium ion battery satisfies the project specifications.

6.1.8 Amplifier

The amplifier that we ended up deciding upon was the mini Hi-Fi 12V Mini Hi-Fi PAM8610 2X10W Audio Stereo Amplifier Board Dual Channel. This audio amplifier was by far the most cost effective amplifier. While the TPA2100P1 NanoFree™ 1-Channel (Mono) Output Class D Audio Amplifier Evaluation Board was far more power budget friendly (2.5V - 5.5V compared to 12V), it costs nearly ten times more than the mini Hi-Fi audio amplifier. Our own design for the audio amplifier would have been optimal for size as it would sit on the PCB along with the microcontroller and Bluetooth module, however through the testing of our design, there was a constant humming noise in the output of the speakers. Also, at this point of the semester, the PCB designs were already finalized and ordered, so using this design would have needed more time, effort, and budget than we were willing to give. The Geree Audio amplifier was more expensive than the final choice as well as far less power budget friendly, with a max voltage supply of 32V. The 12V mini Hi Fi audio amplifier is small, light, as well as inexpensive, making it our choice for the audio amplifier for the final design. Through the testing, the audio amplifier allowed our chosen speaker to produce a loud and clear sound from the DAC output of the microcontroller.

6.2 Software

6.2.1 Arduino IDE

Initially, it was decided that all of the algorithms for the American Sign Language Interpreter project be written in Arduino for the sole purpose of initial testing. However, due to the time constraint, all algorithms will continue to use Arduino libraries. Arduino, in essence, uses C programming functions, which means all standard C constructs should work perfectly together with the Arduino libraries.. In the end, the benefits in efficiency by not utilizing

Arduino libraries did not justify the cost in the amount of time it would take to undergo such a procedure of conversion.

6.2.2 Sign-to-Letter Algorithm

The low accuracy of the initial algorithm emphasized how much small changes in the way people sign affect the way each hand sign can be classified. Additionally, it is difficult for a single user to consistently sign the same way every time they sign a letter. Thus, two different methods for characterizing the sensor signals were developed in parallel.

For the interests of time and the completion of this project, getting the algorithm working as accurately as possible became a priority over improving device accuracy across a spectrum of users. The chosen method was then optimized by incorporating more data variable values from the sensor network and by adjusting threshold values to account for a greater range of error from inconsistent signing.

Chapter 7: Results of Fully-Assembled System:

The fully-assembled system integrates all the hardware and software components, as specified and explained in the previous chapters, into the two main components of the project, the glove module and the speaker module. After incorporating components beginning with the flexible and motion sensors and ending with the amplifier and speaker, the entire American Sign Language Interpreter system operates as specified in the project requirements.

One of the major differences in terms of testing the hardware components individually and as a whole is the power source. Throughout the testing phase, the glove and shirt modules are powered through the USB connection on the Teensy 3.2 development board. In order to test the full functionality of the interpreter system, the project needs to run entirely off of the lithium-ion batteries. The two fully charged batteries located on the glove module have proven that they are capable of supplying the appropriate voltage without any issues. Likewise, the five batteries located on the speaker module can successfully power 12V required by the audio amplifier as well as the 3.3V required by the remaining module components due to the voltage regulator.

While running full-system tests using the USB connection on the Teensy 3.2 development board as the main source of power, all components, excluding the lithium-ion batteries, operated successfully. In order to individually test different aspects of the project and to program the controllers, the batteries were not connected to the full system once the design was soldered on the PCBs. Debugging was performed via the Serial Monitor provided by the Arduino IDE, which outputs sensor data, wireless communication information, and microSD card successes and failures.

After power is supplied to both modules of the interpreter system, the next component that was tested was the flexible and motion sensors. Based on the results as displayed on the Serial Monitor, all sensors are returning reasonable data values. On the software side, the sign-to-letter algorithm compiles and interprets each of the data values from the sensors and characterizes them into an alphabet letter. After hours of dedication and arduous work from the project team members, 89.45% of the signed letters can be recognized by the interpreter system.

Once the sign-to-letter algorithm detects an alphabet letter from the raw sensor data, the character is sent via UART to the transmitting HM-11 Bluetooth module on the glove subsystem. The Bluetooth algorithm runs prior to any other code in the interpreter system, which attempts to detect an active Bluetooth device on the glove module. From all of the full-system tests that the team has done, the Bluetooth module is fully detectable at the 115200 baud rate and pairs successfully with the Bluetooth device located on the speaker module. On one occasion, the Bluetooth devices did not pair correctly, which can be attributed to the invalid MAC address of the receiving Bluetooth that is hardcoded into the Bluetooth algorithm. After correcting this issue, the Bluetooth devices pair correctly. In terms of actual wireless communication of characters between the glove and speaker subsystems, the two can transmit and receive characters accurately. The one caveat of the wireless communication component is that the Bluetooth devices will occasionally and randomly disconnect and reconnect with each other, which disrupt the entire system as a whole.

After the transmission of the character from the Bluetooth module located on the glove is completed, the focus shifts to the speaker module. The speaker module has its own algorithm that it implements with the Bluetooth algorithm implementing first, which operates similarly to the Bluetooth algorithm on the glove subsystem. From the full-system tests, the receiving Bluetooth is capable of receiving characters at a steady and understandable pace. Afterwards, the sound file detection algorithm initiates, parses through the microSD card for a corresponding sound file, and sends the file through the DAC. Due to the utilization of basic conditional statements, the algorithm functions properly 100% of the time based on the full-system tests performed. Occasionally, the system fails due to an initialization error caused by the microSD card. The cause of this failure is most likely attributed to a faulty microSD card approaching its end-of-life and can be easily remedied by replacing it.

Finally, the sound file reaches the amplifier and outputs through the speaker. Many wired speakers and amplifiers had been considered for the speaker module. However, by the time of the full-system test, the appropriate speaker and audio amplifier were selected. With these two components combined, the audio files can be easily heard and understood by everybody in the vicinity.

7.1 Dollar Budget

The allotted dollar budget for the project was $200. Ultimately, the final cost of the deliverable was $218.76, which is $18.76 over-budget. The reason for the miscalculations in budget were due to the design change midway through the semester that abandoned the Bluetooth-capable speaker in favor of designing a separate speaker circuit. This design alteration was due to the inability to properly implement I2S to communicate with the Bluetooth module. Thus, a third PCB was designed to allow the modules to transmit a letter instead of a sound file, and additional components needed for this circuit were purchased.

Item Type	Model	Price	Quantity	Total Price	Seller
Flex Sensor	4”	$12.95	3	$38.85	SparkFun
Flex Sensor	2.2”	$7.95	2	$15.90	SparkFun
Microcontroller	Teensy 3.2	$19.80	2	$39.60	PJRC Store
Bluetooth	HM-11	$12.90	2	$25.80	Digikey
Battery	Lithium Ion Polymer Battery - 3.7v 150mAh	$5.95	7	$41.65	Adafruit
MicroSD Card Shield	SparkFun microSD Transflash Breakout	$3.95	1	$3.95	SparkFun
Glove	All Purpose Glove	$10	1	$10	Home Depot
Speaker	4 Ohm 3 Watt	$1.95	1	-	Adafruit/Inventory
Shirt	100% cotton shirt	-	1	-	Liane
Surface Mount Resistor	Thick Film Resistors - SMD 1/8watt 390ohms 1%	$0.01	1	$0.01	Digikey
Surface Mount Resistor	240 OHM 1/8W 1% 0805 SMD	$0.01	1	$0.01	Digikey
Surface Mount	20K OHM 1/8W 1%	$0.01	5	$0.05	Digikey

Resistor	0805 SMD				
Surface Mount Capacitor	220µF ±20% 4V Ceramic Capacitor X5R 1206	$1.32	1	$1.32	Digikey
Surface Mount Op Amp	General Purpose Amplifier 2 Circuit 8-SOIC	$0.47	3	$1.41	Digikey
Voltage Regulator	TC1262 Voltage Regulator	$0.53	1	$0.53	Digikey
Voltage Regulator	LM317 3 Terminal Adjustable Regulator	$0.68	1	$0.68	Digikey
JST Connectors	JST-PH 2-Pin SMT Right Angle Connector	$0.75	14	$10.50	Adafruit
Motion Sensor	MPU-6050	$3.50	1	$3.50	Amazon
MicroSD Card	1 GB MicroSD Card	-	1	-	Inventory
PCB	PCB for flex sensors, microcontroller, and motion sensor	$6.56	1	$6.56	Osh Park
PCB	PCB for bluetooth transmitter	$2.34	1	$2.34	Osh Park
PCB	PCB for speaker module	$9.17	1	$9.17	Osh Park
Amplifier	2V Mini Hi-Fi PAM8610 2X10W Audio Stereo Amplifier Board Dual Channel	$5.99	1	$5.99	BangGood
Total				$219.77	

Table 16. Full System Component Prices

7.2 Power Budget

	# of device(s)	Device(s)	Voltage Range (V)	Voltage (Typical V)	Current (mA)	Power (mW)
	1	Motion sensor- accelerometer/ gyroscope	2.375-3.46	3.3	3.9	12.87
	1	Microcontroller Teensy 3.2	1.71-3.6	3.3	34	112.2
	5	Flex Sensor	–	3.3	2.36	7.788
	1	Bluetooth BLE HM-11	-0.3-3.9	3.3	15	49.5
					Total Current (mA):	55.26
					Total Power needed (mW):	182.358
					Battery Life (hours):	2.74

Table 17. Glove Module Devices Electrical Specifications

	# of device(s)	Device(s)	Voltage Range (V)	Voltage (Typical V)	Current (mA)	Power (mW)
	1	Speaker	–	12	20	240
	1	Microcontroller Teensy 3.2	1.71-3.6	3.3	34	112.2
	1	SparkFun microSD Transflash Breakout	–	3.3	100	330
	1	Bluetooth BLE HM-11	-0.3-3.9	3.3	15	49.5
					Total Current (mA):	169
					Total Power needed (mW):	731.7
					Battery Life (hours):	0.89

Table 18. Speaker Module Devices Electrical Specifications

Tables 17 and 18 show each of the hardware component electrical specifications which includes the voltage range, operating voltage, operating current, and power consumption. The total power consumed in our system is about 584 mW. Using the 3.7V 150mAh batteries, the glove module should have an operation time of about 2.7 hours. For the speaker module, operation should last for 53 minutes before another recharge.

Chapter 8: Conclusions

8.1 System Performance

The final deliverable consists of a fully functional system that can correctly identify a signed letter nearly 90% of the time, just under the target system accuracy. While the system is accurate when used by one person, the system accuracy diminishes when used across a wider selection of users. The algorithm developed with the help of Tristan Watts-Willis had promising capability to solve this issue. However, due to manpower and time limitations, this algorithm was not able to be implemented in the final device. All aspects of the system are functional to a large extent. Further debugging time is required to determine why this occurs. The algorithm to choose the correct sound file correctly sends the appropriate sound file that corresponds to the letter chosen by the signal characterization algorithm to the speaker, which outputs the sound at an appropriate volume.

8.2 Team Dynamic

Overall, the team worked together well and had a positive work dynamic. The work was split evenly among the members and the members all did their part in the completion of the project in a timely fashion. In addition to completing each individually assigned tasks, team members also consistently assisted other team members when needed. Though disagreements occurred throughout the semester, conflict resolution occurred in a timely and professional manner. When team members finished their tasks, they generally assisted in other areas in order to maintain productivity.

One drawback this semester was that only two of the four personal laptops had the capability to compile and run the code for the Teesy controllers. This lead to inconveniences to both the team members Bryan and Ryan, who had capable laptops, and Liane and Taryn, who were unable to program the Teensy directly. Those with the capable laptops had to yield their computers so that the team members working on the signal characterization algorithm could test and optimize the threshold values to characterize the sensor signals, which inconvenienced both parties. Either Bryan or Ryan had to give up their laptops for some time, which mean less time for them to work on their computer. Meanwhile, Liane and Taryn had to wait for either Bryan or

Ryan to be able to lend their laptops for testing, which led to uncertainties of when they would be able to test, and meant they often had to run tests late at night.

Chapter 9: Applications in Real World, Social, Environmental, and Economical Impacts

9.1 Applications in Real World

The American Sign Language Interpreter is an effort to bridge the communication gap between individuals with the inability to communicate verbally and the general population that lacks knowledge of American Sign Language. When integrated into a conversation between a mute person and an individual not fluent in American Sign Language, this design will encourage communication to progress at a natural pace and mimic an ordinary conversation with the integration of audible words output through the speaker.

Some previous attempts by other parties to create a similar Sign Language Interpreter had design features that prevented these previous efforts from being accessible to the maximum number of people possible. One previous design output the translation in text-form. Though this feature is effective in communicating with many people, it excludes illiterate people, including young children who have not learned to read yet. With the incorporation of sound, the value of the interpreter then becomes available to the general population that can hear and that does not have knowledge of American Sign Language. The use of sound allows for the inclusivity of all who can benefit from the interpreter.

Though there have been past efforts to create a Sign Language Interpreter, these designs also had issues with portability, cost, or accuracy. The lightweight, portable design of our American Sign Language Interpreter increases mobility for the user, and the lack of a necessary laptop to aid in the processing and computing allows the user to move freely. Additionally, the use of cost-effective parts allows the device to be affordable to a wider population. By taking in these considerations and choosing options that encouraged the greatest accuracy, the American Sign Language Interpreter aims to be the best wearable American Sign Language translator on the market.

The American Sign Language Interpreter has several social implications. By translating a lesser-known form of communication, the Sign Language Interpreter will make communicating to the general population easier for a person who cannot speak. Mobility was a major design consideration. Previous attempts at creating a similar sign language interpreter yielded designs that relied on laptops to perform signal processing and analysis or bulky parts that limited

mobility. By creating a lightweight unit that fits completely on the user's body, the American Sign Language Interpreter allows the user to move freely and bring the translator with them to perform everyday activities.

9.2 Social Impacts

The American Sign Language Interpreter helps connect people that previously would have difficulties communicating. Many people that do not grow up with the need for American Sign Language do not learn enough of it in their lifetime to be able to hold a conversation. This provides limitations in communication when they find themselves in a situation where someone without the ability to speak is trying to communicate to them. On the other end, the people who require the use of American Sign Language to communicate with those around them are at a minority in the general society. This demographic often is faced with having to communicate with people who do not understand them. By creating an easier way to create natural conversation, the interpreter allows people with the inability to speak to be able to communicate their thoughts and ideas more effectively to those around them without knowledge of American Sign Language.

Encouraging natural discourse between two previously isolated populations allows for the exchange of diverse ideas. People that have limitations in their form of communication have a different perspective of the world than people without these constraints. In addition, people with handicaps often have innovative and original ways to solve problems. Being able to exchange these different perspectives on the world benefits both groups and allows everyone to grow and learn upon both past failures and successes.

9.3 Environmental Impacts

The American Sign Language Interpreter also has environmental implications. One important aspect of the design was longevity and reusability. Choosing a reusable battery allows the user to produce less waste by not having to discard used batteries. The sensors were chosen with lifespan in mind. By incorporating sensors that can be used many times, the lifespan of the device increases and requires fewer replacement parts. Connecting the sensors to the device

such that they can be easily replaced also allows the majority of the interpreter device that has a longer lifespan than the sensor to be reused with replacement sensors. The creation of any electronic device is going to have a negative impact on the environment once the lifespan of the device runs out. However, by increasing the time between when parts need to be replaced, fewer parts will be discarded over time.

9.4 Economic Impacts

One major design aspect considered during the development of the American Sign Language Interpreter was the cost. As a result of this design aspect, one of the goals of this project is to produce a cost-effective option to increase affordability to a greater population of people. The initial design involved a lot of careful decision making in terms of the parts for the American Sign Language Interpreter. These parts included the sensors, microcontrollers, batteries, Bluetooth modules, and speaker. The main criteria that was considered for these parts was that they needed to offer incredible performance but had to be cost-effective as well. The parts that were chosen for this project meet this criteria and their performance is reflected in the final product.

The creation of an economical Sign Language Interpreter has the potential to further impact the economy. This design has marketability, and could easily be competitive with any current sign language translator. By introducing this design to the global consumer, the American Sign Language Interpreter can stimulate the economy and create a niche market for affordable, easy-to-use translator devices. The advent of this new market can also encourage further advancements in the development of translation devices for a larger variety of languages, both signed and spoken, and potentially lead to the development of a universal language translator.

Chapter 10: Lessons Learned and Future Improvements to the Project

10.1 Lessons Learned

The American Sign Language Interpreter project is a complex and comprehensive challenge that covers a wide variety of disciplines and techniques. This project incorporated topics from several past courses, such as power, signal processing, communication, microcontrollers, circuit design, and programming. In addition to these disciplines, there were certain topics that the team was not familiar with, such as machine learning and Bluetooth protocols.

An important lesson learned from this senior project was the importance of maintaining concise and coherent documentation. On multiple occasions, there were issues, questions, or concerns about certain project components that had been researched and tested. However, due to the lack of proper documentation or communication, the team made its best logical assumptions on the problem at hand. Due to incomplete documentation of past work done by some team members, unnecessary repeated researching and testing occurred that could have been avoided with coherent documentation from the team members. As the semester progressed, however, documentation among team members became more thorough, issues could be handled more efficiently, and team productivity increased overall.

Another lesson that was learned during the project was the importance of sharing important information, such as datasheets used, links used, progress done, and so forth. The consistent sharing of information could have encouraged a faster and more efficient integration of different components of the project. There were occasions when one or two group members had to halt progress on testing and development because the actual group member with the relevant information was preoccupied or not available to explain the information needed. A method that could have been implemented to resolve this issue was to have a centralized file system in which all documents and updates could be posted. Although there was a shared Google Drive used among the group, documents, schematics, and programming updates were not regularly uploaded but rather kept on personal computers. The team members learned that for convenience and efficiency, all relevant documents and information should be readily available to all members of the project team. In the case of this team, the software and accompanying

system algorithms needed to test the project were only available on two computers, which slowed the overall progress of the project. The team has acknowledged its shortcomings and have taken note of them for future collaborative projects.

10.2 Future Improvements to the Project

The current deliverable has the potential for future improvements to the system if supplementary time and resources were available. One improvement is the incorporation of a power switch for the glove module. The project deliverable currently does not have a power switch and to turn the system on and off requires one to plug and unplug the batteries from the JST connectors. This process is undesirable from a user standpoint and could be frustrating while using the device, since turning the device on and off by unplugging each of the batteries is impractical.

Another improvement that would be made is an upgrade of the glove itself. The glove that is being used right now allowed a variety of hand sizes to fit snuggly to allow for the most consistent readings. However, this glove, as noted by several of the testers in a focus group geared towards device usability, is not very easy to sign with. A more flexible and lightweight glove would be chosen and developed if there was more time to work on the project.

Additionally, another improvement that would be made would be to continue to improve the accuracy of the algorithm to as close to 100% as possible. This may take a long time as there are several different types of algorithms and methods that could be used or combined to attempt to increase accuracy. The improvements to the glove that would be made if there were extra time and resources given to the group are for a more user friendly experience, as well as to make testing less painful.

The signal characterization algorithm should also be improved to be accurate across multiple users. Currently, the algorithm is very accurate for one specific user, but diminishes in its ability to correctly characterize the sensor data from others, due to the complex nature of sign language and the inconsistencies in how signs vary from person to person. This can be resolved either by allowing each user to train the device to create their threshold values to be accurate to

the way they sign, or by developing and integrating the algorithm using meta-classification that was partially explored with the help of Tristan Watts-Willis.

Another, more distant future development is to integrate an app into the design to act as the speaker module. Since nearly everyone these days has a smartphone or smart device, developing an app to handle playing the sound files would be more convenient and practical. Additionally, using a phone in lieu of the speaker component would cut the number of microcontrollers and parts that are used by the speaker module designed for the shirt, and would ultimately bring down the cost of the project.

References

1. Semiconductor, Inc. Freescale. *K20 Sub-Family* (n.d.): n. pag. Web. <http://www.pjrc.com/teensy/K20P64M72SF1.pdf>.
2. "A Sample Datasheet." *Embedded Systems Design Using the TI MSP430 Series* (2003): 229-76. Web. <http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_datasheet.pdf>.
3. Slas368G, Texas Instruments Incorporated. *MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller (Rev. G)* (n.d.): n. pag. Web. <http://www.ti.com/lit/ds/symlink/msp430f1611.pdf>.
4. "Personal Data Sheet." *PsycTESTS Dataset* (n.d.): n. pag. Web. <https://people.ece.cornell.edu/land/courses/ece4760/PIC32/Microchip_stuff/2xx_datasheet.pdf>.
5. Stmicroelectronics. (n.d.): n. pag. Web. <http://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf>.
6. "NUCLEO-F303K8." *STMicroelectronics*. N.p., n.d. Web. 01 Nov. 2016. <http://www.digikey.com/product-detail/en/NUCLEO-F303K8/497-15981-ND/5428805?WT.mc_id=IQ_7595_G_pla5428805&wt.srch=1&wt.medium=cpc&WT.srch=1&gclid=CN6Zs-2KhNACFZKGfgodqe8GFQ>.
7. "Teensy USB Development Board." *Teensy USB Development Board*. N.p., n.d. Web. 01 Nov. 2016. <https://www.pjrc.com/teensy/index.html>.
8. "Teensy 3.6 Pins." *Teensy and Teensy++ Pinouts, for C Language and Arduino Software*. N.p., n.d. Web. 01 Nov. 2016. <https://www.pjrc.com/teensy/pinout.html>.
9. K20P64M50Sf0, Document Number:, and Rev. 4. 5/2012. *Document Number: K20P64M50SF0* (n.d.): n. pag. Web. <https://www.pjrc.com/teensy/K20P64M50SF0.pdf>.

10. "GCC - Open Source Compiler for MSP Microcontrollers." - *MSP430-GCC-OPENSOURCE*. N.p., n.d. Web. 01 Nov. 2016. <http://www.ti.com/tool/msp430-gcc-opensource>.
11. <http://ww1.microchip.com/downloads/en/DeviceDoc/50001686J.pdf>.
12. "MPLAB® XC Compilers." *MPLAB- XC Compilers*. N.p., n.d. Web. 01 Nov. 2016. <http://www.microchip.com/mplab/compilers>.
13. "What Should I Use to Develop on STM32 ?" *What Should I Use to Develop on STM32 ?* N.p., n.d. Web. 01 Nov. 2016. <http://www.emcu.it/STM32/What_should_I_use_to_develop_on_STM32/What_should_I_use_to_develop_on_STM32.html>.
14. "BlueDuino Rev2." - *Wiki for April Brother*. N.p., n.d. Web. 01 Nov. 2016. <http://wiki.aprbrother.com/wiki/BlueDuino_rev2>.
15. N.p., n.d. Web. <http://www.robotshop.com/en/blueduino-rev2-arduino-compatible-microcontroller.html?gclid=CPrU9YzBhtACFYJYfgodQOsOEQ>.
16. "Lithium Ion Polymer Battery - 3.7v 150mAh." *Adafruit Industries Blog RSS*. N.p., n.d. Web. 5 Nov. 2016.
17. "LM317T Datasheet(2/25 Pages) NSC | 3-Terminal Adjustable Regulator." *LM317T Datasheet(2/25 Pages) NSC | 3-Terminal Adjustable Regulator*. N.p., n.d. Web. 5 Nov. 2016.
18. Sabatini, Matthew. "Lithium Ion vs. Lithium Polymer - What's the Difference?" *Android Authority*. N.p., 27 Oct. 2013. Web. 1 Nov. 2016.
19. Nguyen, My. "Advantages And Limitations Of Wearable Battery Types." *Wearable Technologies*. N.p., 15 Oct. 2015. Web. 15 Oct. 2016.
20. PaulStoffregen. "PaulStoffregen/Audio." *GitHub*. N.p., n.d. Web. 25 Apr. 2017.

Appendices

Appendix A: User's Manual

Section 1. Glove Module Component Overview

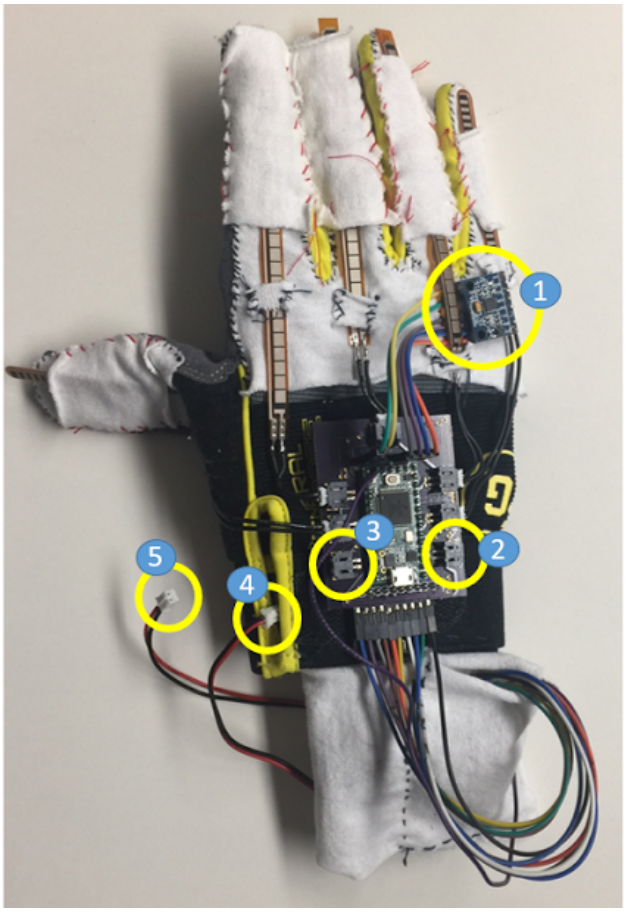


Figure A1. Glove Module

1. ON LED location

2. - 3. Where to connect battery

4. - 5. Battery connector

Section 2. Speaker Module Component Overview

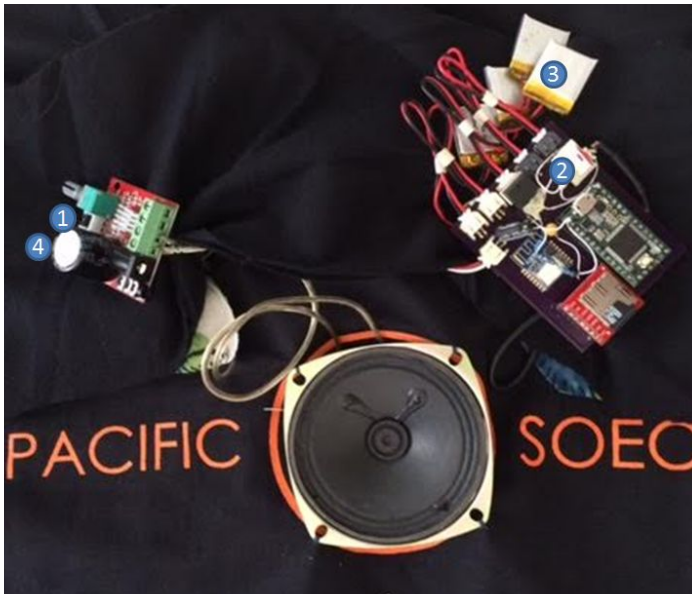


Figure A2. Speaker Module

1. Volume Control
2. Power Switch
3. Batteries
4. ON LED location

Section 3. Powering On/Off Glove

To turn on the glove module, connect the batteries to the open connections.

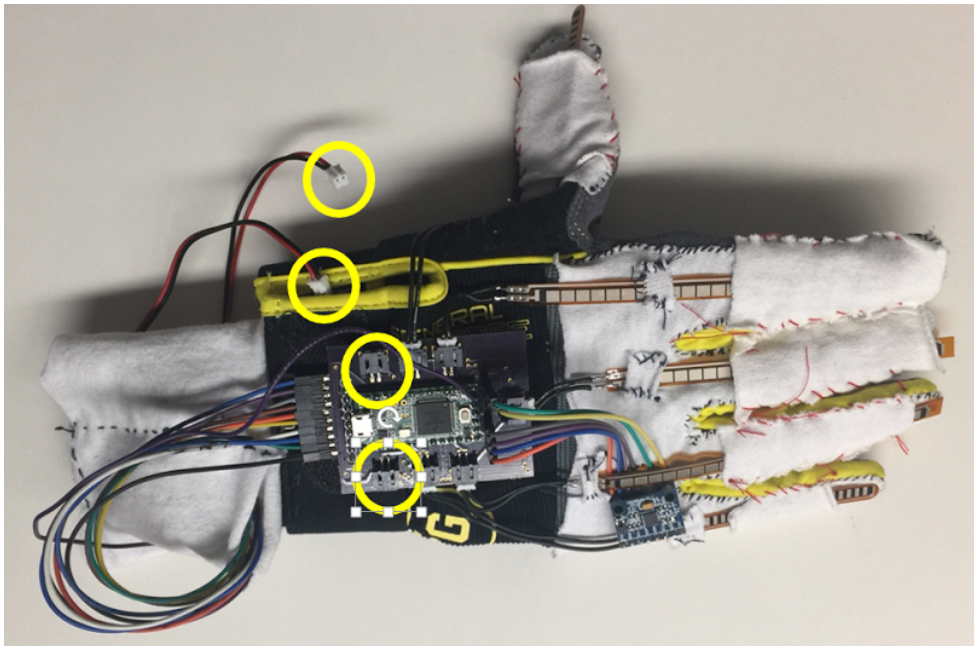


Figure A3. Unconnected Connectors on Glove

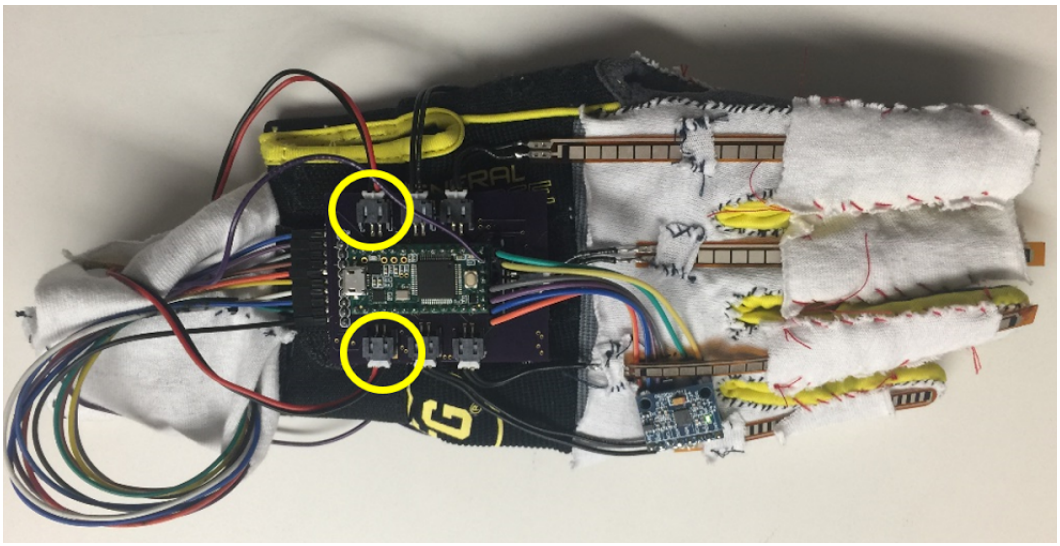


Figure A4. Connected Connectors on Glove

Once powered on, the a light on the glove should glow green as shown in Figure A6.

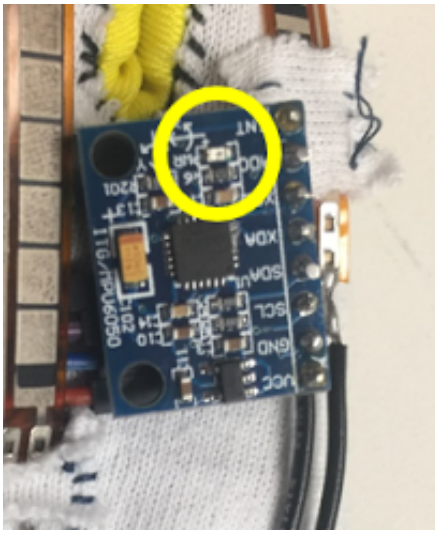


Figure A5. Powered off Glove

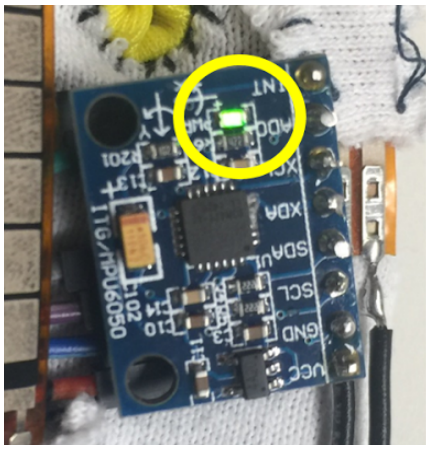


Figure A6. Powered on Glove

To power off the glove module, disconnect the batteries from their connections on the board shown in Figure A3. Once the glove is powered off, the LED on them should not be glowing anymore as shown in Figure A5.

Section 4. Powering On/Off Speaker

To turn on the speaker module, first make sure that all batteries are connected to the board by checking for open connectors. Then hit the power switch on the side of the board.

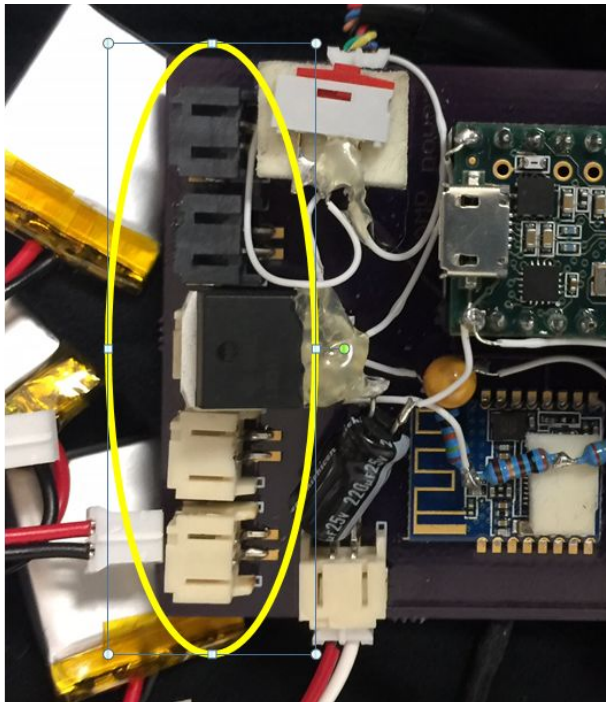


Figure A7. Unconnected Connectors on Speaker

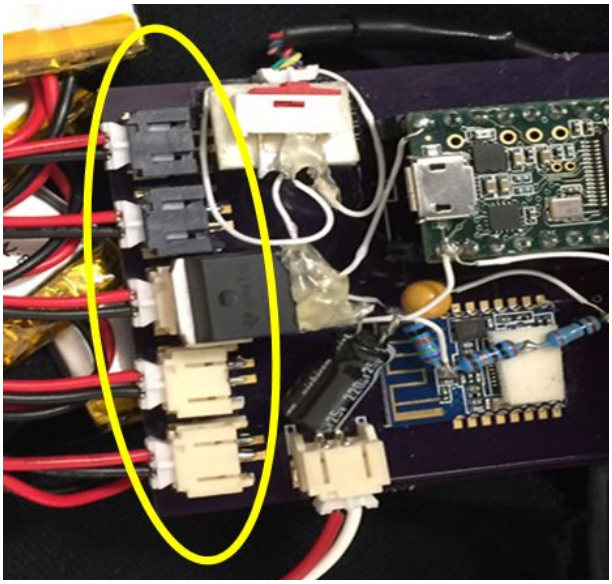


Figure A8. Connected Connectors on Speaker

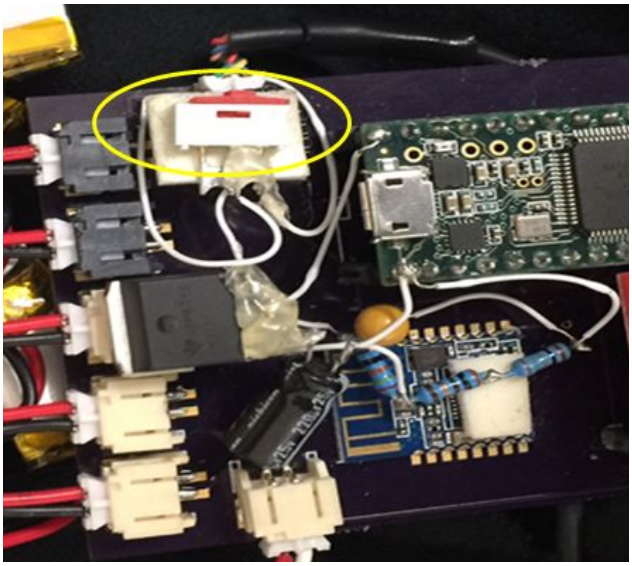


Figure A9. Power switch

Once powered on, the a light on the speaker should glow red as shown in Figure A11.

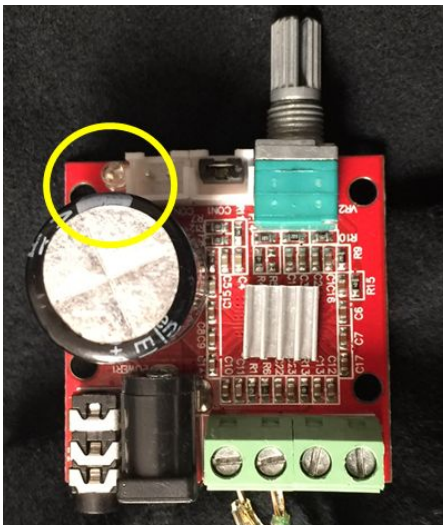


Figure A10. Powered off Speaker

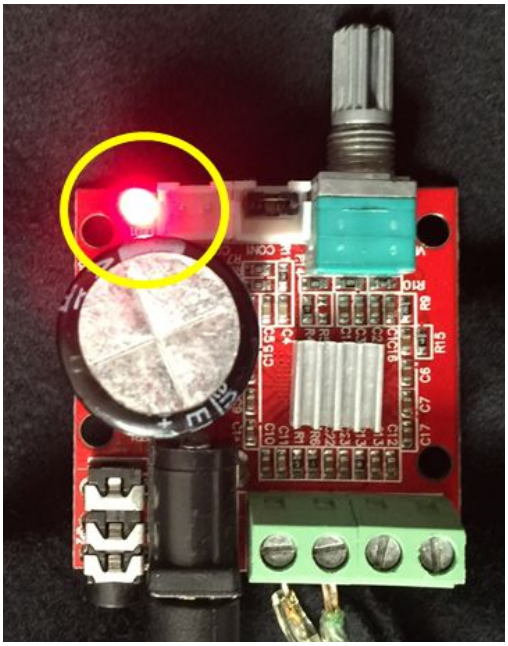


Figure A11. Powered on Speaker

To power off the speaker module, turn the power switch to the off position. Once the glove is powered off, the LED on them should not be glowing anymore as shown in Figure A10.

Section 5. Speaker

To adjust the volume of the speaker, turn the volume control knob on the side of the speaker module. Turn the knob clockwise to turn volume up and turn counter clockwise to turn volume down.

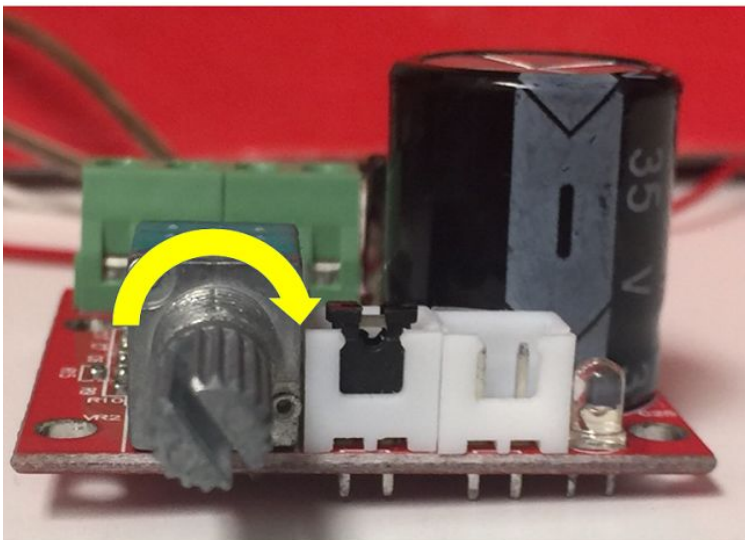


Figure A12. Raise Volume

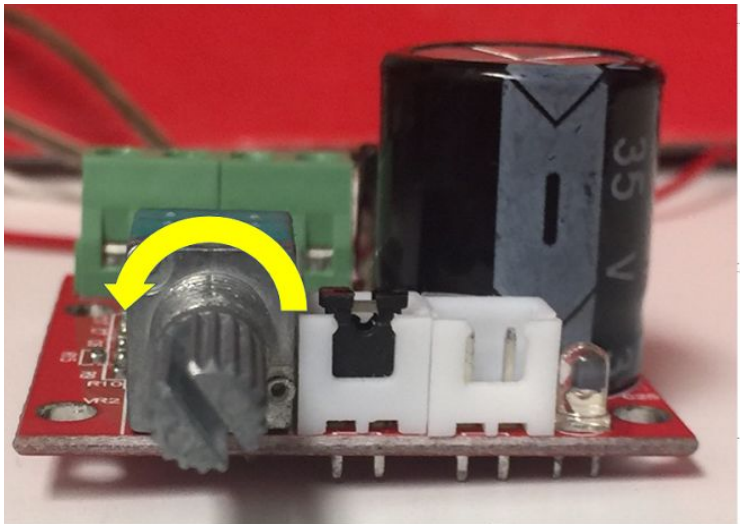


Figure A13. Lower Volume

Section 6. Wearing System

The glove is to be put on your right hand and the speaker will be on your shirt as shown below in Figure A14.

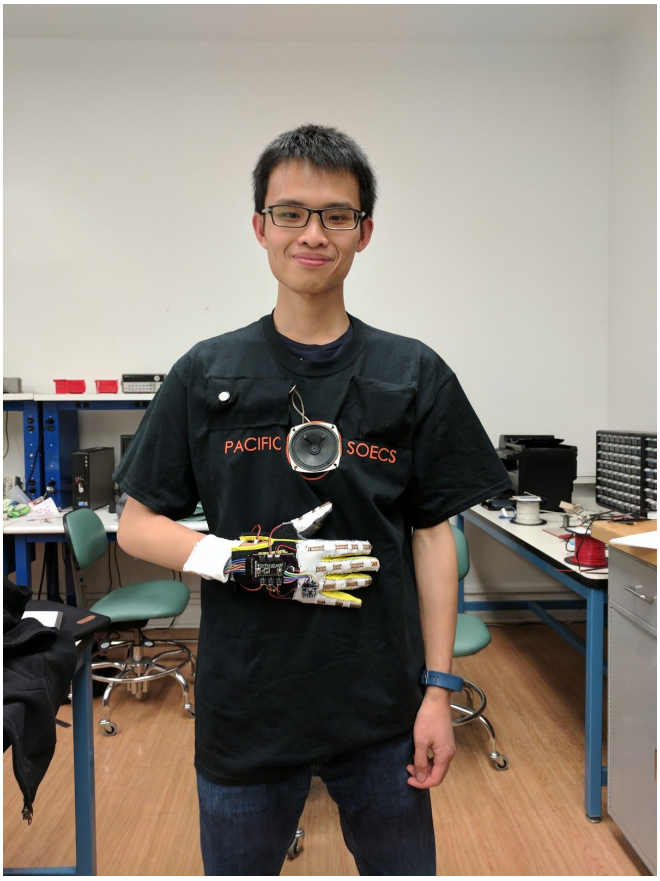


Figure A14. How to wear full system

Once both components are powered on, shown in sections 3 and 4, the Sign Language Interpreter is now ready to begin interpreting the American Sign Language Alphabet.

Appendix B: Installation Procedure

The Installation Procedures section will describe the steps needed to solder all of the components onto the PCB. There will be two sections described: the glove module and the speaker module.

Glove Module:

1. **Teensy 3.2 Development Board**

The glove module uses a majority of the available through-hole connectors. These pins include the following:

1. GND (above digital pin 0): ground
2. Vin (across from GND): voltage from regulator
3. 3.3V (above digital pin 23): 3.3V power supply from Teensy
4. Pin A0: ring finger flexible sensor
5. Pin A1: thumb flexible sensor
6. Pin A2: index finger flexible sensor
7. Pin A3: middle finger flexible sensor
8. Pin A4: motion sensor SDA
9. Pin A5: motion sensor SCL
10. Pin A6: pinky finger flexible sensor
11. Digital Pin 2: motion sensor INT
12. Digital Pin 7: TX of the Bluetooth module; RX of the Teensy
13. Digital Pin 8: RX of the Bluetooth module; TX of the Teensy

The Teensy uses a combination of eight and six pin straight male headers to interface with the PCB. Because the straight male headers are only available as six and eight pins, a few of the header pins may need to be clipped. Please refer to Figure B1 for a sample image of how the connectors should be implemented.

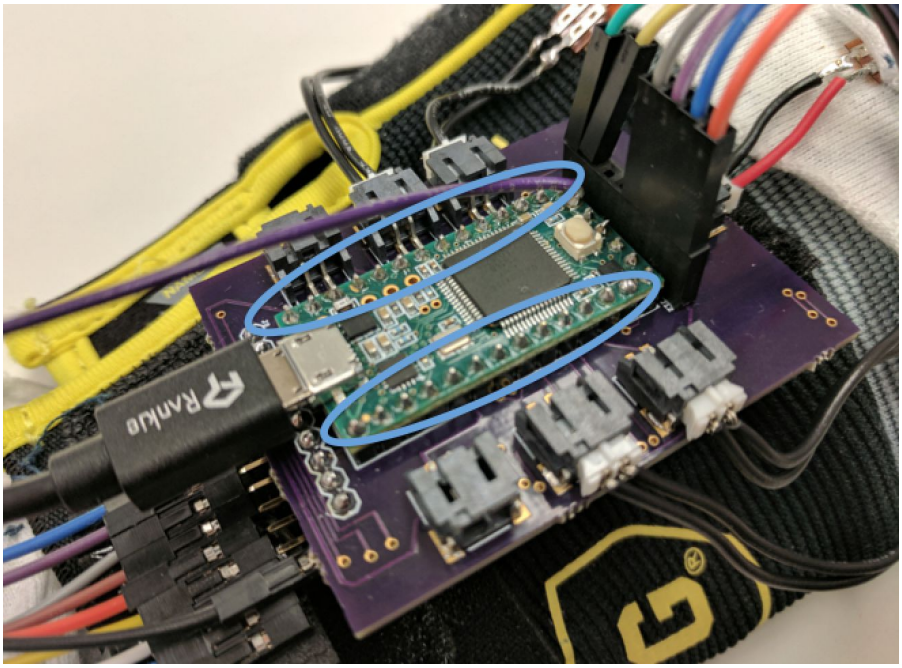


Figure B1. Straight Male Headers Attached to Glove Module PCB

2. **Flexible Sensors**

The flexible sensors each have two pins, which need to have the appropriate wires solder to them. The opposite ends of the wires need to connect into the 2-Pin JST Male Connectors. These male connectors will plug directly into the surface mount female connectors that should be already soldered onto the PCB. Refer to Figure B2 for more details.

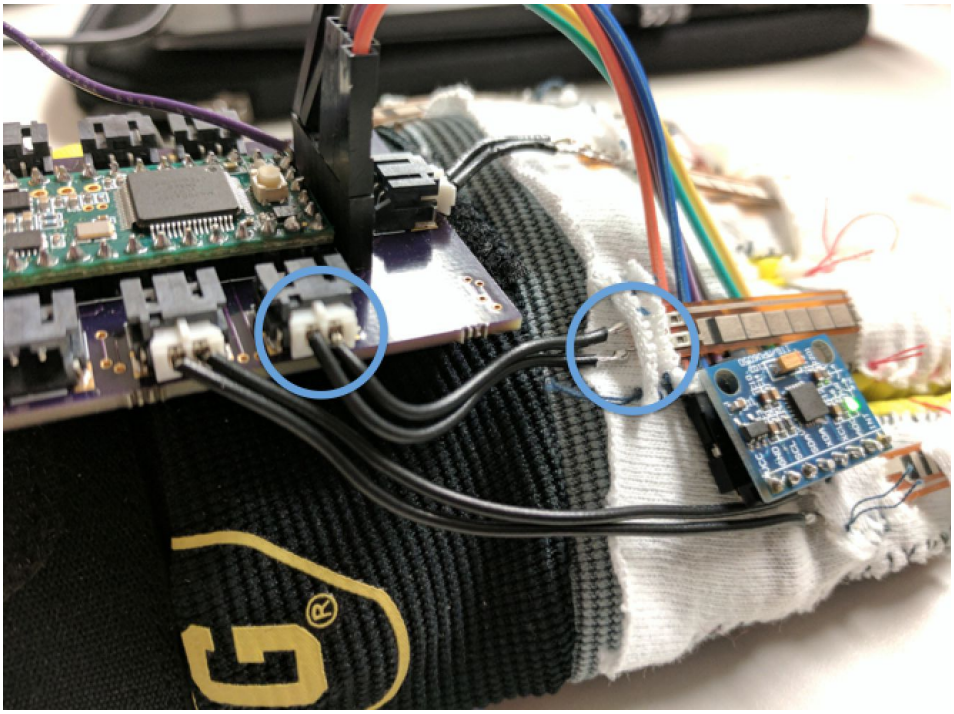


Figure B2. Flexible Sensor Connectors

3. **Motion Sensor**

For this project, six out of the eight pins of the motion sensor are used. These pins include the following:

1. VCC
2. GND
3. SCL
4. SDA
5. ADC
6. INT

The motion sensor should have an eight pin right angle header already solder onto the device. Eight female-to-male wires will be needed to interface with the PCB. The PCB should already have an eight pin female header soldered. The male end of the female-to-male wire will be connected to the eight pin female header on the PCB as shown in Figure B3.

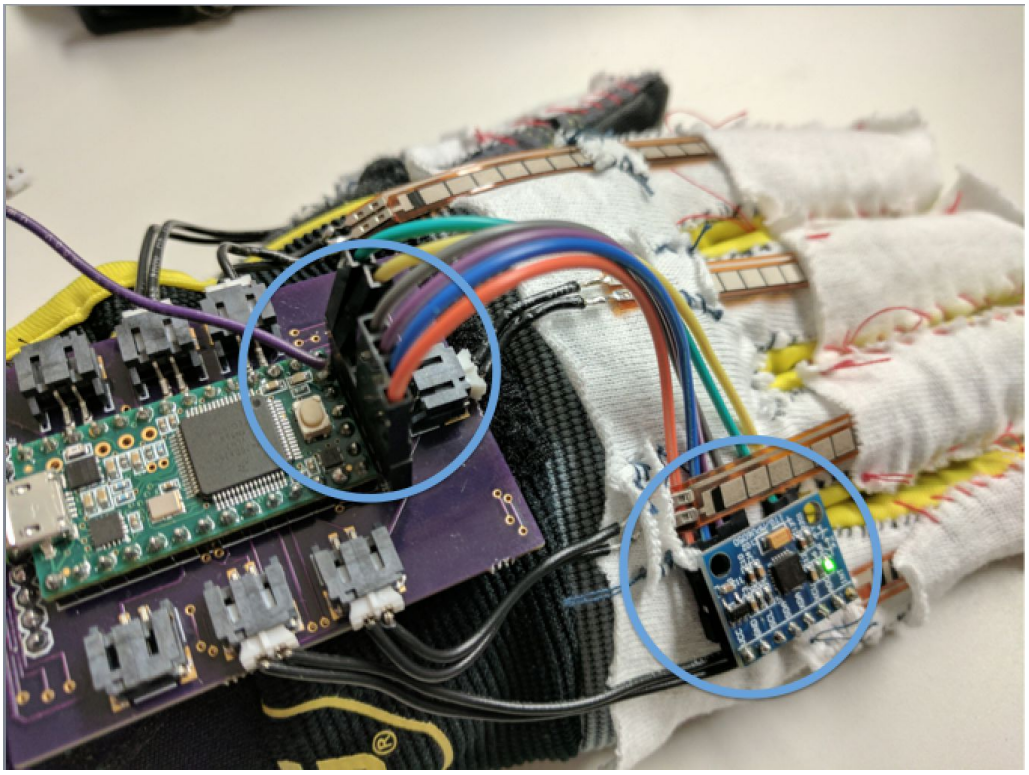


Figure B3. Motion Sensor Connectors

4. **Bluetooth Module**

The PCB design on both the glove and the speaker modules utilizes only a few of the surface mount pins of the Bluetooth modules. These pins include the following:

1. Pin 1: UART_RTS
2. Pin 2: UART_TX (transmit)
3. Pin 3: UART_CTS
4. Pin 4: UART_RX (receive)
5. Pin 9: VCC
6. Pin 11: RESETB
7. Pin 12: GND
8. Pin 15: GPIO1

Only Pins 2, 4, 9, and 12 are actively used in the final project design. The Bluetooth modules do not need any additional headers or connectors to interface with the PCBs. The PCBs should

already handle all the interconnections. The Bluetooth modules simply need to be soldered onto the PCB via solder paste as shown in Figure B4.

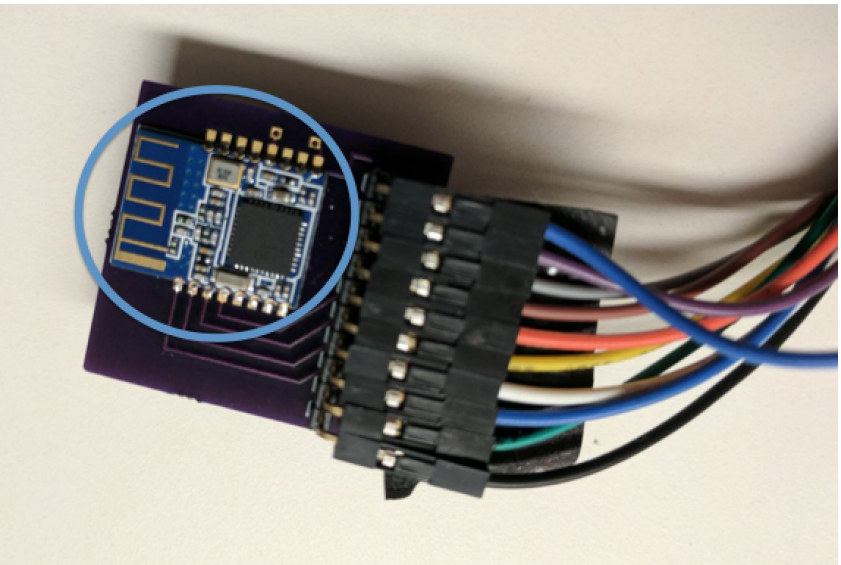


Figure B4. Bluetooth Module Soldered onto Glove PCB

Speaker Module:

1. **Teensy 3.2 Development Board**

The speaker module uses only a handful of the available through-hole connectors. The pins used by the speaker module differ slightly from the glove module:

1. GND (above digital pin 0): ground
2. Vin (across from GND): voltage from regulator
3. 3.3V (above digital pin 23): 3.3V power supply from Teensy
4. DAC (aka A14): digital to analog converter
5. Digital Pin 7: TX of the Bluetooth module; RX of the Teensy
6. Digital Pin 8: RX of the Bluetooth module; TX of the Teensy
7. Digital Pin 10: CS
8. Digital Pin 11: DOUT
9. Digital Pin 12: DIN
10. Digital Pin 13: SCK

The Teensy uses a combination of eight and six pin straight male headers to interface with the PCB. Because the straight male headers are only available as six and eight pins, a few of the header pins may need to be clipped. Refer to Figure B5.

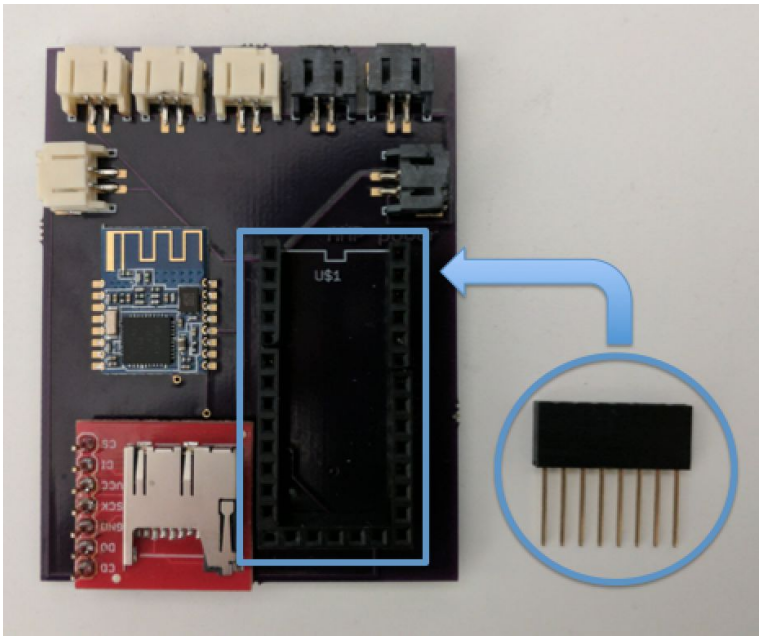


Figure B5. Female Headers Attached to Speaker Module PCB

2. **microSD Card Shield**

The microSD card shield uses all but one of its pins. The pins used for this project include the following:

1. DO (data out)
2. GND
3. SCK
4. VCC
5. DI (data in)
6. CS (chip select)

The microSD card shield can easily be interfaced with the PCB by using a six pin straight male header as shown in Figure B6.

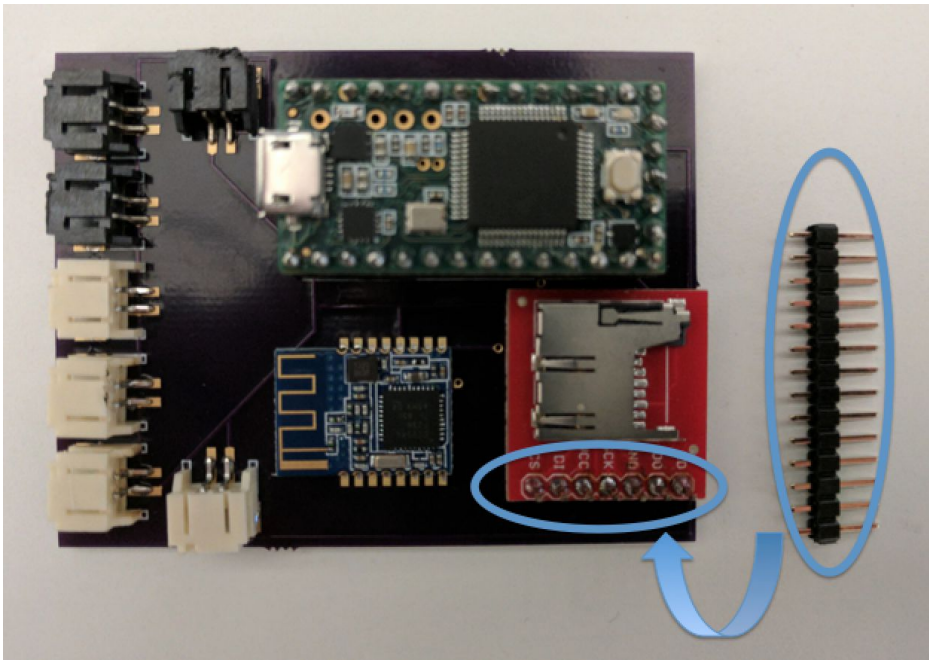


Figure B6. microSD Card Connected with Straight Male Headers

3. **Wired Speaker and Audio Amplifier**

The wired speaker has two soldered wires, which should be connected to the Phoenix connector of the audio amplifier. The audio amplifier has a connector for the audio jack. This particular audio jack has two wires connected to a 2-pin JST Male Connector, which will interface with the PCB via a female JST. The power supply jack interfaces with the PCB in a similar manner as the audio jack.

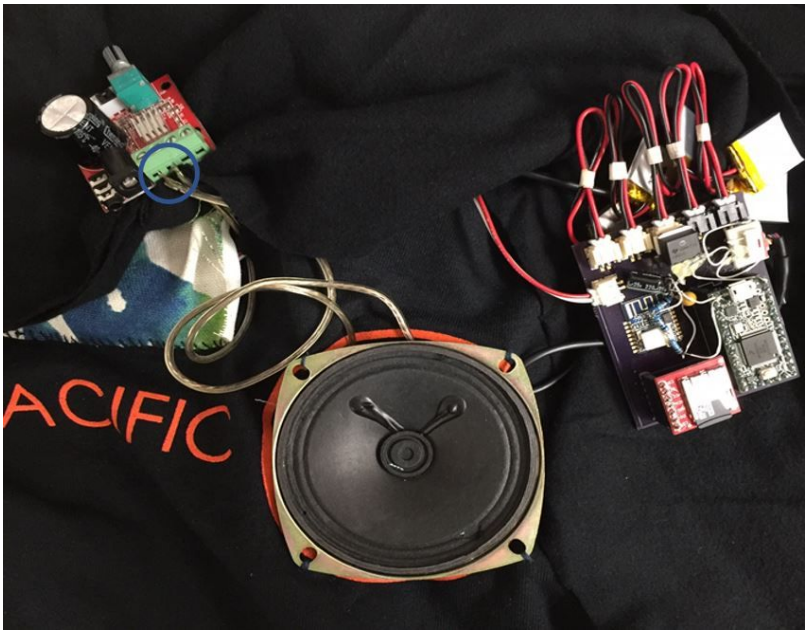


Figure B7: Speaker Interfaced with Amplifier

4. **Bluetooth Module**

Refer to the above *Glove Module* section 4 on *Bluetooth Module*. Figure B8 depicts the placement of the Bluetooth module on the speaker PCB.

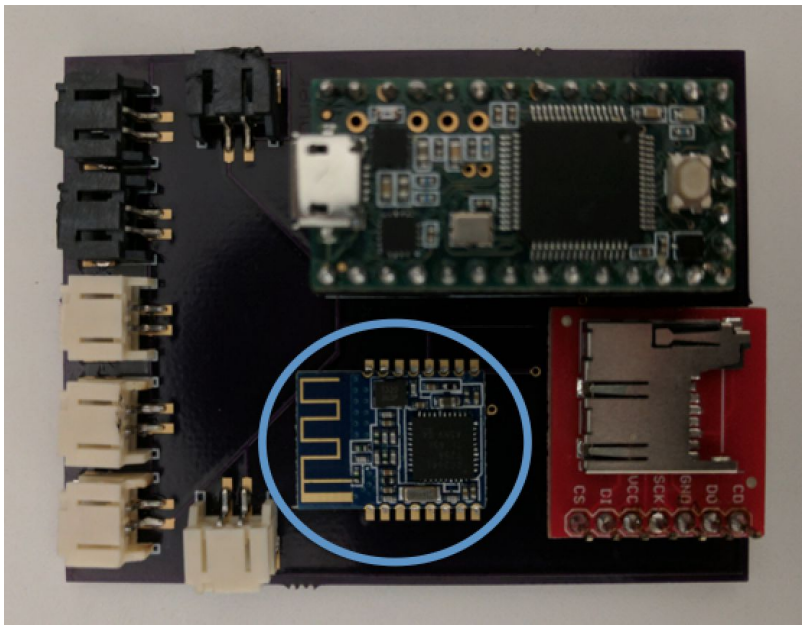


Figure B8. Bluetooth Module Soldered onto Speaker PCB

Appendix C: Troubleshooting Procedures

Section 1. Common Problems

1. Make sure both modules are powered on.
    a. See Appendix A sections 3 and 4
2. Make sure connections on glove are correct.
    a. See figures C1 to C5 below

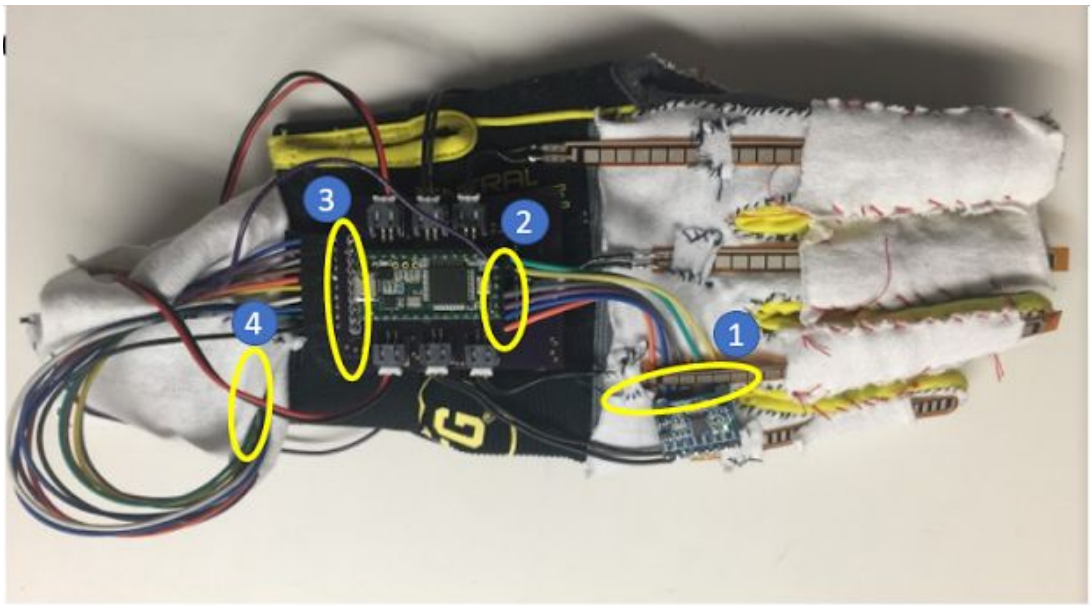


Figure C1. Possible Connection Failure Areas

1. Motion Sensor Connection
2. Motion Sensor to Development Board Connection
3. Bluetooth to Development Board Connection
4. Bluetooth Connection

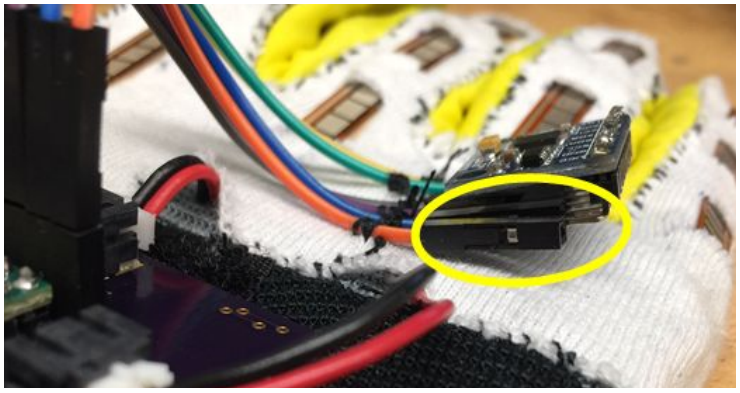


Figure C2. Disconnected Motion Sensor Connection

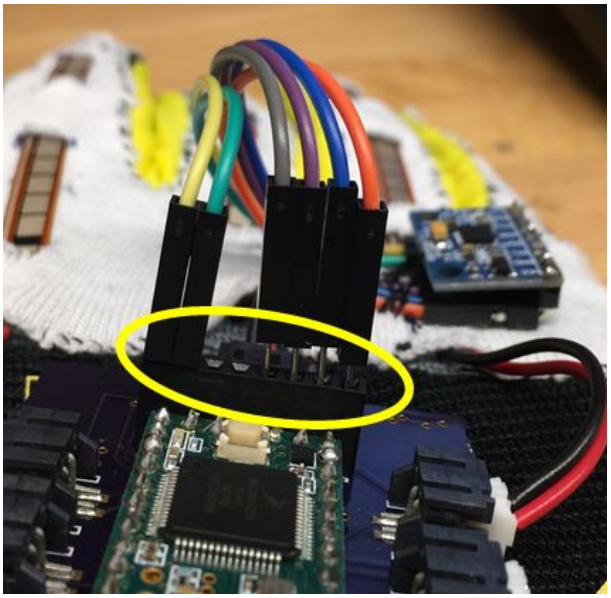


Figure C3. Disconnected Motion Sensor to Development Board Connection

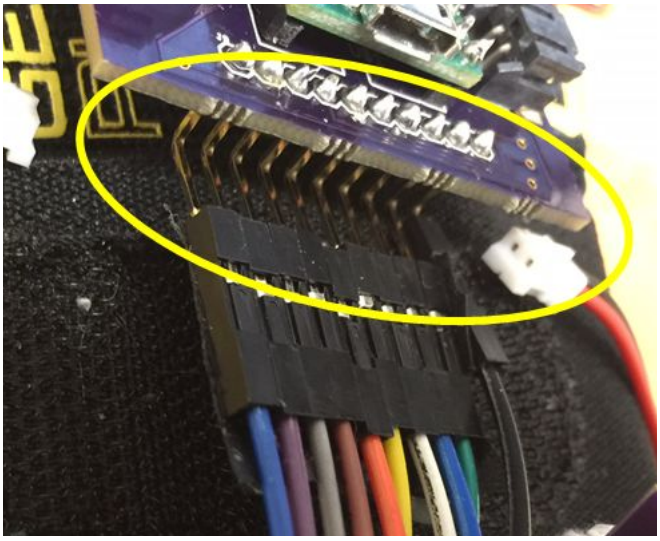


Figure C4. Disconnected Bluetooth to Development Board Connection

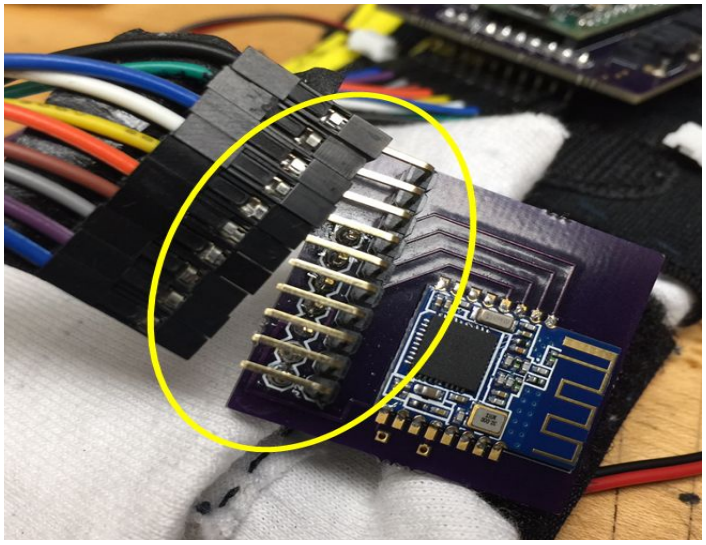


Figure C5. Disconnected Bluetooth Connection

3. Make sure all batteries are charged
4. Make sure speaker volume is up
    a. See Appendix A section 5

Section 2. Bluetooth

The Bluetooth can show that it is having connectivity issues if the system outputs a string of letters that reads “OK+LOST”. When this happens restart the system by powering both modules off (See Appendix A Sections 2 and 3 on how to turn the modules on/off). Once the system restarts, the speaker will read out a string of letters that reads “OK+CONN[*para*]”, in which [*para*] can be A, E, or F. The meaning of each of the three parameters is described as follows:

1. A: connection accepted by the receiving or *slave* Bluetooth module
2. E: connection error
3. F: connection failure

Section 3. Saying the Incorrect Letters

If the speaker is saying letters that you are not signing this may be because of the orientation of the flex sensors. Turn off the system and re-arrange the flex sensors. Power the system back on and try again.

Section 4. Microcontroller

If none of the above sections describes the trouble correctly, the microcontroller may need a factory reset. Contact your vendor and they will reset the microcontroller for you.

Appendix D: Component Layout Diagram

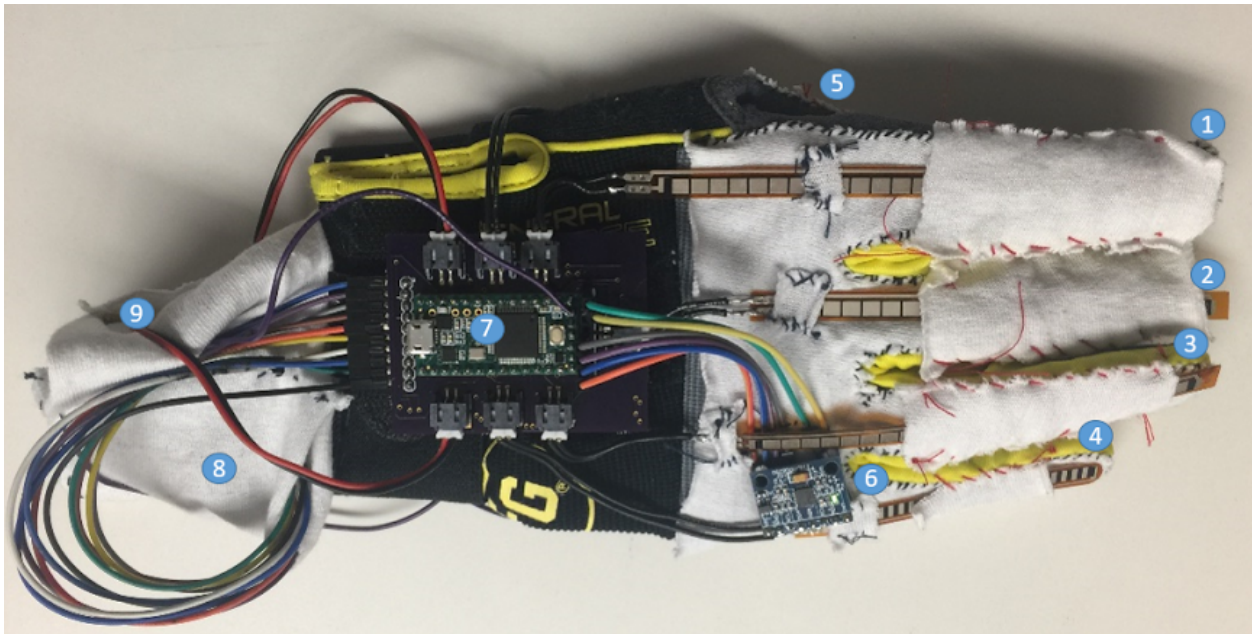


Figure D1. Labeled Glove Module

1. 4” Flex Sensor
2. 4” Flex Sensor
3. 4” Flex Sensor
4. 2.2” Flex Sensor
5. 2.2” Flex Sensor
6. MPU 6050 Motion Sensor
7. Teensy 3.2 Development board
8. Pocket Holding Bluetooth Transmitter (see figure D2)
9. Pocket Holding two batteries (see figure D3)

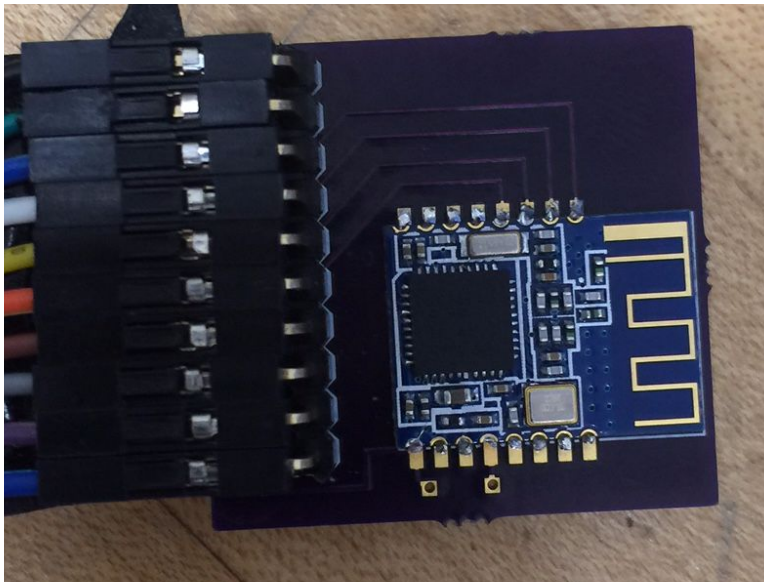


Figure D2. Bluetooth on PCB

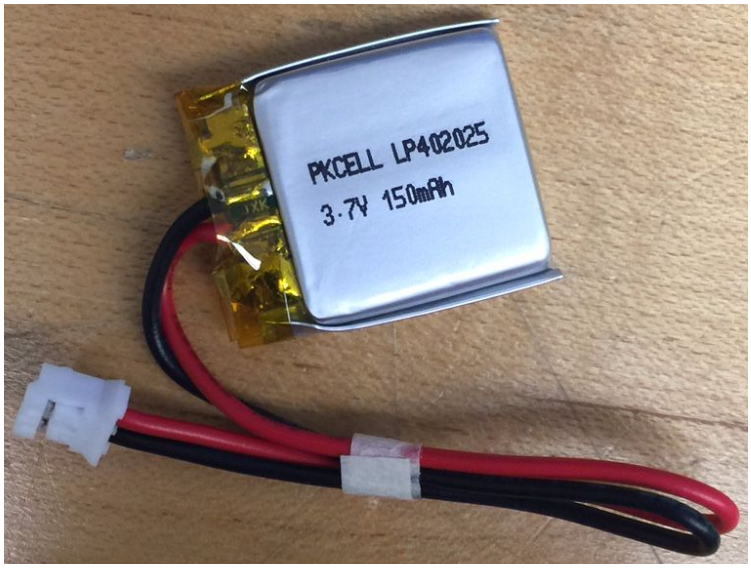


Figure D3. Lithium Ion Battery

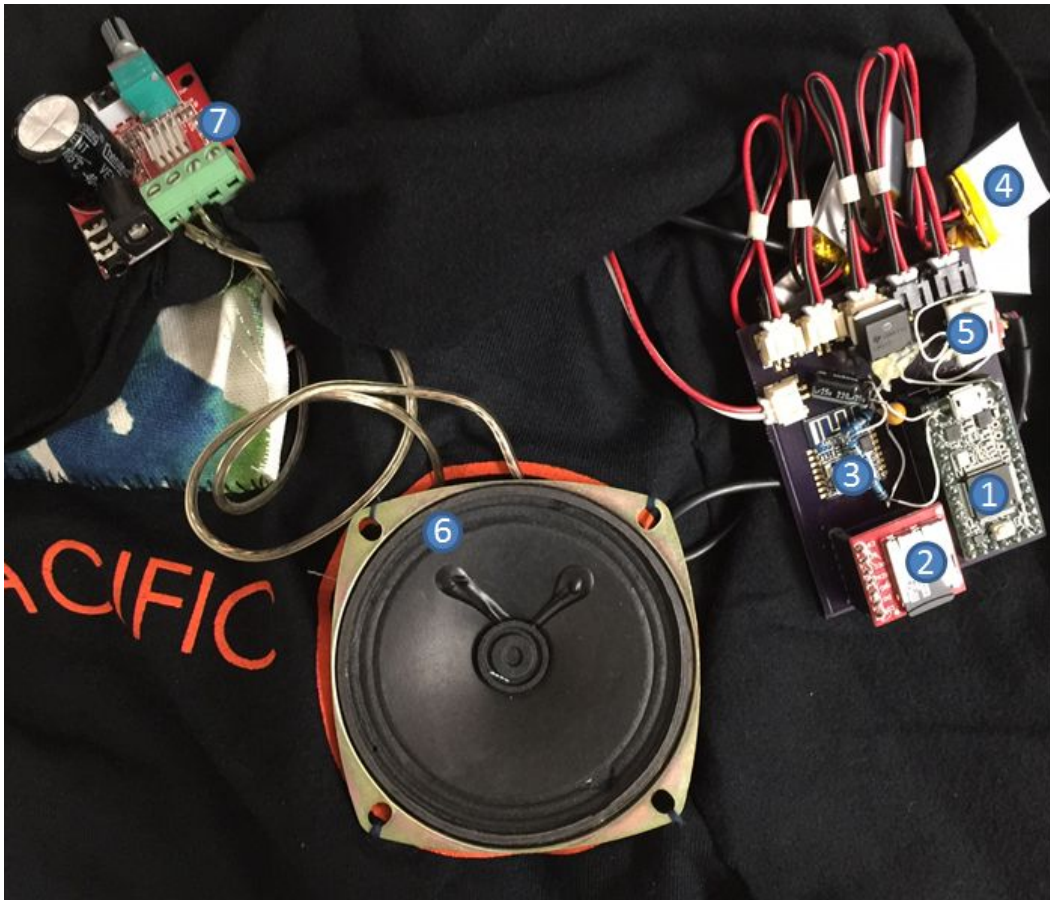


Figure D4. Labeled Speaker Module

1. Teensy 3.2 Development Board
2. MicroSD Card Shield with MicroSD Card
3. Bluetooth Receiver
4. Batteries
5. Power Switch
6. Speaker
7. Audio Amplifier

Appendix E: Principal Component Data Sheets and Other Relevant Material

Section 1. Development Board (Teensy 3.2)

a) MK20DX256 Datasheet: https://www.pjrc.com/teensy/K20P64M72SF1.pdf
b) MK20DX256 User Manual: https://www.pjrc.com/teensy/K20P64M72SF1RM.pdf
c) Teensy 3.2 Specifications and Pinout: https://www.pjrc.com/teensy/teensy31.html
d) Example Code Referenced: https://github.com/PaulStoffregen/Audio/blob/master/examples/WavFilePlayer/WavFilePlayer.ino

Section 2. Battery (Lithium Ion Polymer Battery - 3.7v 150mAh)

a) Datasheet: https://cdn-shop.adafruit.com/product-files/1317/C1515_-_Li-Polymer_402025_150mAh_3.7V_with_PCM.pdf

Section 3. Motion Sensor (MPU-6050)

a) Datasheet: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
b) Example code referenced 1: https://github.com/Electroholics/CodeREPO/blob/master/MPU_6050.ino
c) Example code referenced 2: https://engineering.stackexchange.com/questions/3348/calculating-pitch-yaw-and-roll-from-mag-acc-and-gyro-data

Section 4. Bluetooth (HM-11)

a) Datasheet: http://wiki.seeed.cc/Bluetooth_V4.0_HM_11_BLE_Module/
b) Example code referenced: https://github.com/jpliew/BLEShieldSketch/blob/master/HM_10_Test.ino

Section 5. Flex Sensor (4”)

a) Datasheet:
https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/FLEXSENSORREVA1.pdf

Section 6. Flex Sensor (2.2”)

a) Datasheet:
https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/FLEX%20SENSOR%20DATA%20SHEET%202014.pdf

Section 7. Speaker (4 Ohm 3 Watt)

a) Informational Link: https://www.adafruit.com/product/1314

Section 8. Voltage Regulators

a) TC1262 Voltage Regulator Datasheet (on glove module):
http://www.mouser.com/ds/2/268/21373b-68613.pdf
b) LM317 3-Terminal Adjustable Regulator Datasheet (on speaker module):
http://www.ti.com/lit/ds/symlink/lm317.pdf

Section 9. Amplifier (12V Mini Hi-Fi PAM8610 2X10W Audio Stereo Amplifier Board Dual Channel)

a) Informational Link:
https://www.banggood.com/12V-Mini-Hi-Fi-PAM8610-2X10W-Audio-Stereo-Amplifier-Board-Dual-Channel-p-933675.html?rmmds=search

Appendix F. Code for Glove Module

```
/*
 * THIS ALGORITHM IS WILL ACT AS THE MASTER BLUETOOTH
 * IT WILL BE THE CODE ON THE GLOVE MODULE
 *
 * SENDS THE FOUND LETTER ACCROSS TO OTHER BLUETOOTH
 *
 * DOES NOT CONTAIN ANYTHING NOT NEEDED TO DO THIS
 */

#include <Wire.h> //do I need this?
#include <SPI.h>
#include <SoftwareSerial.h>

#include <SerialFlash.h>

// Bluetooth Stuff
// Bluetooth Buffer
#define BUFFER_LENGTH 100

// Bluetooth Ports
SoftwareSerial ble(7, 8);

// Bluetooth Checking Baud Rate
char buffer[BUFFER_LENGTH];       // Buffer to store response
int timeout = 800;          // Wait 800ms each time for BLE to response, depending on your
application, adjust this value accordingly
//long bauds[] = {9600, 57600, 115200, 38400, 2400, 4800, 19200};
long bauds[] = {115200, 115200, 115200, 115200, 115200, 115200, 115200};

//Flex Sensor
const int flexSensor1 = A0; //analog pin 0 // ring finger
const int flexSensor2 = A1; // thumb
const int flexSensor3 = A2; // index
const int flexSensor4 = A3; //middle
const int flexSensor5 = A6; // pinky

//Motion Sensor
const double pi = 3.14159;
const int MPU_addr = 0x68; // I2C address of the MPU-6050
double AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
float pitch;
float yaw;
float roll;
double velxprev, velyprev, velzprev, AcXprev, AcYprev, AcZprev, v0x, v0y, v0z , velx, vely,
velz, accelx, accely, accelz = 0;
float veltime, veltimenow = 0;

// General
const float VCC = 3.3; //the voltage input
long int count = 0; //just a write a specific amount of data points
int timer;
int sampling_time;
int previous_letter_time;

//Characterization
char letter;

void setup() {
  delay(2000); // just to get everything on and working correctly

  Serial.begin(9600);
  count = 0;
  letter = ' '; //setting it to nothing
  previous_letter_time = 0; // set it to zero at start
```

```
  // Bluetooth
  // Verifying Bluetooth detection
  long baudrate = BLEAutoBaud();

  if (baudrate > 0) {
    Serial.print("Found BLE baud rate ");
    Serial.println(baudrate);
  } else {
    Serial.println("No BLE detected.");
    while (1) {};         // No BLE found, just going to stop here
  }
  // do I need this command to set as master bryan? //
  BLECmd(timeout, "AT+ROLE1", buffer); //set the bluetooth as master
  //_____________________________________________
  BLECmd(timeout, "AT+CONF0C77F9459D8", buffer); // connecting to slave module

  //motion sensor stuff
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);

  //flex sensor stuff
  pinMode(flexSensor1, INPUT);
  pinMode(flexSensor2, INPUT);
  pinMode(flexSensor3, INPUT);
  pinMode(flexSensor4, INPUT);
  pinMode(flexSensor5, INPUT);
}

void loop() {

  timer = millis();

  delay(15);

  count = count + 1;

  //FLEX SENSOR CALC
  int flexAnalog1 = analogRead(flexSensor1); // still only outputs 0
  float flexVoltage1 = flexAnalog1 * VCC / 1023.0;
  int flexAnalog2 = analogRead(flexSensor2); // still only outputs 0
  float flexVoltage2 = flexAnalog2 * VCC / 1023.0;
  int flexAnalog3 = analogRead(flexSensor3); // still only outputs 0
  float flexVoltage3 = flexAnalog3 * VCC / 1023.0;
  int flexAnalog4 = analogRead(flexSensor4); // still only outputs 0
  float flexVoltage4 = flexAnalog4 * VCC / 1023.0;
  int flexAnalog5 = analogRead(flexSensor5); // still only outputs 0
  float flexVoltage5 = flexAnalog5 * VCC / 1023.0;

  //FLEX SENSOR PRINT TO SERIAL MONITOR
  Serial.print(count);
  Serial.print("\t");
  Serial.print(String(flexVoltage2));
  Serial.print("\t");
  Serial.print(String(flexVoltage3));
  Serial.print("\t");
  Serial.print(String(flexVoltage4));
  Serial.print("\t");
  Serial.print(String(flexVoltage1));
  Serial.print("\t");
  Serial.print(String(flexVoltage5));
  Serial.print("\t");
```

```
  //MOTION SENSOR CALC
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr, 14, true); // request a total of 14 registers

  AcXprev = AcX; //storing previous values
  AcYprev = AcY;
  AcZprev = AcZ;

  AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  GyX = Wire.read() << 8 | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY = Wire.read() << 8 | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ = Wire.read() << 8 | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

  //calculate acceleration/velocity here
  double alpha = 0.5;

  v0x = velx;
  v0y = vely;
  v0z = velz;

  //veltime --> the amount of time between samples
  //veltimenow -->
  veltime = timer - veltimenow;
  veltimenow = timer;

  accelx = alpha * AcX + (1 - alpha) * AcXprev;
  velx = v0x + accelx * veltime;
  accely = alpha * AcY + (1 - alpha) * AcYprev;
  vely = v0y + accely * veltime;
  accelz = alpha * AcZ + (1 - alpha) * AcZprev;
  velz = v0z + accelz * veltime;

  Serial.print(String(accelx));
  Serial.print("\t");
  Serial.print(String(accely));
  Serial.print("\t");
  Serial.print(String(accelz));
  Serial.print("\t");

  //yaw pitch and roll calc here
  pitch = atan(AcX / sqrt((AcY * AcY) + (AcZ * AcZ))) * (180 / pi);
  yaw = atan(AcZ / sqrt((AcX * AcX) + (AcZ * AcZ))) * (180 / pi);
  roll = atan(AcY / sqrt((AcX * AcX) + (AcZ * AcZ))) * (180 / pi);

  //MOTION SENSOR PRINT TO SERIAL MONITOR
  Serial.print(yaw);
  Serial.print("\t");
  Serial.print(pitch);
  Serial.print("\t");
  Serial.println(roll);

  sampling_time = timer - previous_letter_time;

  //letter characterization
  letter = signalToLetter(flexVoltage2, flexVoltage3, flexVoltage4, flexVoltage1,
flexVoltage5, yaw, pitch, roll,accelx, accely, accelz);

  if (letter != ' ' && sampling_time > 500) { //if it decided on a letter and it has been
sampling for over half a second since its last sample
    //send letter through bluetooth
    ble.print(letter);
```

```
    Serial.print("The letter chosen was: ");
    Serial.println(letter);
    delay(1000); // give enough time for the slave to receive and play sound

    previous_letter_time = millis(); // save current sampling time
  }
  //if it didnt get a letter just continue to get samples
  // ...
}

/*
 //Uncomment to use test bench and comment the other loop block out
void loop() {
  count = count + 1;

  letter = testbench(count);

  ble.print(letter);
  Serial.print("The letter chosen was: ");
  Serial.println(letter);
  delay(1000); // give enough time for the slave to receive and play sound
}
*/

//FUNCTIONS START HERE ...
char testbench(int counter)
{
  /*
   * This function is for if you just want to see if data is sent over and you don't want to
sign actual letters
   * hard coded to send letters over
   *
   * To use. comment out the entire 1st loop and run the second loops I have instead
   */
  delay(500); // minimum time between letters
  if (counter == 1)
    return ' '; //return nuetral
  else if (counter == 2)
    return 'A';
  else if (counter == 3)
    return 'B';
  else if (counter == 4)
    return 'C';
  else if (counter == 5)
    return 'D';
  else if (counter == 6)
    return 'E';
  else if (counter == 7)
    return 'F';
  else if (counter == 8)
    return 'G';
  else if (counter == 9)
    return 'H';
  else if (counter == 10)
    return 'I';
  else if (counter == 11)
    return 'J';
  else if (counter == 12)
    return 'K';
  else if (counter == 13)
    return 'L';
  else if (counter == 14)
    return 'W';
  else
    return ' ';
}
```

```
// CHARACTERIZATION ALGORITHM
char signalToLetter(float thumb, float index, float middle, float ring, float pinky, float
yaw, float pitch, float roll, float accelx, float accely, float accelz) {
/*
  * Input: Voltage values for each finger from flexible sensor, yaw, pitch, and roll from the
motion sensor
  * Output: Letter (char)
  *
  * This function takes in sensor values, and identifies which letter is indicated through
hand gestures
  */

  int flexVal[5];

  // find thumb values
  if (thumb > 0.93) { // straight thumb
    flexVal[0] = 1;
  }
  else if (thumb > 0.82 && thumb <= 0.91) { // half bent
    flexVal[0] = 2;
  }
  else { // full bent
    flexVal[0] = 3;
  }

  // find index values
  if (index > 1.95) { // straight
    flexVal[1] = 1;
  }
  else if (index > 1.25 && index <= 1.95) { // slight bent
    flexVal[1] = 2;
  }
  else if (index > 0.99 && index <= 1.25) { // half bent
    flexVal[1] = 3;
  }
  else { // full bent
    flexVal[1] = 4;
  }

  // find middle values
  if (middle >= 2.09) { // straight
    flexVal[2] = 1;
  }
  else if (middle > 1.47 && middle < 2.09) { // slight bent
    flexVal[2] = 2;
  }
  else if (middle > 1.28 && middle <= 1.47) { // half bent
    flexVal[2] = 3;
  }
  else { // full bent
    flexVal[2] = 4;
  }

  // find ring values
  if (ring > 1.9) { // straight
    flexVal[3] = 1;
  }
  else if (ring > 1.63 && ring <= 1.9) { // slight bent
    flexVal[3] = 2;
  }
  else if (ring > 1.39 && ring <= 1.63) { // half bent
    flexVal[3] = 3;
  }
  else { // full bent
    flexVal[3] = 4;
```

```
  }

  // find pinky values
  if (pinky > 0.61) { // straight
    flexVal[4] = 1;
  }
  else if (pinky > 0.51 && pinky <= 0.61) { // slight bent
    flexVal[4] = 2;
  }
  else if (pinky > 0.47 && pinky <= 0.51) { // half bent
    flexVal[4] = 3;
  }
  else { // full bent
    flexVal[4] = 4;
  }

  float x1, x2, x3, x4, x5, xsum;
  x1 = flexVal[0] * 10000;
  x2 = flexVal[1] * 1000;
  x3 = flexVal[2] * 100;
  x4 = flexVal[3] * 10;
  x5 = flexVal[4] * 1;
  xsum = x1 + x2 + x3 + x4 + x5;

  //float yaw, pitch, roll; //why is this even here, going to take out for time being

  // if/else if statements to match letter
  if (xsum == 11111 || xsum == 11112) { // neutral
    return '.';
  }

  // G or H
  if (yaw > 40 && yaw < 50 && pitch < 11 && roll > 40 && roll < 50) //originally had only one
'&' between pitch and roll
  {
    if (xsum == 12222 || xsum == 12342 || xsum == 22332 || xsum == 11333 || xsum == 22334) {
//G
      return 'G';
    }
    else if (xsum == 12132 || xsum == 11232 || xsum == 11233 || xsum == 22134) { //H
      return 'H';
    }
    else if (xsum == 14444) { //good job special character
      return '!';
    }
    else {
      return ' ';
    }
  }

  // P
  else if (yaw > 12 && yaw < 30 && pitch < 65 && roll > 20 && roll < 40) //originally had only
one '&' between pitch and roll
  {
    if (xsum == 11221 || xsum == 12221 || xsum == 22231 || xsum == 12231 || xsum == 12232) {
//P
      return 'P';
    }
    else {
      return ' ';
    }
  }

  // Q
  else if (yaw > 12 && yaw < 35 && pitch > 28 && pitch < 55 && roll > 15 && roll < 60) {
    if (xsum == 12332 || xsum == 12333 || xsum == 12322 || xsum == 12332) { //Q
```

```
      return 'Q';
    }
    else {
      return ' ';
    }
  }
  else if (xsum == 34321 || xsum == 34331 || xsum ==34332) { //I or J
    if (yaw > 6 && pitch > 35 && pitch < 55 && roll > 35 && roll < 55) {
      return 'I';
    }
    else {
      return 'J';
    }
  }
  else if (accelx >= 10000 && accely >= 10000) { //Z
    if (xsum == 22334 || xsum == 12343 || xsum == 11322 || xsum == 21211 || xsum == 22333 ||
xsum == 22332 || xsum == 12334) {
      return 'Z';
    }
    else {
      return ' ';
    }
  }

  else if (xsum == 14334 || xsum == 13222 || xsum == 13223 || xsum == 13333) { //A
    return 'A';
  }
  else if (xsum == 32111 || xsum == 31211) { //B
    return 'B';
  }
  else if (xsum == 22222) { //C
    return 'C';
  }
  else if (xsum == 21334 || xsum==31434) { //D
    return 'D';
  }
  else if (xsum == 34433 || xsum == 34434) { //E
    return 'E';
  }
  else if (xsum == 24111 || xsum == 34111) { //F
    return 'F';
  }
  /* else if(xsum == 11333){ //G
     return 'G';
   }
   else if(xsum == 11132){ //H
     return 'H';
   }*/
  else if (xsum == 11122 || xsum == 11132 || xsum == 12232 || xsum == 12242) { //K
    return 'K';
  }
  else if (xsum == 11333 || xsum == 12432 || xsum == 12334 || xsum == 12333) { //L
    if(yaw>35 && yaw <55 && pitch < 15 && roll > 50){
      return 'L';
    }
  }
  else if (xsum == 14231 || xsum == 14232 || xsum == 24231 || xsum == 24331) { //M
    return 'M';
  }
  else if (xsum == 24342 || xsum == 14233 || xsum == 14341) { //N
    return 'N';
  }
  else if (xsum == 33332 || xsum == 23323 || xsum == 33324) { //O
    return 'O';
  }
  else if (xsum == 32231 || xsum == 21234) { //R
```

```
    return 'R';
  }
  else if (xsum == 24444 || xsum == 34444) { //S
    return 'S';
  }
  else if (xsum == 13232 || xsum == 13333 || xsum == 13334 || xsum == 13234 || xsum == 23232
|| xsum == 13221) { //T
    return 'T';
  }
  else if (xsum == 32134 || xsum == 22132 || xsum == 31131 || xsum == 32143 || xsum == 32142
|| xsum == 32132 || xsum == 22122) { //U
    return 'U';
  }
  else if (xsum == 21132 || xsum == 31144 || xsum == 31133 || xsum == 21134) { //V
    return 'V';
  }
  else if (xsum == 21114 || xsum == 31124 || xsum == 31224) { //W
    return 'W';
  }
  else if (xsum == 12234 || xsum == 33344 || xsum == 33334) { //X
    return 'X';
  }
  else if (xsum == 14331 || xsum == 14332 || xsum == 14342) { //Y
    return 'Y';
  }
  /*else if(xsum == 11322){ //Z
   *
    return 'Z';
  }*/


  else {
    return ' ';
  }
}

//BLE Functions
long BLEAutoBaud() {
  int baudcount = sizeof(bauds) / sizeof(long);
  for (int i = 0; i < baudcount; i++) {
    for (int x = 0; x < 3; x++) { // test at least 3 times for each baud
      Serial.print("Testing baud ");
      Serial.println(bauds[i]);
      ble.begin(bauds[i]);
      if (BLEIsReady()) {
        Serial.println("BLE is ready");
        Serial.print("Baud rate is: ");
        Serial.println(bauds[i]);
        return bauds[i];
      }
    }
  }
  return -1;
}

boolean BLEIsReady() {
  BLECmd(timeout, "AT" , buffer);   // Send AT and store response to buffer
  if (strcmp(buffer, "OK") == 0) {
    return true;
  } else {
    return false;
  }
}

boolean BLECmd(long timeout, char* command, char* temp) {
  long endtime;
```

```
  boolean found = false;
  endtime = millis() + timeout; //
  memset(temp, 0, 100);       // clear buffer
  found = true;
  Serial.print("Arduino send = ");
  Serial.println(command);
  ble.print(command);

  while (!ble.available()) {
    if (millis() > endtime) {   // timeout, break
      found = false;
      break;
    }
  }

  if (found) {            // response is available
    int i = 0;
    while (ble.available()) {   // loop and read the data
      char a = ble.read();
      // Serial.print((char)a); // Uncomment this to see raw data from BLE
      temp[i] = a;        // save data to buffer
      i++;
      if (i >= BUFFER_LENGTH) break; // prevent buffer overflow, need to break
      delay(1);           // give it a 2ms delay before reading next character
    }
    Serial.print("BLE reply    = ");
    Serial.println(temp);
    return true;
  } else {
    Serial.println("BLE timeout!");
    return false;
  }
}
```

Appendix G. Code for Speaker Module

```
/*
 * THIS ALGORITHM WILL BE UPLOADED TO THE SHIRT MODULE TEENSY
 *
 * IT WILL ACT AS A SLAVE A RECIEVE A 'char letter' FROM THE GLOVE MODULE TO PLAY ON THE
SPEAKER
 *
 * DOES NOT CONTIAN ANYTHING NOT NEEDED
 */
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SoftwareSerial.h>

#include <Audio.h>
#include <SerialFlash.h>
#include <play_sd_wav.h>

// Bluetooth Stuff
// Bluetooth Buffer
#define BUFFER_LENGTH 100

// Bluetooth Ports
SoftwareSerial ble(7, 8);

// Bluetooth Checking Baud Rate
char buffer[BUFFER_LENGTH];       // Buffer to store response
int timeout = 800;          // Wait 800ms each time for BLE to response, depending on your
application, adjust this value accordingly
long bauds[] = {9600, 57600, 115200, 38400, 2400, 4800, 19200};

//Audio Stuff
AudioPlaySdWav playWav1;
AudioOutputI2S audioOutput;
AudioOutputAnalog dac;
AudioConnection patchCord1(playWav1, 0, dac, 0);
AudioConnection patchCord2(playWav1, 1, dac, 1);

//SD card
const int CS = 10; //card select pin
int sdcard = 0; //to check if the sd card failed

//General
const int led = 13; // the led pin
int count = 0;
int stops = 1;

//Letter found
char letter;
char blei[7] = ""; //for the bluetooth initialization letters

void setup() {
  delay(2000); // just to get everything on and working correctly

  Serial.begin(9600);
  pinMode(led, OUTPUT);
  sdcard = 0;
  count = 0;
  stops = 1;

  // Bluetooth
  // Verifying Bluetooth detection
  long baudrate = BLEAutoBaud();

  if (baudrate > 0) {
```

```
    Serial.print("Found BLE baud rate ");
    Serial.println(baudrate);
  } else {
    Serial.println("No BLE detected.");
    while (1) {};         // No BLE found, just going to stop here
  }
  BLECmd(timeout, "AT+ROLE0", buffer); // set as slave
  Serial.println("----------------------");
  Serial.println("Waiting for remote connection...");

  //sd card stff
  SPI.begin(); //initialize spi
  pinMode(CS, OUTPUT); //initialize pin 10 for sd card
  digitalWrite(CS, HIGH);
  if (!SD.begin(CS)) {
    Serial.println("SD card initialization failed!");
    sdcard = 1; //failed so maybe flash the teensy's lights like crazy?
    return;
  }

  //Audio stuff
  AudioMemory(8);
  SPI.setMOSI(11);
  SPI.setSCK(13);
}

void loop() {
  //all this needs to do is recieve the letter from the master and output a sound for it

  if (sdcard == 1) //if the sd card fails make flashing lights on the TEENSY
  {
    digitalWrite(led, HIGH);
    delay(100);
    digitalWrite(led, LOW);
    delay(100);
  }
  else //if the sdcard works then just do whatever
  {
    if (ble.available())
    {
      letter = (char)ble.read();

      blei[count] = (char)ble.read(); //for the initial setup
      count = count + 1;

      Serial.print("Letter found --> ");
      Serial.println(letter);
      stops = 0;
    }

    //Audio Output
    if (stops == 0) {
      if (blei == "OK CONN") //if(blei[0] == 'O' && blei[1] == 'K' ... so on)
      {
        playFile("INIT.WAV"); // play sound file to say bluetooth initialized
      }
      else
        chooseAudio(letter); //will play the sound
    }
  }
}

//AUDIO
void playFile(const char *filename)
{
  Serial.print("Playing file: ");
```

```
  Serial.println(filename);

  // Start playing the file.  This sketch continues to
  // run while the file plays.
  playWav1.play(filename);

  // A brief delay for the library read WAV info
  delay(5);

  // Simply wait for the file to finish playing.
  while (playWav1.isPlaying()) {

  }
}

//CHOOSE LETTER
void chooseAudio(char letter) {
  if (letter == 'A') //letter recognized is whatever... then play the sound file
    playFile("A.WAV");
  else if (letter == 'B')
    playFile("B.WAV");
  else if (letter == 'C')
    playFile("C.WAV");
  else if (letter == 'D')
    playFile("D.WAV");
  else if (letter == 'E')
    playFile("E.WAV");
  else if (letter == 'F')
    playFile("F.WAV");
  else if (letter == 'G')
    playFile("G.WAV");
  else if (letter == 'H')
    playFile("H.WAV");
  else if (letter == 'I')
    playFile("I.WAV");
  else if (letter == 'J')
    playFile("J.WAV");
  else if (letter == 'K')
    playFile("K.WAV");
  else if (letter == 'L')
    playFile("L.WAV");
  else if (letter == 'M')
    playFile("M.WAV");
  else if (letter == 'N')
    playFile("N.WAV");
  else if (letter == 'O')
    playFile("O.WAV");
  else if (letter == 'P')
    playFile("P.WAV");
  else if (letter == 'Q')
    playFile("Q.WAV");
  else if (letter == 'R')
    playFile("R.WAV");
  else if (letter == 'S')
    playFile("S.WAV");
  else if (letter == 'T')
    playFile("T.WAV");
  else if (letter == 'U')
    playFile("U.WAV");
  else if (letter == 'V')
    playFile("V.WAV");
  else if (letter == 'W')
    playFile("W.WAV");
  else if (letter == 'X')
    playFile("X.WAV");
  else if (letter == 'Y')
```

```
    playFile("Y.WAV");
  else if (letter == 'Z')
    playFile("Z.WAV");
  else {
    //dont do anything
  }
}

//BLE Functions
long BLEAutoBaud() {
  int baudcount = sizeof(bauds) / sizeof(long);
  for (int i = 0; i < baudcount; i++) {
    for (int x = 0; x < 3; x++) { // test at least 3 times for each baud
      Serial.print("Testing baud ");
      Serial.println(bauds[i]);
      ble.begin(bauds[i]);
      if (BLEIsReady()) {
        Serial.println("BLE is ready");
        Serial.print("Baud rate is: ");
        Serial.println(bauds[i]);
        return bauds[i];
      }
    }
  }
  return -1;
}

boolean BLEIsReady() {
  BLECmd(timeout, "AT" , buffer);   // Send AT and store response to buffer
  if (strcmp(buffer, "OK") == 0) {
    return true;
  } else {
    return false;
  }
}

boolean BLECmd(long timeout, char* command, char* temp) {
  long endtime;
  boolean found = false;
  endtime = millis() + timeout; //
  memset(temp, 0, 100);       // clear buffer
  found = true;
  Serial.print("Arduino send = ");
  Serial.println(command);
  ble.print(command);

  while (!ble.available()) {
    if (millis() > endtime) {   // timeout, break
      found = false;
      break;
    }
  }

  if (found) {            // response is available
    int i = 0;
    while (ble.available()) {   // loop and read the data
      char a = ble.read();
      // Serial.print((char)a); // Uncomment this to see raw data from BLE
      temp[i] = a;        // save data to buffer
      i++;
      if (i >= BUFFER_LENGTH) break; // prevent buffer overflow, need to break
      delay(1);           // give it a 2ms delay before reading next character
    }
    Serial.print("BLE reply    = ");
    Serial.println(temp);
    return true;
```

```
  } else {
    Serial.println("BLE timeout!");
    return false;
  }
}
```