THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

THE GLOVE PROJECT: ALLOWING THE VISUALLY IMPAIRED TO UNDERSTAND AND INTERACT WITH THEIR ENVIORNMENT

DAVID JOSE DE MATHEU DE LIMA
SPRING 2018

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Engineering
with honors in Computer Engineering

Reviewed and approved* by the following:

Vijay Narayanan
Distinguished Professor, Computer Science and Engineering
Thesis Supervisor

Chita Das
Distinguished Professor, Department Head, Computer Science and Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

The Grocery Assistance System (GAS), as part of the NSF expedition Visual Cortex on Silicon, seeks to help visually impaired people to be more independent using technology. The GAS system relies on various subsystems including a Computer Vision module to understand the environment around the user as well as a user interface for the system to communicate with the user. The user interface allows the system to send feedback to the user as well as receive inputs that can aid it.

This thesis explores the process of designing and building a solution that combines hardware and software to solves the communication needs of the system. It introduces the concept of the glove, a device that fits on the user's hand and enables communication between the system and the user. The glove combines the use of off-the-shelf hardware with custom software written in C and C# to enable the system to send haptic feedback to the user and receive input. The goal of this exploration that encompasses 3 iterations of the glove was to test different methods of interaction and communication techniques. Through each of the iterations the text looks at the strengths and weaknesses that each had and how they can be improved. Finally, the text explores the future away from the glove, moving into new and more innovative solutions.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to start by thanking the entire MDL team for the help during the process of doing research and writing this thesis. Specially to Dr. Vijay Narayanan and Dr. Kevin Irick as mentors and leaders of this project. I would also like to thank Gus Smith, Peter Zientara, Christopher Pratt and Daniel Troncoso for their help and support with the Glove Project. Finally, this work was supported in part by the NSF Expedition in Computing Program: Visual Cortex on Silicon CCF 1317560.

On a more personal note, I would like to thank my parents and brothers for their unconditional support and help throughout the past four years of college and through the writing of this thesis. I would also like to thank my friends here at Penn State who have supported me through hard and stressful times. Without my family and friends this would have not been possible. This thesis marks the culmination of one of the most challenging yet rewarding experiences that I have had in my life. I am beyond grateful for everyone who made it possible!

Chapter 1 : Introduction

In the last decade, significant improvements in computing systems has allowed the field of Artificial Intelligence to develop at an unprecedented rate. It has allowed to realize dreams of early pioneers in Artificial Intelligence of creating computers that would have human-like capabilities. Developments have been seen across the board in fields like Machine Learning, Natural Language Processing, Computer Vision and Reasoning among others. Here at Penn State, the Microsystems Design Lab (MDL) is part of a National Science Foundation backed exploration called the Visual Cortex on Silicon. [1]

What is the Visual Cortex on Silicon Exploration?

The Visual Cortex on Silicon Exploration seeks to explore the holistic design of machine vision systems that have the potential to approach and eventually exceed the capabilities of human vision systems. The project encapsulates a variety of disciplines from hardware design, computer vision foundations and algorithms as well as Human Computer Interaction. The end goal of the project is to fuse research in all these disciplines to create vision systems that have the same capabilities as the human eye in terms of detection, analysis, and interaction. To achieve this goal advances in all the fields related to the project must be done, from making new hardware to handle the computing capacity necessary, to developing new algorithms that are better at detecting and analyzing objects in a scene, all the way to creating interaction models for people to use these systems. [1]

Inside the Visual Cortex on Silicon Exploration, there is a sub-project to which this thesis is specific to. The project is called Grocery Assistance System (GAS) which is centered in the applications of computer vision systems to be used for different scenarios as an aid for people. Specifically, the main scenario on which GAS is concentrated is working with the visually impaired community in creating computer vision systems that can help visually impaired people in their day to day life. Such as like the name says it going grocery shopping.

Grocery Store Scenario

The main scenario on which the GAS project concentrated was the grocery store. The idea of a visually impaired person being able to enter a grocery store and through the aid of the computer vision system be able to pick up items from shelves. This scenario requires a couple of crucial parts starting with the navigation aspect of being able to direct the person around the store. Then being able to locate the item in the shelf and finally being able to guide the person's hand up to the item.

Role of Human Computer Interaction Design

Computer vision systems are complicated by nature, as they are made from a multitude of systems that are working in perfect unison to detect, classify and analyze objects. It is necessary to take deep consideration at the Human Computer Interaction aspect to make these elaborate and complex systems useful for people. For the GAS project, main users are visually impaired people, so the human computer interaction design is focused in bringing value to this demographic. The main component of the HCI design for the GAS project is the glove, it is the communication device between the computer vision system that is processing the space around them and the user. The glove allows the system to send haptic feedback responses based on events captured by the system as well as the ability of the user to input

commands to the system. It has gone through various iterations that looked at improving the interaction between the user and the computer vision system.

Throughout the process the team has been able to answer questions regarding what the best ways are to interact with visually impaired people (glove, gestures, minimalist interfaces, etc.) and how our experiences with this demographic can extrapolate to other demographics and scenarios. This process can be seen in the flow chart found in figure 1 below. The flow chart shows the main differentiation features that each iteration brought to contribute to the overall project. This exploration will concentrate on the process of designing and building the glove throughout its iterations and what the future awaits past the glove.

Figure 1. Glove Iteration Flowchart

Chapter 2 : Glove 1.0: The First Model

Introduction

When the GAS project started it was understood that the team needed to solve the problem of interacting with the user. Visually impaired people are not able to use conventional displays that many take for granted. It was key then developing a new way of interacting with this demographic while acknowledging their needs. Before the concept of the glove was decided, a process of design was done to develop the specification for solution in terms of what were the things that the interaction device had to be able to do and at the same time not do for it to be useful. The overall main scenario for which the device was being designed for was the grocery store.

Design Specification

For the first of iteration, the main design criteria were catering to the visually impaired demographic and figuring out what information was to be conveyed.

Who is the user?: This group of people have a couple of main characteristics that were key to take into consideration:

- To be classified as visually impaired, it means that their vision is either severely impaired (can detect light, shadows and sometimes minimum color) or have complete blindness.
- People rely on two main mediums to navigate their environment. The first being their walking sticks that allows them to detect obstacles in front of them. The second being sound which they use to hear echoes in spaces and detect obstacles. The solution must take these two mediums into consideration as to not interfere with them.

What information needs to be conveyed?: In this case, five main commands wanted to be transmitted to the user being: forward, left, right, back and stop. This command allows the system to guide the person around. Specifically, to the grocery store scenario, the device had to able to help guide the person's hand towards an object on a shelf.

Given the criteria, the glove was chosen as the device to be built. The glove was to be equipped with 4 motors located in the thumb, middle finger, pinky finger and wrist to give haptic feedback to the user. Each location was to be attached to one of the directions being forward, backward, right, and left. Using a glove allowed the person to still be able to use their hand to reach out for objects with close to no obstruction.

Hardware Specification

Table 1. Glove 1.0 Specification

Main Chip	**Teensy 3.1** [4]
Haptic Engines	1.3VDC Vibrating Motors [6]
Glove	Everlast Hand Wraps (Glove with no fingers)
Board	Custom printed board with connections for the main chip as well as the 4 haptic engines
Connectivity	Wired (Chip is directly connected to the computer)

Results

Figure 2. Glove 1.0 Prototype being worn during a test [14]

Glove 1.0 was a great initial iteration on the glove project. The biggest achievement that the Glove 1.0 had was to be able to show a tangible prototype to other research labs in Penn State as well as the visually impaired community to help explain the overall project and what the future might look like. It ultimately showed promise which created excitement around the project. Glove 1.0 was integrated into a variety of studies and tests to gather data on human computer interaction. In the figure 2 above can be seen Glove 1.0 during a test.

Table 2. Feature Summary for Glove 1.0

Main Features	Strengths	Weaknesses	Conclusions
Haptic Feedback	-Good Motors -Good Placement for commands	-Vibration Strength should be controlled -More controlled can/should be given to developer	Using motors to give haptic feedback was successful
Custom PCB	-Allows for easy connections and organization	-Connections were tight and sometimes a bit hard to solder	Using a custom PCB is a correct approach as it simplifies the process of organizing and soldering components
Wiring on Chip	-N/A	-Proved to cause the most amount of problems in terms of build quality	A new approach must be put together to tackle the wiring of the components on the chip
Wired Connectivity	-Command latency is very small -Not battery necessary as power supplemented by the PC	-Not mobility -Glove must always be connected for functionality	Although a wired connection offers the ability to have unlimited power and low latency, mobility is a key criterion that must be considered.

Haptic Feedback: Haptic feedback was one of the main strengths of the first prototype of the glove. It did a suitable job at communicating commands from the computer to the user. The motors gave enough haptic feedback for users to be able to easily identify what direction vibrations were coming from. Vibration strength could be improved somewhat to make the feedback even better. The main drawback was more from a software perspective, in terms of building a better interface for developers to be able to use and control the glove. This means giving developers the ability to tweak the motors strength as well as create patterns of motor vibrations.

Building Quality (Custom PCB and Wiring on Chip): The use of a custom PCB was a great advantage from a build quality perspective as it allowed to use less cables and for all components to be organized. All components were directly soldered onto the board which presented a build quality disadvantage. If something broke, repairing it was difficult. Additionally, cables and connections were exposed which meant that any minor pull or crash could easily make cables break and make the entire device unusable.

Wired Connectivity: Using a wired design allowed for low latency for commands to be sent as well as no concerns regarding battery life, but it meant movement was compromised. For visually impaired people, feeling free to move around and more importantly be able to move their arms is key. Their hands are their main source of tactile feedback to know if obstacles are near them. By being tethered it meant they wouldn't really move much other than the amount permitted by the cable. Being the first prototype, this was fine, but to move forward the glove will have to go into a wireless configuration which will present new challenges.

Chapter 3 : Glove 2.0: Iterating in the Design

Design Specification

Glove 1.0 left many lessons that need to be addressed and improved for the second generation. Overall, it was found that the glove concept worked well, it allowed the visually impaired people to feel like they have freedom to grab and move, yet it did its job properly at giving commands. For the next generation the three main criteria that were to be improved were Connectivity, Build Quality and Developer Extensibility.

Connectivity: The wired design of the first model served well to prototype and test the concept of the glove, but for the second iteration it was key to move to a better functional design. It meant the glove had to be wireless. This allows the user to not be tethered to the computer running the main system. This allowed for a wider motion range. Making the glove wireless required the use of a different microcontroller that supported Bluetooth technology as well as the creation of a Bluetooth library for the glove to communicate with the computer. A couple of expected issues that will have to be dealt with are the battery life of the microcontroller as a battery will be needed to power it as well as the latency of commands being sent.

Build Quality: The build quality of the first prototype was good, it served its purpose properly. Moving on though, the build quality needed to be improved. The main complaint in terms of build quality that the other glove had was the difficulty to fix it if any cables broke or motors stopped working as everything was directly soldered onto the board. For this new version, more modular approach was taken where motors and the chip itself can be easily swapped in an out without the need to solder anything. This improves the ability to repair the glove. To also minimize how often cables and motors would break, a couple actions were taken in terms of how cables where managed and soldered. Heat shrink was used

throughout the glove to secure solders, pair cables together and give the overall connections more durability.

Developer Extensibility: The final area that needed improvements was the ability of developers to have more control over the commands that the glove can act upon. For this, as part of the Bluetooth library that was created, additional functionality was introduced to give further control over the glove to developers. This included the ability to control motors in sequences to create patterns as well as the use of preprogrammed patterns that were built in. This allowed for more intricate commands to be sent to the user through haptic feedback.

Hardware Specification

Table 3. Glove 2.0 Specification

Main Chip	**Adafruit Feather 32u4 Bluefruit LE** [7]
Haptic Engines	Jameco 1.3VDC Vibrating Motors [6]
Glove	Vibger Winter Gloves
Board	Custom printed board with connections for the main chip as well as the 4 haptic engines
Connectivity	Wireless (Chip supports Bluetooth)

Implementation

For the full architecture diagram of the Glove 2.0 implementation look at Appendix A.

Hardware Implementation

The hardware implementation of glove 2.0 has three main components, first the custom PCB that was created for the glove, second the wiring work and finally, the positioning and securing of the haptic motors.

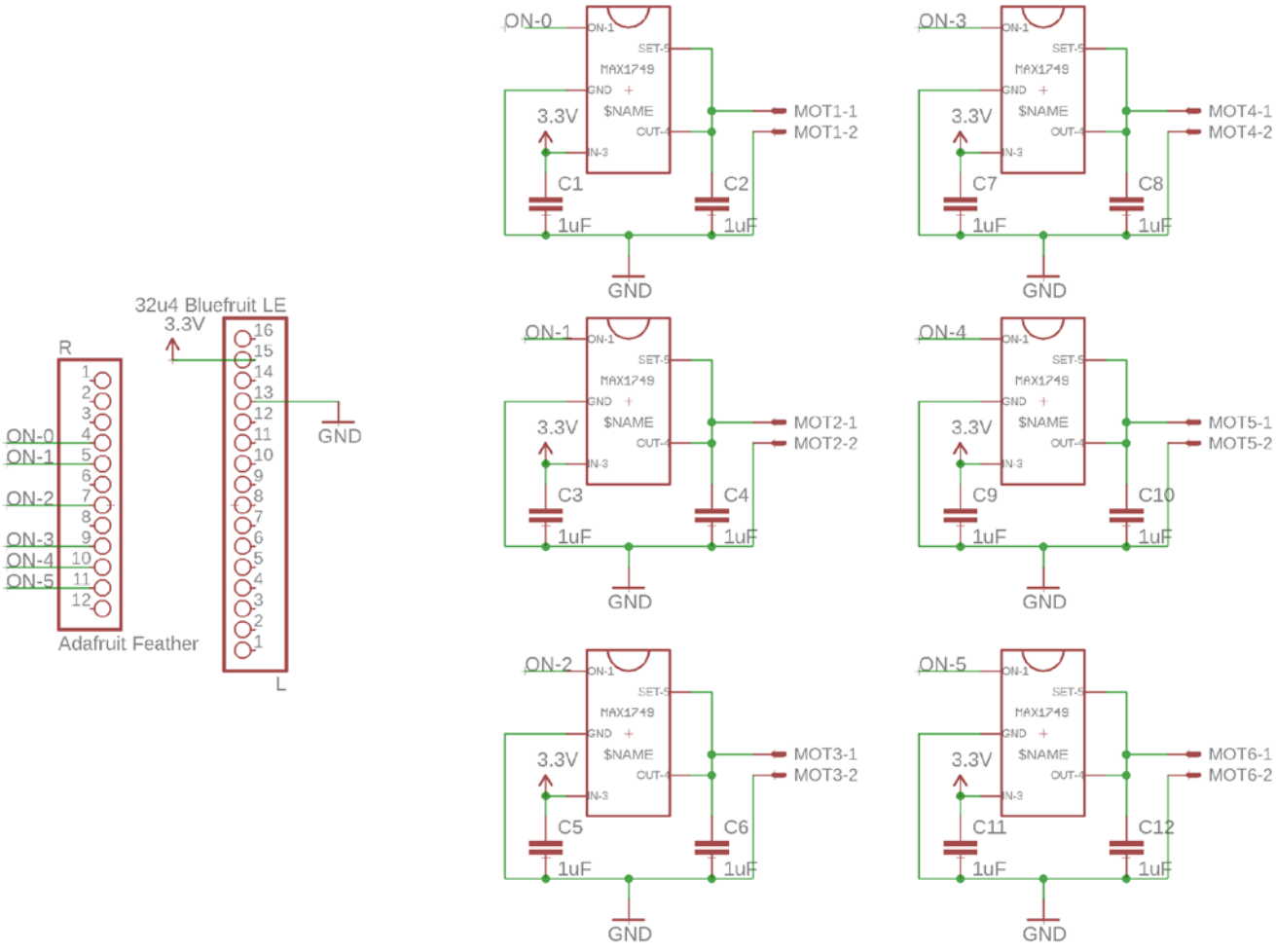


Figure 3. Glove 2.0 Schematics

Custom PCB: For the glove 2.0, a custom PCB was created that allowed for the integration of most of the components into a single chip. It also allowed to design the glove with modularity in mind to improve build quality and make it easier to fix the glove in case of problems. The custom PCB schematic can be found above in figure 3. The PCB itself had a couple of components to it. The first was the central connection made with female pins that allowed for main microcontroller to dock onto the board using male pins. This allowed for the board to be replaced if necessary without having to get rid of the entire board. The second component was the hubs for the motors. These hubs were made of two capacitors, a motor controller and a JST male connector. The motors were then attached using female JST connectors which allowed for easy attaching and detaching of motors if they broke. The final product can be seen below in figure 4.

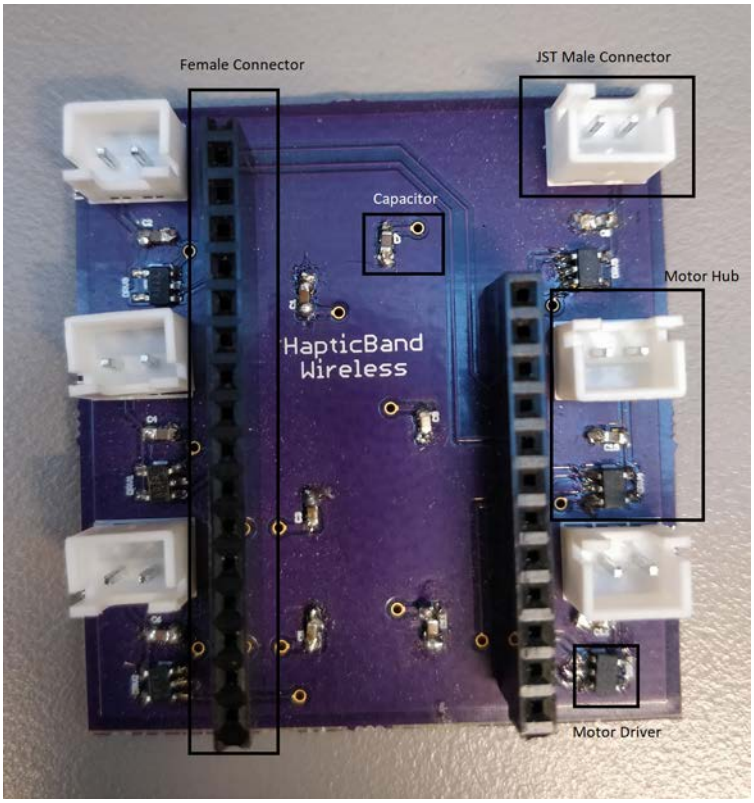


Figure 4. Board Diagram

Wiring: For the wiring of the glove, a couple of specific techniques were used to protect the wires from wear and tear. The first was the use of heat shrink along the wires to help protect the wire as well as organize wires together. Glue heat shrink was used in specific places to close spaces and add a level of waterproofing to the design especially close to the motors in the finger tips as they are the area most exposed. The JST female connectors where crimped on the wire to make sure connections are secured. Velcro was attached to the wires and on the glove to help maintain wires in place. The final product can be seen in figure 5 below.

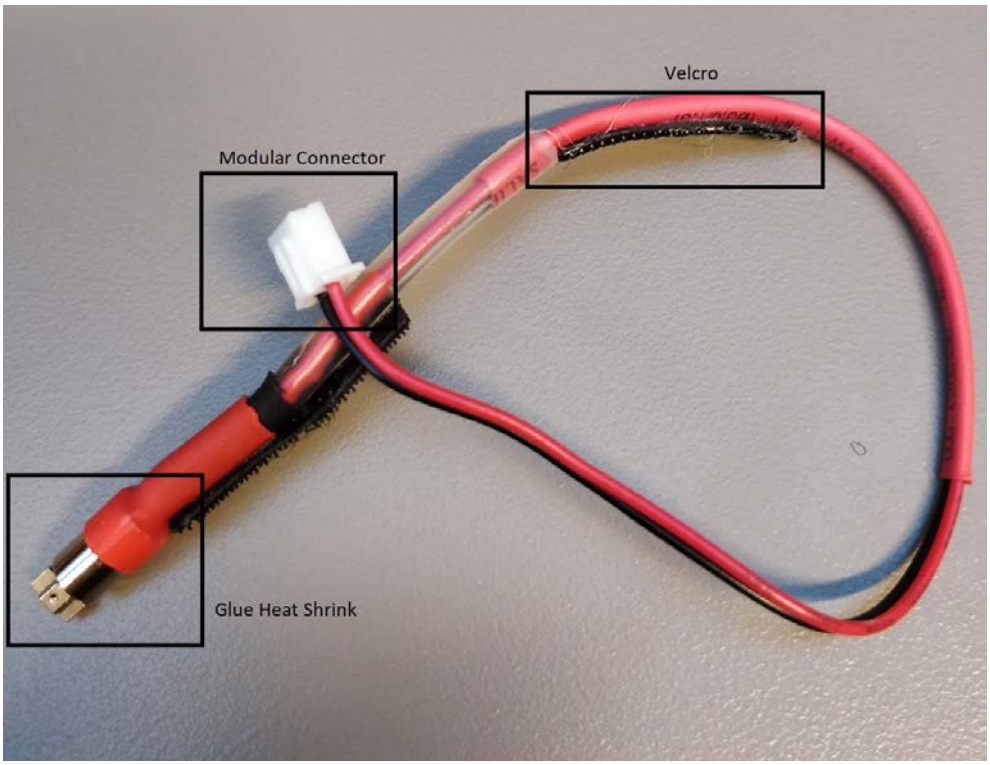


Figure 5. Wire Diagram

Haptic Motors: The haptic motors were secured to the glove using custom 3D printed housings. The housings themselves were hand sown onto the glove. The housing ultimately secures the motors down and make sure that it won't go loose while still allowing the motor to vibrate correctly. See the final product below in figure 6.

Figure 6. Motor Housing (Left) and the Motor Inside the Housing (Right)

Software Implementation

The software implementation of Glove 2.0 is made up of two main components, the actual glove which is controlled using a microcontroller and the software running on a separate PC which connects with the vision system. The two components interface using the Low Energy Bluetooth GATT profiles which allows the connecting systems to communicate through defined characteristics and services.

Low Energy Bluetooth (BLE) works by allowing the central device (PC) to connect to multiple peripheral devices at the same time. Once an exchange of information is required special ports are set up for each device to communicate. Connections are not necessarily active all the time, rather they are passively connected to preserve energy and make for a more efficient protocol. BLE has many advantages

over traditional Bluetooth, but probably one of the ones that makes it ideal for wearable technology is the battery saving capabilities. BLE can use two different types of communication protocols GAP or GATT, for this project GATT was used as both the microcontroller and most PCs support it. [2]

GATT is a Bluetooth protocol built on top of ATT which is a generic data transfer protocol. The best way to understand GATT communication is to think about the main software running on the PC as the client that is trying to communicate with the server which is the microcontroller that is attached to the glove. The server has several services it can perform which in turn have characteristics inside of them. These characteristics can be thought about as attributes or single data points (ex. State for Motor 1). Services and characteristics are organized in a look up table which uses UUIDs are unique keys. UUIDs can either be a 16 or 128-bit number is. The protocol has predefined services which specific characteristics. The glove custom services and characteristics for which UUIDs are 128 bits that are randomly generated. (See Figure 7 for architecture of the GATT Protocol) [12]

From the client side, it can pair with the server through the Bluetooth protocol. Once paired it can use the GATT data protocol to send data requests to the server. By knowing the UUIDs of the services and characteristics it wants to use, the server can access them to either read or write them depending on the configuration of each characteristic. The server can then constantly be checking for changes in its characteristics attached its services.

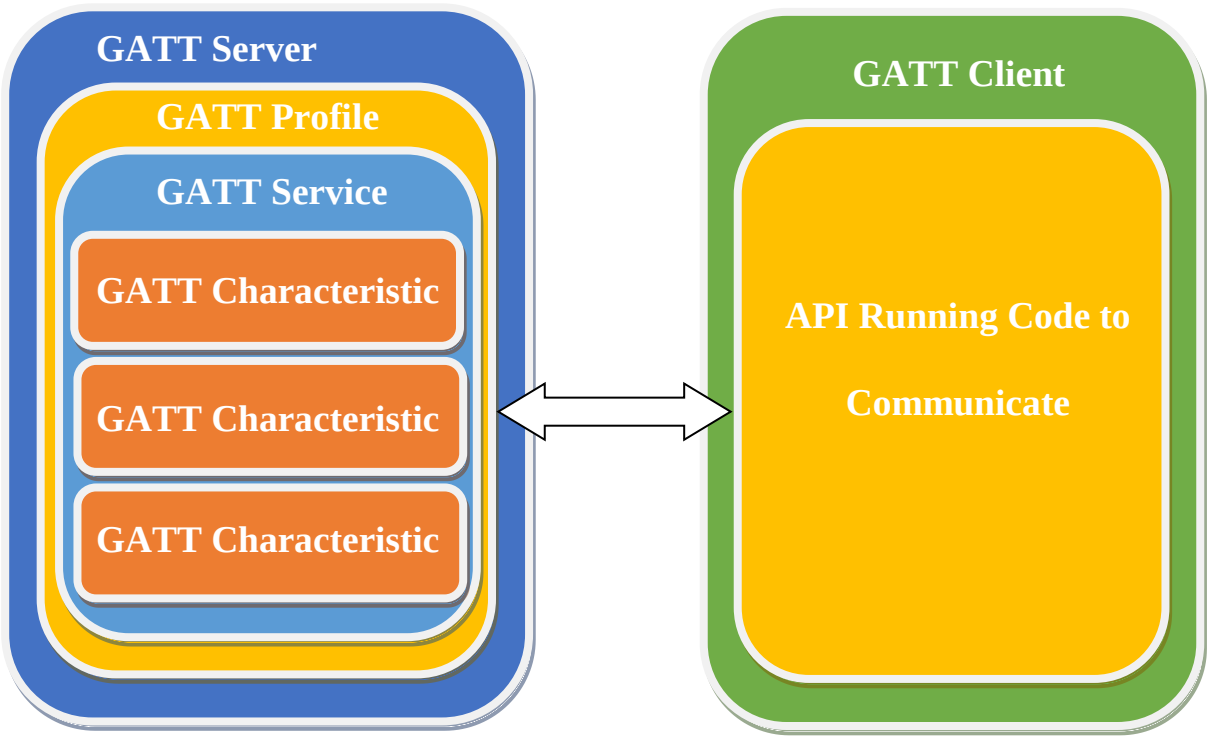


Figure 7. GATT Protocol Architecture

The microcontroller used for this iteration was the Adafruit Feather 32u4. To program the microcontroller, the Arduino IDE was used which supports the Adafruit Feather Library which is written in C. This library is necessary to be able to interface correctly with the microcontroller when deploying the code and making use of the Bluetooth functionality that the chip provides. [7]

Specific to the architecture behind Glove 2.0, there was a single service defined for the control of the motors. Inside that service there was a single characteristic which defined the state of all the motors. The microcontroller was consistently checking thousands of times a second for any changes in the characteristic made by the library to trigger motors. Once changes where found it would read the new data and act based on it. For this custom code was written for the microcontroller to interface with the GATT commands and the motors. (See Figure 8 for a diagram of the architecture.)

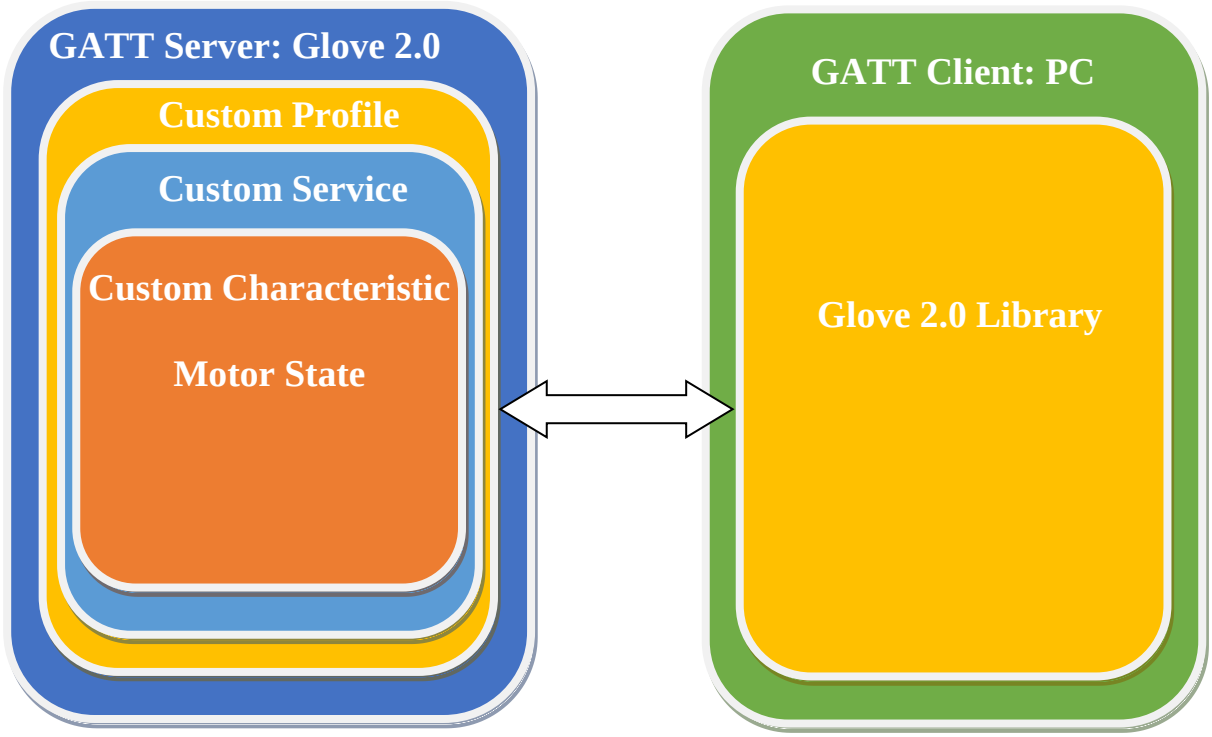


Figure 8. GATT Protocol Architecture for Glove 2.0

Going deeper into the details behind the microcontroller code, there are a couple main functions that the microcontroller performed. Once the microcontroller turned on, it went through a setup process where the services and characteristics were set up and checked. To do this, the UUIDs for the services and characteristics were set to a static value and tested to make sure they had been correctly set. Once the setup was complete, the microcontroller entered an infinite loop where it would check for changes on the characteristics. If a change was detected, then the characteristic was read. Characteristics were modelled as value system where specific values are matched to different functions and patterns. The value scheme is shown in table 4 below. Once read, the buffers were parsed, the motors specified were vibrated or the patterns were enabled. For the complete code see Appendix B.

Table 4. Value Scheme for Functions and Patterns

Pattern	Value
Reach Forward	0x0
Left	0x1
Right	0x2
Up	0x3
Down	0x4
Step Forward	0x5
Step Backward	0x6
Stop	0x7

Motors were controlled by the microcontroller using the digital IO library provided by microcontroller library. This is done by setting pins as either inputs or outputs enabling the functions of reading for inputs and writing for outputs. The motor pins that connected the motors and main chip, were defined as outputs. [11] Once defined, the microcontroller program can trigger the motors HIGH or LOW to turn them on or off. In turn the motors will give haptic feedback to the user in one of the direction predefined or through patterns.

The second component to highlight is the Bluetooth library that allowed the glove to be used in a variety of applications. The Bluetooth library was written in C# specifically to be used with devices running Windows. It made use of the same GATT protocol, only that this time client-side protocol was used. This meant that instead of having services and characteristics, the client will make use of the services and characteristics defined by the server (the glove).

The library has two main functions that it performed. First, verify the connection to the device, including the service and characteristic associated to it. For this the library scans the devices that have been paired with the computer and select the device that matches the name, service, and characteristic it is looking for. The Windows BLE library is used to perform these actions [13]. The second function is to define the process of running a motor or patterns defined by Table 4. The function takes a single parameter defined by the value associated to a specific pattern. It puts together a package that can be read by the microcontroller on the glove and it sends it over BLE using the GATT protocol.

To test the library, an application created which tested basic functionality such as running motors to signal directions as well as running patterns to give more complex commands. The test application required the glove to be paired to the computer on which it was running. (See figure 9 for the UI of the application) The full code for both the library and application are included in Appendix C and D respectively.

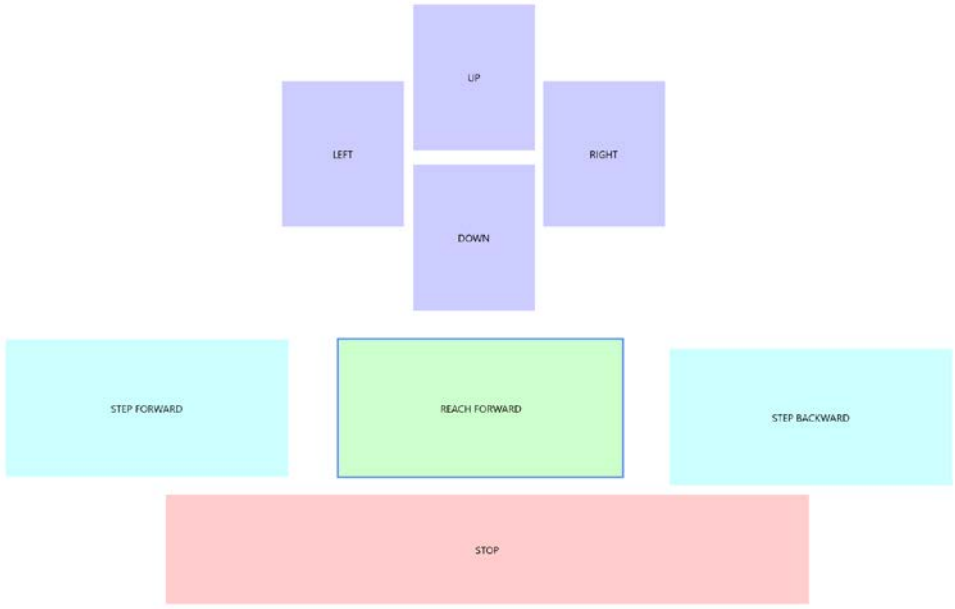


Figure 9. Test Application for Bluetooth Glove

Results

Figure 10. Finished Prototype of Glove 2.0

Following the first iteration, Glove 2.0 was able to advance the glove project in key areas. This iteration of the glove has started to be tested in specific research experiments and studies. It is expected for this version of the glove to be used even further in the next couple of semesters as they are implemented into increasingly studies and tests which currently use the Glove 1.0.

Table 5. Feature Summary for Glove 2.0

Main Features	Strengths	Weaknesses	Conclusions
Wireless	-User mobility increased	-Battery life concerns -Bluetooth reliability	Wireless improved the user experience greatly, although it presented drawbacks that need to be worked on.
Modular Design	-Easy to Fix -Variety of Configurations	-Cable management needs to be improved	Modularity made the overall glove design to be better.
Developer Extensibility	-Library for easy interfacing -Customizable patterns	-Can still improve to make latency and reliability better (linked to Bluetooth issues)	Having a proper library for developers to use will help the glove be adopted much more easily.

Wireless Capabilities: The wireless capabilities allow users an extended degree of freedom. Compared to Glove 1.0, users feel like they can move their hands around much more and feel more comfortable as they can reach faster in case they want to touch or feel something with not tension from a cable slowing them down. More importantly, the experience itself was not affected by the wireless capabilities as the glove still performed well in its task of navigating people through commands, for the most part.

The wireless capabilities introduced two specific issues. The first being the need of a battery to power the device which means battery life is a concern. Given that the glove used a microcontroller with low energy Bluetooth, battery was efficiently used. The battery life of the glove was about 4-5 hours which is not bad but needs improvement. The second drawback was related to the Low Energy Bluetooth implementation used by the Windows where it drops the connection of the Bluetooth design every time the program ends. So re-pairing is necessary every time it will be reused. Specifically, this problem became even more present when the glove is used with a device like the HoloLens. This is not something can be necessarily fixed easily as it requires Windows to change some of its functionality, but some fixes can be introduced.

Modular Design: The second areas that the Glove 2.0 has been a great improvement is the build quality and ease of fixing the glove in case of problems. In the time that the glove has been built, there has been a couple build issues that have been worked through, but overall it has been much easier to build as replacement motors can be made and have on hand it is easy to just connect or disconnect motors. Chips can also be easily swapped to test new code which makes the iteration process faster. Additionally, different motors configuration can be easily done depending on the numbers of commands wanted. Configuration with 3,4,5 or 6 motors are all possible. The main drawback related to cable management as well as the motor placement. Due to the modular design, organizing cables was hard as there were

multiple possible configurations and wiring paths that could be used, therefore a better approach to modular cable management must be introduced.

Developer Extensibility: The last area that has really been highlighted is the ability to create patterns on the glove. It has been a nice surprise to users to feel different unique patterns being created using the glove. From patterns that symbolize movements forward, backwards or sides as well as more intricate patterns to command the user to stop or turn around. This connects to the design specification outlined around the extensibility of the Bluetooth library to be used by developers. The glove will allow different teams and studies to create patterns and glove commands that can be tweaked easily to match their experiments or needs. The main drawback of the library created was due to the overall reliability issues of the Bluetooth libraries for Windows.

Overall, Glove 2.0 was a great iteration that placed the glove project closer the original concept. It improved in many capabilities from the first generation. It also helped bring up key questions regarding where the project could go in the future and key functionality that could be added to further the project. It was interest to the team to not try to perfect the current iteration by continuing to work on build quality, but instead move forward into thinking what else the glove could do other than just vibrate in patterns. What new functionality could make the glove more useful?

Chapter 4 : Glove 3.0: The next steps of the glove

Design Specification

The previous generation of the glove improved on the prototype from the first generation by adding key functionality like wireless capabilities and more developer control over the hardware as well as better build quality. The Glove 2.0 was a very solid interpretation of the original concept of a glove that was able to communicate with the user. For Glove 3.0, the key driving question was "What else can the glove do to make it more useful?" In some way Glove 3.0 is in many ways a parallel iteration to Glove 2.0 as it doesn't really build up on the functionality that Glove 2.0 had but rather tries to explore new areas that Glove 2.0 didn't.

When designing the third generation, the idea was to improve the glove by thinking freely at what new functionality could added that would give the user more control over the system. In some way leaving behind what had already been done and look forward. In the past two generations the glove only responded to commands from the computer, for this generation the main design specification criteria were to make the glove a two-way communication device.

Two-Way Communication: To achieve this design specification, a new Bluetooth library was created to allow commands to be send from the glove to the computer with the Bluetooth communication protocol. This new library compliments the library from Glove 2.0 which allowed the computer to send commands using the Bluetooth library.

A new scenario was developed to help guide the design process of two-way communication functionality. Thinking inside a supermarket situation, the user might want to command the system to look for a specific item inside the store. The system would work in the following way. It would read a list of categories of items to the user. The user can select a category. Then the system would read a list of

items in that category. The user can select and item or go back. Once selected, the item is added to a shopping list from which the overarching system takes items and looks for them. This was a key yet easy scenario to test the two-way communication on.

To fulfill the scenario there are three main commands that the glove had to enable, and system had to be able to get notification of. The first is the ability of the user to select. As the system reads the items, the user can select the category or items. The second is back. As the system is reading items, the user should be able to go back given that the category they chose was not the write one. The third, is the ability of the user to go back and forth through the category and items list. If the system is simply reading categories or items, it means that the user must be fast enough to select at the correct time. This gives further control to the user and ultimately makes it a more challenging test for the two-way communication system.

The select command is simply defined at the bending of the index finger in a tap motion. To implement this command, a flex sensor can be used to detect finger movement. A flex sensor is basically a resistance which changes depending on the degree of flex that is applied to it. Similarly, the back command will be implemented in the same way using a flex sensor attached to thumb. (See Figure 11 for details)

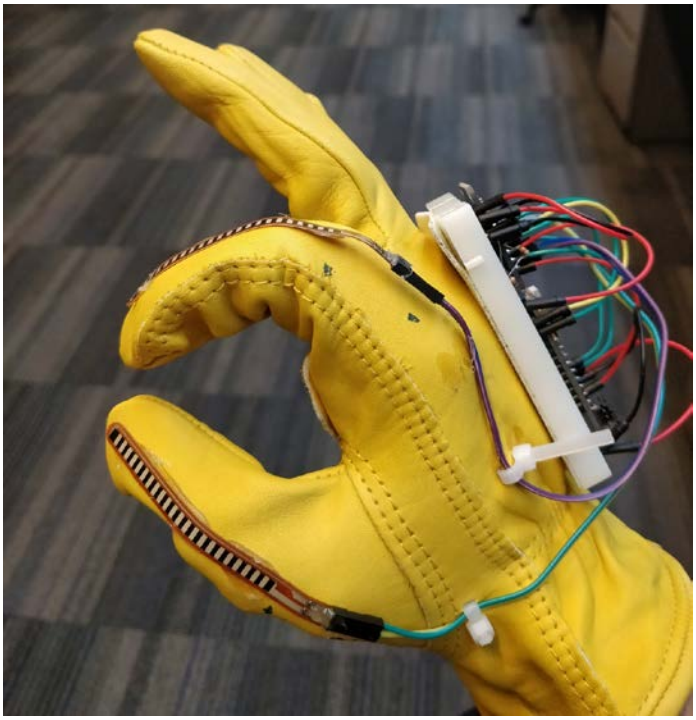


Figure 11. Index and Thumb Bend Example

The back and forth commands could potentially be implemented with a similar approach using flex sensors. A more interesting approach to test a new interaction method is the use of gestures. Using an accelerometer, gyroscope and magnetometer, gestures can be inputted into the system. The back and forth gestures will be modelled as the back and forth movement of the hand in the Y direction. (See Figure 12 for details)

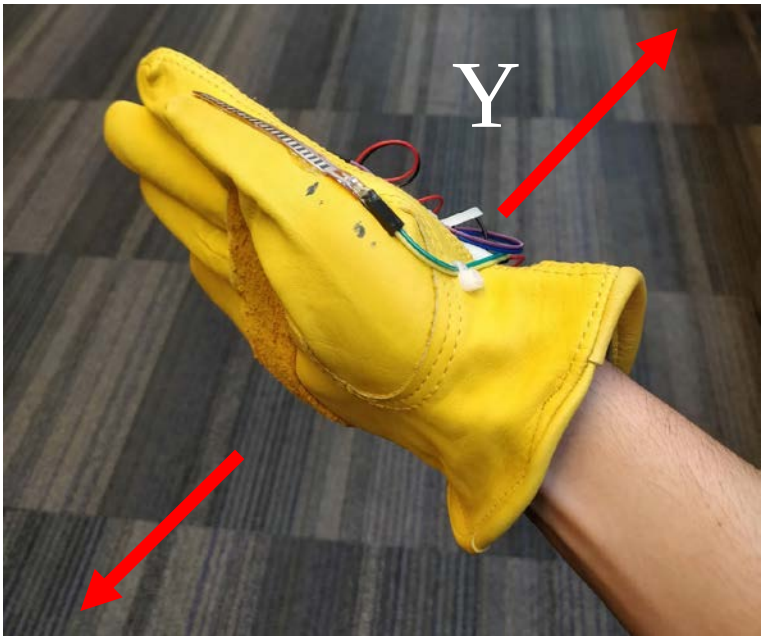


Figure 12. Back and Forth Command Example

Hardware Specification

Table 6. Glove 3.0 Specification

Main Chip	**Adafruit Feather 32u4 BLE [7]**
Haptic Engines	None
Glove	Yellow Leather Work Gloves
Board	Custom printed board with connections for the main chip as well as the 4 haptic engines
Connectivity	Wireless (Chip supports Bluetooth)
Accelerometer, Gyroscope and Magnetometer	LSM9DS1 – 9DOF [8]
Flex Sensor	Short Flex Sensor [3]

Implementation

For the full architecture diagram of the Glove 3.0 implementation look at Appendix E.

Hardware Implementation

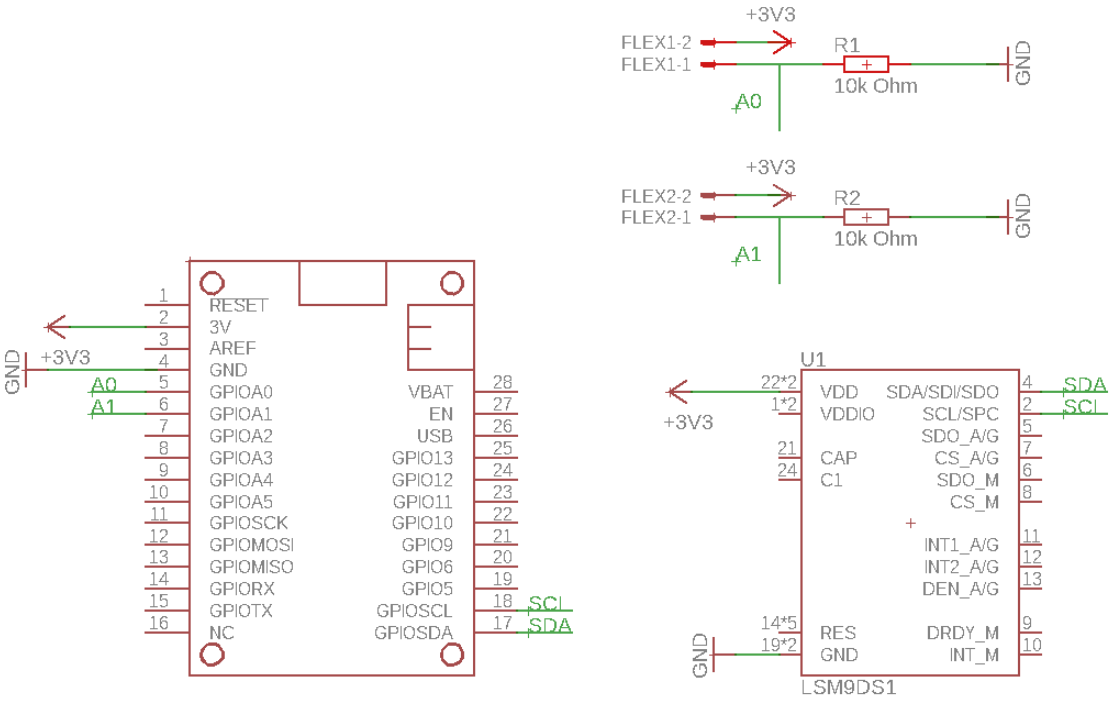


Figure 13. Glove 3.0 Schematic

The prototype of Glove 3.0 was built using a breadboard to connect all the components which was attached to the actual glove. The reason behind using a breadboard was to be able to prototype freely as this was the first time that some of the components were being used. On the breadboard there were two main components: The microcontroller which in this case was the Adafruit 32u4 and the LSM sensor

which was the Adafruit LSM9DS1 (Seen in figure 14). The microcontroller and the LSM chip were connected using the I2C bus (Inter-Integrated Circuit) which is a simple protocol that only requires two wires. This uses the SDA (Data) and SCL (Clock) pins on the microcontroller to collect data. [17] The connections can be seen in the schematic above in figure 13.

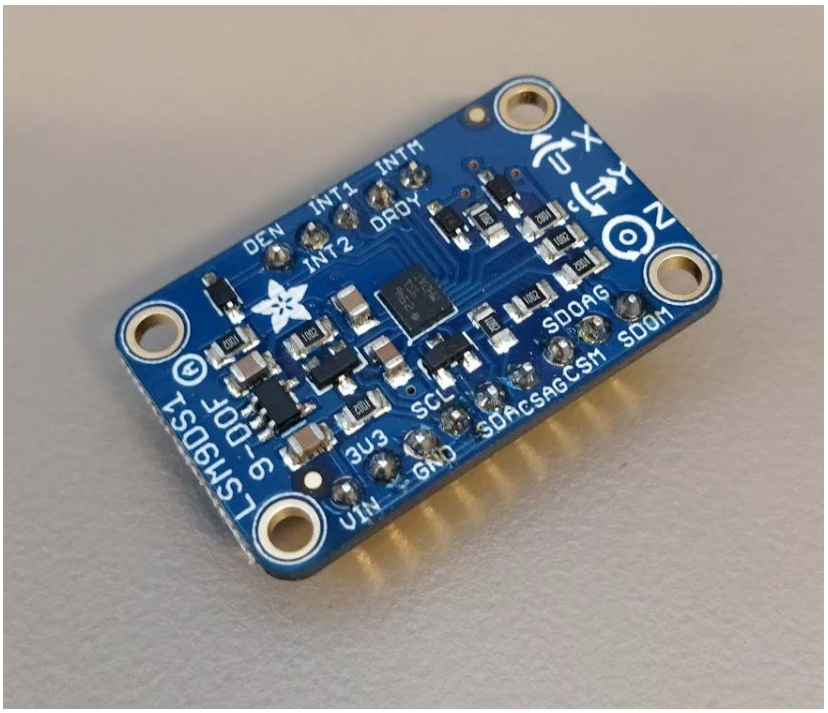


Figure 14. LSM9DS1 Chip

The other two main components of the glove 3.0 hardware implementation were the two flex sensors attached to the index finger and thumb. (See figure 15) These sensors were connected directly to the microcontroller with a pulldown resistor to create a voltage divider which allows to reduce the voltage being inputted back to the microcontroller to not burn (See schematic above). [9] The sensors were attached to the glove using hot glue on the back. Cables were then soldered onto the ends and hot glued to increase durability. The cables were then connected back to the microcontroller through analog pins where the voltage could be measured.

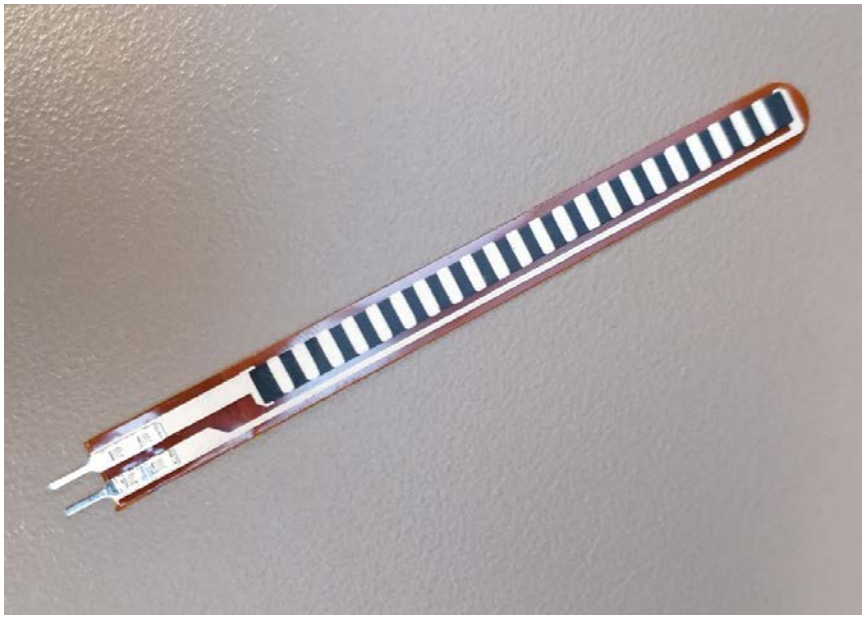


Figure 15. Flex Sensor

To connect everything together, jumper cables were used as they allow for easy prototyping. The breadboard containing the main components was secured using adhesive to the glove and zip ties were used to secure cables and keep everything organized.

Software Implementation

The library was built to allow commands to be sent from the glove to the computer was based on the BLE GATT Protocol, which was explained in the previous chapter. For this implementation, the glove still maintained the role of the server and the computer the role of the client as to maintain the overall architecture from Glove 2.0. The key difference between the implementations is the use of subscription enabled characteristics. The idea behind subscribing to characteristics is to be able to get notifications whenever a characteristic value changes. This is done through characteristics that are configured to allow for notifications. Once notifications are enabled, the client can start listening to changes in the characteristic value. If a change is detected, the client can retrieve the new value. [2]

Specific to the architecture behind Glove 3.0, a custom service was defined in the glove with a generated UUID. Attached to that service was a custom characteristic with a generated UUID. This characteristic represents the action the user performed (i.e. Flexed or moved hand). This characteristic was slightly different than the one used for the previous Glove 2.0 implementation. This time the configuration of that characteristic was given properties to be able to handle notifications. This was done by setting the correct flags (0x08) on the properties parameter with which the characteristic is defined. (See figure 16) [12] The glove's configuration is ready to take notifications requests.

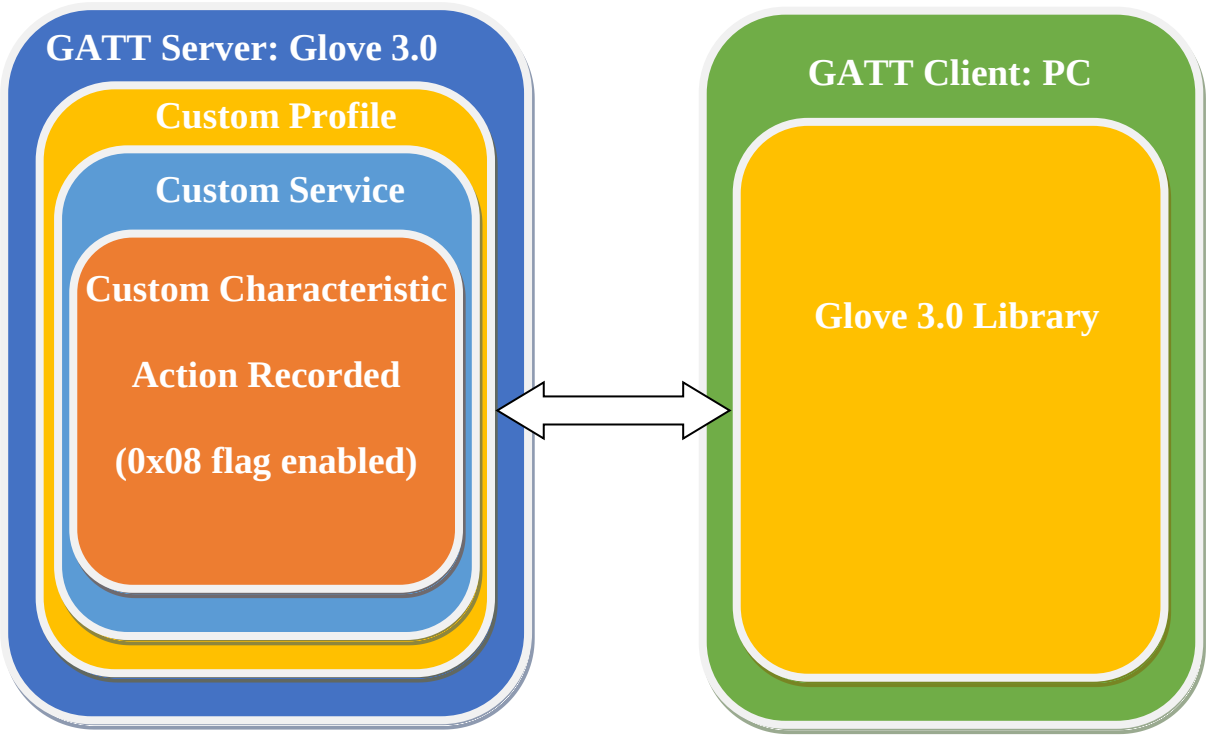


Figure 16. GATT Protocol Architecture for Glove 3.0

The characteristics in the glove were updated based on the state of the different sensors attached to the microcontroller, two flex sensors and one accelerometer/gyroscope/magnetometer (LSM) sensor. A value scheme was created for the characteristic to represent the various actions that the user could perform given by the following table. This scheme is used both in the glove and in the library that runs on the computer to be able to correctly identify the actions performed.

Table 7. Value Scheme for Input Actions

Action	Value
Nothing	1
Index Bend	2
Thumb Bend	5
Left Gesture	3
Right Gesture	4

The flex sensor is controlled using the Analog IO library. Like the Digital IO library, it allows to define pins as either input or output, where the flex sensor was an input. The flex sensor required an Analog IO library as the signal it emits is analog by nature as it reflects a voltage. [11] The values read from the flex sensor were normalized and processed based on a scale to extract whether the sensor was flexed or not. This approach was done on both the flex sensors. This data is then converted into the value scheme above if a flex is detected.

The LSM sensor is controlled by a proprietary library created by Adafruit, the manufacturer of the sensor. The library takes care of the setup process of the sensor and its calibration. As a developer, the only options that can be tweaked have to do with the detection ranges that change the precision of the sensor. For the glove, this configuration was left to its default value. The library summarizes most of the complexity of the sensor and makes it easy to simply retrieve values for each of the parameters. [9]

In this case, the main parameter that was used to detect movement was the acceleration in the y axis. (See figure 12 for details on the direction). Using that acceleration parameter, two measurements were taken with 100 milliseconds of delay between them. Then a line was plotted between the two points and the integral of the was taken to get a velocity. That velocity was then normalized and compared to a set of bounds to extrapolate the direction of the movement. This approach proved to give less false

positives and therefore a higher accuracy at detecting gestures. The results were then converted to the value scheme above. (The microcontroller code can be found in Appendix F.)

On the client side, a C# library received notifications of the changes in the characteristics. This library used a similar approach as the one made for Glove 2.0. It has two main components to it. The first is the connection and verification function. Like the previous library, it makes sure that the correct UUIDs are provided, set ups the connection to the device making sure that the UUIDs match and the device name is correct. The second component of the library is the key to allowing notifications of changes in the characteristics to show up. This is done through an asynchronous process which enables notifications on the glove through making changes on its configuration. Once this is done successfully, the developer can use the C# event framework to set up a callback function that will run every time the characteristic value changes. The callback function structure makes use of the C# event system. Inside that callback function, the developer can read the new value and know what action was taken based on the value scheme. [12]

To test the library, an application was created to mimic the scenario discussed above regarding a shopping list. Inside the test app, the library calls on the glove and enables notifications, set ups the call back function for the events and processes the different options to allow the user to control take control over the functionality. The application allowed the user to select categories and items using their index finger, move through categories and items using gestures and go back using their thumb. The test application required the glove to be paired to the computer on which it was running. The set-up process is done once the start button is hit. It also allows for manual control to test different parts of the app. The center text which by default starts with the first category, “Cereal”, shows the current category or item. (See figure 17 for the UI of the application) The full code for both the library and application are included in Appendix G and H respectively.

Figure 17. Test Applications for Glove 3.0

Results

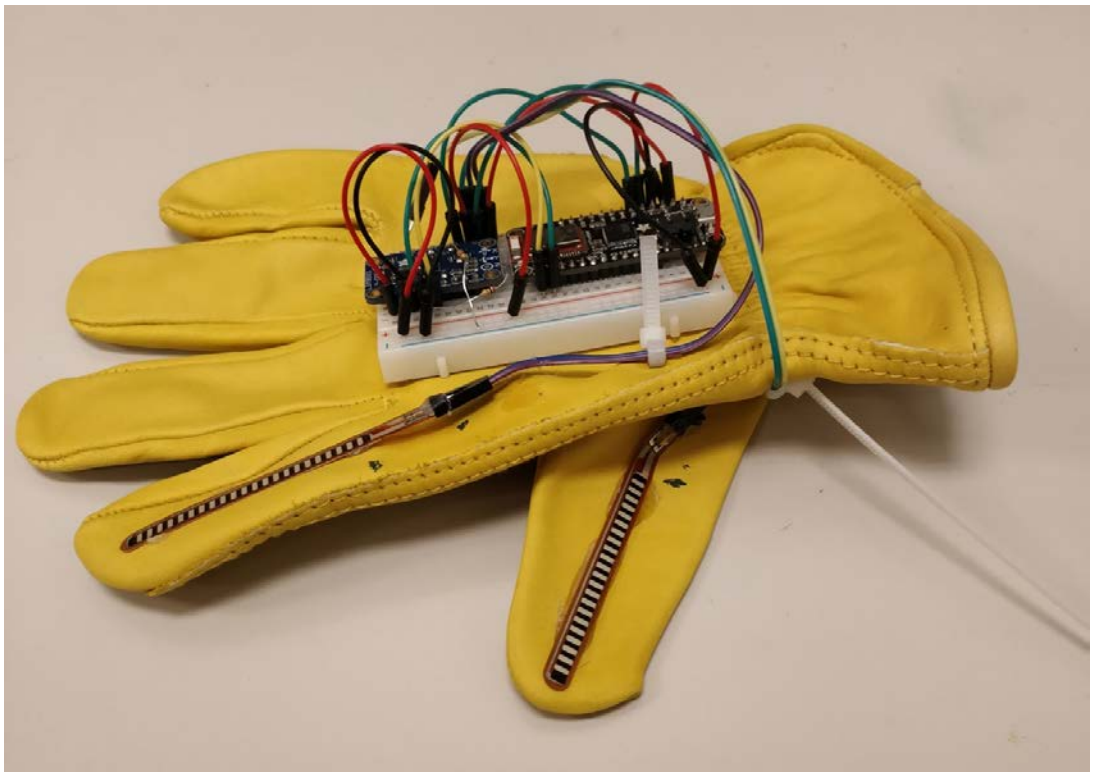


Figure 18. Glove 3.0 Prototype

Glove 3.0 was in many ways a parallel iteration to Glove 2.0. It didn't really touch at all on the haptic feedback using motors concept that Glove 1.0 and 2.0 did. In contrast, it served as an experimental prototype looking to understand if finger detection and gestures might be the way to help users communicate with the system and enable a broader spectrum of scenarios. The result is a prototype that left mixed feedback in a variety of areas. Overall, the device worked as intended and highlighted the potential of building a full glove that had both haptic feedback and user input functionality to be integrated into studies and research efforts, as well as trickling down to real life users.

Table 8. Feature Summary for Glove 3.0

Main Features	Strengths	Weaknesses	Conclusions
Finger Detection	-Intuitive to Use -High precision	-Sensors break easily	Should look at new approaches to doing finger detection
Gesture Detection	-Intuitive to use	-Sensor precision was low	A new data processing algorithm should be used to increase reliability
Developer Library	-Easy to implement	-Reliability issues related to Bluetooth	The library worked as intended yet like the library for Glove 2.0 needs some work on the Bluetooth side.

Gesture Detection: Looking at the gesture system, it felt innovative and convenient to use. During demos it worked decently well, where gestures would be captured correctly with only between 25%-30% false positives. Even though the false positive percentages seem low, it still a high number to have if the device were to be used in real world scenarios outside of research. Specifically, great part of the false positives was due to the inaccuracy of the sensors used, which would require a significant amount of work to clean the data more and use better extrapolation methods to get better results. Specific gestures like precise movements are increasingly more difficult to detect well using just sensors. If the detection where to be of a simple shake or imprecise movement it becomes much easier to detect. When compared to gestures done through computer vision, those algorithms tend to perform better and at lower costs than buying extremely precise accelerometers to try to get better results.

Finger Detection: The finger detection using the flex sensors was probably the biggest weakness of the prototype. Overall the detection of finger movement is something easy to do and which delivers very high reliability scores. In demos done with the device false positives only accounted for 5-10% of the captured movements. The problems with the flex sensors started happening after being used a couple times. The sensors tend to "break" easily which results in increasingly bad results. What is meant for breaking in this case, is not a complete malfunction of the sensor. Instead due to the nature of the sensor which uses resistance to measure flex, as the sensor is flexed over and over, the resistance tends to be

lower, until a point where the sensor being flexed or not look the same when testing the voltage across it. Therefore, from a build quality perspective the finger detection using flex sensors is not a good path to take. With this knowledge, a couple of alternatives explored which would enable better build quality, but some were either not miniature enough to integrate into a glove or presented other types of problems like high cost.

Developer Library: The developer library allowed for easy integration of the glove and its functionality into an application. It still suffered similar reliability issues as the library for Glove 2.0 which need to be improved.

Overall, Glove 3.0 was not necessarily a failure. Ultimately, it showed potential of using gestures and hand movements as possible input methods for the system. Additionally, it showed the ability to create devices that can have two-way communication with a central device to both read and write commands. Together with Glove 2.0, they painted a complete picture of what was possible to do with the glove which was of extreme value. It also leads to more questions that are currently helping guide the vision of the project and the next iteration of the interaction model for the vision system. Questions in the realm of wearables regarding other potential devices that could be used to help guide the visually impaired. Also questions in the realm of computer vision regarding the use of computer vision systems to looks at gestures and hand movement.

Chapter 5 : Moving Past the Glove

Debrief on Glove Concept

For three generations, the glove development left a lot of knowledge regarding the process of creating systems to interact with visually impaired people. The glove project became a leading example across the Penn State community and across other organizations of human centered design. It went on to be part of the award-winning proposal that was selected nationally by the IEEE. [10] For the next couple of years, the glove will be used for experiments and studies that deal with visually impaired individuals. Additionally, it has also lead to questions regarding the use of similar technology to be used on industrial settings as guidance systems for workers to be navigated across giant warehouses, constructions sites and other spaces.

The glove was a great prototype to identify the key functionality that users loved and need. As outlined across the previous chapters many were the strengths of the 3 generations and more importantly many were the weakness that still need to be worked on. Some of the key strengths include:

- **Wireless Capabilities:** The ability to be a separate entity from the computer running the main system, allowing for free movement.
- **Two-Way Communication:** The ability of the glove to receive and send commands for the user and the system to communicate.
- **Hands Free Design:** The glove allowed for the hand on which it was worn to still open and close with no obstructions.
- **Developer Extensibility:** The ability of developers to integrate the glove into their projects and tweak its functionality accordingly.

On the other hand, some of the key weaknesses include:

- **Build Quality:** The glove suffered from multiple build quality issues that although they were improved upon, keep being present throughout its lifetime.
- **Compatibility:** The library that controlled the glove was extensible only to a certain criterion of devices, which made the glove not fully compatible with devices like phones running Android or IOS.
- **Reliability:** Related to the build quality and compatibility issues, the glove was ultimately not a truly reliable device when thinking outside of a research setting where real users would wear it and use it daily.

Given this strengths and weaknesses, it was clear that there was still space to keep innovating and building new interaction models for users. It is key to keep innovating and thinking about what is the next frontier that needs to be broken and dealt in.

Next Steps: Spatial Understanding and Simulation

As the project moves away from the glove, its key to come up with solutions that can build on the strengths of the glove while improving on its weakness all meanwhile keeping the main functionality intact. One of the main solutions that the team has been looking at is the use of the Microsoft HoloLens. The HoloLens is a device designed for Augmented Reality experiences, but much of its technology can be applied to the same scenarios as the glove. The HoloLens is equipped with various cameras that track the environment around it and that give high resolution images at high frame rates. The HoloLens has a spatial sounds system that emits sound in 360 degrees. Let’s now explore how the HoloLens capabilities can be used to create a new user feedback system.

Spatial Sound and Audio: Audio capabilities of the HoloLens can take the place of the haptic feedback used on the glove. The first approach is to use the integrated speakers to simply give commands to the user. This is a good initial approach given that the speakers are already integrated into the device which makes it easy to use. One step further, spatial sound can be used to augment this functionality. Spatial sound technology gives the user the feeling that sound is being emitted from a specific direction or at a distance. Using spatial sound on HoloLens, sound beacons can be virtually placed around the user's environment. The beacons emit uninstructive sound that simply helps the user identify the direction towards they should move. (See figure 19) Using sound beacons in the space, the entire experience can be gamified so that the user is simply following sound beacons around an environment. Direct audio can be used in the case of asking the user to stop due to an obstacle or change in the path. [16]

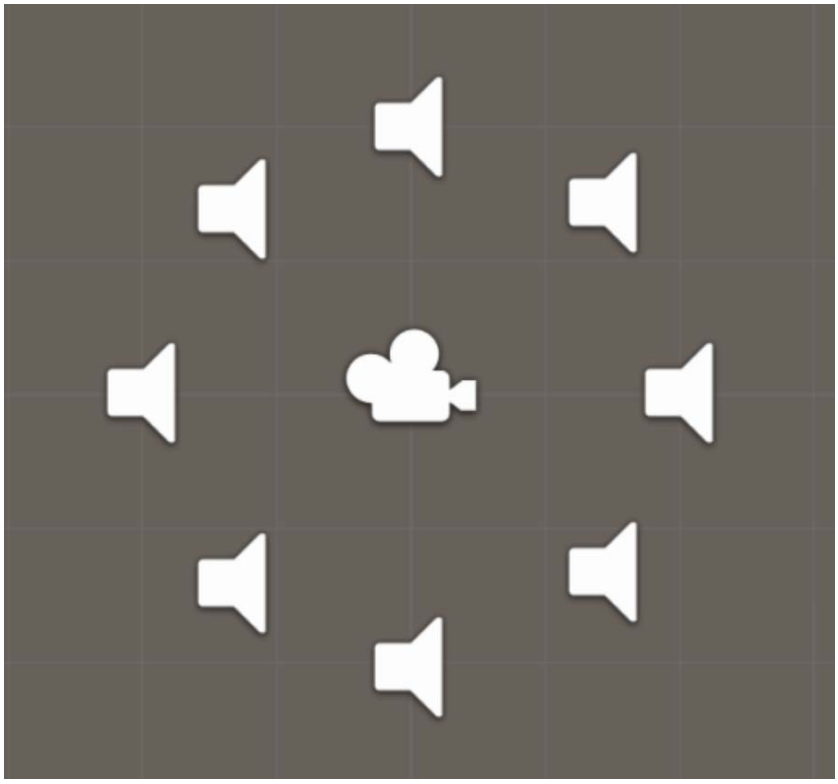


Figure 19. Spatial Sound Diagram

Camera System: The ability to take images and process them all within the same unit can unlock possibilities in terms of input methods. Image processing can be used to identify, and track gestures and hand movements made by users. (See figure 20) This can yield better results than using hardware to identify the same gestures and commands. Additionally, tracking the position of the hand can help guide it toward the specific object on a shelf or area. The main limitation is the necessity for the hands to be tracked in front of the cameras, but as the technology advances there is potential for cameras to be placed in more locations and with wider lenses. The same image processing technology can be used on any device other than the HoloLens, although the HoloLens is equipped with high end graphical processing units that allow for faster processing. The camera system is already a necessity as it is used by the computer vision system that identifies objects. [5]

Spatial Mapping: Spatial mapping is not a feature that will affect input methods but will improve the overall perception that the system has of its environment. Cameras and depth sensors are used to create 3D meshes of the environment that can be used to navigate the user and understand any obstacles or items that appear in front of them. This meshes are a 1:1 representation of the actual space. (See Figure 21) The meshes update multiple times a second which means that obstacles can be detected. This functionality is truly what differentiates the HoloLens from other devices and shows the most potential. This truly makes it possible to create a system that can understand the environment around a person, what obstacles and objects are in it and the true position of everything. This means that from a feedback perspective, better and more accurate commands can be given to users. Combining the spatial map meshes with spatial sound allow for sound beacons can be placed directly on top of the real physical space as the virtual and physical space are the same. This makes for more accurate and safe systems to guide users through even unknown spaces. [15]

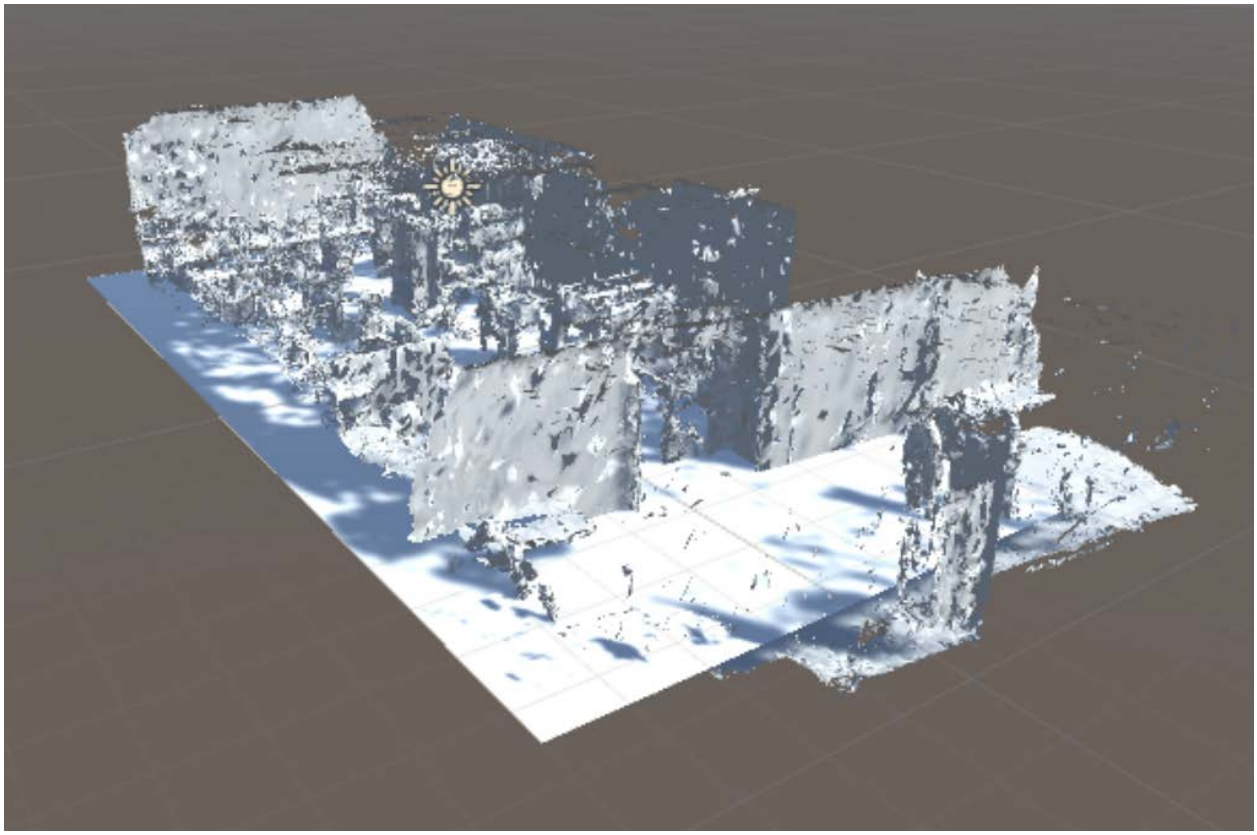
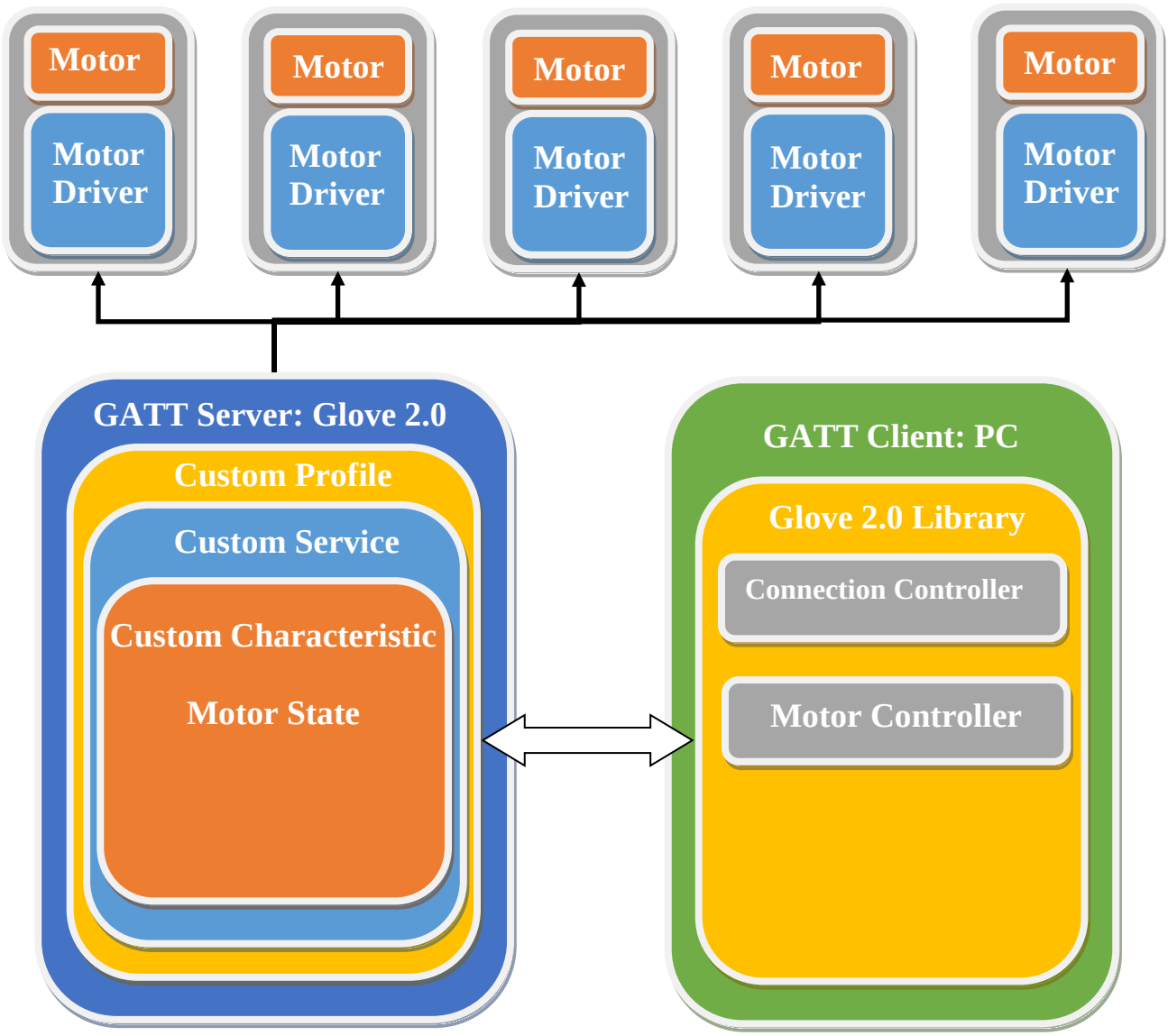


Figure 20. Spatial Map created by HoloLens

Overall, a device like the HoloLens has the potential to impact the project positively from a variety of perspectives. Starting with allowing for new input and feedback methods. Providing new innovative technology in spatial mapping and environment detection that can help navigate and track users in their space. Ultimately the HoloLens is a unified platform that encompasses both the necessary software and hardware in a single package. That unification comes at a steep cost of $3000. This will require to project to keep innovating to look at cheaper approaches to providing the same functionality. The HoloLens serves as a great prototyping platform for the time being. The hope is for the technology to become cheaper and available to a wider range of people, but while that happens the next moves will be in creating more democratized solutions.

Appendix A: Glove 2.0 Complete Architecture



Appendix B: Glove 2.0 Microcontroller Code

```
/*
 Code developed by Chris Pratt in conjunction with David de Matheu and Gus Smith

 The Bluetooth LE Glove Listener is part of the VCoS NSF project, and is intended to listen for updates to a BLE characteristic.
 This characteristic contains a byte array that indicates which motor(s) should vibrate to provide haptic feedback to the user.
 The current interface is as follows:
 [Motor (1 - 6 or 7). Each number corresponds to a motor, or 7 for all motors)]
 [Intensity (0..100 percent)]
 [Duration (1..100 percent of a second)]
 These values are sent in a byte array in the following format:
 [motor, intensity, duration]

 Every time the characteristic is updated, it will overwrite the duration of the previous run, terminating it immediately.
 */

#include <Arduino.h>
#include <SPI.h>
#if not defined (_VARIANT_ARDUINO_DUE_X_) && not defined (_VARIANT_ARDUINO_ZERO_)
  #include <SoftwareSerial.h>
#endif
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"
#include "Adafruit_BLEGatt.h"
#include "BluefruitConfig.h"

// Play with these (INTENSITY_SCALE has minimum of 3)
#define INTENSITY_SCALE 70
#define INTENSITY_MIN 25
#define INTENSITY_MAX 64
//Value Scheme
#define LEFT 1
#define RIGHT 2
#define UP 3
#define DOWN 4
#define STEP_FORWARD 5
#define STEP_BACKWARD 6
#define STOP 7
#define REACH_FORWARD 0

// Pin Mapping for IST Glove: MOT0 - 3 MOT1 - 8 MOT2 - 12 MOT3 - 23
int motorpins[7] = {
  0,13,12,10,6,5,3 }; // motor driver pins 0 through 5, ideally spaced 60 degrees apart

/* Pin Mapping for Dev Glove: MOT0 - 3 MOT1 - 8 MOT2 - 12 MOT4 - 19
int motorpins[6] = {
    3,8,12,19,23,14 };
*/

// Create the bluefruit object
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

Adafruit_BLEGatt gatt(ble);

// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
```

```
  while (1);
}

int32_t gattServiceId;
int32_t gattCharId;
uint8_t motor = 0;
uint8_t intensity = 0;
long duration = 0;

void setup(void) {
  Serial.begin(115200);

  for(int i = 0; i < 7; i++) {
    pinMode(motorpins[i], OUTPUT);
  }
  digitalWriteAll(LOW);

  if (!ble.begin(VERBOSE_MODE))
  {
    error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check wiring?"));
  }

  /* Perform a factory reset to make sure everything is in a known state */
  Serial.println(F("Performing a factory reset: "));
  ble.factoryReset();

  ble.sendCommandCheckOK(F("AT+GAPDEVNAME=HapticGloveB"));
  ble.sendCommandWithIntReply( F("AT+GATTADDSERVICE=UUID128=15-DB-5D-20-50-D4-43-70-A4-39-75-4E-
71-82-CB-54"), &gattServiceId);
  ble.sendCommandWithIntReply( F("AT+GATTADDCHAR=UUID128=15-DB-5D-21-50-D4-43-70-A4-39-75-4E-71-
82-CB-54,PROPERTIES=0x08,MIN_LEN=1, MAX_LEN=3, VALUE=0x000000"), &gattCharId);
  ble.reset();

  /* Disable command echo from Bluefruit */
  ble.verbose(false);
  ble.echo(false);
}

// Write to every motor
void digitalWriteAll(uint8_t state) {
  for (int m = 1; m < 7; m++)
  {
    digitalWrite(motorpins[m], state);
  }
}

void parsePacket(String packet) {
  String motorBuffer = packet.substring(0, 2);
  String intensityBuffer = packet.substring(3, 5);

  const char* motorStr = motorBuffer.c_str();
  const char* intensityStr = intensityBuffer.c_str();

  motor = constrain(strtoul(motorStr, NULL, 16), 0, 7);
  intensity = constrain(strtoul(intensityStr, NULL, 16), 0, 100);
  if (intensity != 0) { Serial.print("Int1: "); Serial.println(intensity); }
  intensity = map(intensity, 0, 100, INTENSITY_MIN, INTENSITY_MAX);
  if (intensity != 25) { Serial.print("Int2: "); Serial.println(intensity); }
}

bool checkCharacteristic() {
  // Read the buffer
  ble.println("AT+GATTCHAR=1");
  ble.readline();
  if (strcmp(ble.buffer, "OK") == 0) {
    // no data
    return;
  }

  // Parse our data
```

```
  parsePacket(String(ble.buffer));

  if (motor != 0) {
    return true;
  }
}

void runMotor() {
  // Reset the characteristic
  String command = "AT+GATTCHAR=1,0x000000";
  ble.println(command);

  duration = 1000000;
  intensity = 64;

  Serial.print("Running motor: "); Serial.println(motor);
  Serial.print("    Intensity: "); Serial.println(intensity);

  unsigned long start_time = micros();

  // Calculate length of the duty cycle
  word cycle_time = INTENSITY_SCALE * intensity;

  // Make sure every motor is off when running a new motor
  digitalWriteAll(LOW);


  // Keep running until the full duration has passed
  while(start_time + duration > micros()) {
    // Pulse the motors based on our duty cycle
    if(motor == 7) {
      digitalWriteAll(HIGH);
    }
    else {
      digitalWrite(motorpins[motor], HIGH);
    }
    delayMicroseconds(cycle_time);

    // If there's another byte waiting, read it right away
    if(checkCharacteristic()) {
      break;
    }

    // Pulse the motors based on our duty cycle
    if(motor == 7) {
      digitalWriteAll(LOW);
    }
    else {
      digitalWrite(motorpins[motor], LOW);
    }

    delayMicroseconds(((100 * INTENSITY_SCALE) - cycle_time));
  }
}

void loop(void) {
  motor = 0;
  intensity = 0;
  duration = 0;

  checkCharacteristic();
  // Check our motor characteristic to see if we need to run a motor
  if (motor == LEFT) {
    motor = 4;
    duration = 10 * 10000;
    runMotor();
  }
  else if (motor == RIGHT) {
     motor = 5;
     duration = 10 * 10000;
```

```
    runMotor();
  }
  else if (motor == UP) {
    motor = 2;
    duration = 10 * 10000;
    runMotor();
  }
  else if (motor == DOWN) {
     // STOP: 1s all 1s all
     motor = 3;
     duration = 10 * 10000;
     runMotor();
  }
  else if (motor == STEP_FORWARD) {
    motor = 3;
    duration = 5 * 10000;
    runMotor();
    digitalWriteAll(LOW);
    delay(100);
    motor = 2;
    duration = 10 * 10000;
    runMotor();
  }
  else if (motor == STEP_BACKWARD) {
     motor = 2;
     duration = 10 * 10000;
     runMotor();
     digitalWriteAll(LOW);
     delay(100);
     motor = 3;
     duration = 5 * 10000;
     runMotor();
  }
  else if (motor == STOP) {
    motor = 7;
    duration = 10 * 10000;
    runMotor();
  }
  else if (motor == REACH_FORWARD && intensity > 26) {
     motor = 7;
     duration = 5 * 10000;
     runMotor();
     digitalWriteAll(LOW);
     delay(50);
     motor = 7;
     duration = 5 * 10000;
     runMotor();
     digitalWriteAll(LOW);
     delay(50);
     motor = 7;
     duration = 5 * 10000;
     runMotor();
  }

  // Make sure every motor is off
  digitalWriteAll(LOW);
}
```

Appendix C: Glove 2.0 Library Code

```
/*
Developed by Daniel Troncoso and David de Matheu
Haptic Glove Bluetooth Library
*/
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using Windows.Devices.Bluetooth.GenericAttributeProfile;
using Windows.Devices.Bluetooth;
using Windows.Devices.Enumeration;
using System.Threading.Tasks;

namespace HapticGloveApp
{
    class HapticGlove
    {
        // Define BLE current device getters and setters.
        BluetoothLEDevice currentDevice { get; set; }

        //private members
        private string deviceName;
        private string serviceID;
        private string characteristicID;

        // Define the selectedCharacteristic variable.
        private GattCharacteristic selectedCharacteristic;
        // Define the selectedService variable.
        private GattDeviceService selectedService;
        private const int CHARACTERISTIC_INDEX = 0;
        private const GattClientCharacteristicConfigurationDescriptorValue
CHARACTERISTIC_NOTIFICATION_TYPE = GattClientCharacteristicConfigurationDescriptorValue.Notify;

        //constructor
        public HapticGlove(string theName, string theServiceID, string theCharacteristicID)
        {
            this.deviceName = theName;
            this.serviceID = theServiceID;
            this.characteristicID = theCharacteristicID;

        }

        //private private
        async Task<BluetoothLEDevice> getDevices(string deviceName)
        {
            foreach (DeviceInformation di in await
DeviceInformation.FindAllAsync(BluetoothLEDevice.GetDeviceSelector()))
            {
                BluetoothLEDevice bleDevice = await BluetoothLEDevice.FromIdAsync(di.Id);
```

```
                //Out of the list of devices available to the computer the specific one is
selected and added to the device list.
                if (bleDevice != null && bleDevice.Name == deviceName)
                {
                    currentDevice = bleDevice;
                    Debug.WriteLine(bleDevice.Name);
                }
                break;
            }
            return currentDevice;
        }

        private GattDeviceService selectService(BluetoothLEDevice device, Guid service)
        {
            return selectedService = device.GetGattService(service);
        }
        private GattCharacteristic selectCharacteristic(GattDeviceService service, Guid
characteristic)
        {
            return selectedCharacteristic =
service.GetCharacteristics(characteristic)[CHARACTERISTIC_INDEX];
        }
        private async Task<byte[]> read()
        {
            byte[] response = (await selectedCharacteristic.ReadValueAsync()).Value.ToArray();
            Array.Reverse(response, 0, response.Length);
            return response;
        }
        private async Task<GattCommunicationStatus> write(GattCharacteristic characteristic,
byte[] data)
        {
            return await characteristic.WriteValueAsync(data.AsBuffer(),
GattWriteOption.WriteWithResponse);
        }

        /*Patterns
        {
            REACH_FORWARD   = 0x0,
            LEFT            = 0x1,
            RIGHT           = 0x2,
            UP              = 0x3,
            DOWN            = 0x4,
            STEP_FORWARD    = 0x5,
            STEP_BACKWARD   = 0x6,
            STOP            = 0x7
        }
        */

        //public methods
        public async Task RunPattern(int pattern)
        {
            Guid serviceId = new Guid(this.serviceID);
            Guid characteristicId = new Guid(this.characteristicID);
            List<string> characteristicList = new List<string>();
            GattCommunicationStatus status = 0;
            byte[] data = { (byte)pattern, 0x7 };

            this.currentDevice = await this.getDevices(this.deviceName);
            Debug.WriteLine(this.currentDevice);
            this.selectedService = this.selectService(this.currentDevice, serviceId);
            Debug.WriteLine(this.selectedService.Uuid);
            this.selectedCharacteristic = this.selectCharacteristic(this.selectedService,
characteristicId);
            Debug.WriteLine(this.selectedCharacteristic.Uuid);
            status = await this.write(this.selectedCharacteristic, data);
            Debug.WriteLine(status);
        }

        public void setCharacteristicID(string id)
        {
```

```
            this.characteristicID = id;
        }
        public void setServiceID(string id)
        {
            this.serviceID = id;
        }
        public string getName()
        {
            return this.deviceName;
        }
        public string getCharacteristicID()
        {
            return this.characteristicID;
        }
        public string getServiceID()
        {
            return this.serviceID;
        }
    }
}
```

Appendix D: Glove 2.0 Test Application Code

MainPage.xaml.cs

```
/*
Developed by Daniel Troncoso and David de Matheu
Haptic Glove Bluetooth Library - Test App CS File
*/
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using Windows.Devices.Bluetooth.GenericAttributeProfile;
using Windows.Devices.Bluetooth;
using Windows.Devices.Enumeration;
using System.Threading.Tasks;
using Windows.UI.ViewManagement;

// The Blank Page item template is documented at
http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace HapticGloveApp
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        HapticGlove bluetoothDevice = new HapticGlove("HapticGloveB",
                                                      "15DB5D20-50D4-4370-A439-
754E7182CB54",
                                                      "15DB5D21-50D4-4370-A439-
754E7182CB54");
        public MainPage()
        {
            this.InitializeComponent();
            // Set the preferred launch view size to 360 * 550
            ApplicationView.PreferredLaunchViewSize = new Size { Height = 360, Width = 640 };
            ApplicationView.PreferredLaunchWindowingMode =
ApplicationViewWindowingMode.PreferredLaunchViewSize;

        }

        public enum Pattern
        {
            REACH_FORWARD = 0,
            LEFT = 1,
            RIGHT = 2,
            UP = 3,
            DOWN = 4,
```

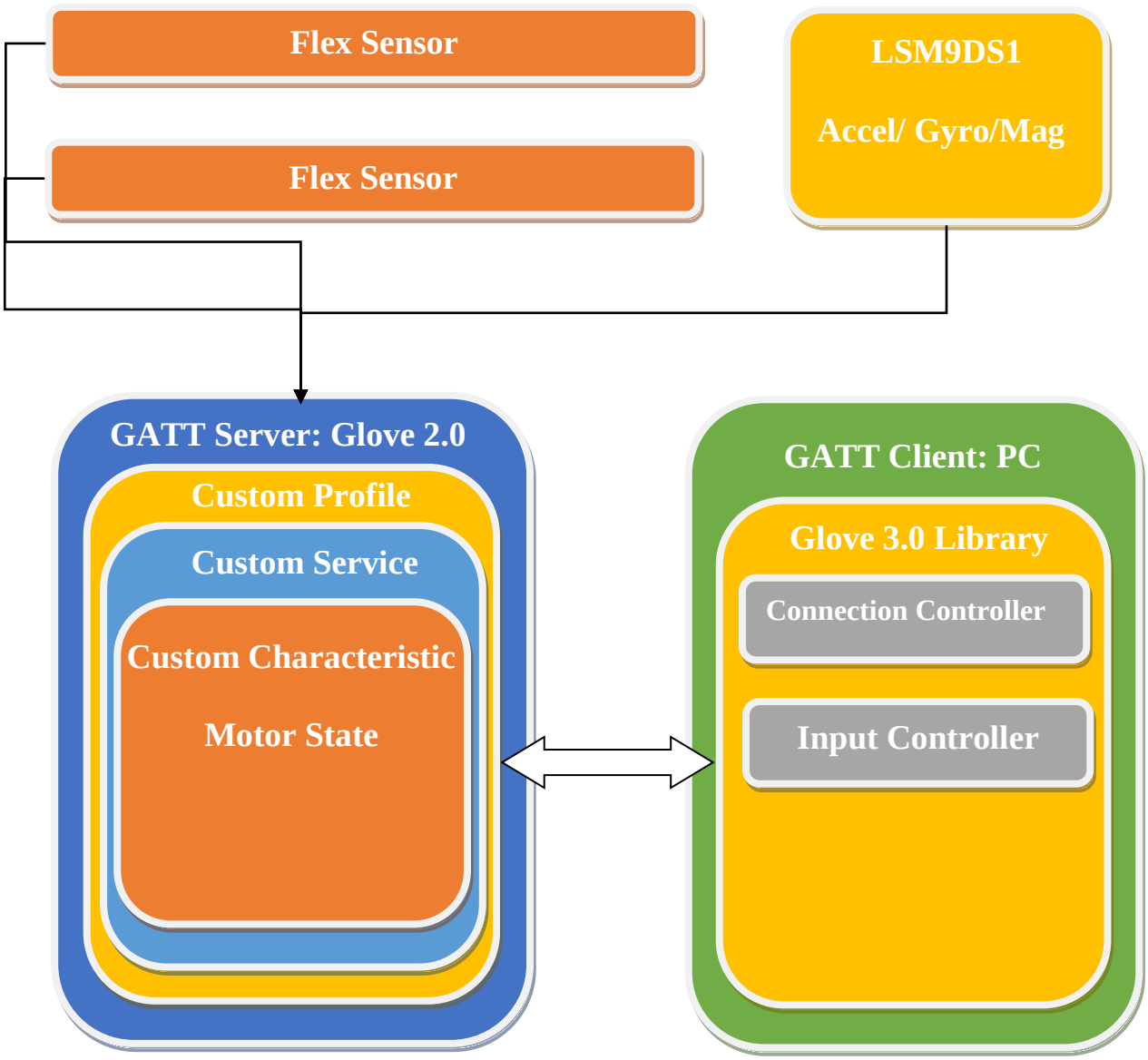
```
            STEP_FORWARD = 5,
            STEP_BACKWARD = 6,
            STOP = 7
        }
        //LEFT
        private async void Left_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.LEFT);
        }
        //RIGHT
        private async void Right_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.RIGHT);
        }
        //UP
        private async void Up_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.UP);
        }
        //DOWN
        private async void Down_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.DOWN);
        }
        //STEP_FORWARD
        private async void Step_Forward_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.STEP_FORWARD);
        }
        //STEP_BACKWARD
        private async void Step_Backward_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.STEP_BACKWARD);
        }
        //REACH_FORWARD
        private async void Reach_Forward_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.REACH_FORWARD);
        }
        //STOP
        private async void Stop_Click(object sender, RoutedEventArgs e)
        {
            await bluetoothDevice.RunPattern((int)Pattern.STOP);
        }
    }
}
```

MainPage.xaml

```
/*
Developed by Daniel Troncoso and David de Matheu
Haptic Glove Bluetooth Library - Test App XAML File
*/
<Page
    x:Class="HapticGloveApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:HapticGloveApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" Loaded="button8_Click">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Button x:Name="Left" Content="LEFT" HorizontalAlignment="Left" Margin="451,148,0,0"
VerticalAlignment="Top" Height="216" Width="180" Click=" Left_Click" Background="#330000FF"/>

        <Button x:Name="Right" Content="RIGHT" HorizontalAlignment="Left" Margin="837,148,0,0"
VerticalAlignment="Top" Height="216" Width="180" Click=" Right_Click" Background="#330000FF"/>
```

```
        <Button x:Name="Up" Content="UP" HorizontalAlignment="Left" Margin="645,34,0,0"
VerticalAlignment="Top" Height="217" Width="180" Click=" Up_Click" Background="#330000FF"/>

        <Button x:Name="Down" Content="DOWN" HorizontalAlignment="Left" Margin="645,272,0,0"
VerticalAlignment="Top" Height="217" Width="180" Click=" Down_Click" Background="#330000FF"/>

        <Button x:Name="Step_Forward" Content="STEP FORWARD" HorizontalAlignment="Left"
Margin="42,532,0,0" VerticalAlignment="Top" Height="204" Width="418" Click=" Step_Forward_Click"
Background="#3300FFFF"/>

        <Button x:Name="Step_Backward" Content="STEP BACKWARD" HorizontalAlignment="Left"
Margin="1024,546,0,0" VerticalAlignment="Top" Height="203" Width="418" Click="
Step_Backward_Click" Background="#3300FFFF"/>

        <Button x:Name="Reach_Forward" Content="REACH FORWARD" HorizontalAlignment="Left"
Margin="536,532,0,0" VerticalAlignment="Top" Height="204" Width="418" Click="
Reach_Forward_Click" Background="#3300FF00"/>

        <Button x:Name="Stop" Content="STOP" HorizontalAlignment="Left" Margin="279,763,0,0"
VerticalAlignment="Top" Height="162" Width="951" Click=" Stop_Click" Background="#33FF0000"/>

    </Grid>
</Page>
```

Appendix E: Glove 3.0 Complete Architecture



Appendix F: Glove 3.0 Microcontroller Code

```
/*
Developed by David de Matheu
Input Glove Microcontroller Code
*/
#include <Arduino.h>
#include <SPI.h>
#include <Adafruit_LSM9DS1.h>
#include <Adafruit_Sensor.h>
#if not defined (_VARIANT_ARDUINO_DUE_X_) && not defined (_VARIANT_ARDUINO_ZERO_)
  #include <SoftwareSerial.h>
#endif
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"
#include "Adafruit_BLEGatt.h"
#include "BluefruitConfig.h"

#define LSM9DS1_SCK A5
#define LSM9DS1_MISO 12
#define LSM9DS1_MOSI A4
#define LSM9DS1_XGCS 6
#define LSM9DS1_MCS 5

// Create the bluefruit object
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
Adafruit_BLEGatt gatt(ble);
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

//Parameters Declaration
int32_t gattServiceId;
int32_t gattCharId;
int prevValue = 0;
int prevValue2 = 0;

//Configure Bluetooth
void setupBluetooth(){
  if(!ble.begin(VERBOSE_MODE))
  {
    Serial.println("error");
  }
  //Name and Clear
  ble.sendCommandCheckOK(F("AT+GAPDEVNAME=InputGlove"));
  ble.sendCommandCheckOK( F("AT+GATTCLEAR"));

  //Services and Characteristics
  ble.sendCommandWithIntReply( F("AT+GATTADDSERVICE=UUID128=10-AF-FC-36-65-B9-43-91-BA-A9-DE-95-
27-E9-BA-66"), &gattServiceId);
  //Set Notification Property 0x18
  ble.sendCommandWithIntReply( F("AT+GATTADDCHAR=UUID128=97-61-0D-B9-BB-62-43-45-A1-C6-85-33-74-
28-86-9D,PROPERTIES=0x18,MIN_LEN=1,MAX_LEN=4,VALUE=0x0"), &gattCharId);

  ble.reset();

  Serial.println("Characteristics and Services");
  ble.sendCommandCheckOK(F("AT+GATTLIST"));

  Serial.println("Requesting Bluefruit info:");
  /* Print Bluefruit information */
  ble.info();
```

```
  ble.verbose(false);
  ble.echo(false);
}

//Set up all the sensors
void setupSensor()
{
  //Analog Sensors
  pinMode(18,INPUT);
  pinMode(19,INPUT);

  //LSM Setup
  if (!lsm.begin())
  {
    Serial.println("Oops ... unable to initialize the LSM9DS1. Check your wiring!");
    while (1);
  }
  Serial.println("Found LSM9DS1 9DOF");
  lsm.setupAccel(lsm.LSM9DS1_ACCELRANGE_4G);
  lsm.setupMag(lsm.LSM9DS1_MAGGAIN_4GAUSS);
  lsm.setupGyro(lsm.LSM9DS1_GYROSCALE_2000DPS);
  lsm.read();  /* ask it to read in the data */

  /* Get a new sensor event */
  sensors_event_t a, m, g, temp;
  lsm.getEvent(&a, &m, &g, &temp);
  int initY = a.acceleration.y;

}
bool input = false;
//Check for Input
int checkInput() {
  while(!input)
  {
      //Check Index first
      int x = analogRead(18);
      if(abs(x-prevValue) > 30)
      {
        if(x<60)
        {
          prevValue = x;
          return(2);
        }
        else
        {
          prevValue = x;
          return(1);
        }
      }
      else
      {
        int i = checkInput2();
        if(i == 1)
        {}
        else return(i);
      }
  }
}

int checkInput2() {
  while(!input)
  {
    //Check Thumb Next
      int x = analogRead(19);
      if(abs(x-prevValue2) > 30)
      {
        if(x<15)
        {
          prevValue = x;
          return(5);
```

```
        }
        else
        {
          prevValue = x;
          return(1);
        }
      }
      else
      {
        int i = checkGesture();
        if(i == 1)
        { }
        else return(i);
      }
  }
}

int initX = 0;
int initY = 0;
int initZ = 0;

//Check Gesture if no flex
int checkGesture()
{
  lsm.read();
  sensors_event_t a, m, g, temp;
  lsm.getEvent(&a, &m, &g, &temp);

  int initY = a.acceleration.y;

  delay(100);

  lsm.read();
  sensors_event_t a2, m2, g2, temp2;
  lsm.getEvent(&a2, &m2, &g2, &temp2);

  int Y = a2.acceleration.y;


  //calculate integral based on line position
  if(Y == initY)
  {
    return(1);
  }
  if(Y > initY)
  {
    if(Y <0)
    {
      //Y negative and initY negative
      double h1 = (double)abs(Y);
      double h2 = (double)(abs(initY) - abs(Y));
      double area = 0-(h1*0.1)+(h2*0.05);
      //Serial.println(area);
      if(abs(area) > abs(0.5))
      {
        Serial.println("Going Right 1");
        Serial.println(area);
        return(4);
      }
    }
    else
    {
      if(initY <0)
      {
        //initY negative, Y positive
        double h1 = (double)initY;
        double h2 = (double)Y;
        double area1 = initY*0.025;
        double area2 = Y*0.025;
        double area = area1 + area2;
```

```
      //Serial.println(area);
      if(abs(area) > abs(0.5))
      {
        Serial.println("Going Left 1");
        Serial.println(area);
        return(3);
      }
    }
    else
    {
      //inity Y postive, Y positive
      double h1 = abs(Y);
      double h2 = abs(initY) - abs(Y);
      double area = (h1*0.1)+(h2*0.05);
      //Serial.println(area);
      if(abs(area) > abs(0.5))
      {
        Serial.println("Going Right 2");
        Serial.println(area);
        return(4);
      }
    }
  }
}
else
{
  if(initY <0)
  {
      double h1 = (double) abs(initY);
      double h2 = (double) (abs(Y) - abs(initY));
      double area = (h1*0.1)+(h2*0.05);
      //Serial.println(area);
      if(abs(area) > abs(0.5))
      {
        Serial.println("Going Right 2");
        Serial.println(area);
        return(1);
      }
  }
  else
  {
    if(Y <0)
    {
      //initY positive, Y negative
      double h1 = (double)initY;
      double h2 = (double)Y;
      double area1 = initY*0.025;
      double area2 = Y*0.025;
      double area = area1 + area2;
      //Serial.println(area);
      if(abs(area) > abs(0.5))
      {
        Serial.println("Going Right 3");
        Serial.println(area);
        return(4);
      }
    }
    else
    {
      //inity Y postive, Y positive
      double h1 = (double)abs(Y);
      double h2 =  (double) (abs(initY) - abs(Y));
      double area = (h1*0.1)+(h2*0.05);
      //Serial.println(area);
      if(abs(area) > abs(0.5))
      {
        Serial.println("Going Left 3");
        Serial.println(area);
        return(3);
      }
```

```
      }
    }

  }

  return(1);
}

void setup() {
  while (!Serial);  // required for Flora & Micro
  delay(500);
  Serial.begin(115200);
  setupBluetooth();
  setupSensor();
}

void loop() {
  int res = checkInput();
  if(res == 1)
  {
    ble.println("AT+GATTCHAR=1,0x1");
  }
  else if(res == 2)
  {
    Serial.println("Bent");
    ble.println("AT+GATTCHAR=1,0x2");
  }
    else if(res == 4)
  {
    Serial.println("right");
    ble.println("AT+GATTCHAR=1,0x4");
  }
    else if(res == 3)
  {
    Serial.println("left");
    ble.println("AT+GATTCHAR=1,0x3");
  }
      else if(res == 5)
  {
    Serial.println("Thumb Bend");
    ble.println("AT+GATTCHAR=1,0x5");
  }

  delay(100);
}
```

Appendix G: Glove 3.0 Library Code

```
/*
Developed by David de Matheu
Input Glove Microcontroller Code
*/
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using Windows.Devices.Bluetooth.GenericAttributeProfile;
using Windows.Devices.Bluetooth;
using Windows.Devices.Enumeration;
using System.Threading.Tasks;
using Windows.Storage.Streams;

namespace _2WayCommApp
{
    class _2Glove
    {
        BluetoothLEDevice currentDevice { get; set; }

        //private members
        private string deviceName;
        private string serviceID;
        private string characteristicID;

        // Define the selectedCharacteristic variable.
        private GattCharacteristic selectedCharacteristic;
        // Define the selectedService variable.
        private GattDeviceService selectedService;
        private const int CHARACTERISTIC_INDEX = 0;
        private const GattClientCharacteristicConfigurationDescriptorValue
CHARACTERISTIC_NOTIFICATION_TYPE = GattClientCharacteristicConfigurationDescriptorValue.Notify;

        //constructor
        public _2Glove(string theName, string theServiceID, string theCharacteristicID)
        {
            this.deviceName = theName;
            this.serviceID = theServiceID;
            this.characteristicID = theCharacteristicID;

        }
        //private private
        async Task<BluetoothLEDevice> getDevices(string deviceName)
        {
            Debug.WriteLine("Getting Devices");
            var devices = await
DeviceInformation.FindAllAsync(BluetoothLEDevice.GetDeviceSelector());
            Debug.WriteLine("Devices Enumerated");
            foreach (DeviceInformation di in devices)
```

```
            {
                BluetoothLEDevice bleDevice = await BluetoothLEDevice.FromIdAsync(di.Id);
                //Out of the list of devices available to the computer the specific one is
selected and added to the device list.
                if (bleDevice != null && bleDevice.Name == deviceName)
                {
                    currentDevice = bleDevice;
                    Debug.WriteLine(bleDevice.Name);
                    break;
                }
            }
            return currentDevice;
        }

        private async Task<GattDeviceService> selectService(BluetoothLEDevice device, Guid
service)
        {
            var list = await device.GetGattServicesAsync();
            foreach(GattDeviceService serv in list.Services)
            {
                if(serv.Uuid == service)
                {
                    Debug.WriteLine(serv.Uuid);
                    return serv;
                }
            }
            return null;
        }

        private async Task<GattCommunicationStatus> write(GattCharacteristic characteristic,
byte[] data)
        {
            return await characteristic.WriteValueAsync(data.AsBuffer(),
GattWriteOption.WriteWithResponse);
        }

        private async Task<GattCharacteristic> selectCharacteristic(GattDeviceService service,
Guid characteristic)
        {
            var list = await service.GetCharacteristicsAsync();
            foreach(GattCharacteristic charc in list.Characteristics)
            {
                if(charc.Uuid == characteristic)
                {
                    Debug.WriteLine(charc.Uuid);
                    return charc;
                }
            }
            return null;
        }

        public async Task ConnectDevice()
        {
            Debug.WriteLine("Connecting to Device");
            Guid serviceId = new Guid(this.serviceID);
            Guid characteristicId = new Guid(this.characteristicID);
            this.currentDevice = await this.getDevices(this.deviceName);
            Debug.WriteLine(this.currentDevice);
            this.selectedService = await this.selectService(this.currentDevice, serviceId);
            Debug.WriteLine(this.selectedService.Uuid);
            this.selectedCharacteristic = await this.selectCharacteristic(this.selectedService,
characteristicId);
            Debug.WriteLine(this.selectedCharacteristic.Uuid);

            var writer = new DataWriter();
            // WriteByte used for simplicity. Other commmon functions - WriteInt16 and
WriteSingle
            writer.WriteByte(0x0);
            // Parse the data however required
```

```
            GattCommunicationStatus result = await
this.selectedCharacteristic.WriteValueAsync(writer.DetachBuffer());
            if (result == GattCommunicationStatus.Success)
            {

            }
        }

        public async Task<GattCommunicationStatus> EnableNotification()
        {
            await ConnectDevice();
            Debug.WriteLine("Changing Notification Settings");
            GattCommunicationStatus status = await
this.selectedCharacteristic.WriteClientCharacteristicConfigurationDescriptorAsync(
                GattClientCharacteristicConfigurationDescriptorValue.Notify);
            return status;
        }

        public void setCharacteristicID(string id)
        {
            this.characteristicID = id;
        }
        public void setServiceID(string id)
        {
            this.serviceID = id;
        }
        public string getName()
        {
            return this.deviceName;
        }
        public string getCharacteristicID()
        {
            return this.characteristicID;
        }
        public string getServiceID()
        {
            return this.serviceID;
        }

        public GattCharacteristic GetGattCharacteristic()
        {
            return selectedCharacteristic;
        }
    }
}
```

Appendix H: Glove 3.0 Test Application Code

MainPage.xaml.cs

```
/*
Developed by David de Matheu
Input Glove Test Application Code - CS
*/
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using System.Threading;
using System.Threading.Tasks;
using Windows.Devices.Bluetooth.GenericAttributeProfile;
using Windows.Storage.Streams;
using System.Diagnostics;
using System.Collections.ObjectModel;

namespace _2WayCommApp
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    ///
    public sealed partial class MainPage : Page
    {
        _2Glove device = new _2Glove("InputGlove", "10AFFC36-65B9-4391-BAA9-DE9527E9BA66",
"97610DB9-BB62-4345-A1C6-85337428869D");

        public struct Category
        {
            public int numElements;
            public string name;
            public Category[] SubCategories;

            public Category(string name, Category[] Cat)
            {
                this.name = name;
                this.SubCategories = Cat;
                this.numElements = 0;
            }
        }

        public Category shopping = new Category("Shopping",new Category[10]);
        public int current = 0;
        public Category currentCat;
        public string[] shoppingList = new string[10];
        public int currList = 0;

        public MainPage()
```

```
{
    this.InitializeComponent();

    //Cereal
    Category Cereal = new Category("Cereal", new Category[10]);

    Category Cereal1 = new Category("Cheerios", new Category[10]);
    Cereal.SubCategories[0] = Cereal1;
    Cereal.numElements++;

    Category Cereal2 = new Category("Fruit Loops", new Category[10]);
    Cereal.SubCategories[1] = Cereal2;
    Cereal.numElements++;

    Category Cereal3 = new Category("Apple Jacks", new Category[10]);
    Cereal.SubCategories[2] = Cereal3;
    Cereal.numElements++;

    shopping.SubCategories[0] = Cereal;
    shopping.numElements++;

    Category Meat = new Category("Meat", new Category[10]);

    Category Meat1 = new Category("Chicken", new Category[10]);
    Meat.SubCategories[0] = Meat1;
    Meat.numElements++;

    Category Meat2 = new Category("Beef", new Category[10]);
    Meat.SubCategories[1] = Meat2;
    Meat.numElements++;

    Category Meat3 = new Category("Pork", new Category[10]);
    Meat.SubCategories[2] = Meat3;
    Meat.numElements++;

    shopping.SubCategories[1] = Meat;
    shopping.numElements++;

    Category Bakery = new Category("Bakery", new Category[10]);

    Category Bakery1 = new Category("Bread", new Category[10]);
    Bakery.SubCategories[0] = Bakery1;
    Bakery.numElements++;

    Category Bakery2 = new Category("Donuts", new Category[10]);
    Bakery.SubCategories[1] = Bakery2;
    Bakery.numElements++;

    Category Bakery3 = new Category("Flour", new Category[10]);
    Bakery.SubCategories[2] = Bakery3;
    Bakery.numElements++;

    shopping.SubCategories[2] = Bakery;
    shopping.numElements++;

    Category Produce = new Category("Produce", new Category[10]);

    Category Produce1 = new Category("Bread", new Category[10]);
    Produce.SubCategories[0] = Produce1;
    Produce.numElements++;

    Category Produce2 = new Category("Donuts", new Category[10]);
    Produce.SubCategories[1] = Produce2;
    Produce.numElements++;

    Category Produce3 = new Category("Flour", new Category[10]);
    Produce.SubCategories[2] = Produce3;
    Produce.numElements++;

    shopping.SubCategories[3] = Produce;
```

```
            shopping.numElements++;

            currentCat = shopping;
        }

        private async void Characteristic_ValueChanged(GattCharacteristic sender,
GattValueChangedEventArgs args)
        {
            await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, async () =>
{
                // An Indicate or Notify reported that the value has changed.
                var reader = DataReader.FromBuffer(args.CharacteristicValue);
                byte[] data = new byte[args.CharacteristicValue.Length];
                DataReader.FromBuffer(args.CharacteristicValue).ReadBytes(data);
                int val = data[3];
                if (val != 1)
                {
                    Debug.WriteLine("Value changed");
                }
                if (val == 2)//Select
                {
                    Debug.WriteLine(val);
                    Debug.WriteLine("Select Bent");
                    await Select();
                }
                else if (val == 3)//Right
                {
                    Debug.WriteLine("Right Gesture");
                    //Run Right Subroutine
                    await Right();
                }
                else if (val == 4) //Left
                {
                    Debug.WriteLine("Left Gesture");
                    //Run Left Subroutine
                    await Left();
                }
                else
                {
                    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, ()
=> {
                        TextBlock fieldTB = (TextBlock)this.FindName("SelectT");
                        fieldTB.Text = "No Selection";
                    });
                }
                //Set char back to zero
                var writer = new DataWriter();
                // WriteByte used for simplicity. Other commmon functions - WriteInt16 and
WriteSingle
                writer.WriteByte(0x1);
                // Parse the data however required
                GattCommunicationStatus result = await
device.GetGattCharacteristic().WriteValueAsync(writer.DetachBuffer());
                if (result == GattCommunicationStatus.Success)
                {
                    // Successfully wrote to device
                }
            });
        }

        private async void start_Click(object sender, RoutedEventArgs e)
        {
            try {
                GattCommunicationStatus Result = await device.EnableNotification();
                if (Result == GattCommunicationStatus.Success)
                {
                    Debug.WriteLine("Succesfully enabled Notification");
                    device.GetGattCharacteristic().ValueChanged += Characteristic_ValueChanged;
                }
                else
```

```
            {
                Debug.WriteLine("Error");
            }
        }
        catch
        {
            Debug.WriteLine("try again");
        }
    }

    private async void select_Click(object sender, RoutedEventArgs e)
    {
        await Select();
    }

    ObservableCollection<string> listItems = new ObservableCollection<string>();
    private async Task Select()
    {
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () => {
            TextBlock fieldTB = (TextBlock)this.FindName("SelectT");
            fieldTB.Text = "Selected";
        });

        if (currentCat.SubCategories[current].numElements == 0)
        {
            listItems.Add(currentCat.SubCategories[current].name);
            Debug.WriteLine(string.Format("Adding {0} to the shopping list",
currentCat.SubCategories[current].name));
            currList++;
            currentCat = shopping;
            current = 0;
            await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
            {
                ListView view = (ListView)this.FindName("listView");
                view.ItemsSource = listItems;
            });
            await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
            {
                TextBlock fieldTB = (TextBlock)this.FindName("Cat");
                fieldTB.Text = currentCat.SubCategories[current].name;
            });
        }
        else
        {
            currentCat = currentCat.SubCategories[current];
            current = 0;
            await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
            {
                TextBlock fieldTB = (TextBlock)this.FindName("Cat");
                fieldTB.Text = currentCat.SubCategories[current].name;
            });
        }
        Timer timer = new Timer(timerCallback, null, 1000, Timeout.Infinite);
    }

    private async void left_Click(object sender, RoutedEventArgs e)
    {
        await Left();
    }

    private async Task Left()
    {
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () => {
            TextBlock fieldTB = (TextBlock)this.FindName("SelectT");
            fieldTB.Text = "Left";
        });
        current--;
        if (current < 0)
        {
            current = 0;
```

```
            Timer timer2 = new Timer(timerCallback, null, 1000, Timeout.Infinite);
            await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () => {
                TextBlock fieldTB = (TextBlock)this.FindName("SelectT");
                fieldTB.Text = "No Selection";
            });
            return;
        }
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () => {
            TextBlock fieldTB = (TextBlock)this.FindName("Cat");
            fieldTB.Text = currentCat.SubCategories[current].name;
        });
        Timer timer = new Timer(timerCallback, null, 1000, Timeout.Infinite);
    }

    private async void right_Click(object sender, RoutedEventArgs e)
    {
        await Right();
    }

    private async Task Right()
    {
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
        {
            TextBlock fieldTB = (TextBlock)this.FindName("SelectT");
            fieldTB.Text = "Right";
        });
        current++;
        if (current > currentCat.numElements - 1)
        {
            current = currentCat.numElements - 1;
            Timer timer2 = new Timer(timerCallback, null, 1000, Timeout.Infinite);
            await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
            {
                TextBlock fieldTB = (TextBlock)this.FindName("SelectT");
                fieldTB.Text = "No Selection";
            });
            return;
        }
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
        {
            TextBlock fieldTB = (TextBlock)this.FindName("Cat");
            fieldTB.Text = currentCat.SubCategories[current].name;
        });
        Timer timer = new Timer(timerCallback, null, 1000, Timeout.Infinite);
    }

    private async void timerCallback(object state)
    {
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () => {
            TextBlock fieldTB = (TextBlock)this.FindName("SelectT");
            fieldTB.Text = "No Selection";
        });
    }
  }
}
```

MainPage.xaml

```
/*
Developed by David de Matheu
Input Glove Test Application Code - XAML
*/
<Page
    x:Class="_2WayCommApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_2WayCommApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <TextBlock x:Name="SelectT" HorizontalAlignment="Left" TextWrapping="Wrap" Text="No
Selection" VerticalAlignment="Top" Margin="205,489,0,0" Height="54" Width="472" FontSize="36"
TextAlignment="Center"/>

        <Button x:Name="start" Content="Start" HorizontalAlignment="Left" Margin="356,352,0,0"
VerticalAlignment="Top" Height="72" Width="165" Click="start_Click"/>

        <TextBlock x:Name="Cat"  HorizontalAlignment="Left" TextWrapping="Wrap" Text="Cereal"
VerticalAlignment="Top" Margin="101,222,0,0" Height="143" Width="670" FontSize="72"
TextAlignment="Center" RenderTransformOrigin="0.493,1.049"/>

        <Button x:Name="rightB" Content="Right" HorizontalAlignment="Left" Margin="644,564,0,0"
VerticalAlignment="Top" Height="72" Width="165" Click="right_Click" />

        <Button x:Name="leftB" Content="Left" HorizontalAlignment="Left" Margin="87,564,0,0"
VerticalAlignment="Top" Height="72" Width="165" Click="left_Click"/>

        <Button x:Name="selectB" Content="Select" HorizontalAlignment="Left" Margin="356,564,0,0"
VerticalAlignment="Top" Height="72" Width="165" Click="select_Click"/>

        <ListView x:Name="listView" HorizontalAlignment="Left" Height="636" Margin="1001,50,0,0"
VerticalAlignment="Top" Width="205" BorderBrush="Black"  BorderThickness="3"
RequestedTheme="Light"/>

        <TextBlock x:Name="textBlock" HorizontalAlignment="Left" TextWrapping="Wrap"
Text="Shopping List" VerticalAlignment="Top" Margin="1058,25,0,0"/>

        <TextBlock x:Name="textBlock2" HorizontalAlignment="Left" TextWrapping="Wrap"
Text="Shopping List Demo" VerticalAlignment="Top" Margin="57,25,0,0" FontSize="90"/>

        <Border BorderBrush="Black" BorderThickness="2" HorizontalAlignment="Left" Height="720"
Margin="899,0,0,0" VerticalAlignment="Top" Width="3"/>
    </Grid>
</Page>
```

BIBLIOGRAPHY

"Collaborative Research: Visual Cortex on Silicon," 2013. [Online]. Available:
1] http://www.cse.psu.edu/research/visualcortexonsilicon.expedition/description.html. [Accessed 8 February 2018].

"GATT Overview," Bluetooth, [Online]. Available:
2] https://www.bluetooth.com/specifications/gatt/generic-attributes-overview. [Accessed 8 February 2018].

"Short Flex Sensor," Adafruit, [Online]. Available: https://www.adafruit.com/product/1070.
3] [Accessed 8 February 2018].

"Teensy USB Development Board," PJRC, 2015. [Online]. Available:
4] https://www.pjrc.com/store/teensy31.html. [Accessed 8 February 2018].

"Use Gestures," Microsoft, 30 November 2017. [Online]. Available:
5] https://support.microsoft.com/en-us/help/12644/hololens-use-gestures. [Accessed 8 February 2018].

"Vibrating Motor 1.3 Volt DC 80 mA," Jameco, [Online]. Available:
6] https://www.jameco.com/z/6ZK003-R-Vibrating-Motor-1-3-Volt-DC-80-mA_256306.html. [Accessed 8 February 2018].

L. Ada, "Adafruit Feather 32u4 Bluefruit LE," Adafruit, 5 May 2017. [Online]. Available:
7] https://learn.adafruit.com/adafruit-feather-32u4-bluefruit-le. [Accessed 8 February 2018].

L. Ada, "Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout,"

8] Adafruit, 1 February 2017. [Online]. Available: https://learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout. [Accessed 1 February 2018].

9] L. Ada, "Force Sensitive Resistor (FSR)," Adafruit, 4 May 2015. [Online]. Available: https://learn.adafruit.com/force-sensitive-resistor-fsr. [Accessed 8 February 2018].

10] R. Coakley, "CSE students win second place in Global Student Challenge," Penn State College of Engineering, 29 August 2017. [Online]. Available: http://www.eecs.psu.edu/News/CSE-Global-Student-Challenge.aspx. [Accessed 8 February 2018].

11] Jimbo, "Analog vs. Digital," Sparkfun, [Online]. Available: https://learn.sparkfun.com/tutorials/analog-vs-digital. [Accessed 8 February 2018].

12] M. Satran and M. Jacobs, "Bluetooth GATT Client," Microsoft, 8 February 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/uwp/devices-sensors/gatt-client. [Accessed 8 February 2018].

13] M. Satran and M. Jacobs, "Bluetooth Low Energy," Microsoft, 15 March 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/uwp/devices-sensors/bluetooth-low-energy-overview. [Accessed 8 February 2018].

14] C. Winner, "Third Eye," Penn State, 28 April 2017. [Online]. Available: http://www.psu.edu/feature/2017/04/28/third-eye. [Accessed 8 February 2018].

15] M. Zeller, K. Baker and B. Bray, "Spatial mapping," Microsoft, 21 March 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/spatial-mapping. [Accessed 8 February 2018].

16] M. Zeller, K. Baker and B. Bray, "Spatial sound," Microsoft, 21 March 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/spatial-sound. [Accessed 8 February 2018].

SFUPTOWNMAKER, "I2C," Sparkfun, [Online]. Available:

17] https://learn.sparkfun.com/tutorials/i2c. [Accessed 8 February 2018].

Academic Vita

David José de Matheu

300 S Pugh St • State College, PA 16801 • (814) 321-6323
ddematheu@psu.edu • linkedin.com/in/ddematheu

Education

- **The Pennsylvania State University, University Park, PA**
  - Bachelor of Science in Computer Engineering, May 2018
  - Schreyer Honors College at Penn State University
  - Courses:
    - CMPEN 431: Computer Architecture and Design
    - EE 310: Circuits and Devices
    - CMPSC 473: Operating Systems
    - CMPSC 311: Systems Programming
    - CMPSC 465: Introduction to Algorithms
    - EDESGN 100H: Engineering Design Honors

Work Experience

- **Microsoft Technical PM – Windows and Devices – Xbox MR (Summer 2017)**

- **Microsoft Technical PM – Windows Fundamentals Team (Summer 2016)**
  - Worked on the area of Windows Compatibility designing, specing and overseeing development of a Bot entity that surfaced data regarding compatibility regressions using the Microsoft Bot Framework and Natural Language Processing Engine (LUIS.ai).

Extracurricular and Personal Projects

- **EECS Undergraduate Advisory Committee – Committee Member** (2017-2018)
  - Met with EECS Faculty to discuss different issues affecting the EECS school to look for solutions and ways of improving the overall student experience.
  - Held town halls for students to express their areas of frustration in order to properly communicate them to faculty.

- **Undergraduate Researcher – Third Eye, MDL Penn State** (2016-2018)
  - Worked as part of the Third Eye team, using computer vision algorithms to aid visually impaired individuals to navigates spaces.
  - Worked in the development of Bluetooth interfaces between a HoloLens device and a custom Feather based haptic feedback glove
  - Designed and built Haptic Feedback Gloves using motors and Feather microcontroller

- **Formula SAE – Electronics Subsystem Lead** (2015- 2016)
  - Working on the overall design of the electronics components of the car specifically, main wiring harness, engine wiring harness, relay and fuse box, and data telemetry system.
  - Tracking progress and timelines for the Electronics subsystem to report to the overall team and other leads.

Skills

Fluent Spanish and English	*Intermediate C, C++, C#*	*Intermediate NodeJS*
Intermediate Arduino	*Intermediate Python*	*Intermediate SQL*