Universal Gesture Interface

By

Kenobi Carpenter

Kobe Duda

Connor Michalec

Final Report for ECE 445, Senior Design, Spring 2026

TA: Lukas Dumasias

6 May 2026

Team 79

Abstract

The Universal Gesture Interface, is a device which aims to bridge the gap from rudimentary computer control interfaces and fully immersive control interfaces. We went about our design by focusing on developing a glove that utilized an IMU, Flex sensors, Force Sensitive Resistors to build an accurate picture of what a user is doing with their hand. We demonstrated the effectiveness of our project with technical tests such as latency and polling rate, as well as human tests like gesture detention correctness.

One of the biggest challenges we faced throughout this was troubleshooting the IMU, in which we could not receive data until finally discovering that we were not using it as intended, requiring us to come up with an emergency solution to test our software.

Overall our project was a success and we were able to demonstrate accurate gesture detection allowing us to use the glove as a mouse for navigation of the desktop, as well as a game controller.

1. Introduction

There are many ways that humans interact with computer systems and software. The most common are a keyboard and mouse, but there are also more immersive options such as Virtual Reality and other fully immersive options. There are many benefits and drawbacks to any particular device, however most fall into one of two extremes. The first is that they are extremely rudimentary and non-immersive, such as a mouse or keyboard. The other extreme is full immersivity, such as VR, which is very expensive, heavy, and sometimes even vomit inducing. There seems to be a lack of a middle ground, or a device that is simple, lightweight, and intuitive, yet not expensive or too immersive. We set out to solve this challenge by making a gesture interface. In other words, we wanted to design a device capable of recognizing key intuitive human gestures, and use those movements to control a computer, or specific software on them. This project could have many uses from 3D design software to video games. The device is centered around a glove that the user wears. There is an inner and an outer glove sewn together, with all of the necessary sensors and electronics sandwiched in between. Our glove uses flex sensors to measure the amount the user bends their thumb, index, and middle finger. We also use force sensitive resistors (FSR's for short) to detect if the user is pressing something, or pressing two fingers together. Finally, we use an IMU to determine the orientation of the glove, and therefore the hand of the user. Using the data from all these sensors, we obtain a mostly complete picture of what the user is doing with their hand, and we process this data to detect both static and dynamic gestures. Our completed design offers an intuitive method to control a computer, effectively bridging the gap between a mouse and a VR headset.

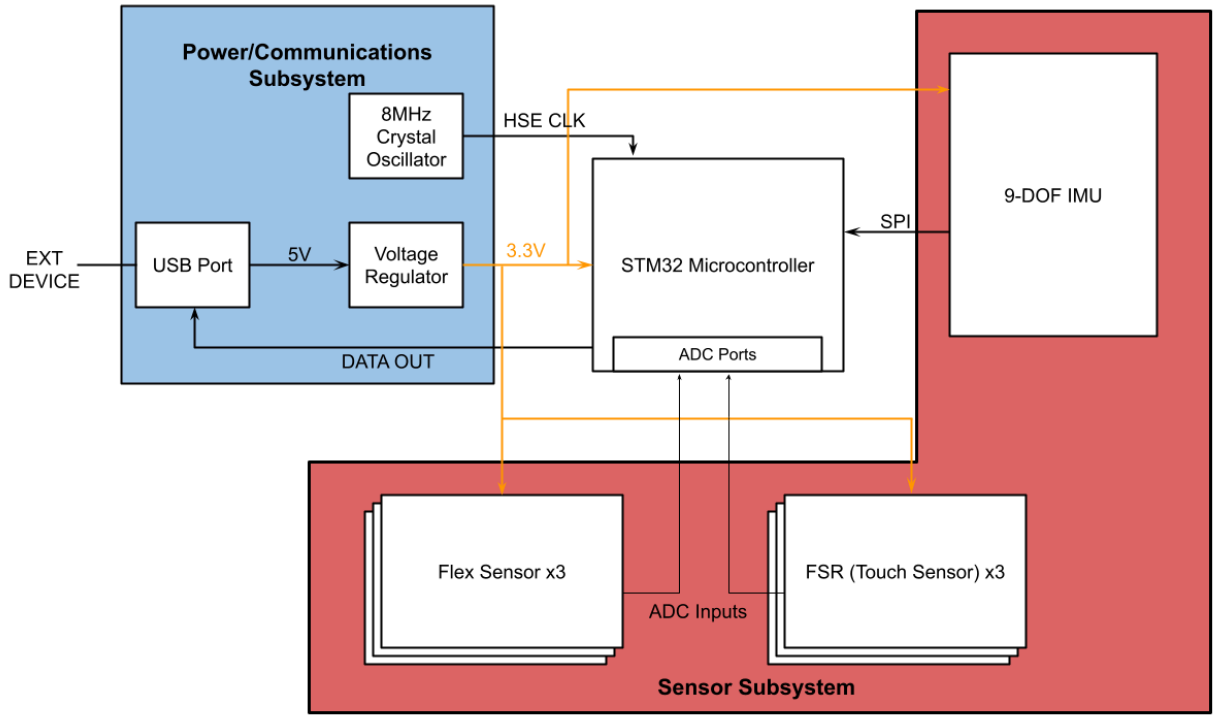


Figure 1 Block diagram of the system.

2 Design

2.1 Design procedures

Our design procedure involved us each attacking different aspects of the project, and bringing them together for testing. We aimed to work on our project in a procedural manner, starting with a simple prototype testbench, with the intention of moving onto something more fleshed out once the testbench proved feasible. This was especially applicable to the PCB's, in which we developed a "testboard" which just provided us basic functionality for our components before designing a "final" PCB board which integrated all of our desired functionality.

The benefit of this is that it simplifies what we need to do at each step. By dealing with a less complicated PCB at first, we can isolate any issues that might come up before introducing complications. This proved critical in troubleshooting some of the issues we encountered throughout our project and helped us isolate what those were amongst many different potential culprits.

2.2 Hardware modules

2.2.1 IMU

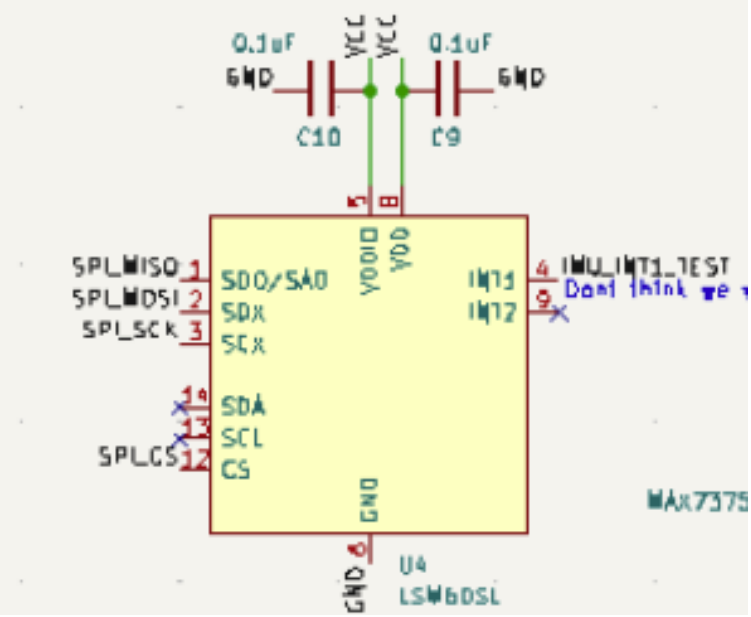


Figure 2 IMU schematic (with decoupling capacitors)

The Inertial Measurement Unit (IMU) provides the information necessary for orientation and acceleration detection which we utilize for detecting hand posture. The IMU we utilized is the LSW6DSL [1], a low cost IMU chosen due to its ability to seamlessly work with the STM32 on the same voltage (3.3V) which eliminates the need for voltage conversion seen with other IMU options.

The IMU contributes to the overall design by providing information on the glove's translational and rotational positioning. In order for the glove to be useful as a Human Interface Device (HID), such as a mouse, this information is essential for providing tracking behavior and would be one of the main methods the user would be able to command translational movements of whatever software they are controlling. The unit can communicate with SPI and will be connected to the microprocessor.

2.2.2 Flex sensors

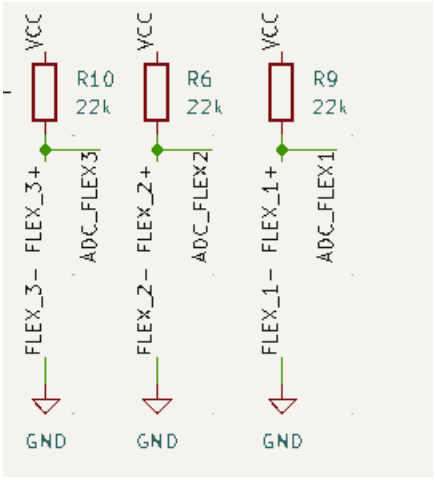


Figure 3 Flex sensor voltage divider circuit

Three flex sensors provide the information necessary for detecting finger flexing and finger positioning. It is key for proper gesture detection since most gestures rely not only on hand posture and movement, but also positioning of fingers over time. Such as thumbs up, thumbs down, finger gun, etc. The flex sensors we are utilizing are the generic 4.5" flex sensors that can be found in the ECE supply shop. Their operation is rudimentary, and can be easily substituted for any make/model of flex sensors that can be found.

The key to their operation is to implement them in a "voltage divider" circuit with a 27kOhm resistor such that the microprocessor can receive a voltage to its ADC given depending on the resistance seen on the flex sensor (their deflection corresponds to a resistance value).

2.2.3 Touch sensors

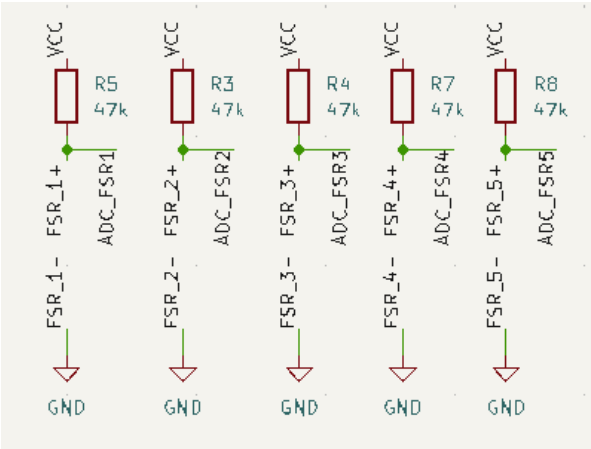


Figure 4 FSR voltage divider circuit

Five force-sensitive resistors serve the purpose of capturing the data required to sense how the user utilizes their fingers to gesture. They must provide an accurate reading that should avoid hitting false positives. They are be attached to the tips of the thumb, index finger, and middle finger. Similar to the flex sensors, they will be wired to ADCs with voltage dividers (47kOhm) to be read by the MCU. Used for detecting pinching, tapping, and pressing. We chose to use the MF01A-N-221-A01 [2] which is cheap and provided us with the force sensitivity required for our application.

The schematic for the touch sensors is identical to that of the flex sensors except they utilize different resistance values to compensate for their different construction.

2.2.4 Microprocessor

The microprocessor unit takes as input all of the aforementioned sensor data and sends USB as output. We chose to use the STM32F405 [3] which was chosen for its DSP capabilities, as processing sensor inputs and identifying them as gestures will constitute a considerable portion of this project. With high end specs such as a 32-bit 168Mhz processor and 192KB of SRAM it is more than enough to handle any signal processing with throw at it.

This is also where most of our design decisions will be integrated. For example, the IMU is prone to drift, meaning we'll have to make UX decisions that mitigate its influence, i.e. only moving the mouse when a finger is down on the desk.

2.2.4 USB Interface and power

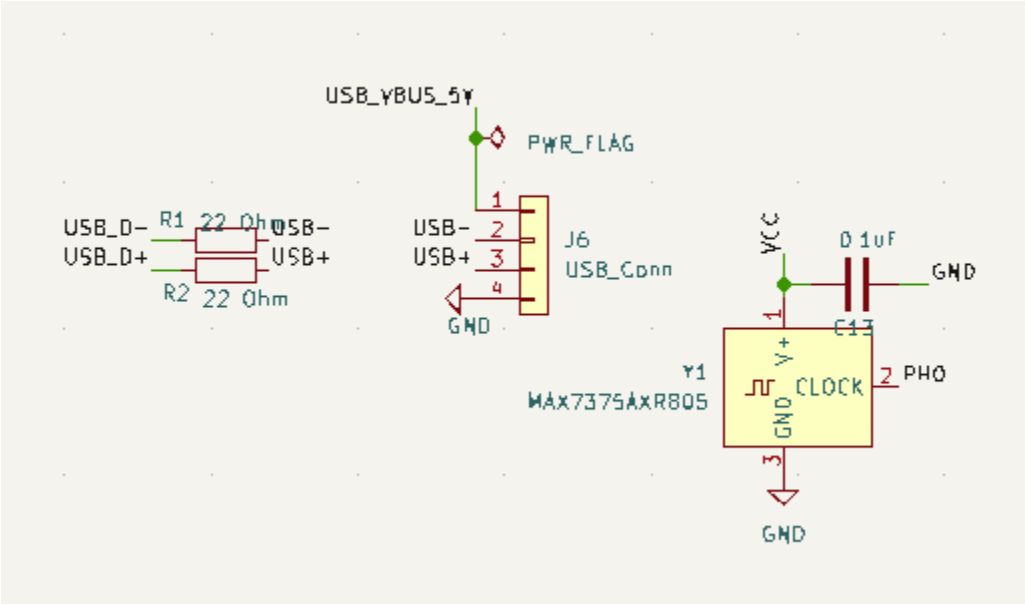


Figure 5 USB connector with resistors and crystal oscillator

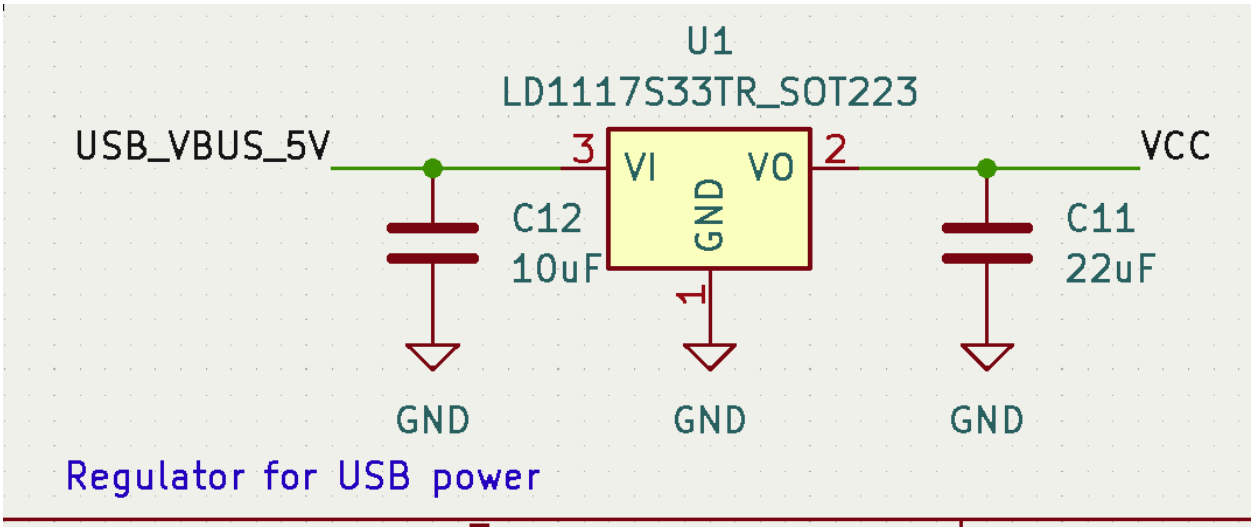


Figure 6 Regulator for USB power

The USB connection module is integrated directly into the PCB of the device. This includes decoupling capacitors, a pull-up on the D+ pin (done in software), and an external 8MHz crystal oscillator. It will act as the 5V power supply to the device, and the LD1117S33TR [4] power regulator will be used to convert this into the expected 3.3V for the microprocessor. We chose this regulator due to its simplicity and ease of operation.

The USB port must reliably supply 5V to the whole device, and the voltage regulator must be able to supply 3.3V to the rest of the device. The USB connection must be able to communicate at full speed (**12 Mbps**). Reports sent to the host device must be transmitted at a rate that latency is undetectable by humans, minimum **100Hz**.

Given that USB should be communicating at full speed (12mbps) - with a 1ms frame. If we assume we are emulating a mouse (to start). We expect to send about 4 bytes per frame (X, Y, clicks, scroll) on HID interface. If we sample at our minimum of (100Hz):

$$4\ bytes\ *\ 100Hz = 400\text{Bps} \tag{2.11}$$

Which is far below the maximum of 12Mbps.

2.2 Design Alternatives

Throughout our design process, there were things that challenged us and required us to utilize quick thinking and flexibility to overcome them given the time constraints we had.

One of the biggest sources of trouble was the IMU. After ordering our first PCB, and after all of the components were soldered to it, the IMU would not send data to the MCU. We realized that our IMU pinout was incorrect, and needed to be changed on future boards. Unfortunately, we had already ordered our second PCB at this time and did not have a reliable way to even begin to test our software for several weeks. This meant that the second board we ordered was not going to work unfortunately, however we could slightly modify it with our third PCB order to get everything working properly. For our third and final PCB order, we corrected the IMU wiring and kept all the modifications we had made from the second PCB.

As we were waiting for our third PCB to arrive, we decided it would be necessary to perform "surgery" on our board to test our software with a working IMU. This required soldering single wire strands to the microscopic pins on the IMU. After many hours spent in the lab, we finally got everything working, allowing us to start incorporating the IMU into our software development.

Figure 7 IMU Surgery

2.3 PCB Layout

There are many things to consider when designing a layout for a PCB, however due to the fast paced nature of the class, we decided to design our first board to serve as a testbench for all of our components. This was done by adding extra test points and spacing out components for easy soldering and debugging. Our goal was to first get a working PCB, and later optimize its shape and routing for the mechanical design of the glove.

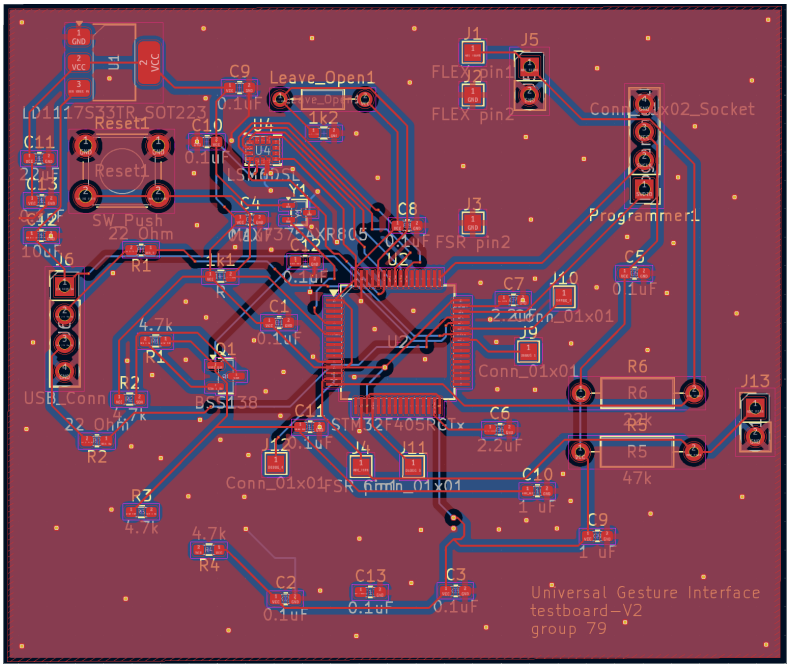


Figure 8 Initial Testboard.

One significant challenge we faced during PCB iteration was the manufacturing times. Unfortunately, our first board had not come in by the time we had to order our second board, so a lot of the testing to ensure proper functionality couldn't be done. For our second PCB order, we re-routed everything to optimize its integration with our physical glove. First, we shaped the edge of the board to somewhat resemble the shape of a palm. We came up with a home-plate shape, which was both simple and allowed the board to easily rest on the top of the user's hand, which is eventually where it would be placed in the glove. We also made sure to route the USB data lines as a differential pair, which was crucial for reliable data transmission.

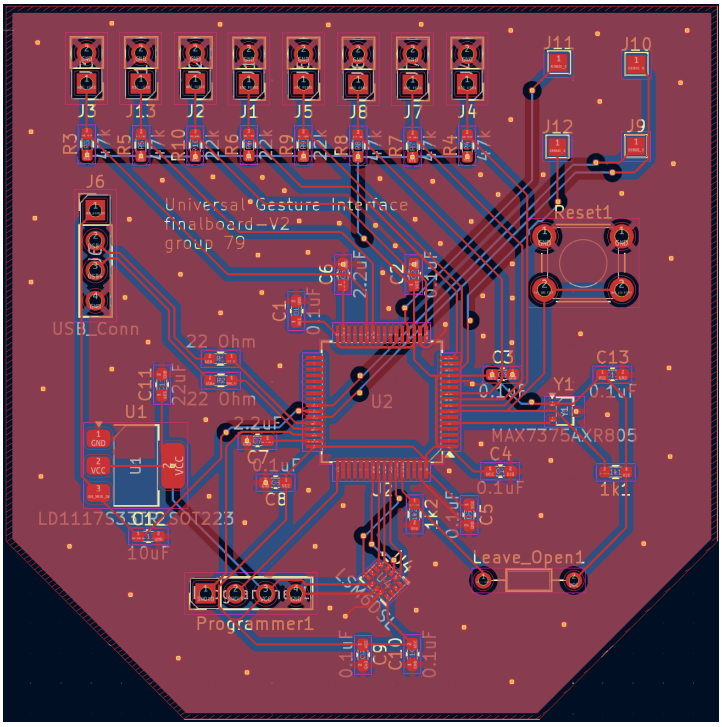
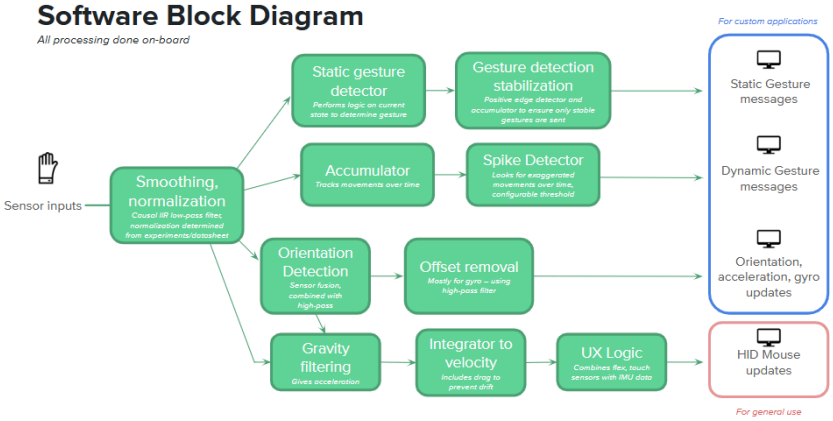


Figure 9 Final Board.

The result was a compact board with efficient wire routing, which would eventually be sewn in between the two gloves on top of the user's hand, which allowed the onboard IMU to accurately determine the hand's orientation and acceleration.

2.4 Software and processing

Our software design is primarily focused on the on-board digital signal processing. The data flow is demonstrated graphically in the below figure to give an overall picture. We will look at each of these parts in detail in subsequent sections



2.4.1 Initial Signal processing

The raw data sent from the sensors to the system is quite messy. It is quite noisy, not normalized to any useful scale, and prone to offsets and inaccuracies. The first issue to address is the noise. Because we are sampling at a much higher rate than we are streaming, we can simply remove the highest frequencies with a low-pass filter. The simplest and most efficient implementation we could find was to use a causal IIR low-pass filter, or essentially a sliding average of the last filter output and the current input. This is implemented with the following equation:

$$y[n] = ax[n] + (1 - a)y[n - 1] \tag{2.41}$$

Here, a is our tunable parameter that adjusts the cutoff frequency. The tradeoff in choosing a value for a is that a lower value results in a much smoother signal, but a slower response time. As mentioned before, the high sampling rate gave us leeway to go with an extra low a value while still not sacrificing too much responsiveness. For flex sensors, which weren't incredibly dependent on quick responsiveness, we chose an aggressive smoothing factor of **0.05**. For acceleration values, which were a bit more dependent on subtle and quick motions, we chose a value of **0.5** for a.

It was also necessary to renormalize values. Unexpectedly, the flex sensors for the three fingers all had different offsets and ranges for their voltage divider outputs, so it was necessary to renormalize them all to a unified scale. We chose to use a linear remapping, with the constants determined experimentally. We used our monitoring software to determine the max and min ADC values for maximum and minimum finger extension, then stretched the inputs accordingly.

Finally, it was incredibly helpful to normalize accelerometer and gyroscope data to a useful scale, as they both are sent as 16-bit integers at a fairly arbitrary scale. To account for any device-specific variations, we determined accelerometer scale experimentally by collecting data while the device was at rest over a period of time and taking the average value. Because the device is constantly experiencing 1g of acceleration at rest, by taking the magnitude of this vector and dividing inputs by this constant, we can normalize all accelerometer data to a scale of 1g. Gyroscope data is even more crucial to measure at an accurate scale and unfortunately more difficult to measure experimentally. We used information from the data sheet to determine a constant that maps our gyroscope data to a scale of radians/sec.

2.4.2 Gesture Detection

We distinguish between two kinds of gestures for the sake of this paper: **static** and **dynamic** gestures. Static gestures are those which can be classified by evaluating the current state of the hand, with no time dependence. This may include motion (i.e. current acceleration values), but it can only be determined by evaluating the current state of the sensor data. On the other hand, dynamic gestures are those which can only be identified over time. This could include waving a hand, conducting an orchestra, or shooting a finger gun. We considered dynamic gesture recognition to be a stretch goal for the scope of this project, and unfortunately were only able to manage a very simple implementation.

Static gesture recognition, at its core, purely comes down to doing basic logic on the current available sensor data. With this logic, we created a sample set of 6 identifiable static gestures:

- Thumbs up
- Thumbs middle
- Thumbs down
- Finger gun
- Pinch index finger
- Pinch middle finger

On top of the basic classification layer, we add an additional interface layer for turning this state identification into discrete messages. For each message type, we pass it through a "stability gate", containing an accumulator and predefined thresholds for activation. This checks to see if a gesture has been held for a long enough amount of time, and if so, triggers an update event (i.e. sending a CDC message) only on the positive edge of the gesture being held. When the accumulator falls below a defined threshold, the state is set back to "no gesture".

We only were able to define one dynamic gesture, and it is built on a framework for detecting energy spikes in a signal. This is a generic framework that could work for any input signal, and keeps a constantly decaying accumulator that triggers an event upon crossing a threshold. Both the threshold and decay rate are configurable parameters, and we chose to select these parameters by hand to get something that felt consistent while avoiding double-activations or false positives. To demo this framework, we made a finger gun shooting gesture, affectionately labeled "BANG!" It has an initial gate of checking for the finger gun static gesture, then uses the spike detector to detect a spike in the gyroscope z-axis. Surprisingly, this resulted in one of our most reliable gestures, passing 50/50 trials in our testing.

2.4.3 Velocity Integration

One of the most useful metrics for both gesture identification and mouse movement is the linear velocity of the device. Unfortunately this is not natively provided from IMUs, and the naive approach of simply integrating velocity leads to a large spread of issues.

We address these issues firstly by adding a drag component to our velocity integration. Because acceleration data is inaccurate and noisy, a running integrator can drift over time. By multiplying the value by a constant drag coefficient, we can restabilize it when acceleration is zero. Of course, this does not solve the double-integration errors of position, but this can be remedied in UX design decisions (i.e. solutions similar to how a user might pick up a mouse that has drifted off-course to re-center).

The greater hurdle to tackle was the gravity offset of the accelerometer. This obviously greatly offsets velocity values depending on the angle of the device, so it must be remedied with signal processing. A general overview of our approach to this can be seen below:

$$\mathbf{g}_{est} \leftarrow \mathbf{g}_{est} + \Delta t \cdot (\mathbf{g}_{est} \times \boldsymbol{\omega})$$

Rotate estimated gravity vector by angular acceleration vector (small timestep approximation, proper normalization necessary)

$$\| \mathbf{a} \|$$

Check if acceleration magnitude is close to 1g

$$\mathbf{g}_{est} = \alpha * \mathbf{a} + (1 - \alpha) * \mathbf{g}_{est}$$

"Pull" the gravity estimate back towards the DC offset of acceleration

In short, we keep a constant "gravity estimate" vector, and utilize gyroscope data for short term adjustments while using accelerometer data for long-term corrections and restabilization.2.5 Physical Design

We prioritized comfort and ease of use in our physical design. We wanted something easy to attach to the body quickly. We wanted to avoid the negative user experience common to VR headsets that are too heavy, and require a lot of fidgeting with multiple straps and buckles to be comfortable and work properly. We settled on a glove for the design, as it is incredibly easy to slip on and off. By putting all of the necessary electronics in the glove, sewed into the proper configuration, simply putting the glove on would automatically place all of the sensors where they needed to be with respect to the hand.

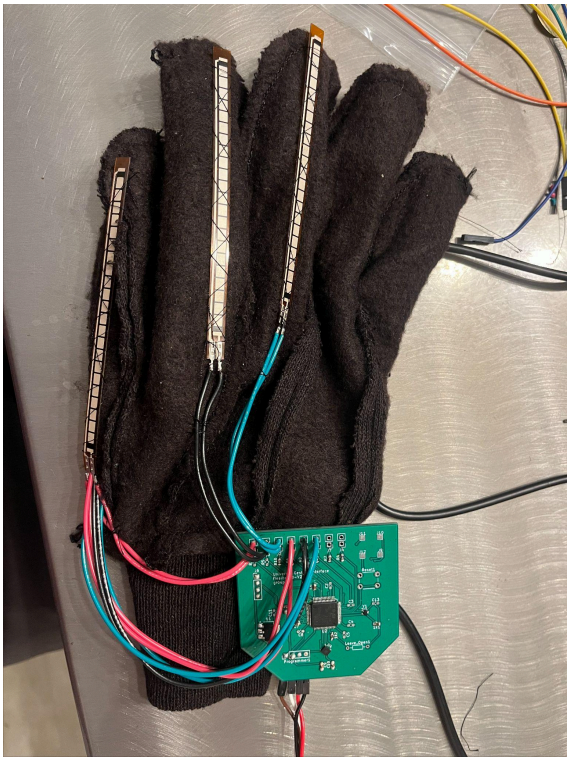


Figure 10 Inner glove assembly

Figure 11 Force Sensitive Resistor Mounts

As for our sensors, we needed to pick sensors that could easily fit on the surface of the glove. We chose extremely thin, small flex sensors and force sensitive resistors, which would fit discreetly on the fingers of the glove and could be easily covered with a cosmetic component.

This approach offered numerous challenges however. Gloves are typically made of some type of fabric, which is hard to securely mount things to. We had the idea of sewing the electronics in place, however we felt this was not enough to hold them securely. We also added a bit of superglue, however this was still insufficient, as the electronics were very exposed and could easily catch on something and be knocked out of place. We finally found that an easy way to cover everything neatly while also securing the sensors firmly in position was to add a second glove, around the first one. Our design essentially became an inner glove and an outer glove, with all of the electronics sandwiched in between. Once everything was working reliably, we sewed the two gloves together to ensure nothing would move. This method proved incredibly effective at preventing sensor misalignment while at the same time neatly covering all of the electronics and sensors, which was the best of both worlds.

This design noteworthily had no straps, buckles, or any moveable members requiring adjustment. Wearing the device is as simple as putting on a glove. Due to the choice of material, the glove is very expandable and can easily fit many hand sizes with no adjustment necessary.

3. Requirements and verification

Our high-level requirements for the project were as follows:

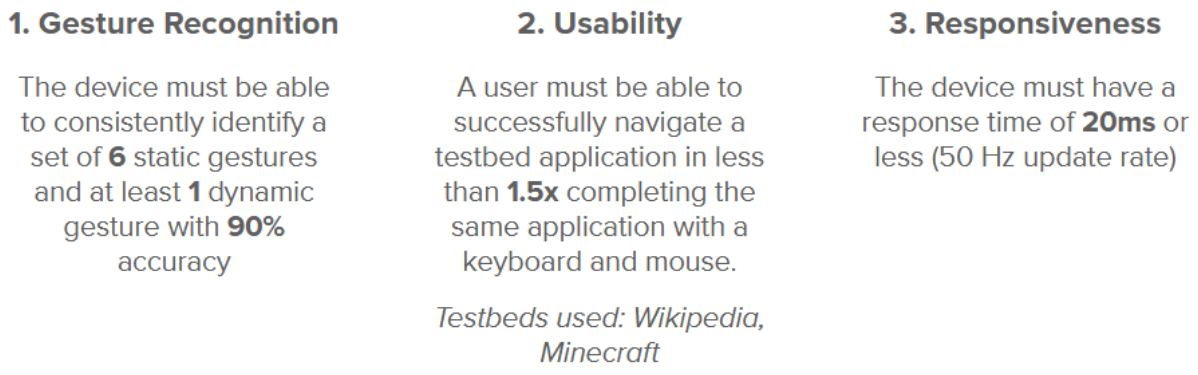


Figure 12 **High-level requirements**

We had one qualitative and two quantitative tests to verify these requirements. Gesture recognition was done by 7 sets of **50** trials, one for each gesture type, counting the number of successful identifications.

Responsiveness was determined by externally measuring refresh rate. Usability was determined through two testbed applications. Our results were as follows:

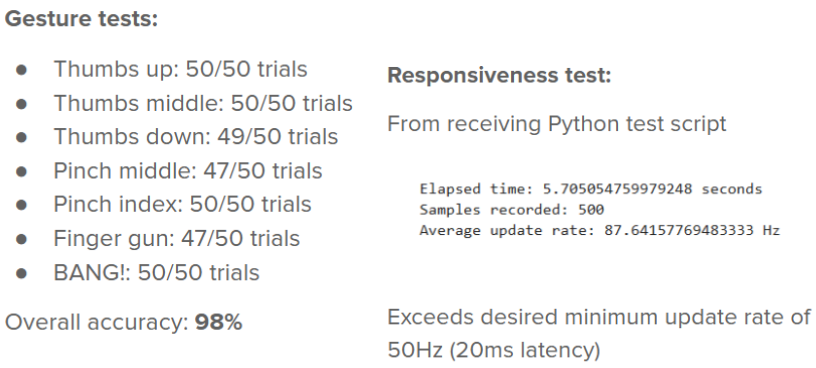


Figure 13 Quantitative Results

4. Costs

4.1 Parts

Table 1 Parts Costs

Part	Quantity	Cost per unit ($)	Total Cost ($)
MF01AN221A01 (FSR)	4	$6.60	$26.40
STM32F405RGT6V (MCU)	4	$9.37	$41.01
511-LSM6DSLTR (IMU)	4	$4.26	$17.04
4.5” flex sensor	5	$7.95	$39.75
LD1117S33TR (regulator)	5	$0.32	$1.60
MAX7375AXR805 (oscillator)	3	$2.20	$6.60
Misc resistors, capacitors	~20	free	free
Total	**45**	**$30.7**	**$132.4**

Note: resistors and capacitors were free since they were found in spare parts bins

4.2 Labor

Most of the labor involved in this project was either soldering or sewing. The biggest challenge with the soldering was the surface mount components, specifically the MCU and the IMU. The IMU was challenging in particular due to its small package size and bottom pad, requiring a heat gun and solder paste in order to solder it properly. The first prototype board required a lot more labor to assemble

because of the surgery required due to the mistake in the schematic. We spent a total of around 15 hours soldering the board.

The other key aspect of the project involving a lot of labor was sewing. All of the sensors needed to be precisely and securely sewn into place to ensure proper operation. Each sensor took approximately an hour to sew in place, and the entire glove consisted of about 8 hours of sewing, including the time taken to sew both gloves together.

Factoring in other miscellaneous tasks, we spent a total of 40 hours of labor to build the final prototype, however we believe we could cut this to about 10 hours for future prototypes. Assuming an hourly rate of $40 per hour for an entry level engineer, one could expect a labor cost of $400 for complete assembly of a prototype.

5. Conclusion

5.1 Accomplishments

We accomplished a lot in the completion of this project. The initial goal was to bridge the gap between very rudimentary and very immersive ways to interact with computers and software. We absolutely succeeded in this goal, as our final design was very easy to put on and use, and served as a very intuitive way to interact with a variety of computer programs. We completed all of our functionality goals, which were to detect 6 static and 1 dynamic gestures with over 90% accuracy. We achieved a 98% average for all the gestures, with no individual gesture going below 90%. We also achieved less than 20ms of latency, which we tested with a python script. We were successfully able to navigate a search browser and play minecraft using the glove, which was another key goal we had. In the end, the project was very successful and proved that a hybrid-immersive controller is both feasible and desirable in many situations.

5.2 Uncertainties

Perhaps the greatest area of uncertainty with this project's final performance is with the velocity integrator. While it is passably functional to work as a product, there is much room for improvement to be made. Misalignments in the gravity estimation and the accelerometer updates cause minor (or sometimes major) "bouncing" artifacts, and occasionally strange behavior when rotating the hand. While we believe the approach we took to remedying this was sufficiently complex, it was not fine-tuned well enough to be a strong competitor to a mouse when evaluating translational movement. However, we believe this to be a very remediable problem with a bit more time and effort, and intend to continue work to get something more satisfactory!

There are also concerns with the scalability of this device. Specifically, many of our sensors exhibited inconsistent behavior, and our codebase contains many normalization constants specific to our device and hardware. It is a valid concern that these constants may not be universally applicable if the device were to be replicated with new parts, and some on-board auto-calibration may be necessary for addressing this if this device were to be mass-produced.

5.3 Ethical considerations

Section II subsection 7 of the IEEE standard states that engineers must "treat all persons fairly and with respect, and to not engage in discrimination based on characteristics such as race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression." Two characteristics that would greatly affect our users' ability to use this product are hand size and handedness. We will need to take care in our design decisions to make sure our product is accessible to as many people as possible. Exact solutions to this issue will require iteration through the design process, but possible avenues include making multiple hand sizes, making left-handed and right-handed gloves, or making the components mount to the hand in a way that accounts for all ranges of hand sizes instead of a typical fabric glove.

The safety concerns of this device are a subset of those of a typical virtual reality headset. Any technology that requires the user moving around physically in a room poses a risk to themselves, their environment, and those around them. This is particularly true when the application involves a high degree of activity or fast, sudden motions. Not only could this result in harm from accidentally striking the environment or a nearby person, but individuals with cardiovascular issues may also be put at risk of heart complications by high activity. These are risks inherent to any physical activity and we cannot completely nullify them through our design, but we intend to include appropriate warnings with our product to encourage safe usage. It is also worth noting that these risks will be slightly magnified for this first iteration of our design due to the wired connection, which increases the risk of damage to surroundings and poses a potential strangling hazard. As this design decision has been made out of necessity for the sake of completing the project within course deadlines, we will not be addressing it this semester, but the goal for a finished product would be to have a wireless connection to improve both usability and safety.

5.4 Future work

Satisfied with our work, there are a few ideas we had for how we could see this project evolving.

1. The first would be developing a wireless interface. Initially, we utilized a wired connection due to the simplicity of USB compared to a wireless solution and speed, but given the nature of a device that is meant to strap to your hand, a wireless model would greatly improve on the design.
2. Another area for improvement would be to utilize double joint flex sensors. Our current design uses one flex sensor per finger, which works very well at detecting basic gestures, but for more advanced gestures it would be greatly beneficial to be able to detect the deflection of both finger joints independently giving more data to use in mapping out gestures to use.
3. Finally, if we were to make this a viable product, we would need to develop a sleeker method to house our components and package our product. Our double-glove method worked as an excellent prototype, but was still bulky with our rigid PCB. One idea for how to develop a sleeker design would be to utilize a flexible PCB. This paired with a wireless interface would result in a truly seamless experience.

In the end, there are many ways to improve on our design. We do not intend to continue the project on our own but have left our development in a open-ended state with software that could easily be added to and

hardware with design intention of being improved upon. We learned a great deal about developing a product in this course and by leaving it open ended we hope others can explore our ideas and even improve upon them.

References

[1] “LSM6DSL Datasheet,” *STMicroelectronics*. https://www.st.com/en/mems-and-sensors/lsm6dsl.html

[2] “MF01A-N-221-A01 Datasheet,” *Mouser.com*, 2026. https://www.mouser.com/datasheet/3/140/1/MF01A_____A01.pdf

[3] “STM32F405/415 - STMicroelectronics,” *STMicroelectronics*, 2025. https://www.st.com/en/microcontrollers-microprocessors/stm32f405-415.html

[4] “LD1117 Datasheet,” https://www.st.com/content/ccc/resource/technical/document/datasheet/99/3b/7d/91/91/51/4b/be/CD00000544.pdf/files/CD00000544.pdf/jcr:content/translations/en.CD00000544.pdf

Appendix A Schedule

Table 2 Schedule

Week	Tasks
2/23	**Kobe:** Begin testing DSP algorithms in Python **Kenobi:** Brainstorm ideas for physical glove design **Connor:** Development of V1 PCB testboard schematic
3/2	**Kobe:** Start getting real sample data **Kenobi:** Design V1 testboard PCB layout and send for manufacturing **Connor:** Begin development of V2 PCB, aiming for a final design which we can put into a glove.
3/9	**Kobe:** Create live visualizations for data for debugging **Kenobi:** Re-design PCB for second round and send for manufacturing **Connor:** Solder and begin testing of the V1 PCB
3/16	*Spring break*
3/23	**Kobe:** Write python code for DSP methods, run on recorded pre-recorded sensor data **Kenobi:** Assemble and debug testbench PCB **Connor:** Implement and test USB connectivity with HID and serial connection.
3/30	**Kobe:** Additional week for researching/testing DSP methods **Kenobi:** Update PCB to fix IMU wiring issue for final round order **Connor:** Troubleshoot IMU wiring issue with “surgery” as solution.

4/6	**Kobe:** **Kenobi:** Start physical glove assembly using dummy PCB for now **Connor:** Build upon Kobe's initial IMU connectivity code, making it work with our model, receiving successful IMU data.
4/13	**Kobe:** Extra time for translating DSP code **Kenobi:** Finish sewing sensors onto glove and brainstorm wire management **Connor:** Solder final PCB and get ready for Kenobi's implementation with glove.
4/20	**Kobe:** Continue refining gesture set **Kenobi:** Assemble final PCB and connect all the glove mounted sensors **Connor:** Revision of HID functionality and develop keyboard interface.
4/27	**Kobe:** Final verification, working on dynamic gesture stretch goal **Kenobi:** Final testing and sewing the outer glove to secure electronics **Connor:** Finalize PCB and develop serial/HID dual mode for communication with computer.

Appendix B Requirement and Verification Table

Table 3 Requirements and verification table

Requirement	Verification Method
Gesture classifier achieves ≥ **90% accuracy** with ≤ **5% false positives**, over ≥ **50 trials per gesture**.	Conduct structured gesture-classification tests with demo
User completes navigation task with ≥ **80% success rate** and ≤ **1.5× baseline time**.	Run user study in testbed app; record completion time and success rate; compare to mouse-keyboard baseline.
IMU reports data at ≤ **500 Hz** with **SPI at 7 MHz** and **drift manageable**	Analyze drift to be usable and SPI connection full
USB interface enumerates within **2 seconds**, operates at **full-speed (12 Mbps)**, and sends HID reports at ≥ **100 Hz**.	Use USB protocol analyzer; measure enumeration time; verify report rate; confirm OS recognition without drivers.
Physical frame must be wearable, secure, and allow full finger motion.	Conduct fit tests with multiple users; evaluate comfort, range of motion, and sensor stability during gestures.