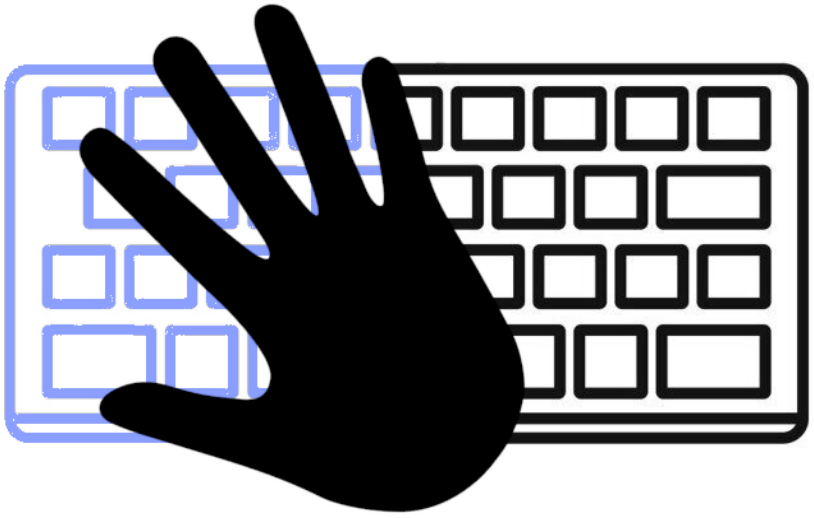
TouchTyper

A finger-tracking keyboard input device.



Nathan Dorson, Aidan Dougherty, Timothy Frieden, Nicholas Gale, Christopher Lackowski, Steven Scangas (Group C5)

Charles DiMarzio

Capstone II

Fall 2023

Abstract

In this paper we discuss the development of the TouchTyper gloves: a pair of gloves which track finger movements and allow translation of the wearer's actions to keystrokes on an imaginary keyboard. The gloves constantly stream sensor data to a host device, where the data are run through algorithms to track all the finger positions, and these positions are mapped onto a virtual keyboard, allowing the wearer to type without the use of an actual keyboard. This set of gloves is more portable than a normal physical keyboard, as well as more discreet. In theory the gloves could be used with varied devices such as smartphones, tablets, laptops, and desktops. The core piece of technology – the finger tracking – also can be imagined to have further applications such as for sign language translation, playing MIDI instruments, or controlling robots, to name a few.

To use the TouchTyper device, the user wears the gloves, which are connected via wires or paired over Bluetooth to the desired computer or other smart device. The host device runs a program to track the inertial measurement units (IMUs) on each fingertip. Acceleration, gyro, and magnetometer data from every finger are transmitted from the gloves to the host device at around 30Hz, allowing the rotation to be tracked and allowing linear acceleration to be twice integrated to determine the position offset of each fingertip. To counteract drift and account for sensor noise, the TouchTyper team implemented a number of filters and heuristics that remove constants of integration and keep the system in a known state at all times. With proper tracking, the keypress detection routine outputs detected keypresses, and also feeds the keypress detections back into the position estimation routine in order to increase confidence in the tracked position with every keystroke.

For testing and demonstration, the TouchTyper team set up a virtual 3D scene containing a keyboard model and sensor models for each tracked finger, and had the sensor models track the physical sensor positions. In this way, it was possible to visually assess whether the sensors were being tracked correctly and were hitting the correct keys. A single set of gloves was created to show the proof of concept of finger tracking and the application thereof for keyboard input. Cost restraints dictated that an affordable IMU be selected for the project, but even more success could be achieved with higher grade sensor components, since sensor inconsistency caused the most issues when trying to achieve accurate tracking and keypress detection. When prices for such sensors drop in the future, these sorts of input devices may be more viable.

Introduction

There are many ways to do textual input on a computer: standard keyboards, chorded keyboards, voice dictation, and more. One niche that has not yet been fully explored in this realm is the wearable keyboard. The advantages of wearable keyboards are their portability, customizability, and adaptability to different work surfaces. There are existing products, such as the TapXR [1], which attach to the hand and allow the user to perform input, but only after learning a completely new input scheme to that of a traditional keyboard. For our capstone project we proposed a pair of gloves that would allow the user to type with their existing muscle memory, as if using a standard QWERTY keyboard.

For this project, the goal was to create a pair of gloves, which we called TouchTyper, that can act as wearable keyboard input to a computer. This was accomplished using IMUs to precisely track the user's fingers and supply virtualized keyboard input to the computer when the user moves fingers as if to press keys on an imaginary keyboard in front of them. The targeted benchmark was to accurately reproduce at least 90% of user keypresses, assuming perfect typing form.

We believed that if TouchTyper could be made to successfully act as a keyboard input device, then there would be many ways to extend the technology. One such extension would be to use the gloves as a virtualized synthesizer that supports the MIDI format. By substituting the conventional typing keyboard's keymap with a customized one for a MIDI controller, users could emulate pressing piano keys and buttons in a comparable manner. Another way to extend the technology would be to use the finger tracking capabilities to do sign language translation in real time. Since the gloves keep track of every finger's position and orientation, this data could be analyzed and used to convert sign language into text. A final interesting project that would expand on TouchTyper's technology would be to use electrodes on the user's forearms to try to detect keypress intention. Since all finger movement is driven by electrical impulses that have to travel via the forearms, these signals could be captured in parallel with TouchTyper use, and a model could be trained to interpret the electrical impulses. This would allow a new, even less intrusive device to be built that would let the user's intended text to be printed on a screen with nothing but an armband.

Similar Projects

In our preliminary research, we sought out existing devices that would be similar to the TouchTyper in design and achievement in order to support our design's plausibility. One of the first examples studied was the Quantum Precision XR, released commercially in 2022 by Manus [2]. Listed at $5,500 USD, the gloves were advertised for their pinpoint accuracy and state of the art virtual reality experience, particularly in the contexts of technical training and virtual prototyping. The Quantum gloves and TouchTyper were similar in design; a single small sensor would be fixed at each fingertip, right on top of the nail bed area. Each sensor would have a short

but flexible flat cable running from the fingertip to a rectangular controller/computer of sorts that is based on the back of the hand. While the two devices were to be near identical in their layout, the sensors on the Quantum fingertips were not IMUs, but “absolute position” sensors, of which there was no elaboration provided in its hardware specifications. Manus advertised the Quantum as featuring zero drift, but it was not clear whether the sensors or the processing was more responsible. While similar in principle, the Quantum gloves were simply out of the scope of our project in terms of budget and complexity, but remained an inspiring product suggesting the plausibility of finger tracking.

The most relatable device we came across was the CIE-DataGlove, an IMU based system that could track the hand posture of its wearer and display it via computer generated hand model [3]. The glove specifically aimed to achieve this feat without the use of mechanical finger resistance, while also retaining posture accuracy as the user handles objects. In general design, the glove was fairly similar to our plan for TouchTyper; IMUs were placed on each finger and wired to a controller board fastened on the back of the hand. The most notable difference from our design was that the DataGlove’s posture tracking system featured multiple IMUs on each finger to most closely capture the location of each finger segment, as to emulate a realistic jointed hand. The DataGlove’s performance was reassuring, and the flexible wiring arrangement featured on the DataGlove was something we were inspired by and aimed to match in both functionality and ergonomics.

Analysis & Design

Design Strategy

Several options were considered in this project to track the fingertips in a way that would be both cost effective and accurate. The first option that was considered was flex sensors, which would be able to determine how much a finger was bent. However, it would be difficult to determine side-to-side motion of a finger, so there would be difficulty determining the actual location of the fingertip. The next sensor option that was considered was string potentiometers. This would have a similar issue of determining lateral motion of the finger, but could be avoided by utilizing more sensors. Unfortunately, string potentiometers are both large and somewhat expensive, so they were decided against. The third sensor option that was considered was inertial measurement units. While these measure acceleration and rotational velocity as opposed to some absolute metric relating to position (as with flex sensors and string potentiometers), they are both small and inexpensive. They require more software to ensure accurate tracking, but are accurate enough and sufficiently constrained by our use case that getting location data is possible.

Inertial measurement units (IMUs) are a fundamental part of the functionality and design of the TouchTyper. Utilized extensively in aircraft navigation and stabilization systems, IMUs consist

of miniscule accelerometers and gyroscopes oriented along three reference axes. These sensors measure the rate of acceleration along each axis, as well as the angular rate about each axis. While IMUs are commonly used to track the location of their host components, they only provide raw data from the internal accelerometers and gyroscopes. As a result of this, they are known for being precise in their measurements but inaccurate to their position; if the acceleration data is integrated to find position without an adequate data filtering process, inevitable signal noise is included in the integration and results in significant drift from the sensor's correct position. Intricate systems involving additional sensors, data filtering and integration algorithms are crucial to properly and accurately obtain position coordinates from an IMU, especially in small scale position tracking applications such as the TouchTyper.

In most IMU applications, including our own, a magnetometer is used in tandem as a way to secure the device's orientation information. The local magnetic field is measured by the magnetometer and used as a secondary orientation reference, aiding to minimize drift from the gyroscopes in the IMU. Calibration is also a very important aspect to our design that is not always associated with the use of IMUs; whether constant or user activated, quick and frequent calibrations allow for the IMUs to have a universally defined starting point to keep the physical and digital IMU locations as uniform as possible.

Several different IMUs were researched to determine the best fit for the TouchTyper. Some requirements for the IMUs were that they be inexpensive, have a reasonably high data collection rate, and be as precise as possible while fulfilling the other requirements. The MPU6050 was selected because it met these criteria, and outperformed the other sensors which were tested in the prototyping phase.

Technical Foundation

Dead Reckoning

The foundation of the method for tracking IMUs is that of dead reckoning. This is a process by which position is calculated using the prior position, as well as estimates for heading, velocity, and change in time. Velocity is known via integration of the reported acceleration from the IMUs. Dead reckoning is accurate on short time scales, but as more time elapses, positional drift grows due to unavoidable errors in sensor reporting. This is a known issue that we take several steps to counteract.

For our specific application, for the position to be tracked in a global frame of reference, the linear acceleration values captured from the IMUs must be corrected at every time step, as they are otherwise in the rotated reference frame of the IMU itself. To do this, the initial orientation of the IMU is known based on some calibration motion, combined with an assessment of the gravity vector and magnetic field, and then subsequent rotation away from the initial orientation is tracked very carefully. The way this is performed is by taking the rotational velocity around the three axes at each time, multiplying by the time step and converting these pitch, roll, and yaw values into a representative quaternion. The rotation quaternion for each time step is multiplied into the global orientation quaternion, tracking orientation over time. However, this is not sufficient, due to possible drift in the IMU gyro data. Therefore a filter is used to combine the precisely tracked rotations over time with magnetometer data and the acceleration due to gravity, which act as somewhat less precise global truths about orientation. Some viable attitude estimators exist which do just this, such as the Madgwick orientation filter, or the Mahony orientation filter [4].

Calibration

IMUs are subject to both noise and persistent bias. Noise cannot be dealt with ahead of time, but each individual sensor can be pre-calibrated to account for fixed errors. One method of doing this is setting up a linear corrective system, where known calibration scenarios are used to find two separate error offsets for each axis of an IMU, and then setting up a linear mapping between the reported values and the correct values.

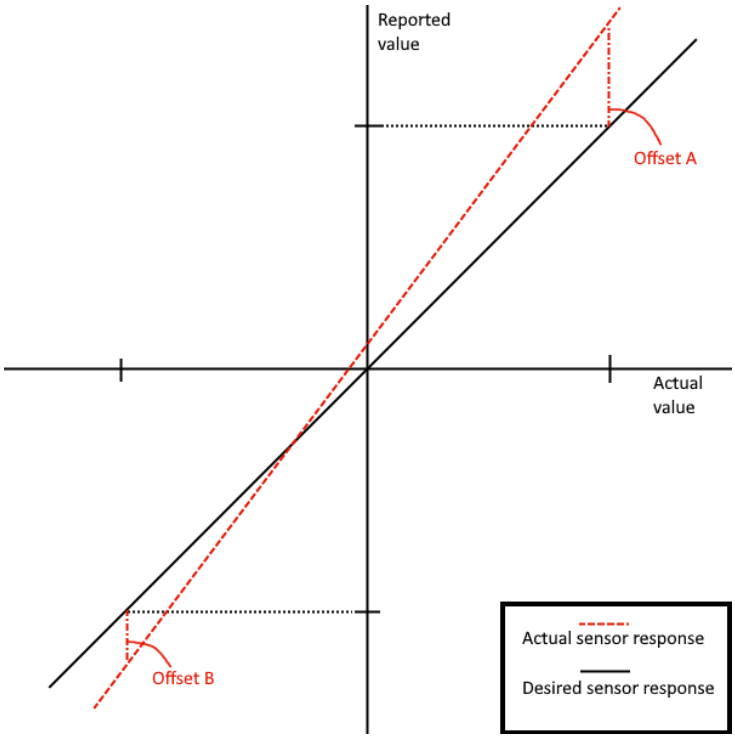


Fig. 1 - Finding a linear offset for a biased sensor response

Filtering and Heuristics

A series of high pass filters may be used to further remove persistent bias in the signals. Even if calibration is done very well, it may still not be able to remove all DC level signals from the sensors. A high pass filter on the acceleration data from the sensors may help with this, before the acceleration data is multiplied by the time step to find the change in velocity. A low pass filter on acceleration could have the effect of reducing the influence of noise on the system. Furthermore, high pass filters on the velocity data and position data could have the effect of "pulling" velocity and position back to zero whenever there is not any active change in those signals, thereby eliminating drift due to uncompensated bias in acceleration [5]. This may be appropriate for our application, seeing as keypresses are generally a single, quick tap of a finger, which then returns to a neutral position. As an example, see the figure below, which depicts the results of using a custom high pass filter to remove the constant offset of -0.8m/s from an example graph of velocity.

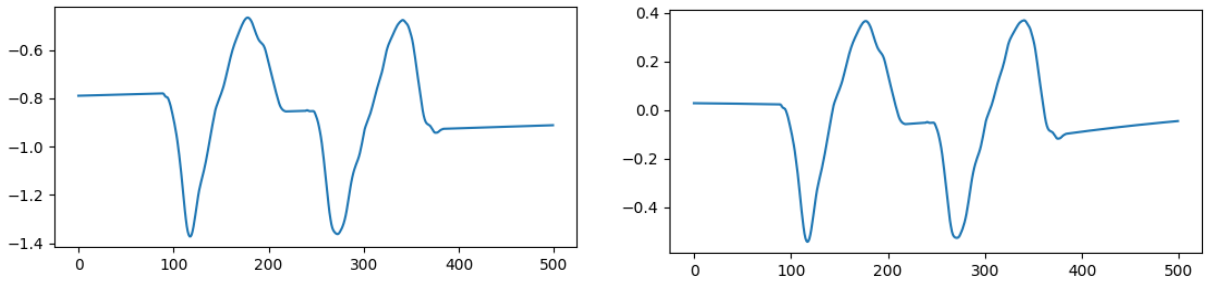


Fig. 2 - Third-order High Pass Filter on Velocity Data

Since our system is running in real time, we are restricted to causal filters, which means that the filtered outputs are slightly time-shifted. We aimed to minimize the latency created by this effect by sampling as quickly as possible and selecting filters with minimal time delay. We also had to carefully consider the cutoff frequencies and order of any filters we use; the figure below depicts the difference in frequency response for different filter constructions. We want to keep in mind that higher order filters may give better results, but also result in a heavier processing load on the host computer when repeated for every sample. We also want to select a cutoff frequency that eliminates as much near-DC level signal as possible, while preserving all of the user actions.

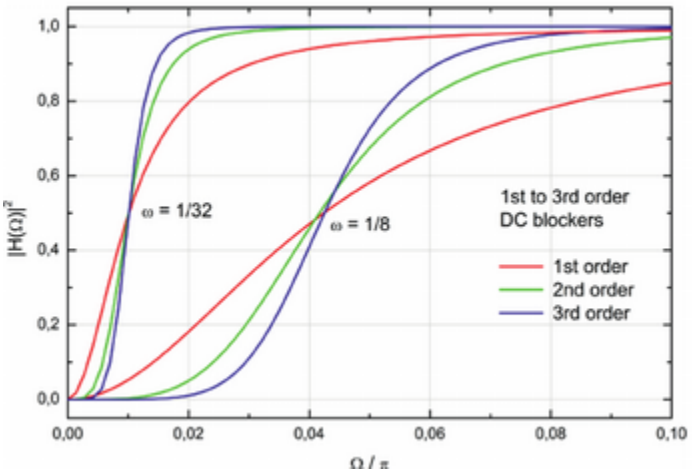


Fig. 3 - DC Blocker High Pass Filter Frequency Responses [6]

Another technique for improving tracking accuracy and reducing drift is resetting velocity when a keystroke is detected. It is known that at the instant of a keystroke, the fingertip should be stationary, so as long as keystroke detection works correctly, the velocity may be frequently reset to zero, which helps to mitigate the accumulation of error over time when integrating acceleration values.

System Block Diagram

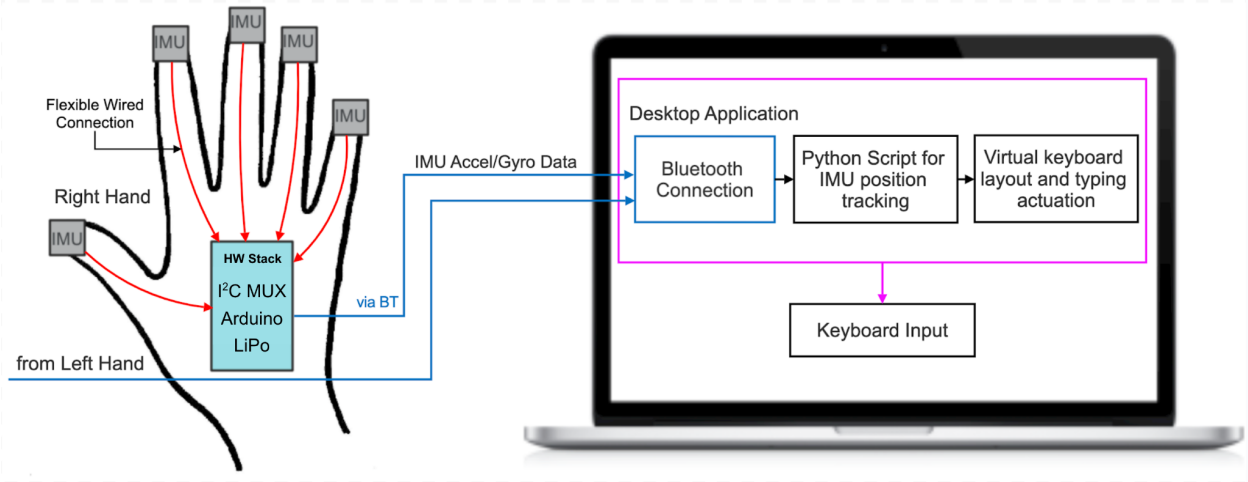


Fig. 4 - Basic System Block Diagram

The above figure presents the basic outline of the entire system. The user wears two gloves, both equipped with an enclosed hardware stack for data multiplexing and wireless transmission, as well as sensor PCBs fastened to each fingertip. Each stack consists of an Arduino Nano 33 BLE, LiPo battery, LiPo/Arduino connector board, and a custom PCB for 5-way I^2C switching. The multiplexer board on each hand is wired out to the five sensor boards, each featuring an IMU and a magnetometer, thus retrieving rotational velocity, linear acceleration, and magnetic field from each fingertip. This data is passed to a host computer via a wired or wireless connection, where it is analyzed to track the fingers and detect keypress attempts to pass along to the operating system as a keyboard input.

Hardware

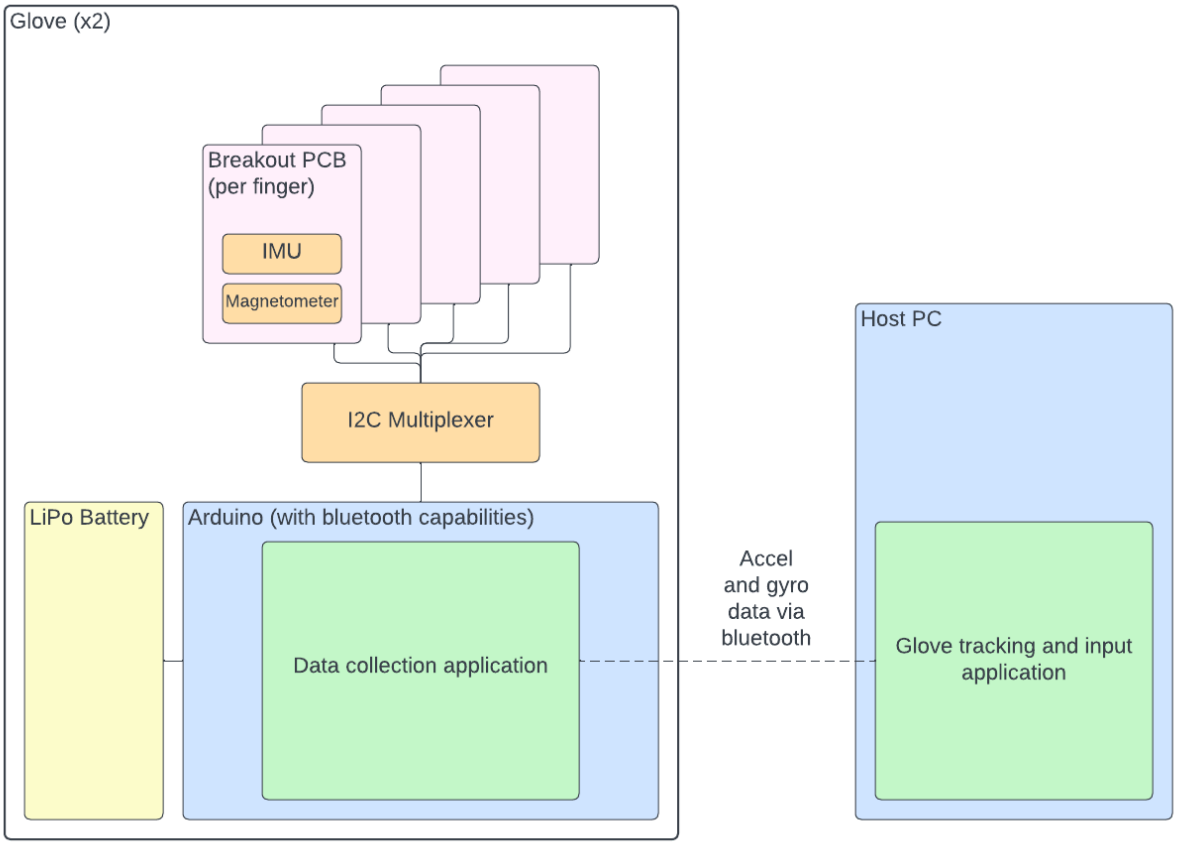


Fig. 5 - System Hardware Block Diagram

The hardware design of the system is as shown above. An Arduino Nano 33 BLE was used as the microcontroller for each glove, and was powered by a lithium polymer (LiPo) battery via an existing adapter PCB [7]. The Arduino is connected via the I2C protocol to the sensors on each finger. Since most I2C sensors of the same model use the same I2C address, we added an I2C multiplexer between the Arduino and the sensors so that all sensors can be addressed independently. The sensors on each finger consist of an IMU and a magnetometer. These sensors are laid out on custom designed breakout boards sized appropriately for mounting on narrow fingers.

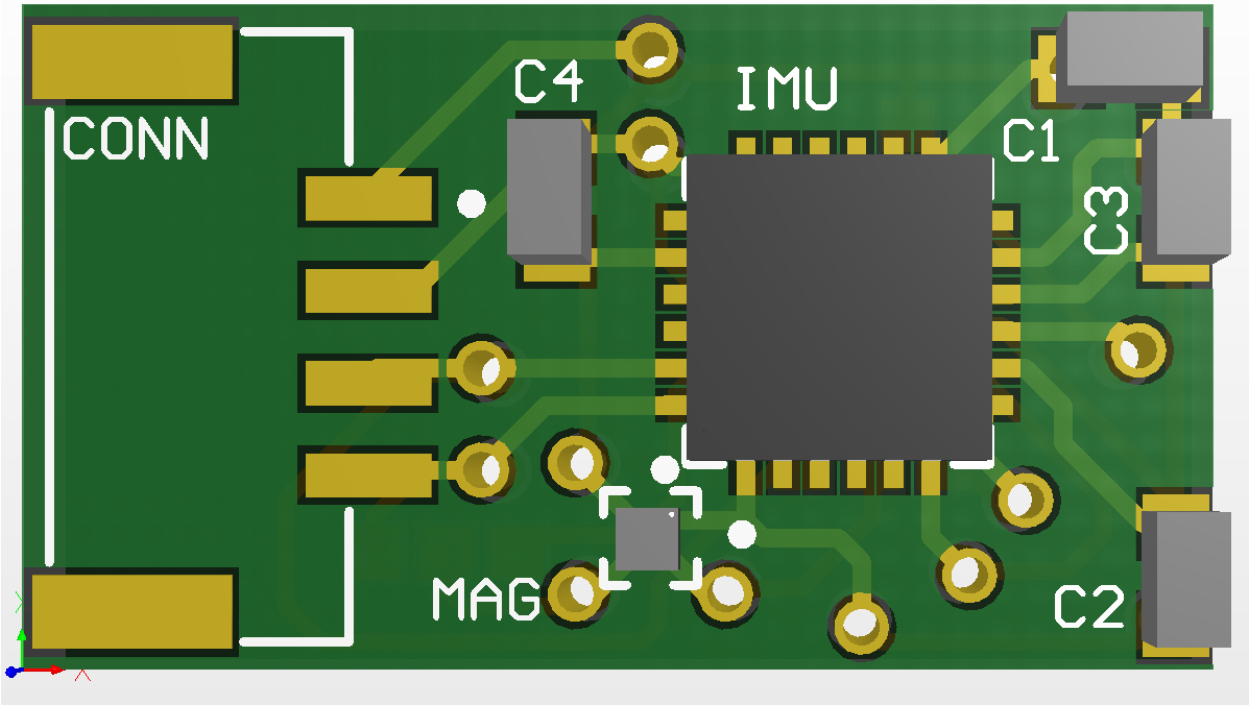


Fig. 6 - Custom IMU PCB Basic Layout

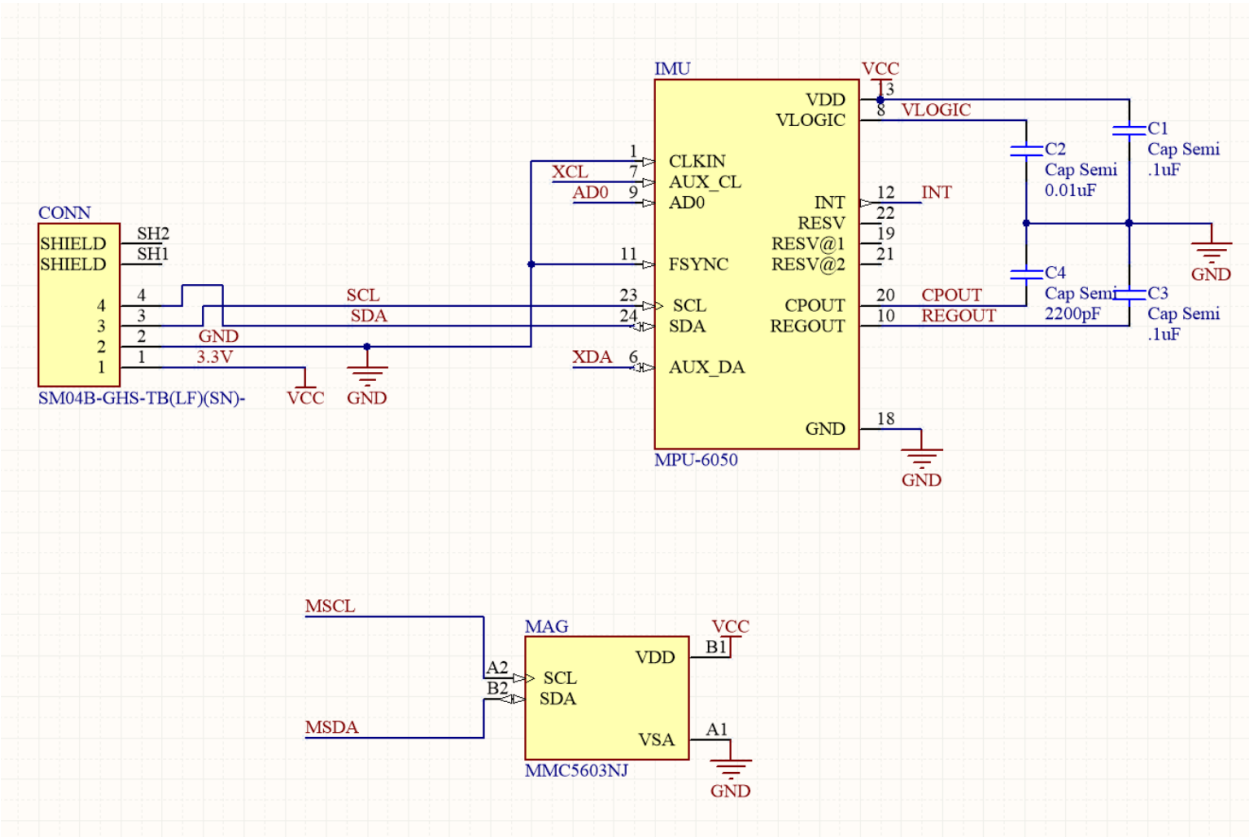


Fig. 7 - Custom IMU PCB Schematic

Shown above in figures 6 and 7 is first, the 3D view of the final design for the IMU PCB and below that is that circuit schematic for said board. The main goal while designing this board was to properly house the IMU and magnetometer sensors and be able to interface with them. The secondary goal was to minimize the size of the PCB in order to create the least amount of interference with the user's typing experience. In total, we assembled ten of these IMU boards, one for each of the user's fingers. The final dimensions of the board was 9mm by 16mm. This meant that the boards were easily mounted on the tips of each finger and did not get in the way of the user's typing experience.

Because of the small size of this board, we opted to only include the necessary circuits required for the sensors to operate and moved all other less critical components to the I2C Switch board. There are three main components that go into this sensor board. First, is the 4-pin connector which is used to interface between the sensors and the Arduino Nano. This connector simply ties 3.3V power, ground, and the two I2C lines together between the sensor board and the microcontroller. The other two components are the IMU and the magnetometer. The IMU has a lot more pins and is physically much larger than the magnetometer. Both sensors are connected together on an I2C bus. The only other parts on this board are the 4 noise reducing capacitors which are not necessary for the IMU's function but did help give more precise motion data.

The assembly of the ten sensor boards was a challenge. All of them were assembled by hand with a soldering iron and a heat gun. We did not use solder paste or a reflow oven. All of the components were extremely small to try to reduce the overall size of the board. This led to a lot of difficulties when trying to solder these sensors with QFN packages. All of the pins for the sensors were located underneath the chips themselves. The magnetometers were especially tricky since they only had 4 pins and they were all completely underneath the main body of the sensor. In the end, we did manage to solder all 10 boards properly and get them all working.

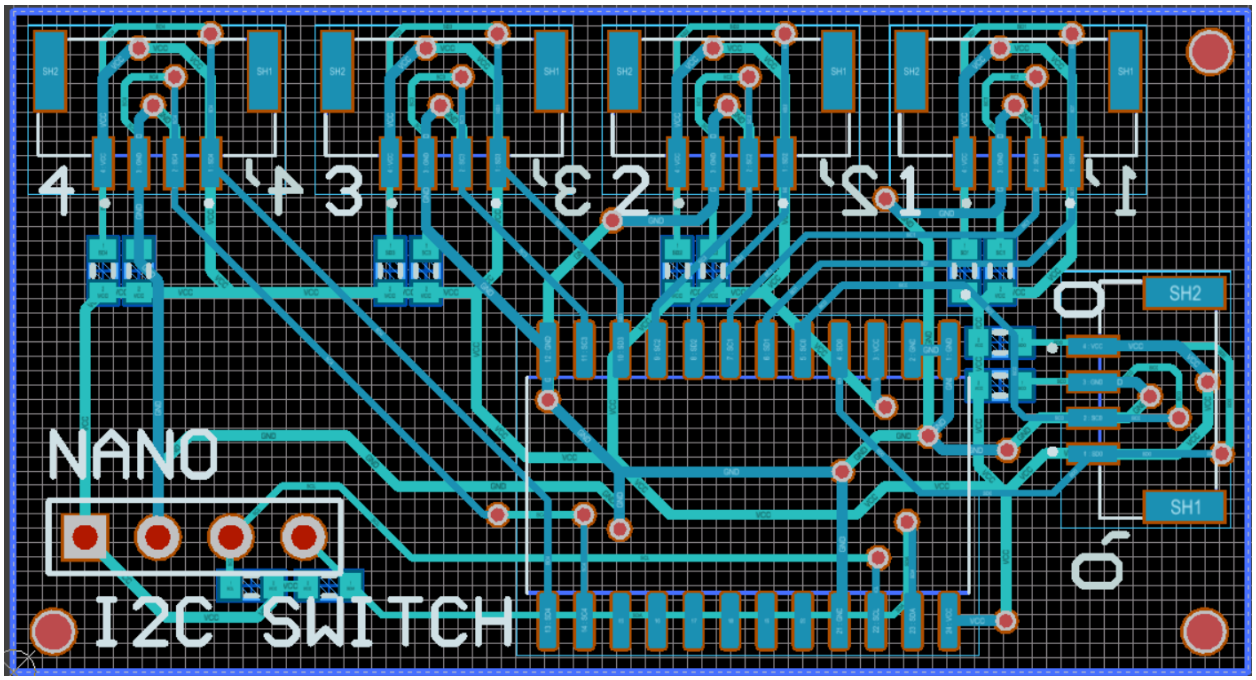


Fig. 8 - Custom I2C Switch Board Basic Layout

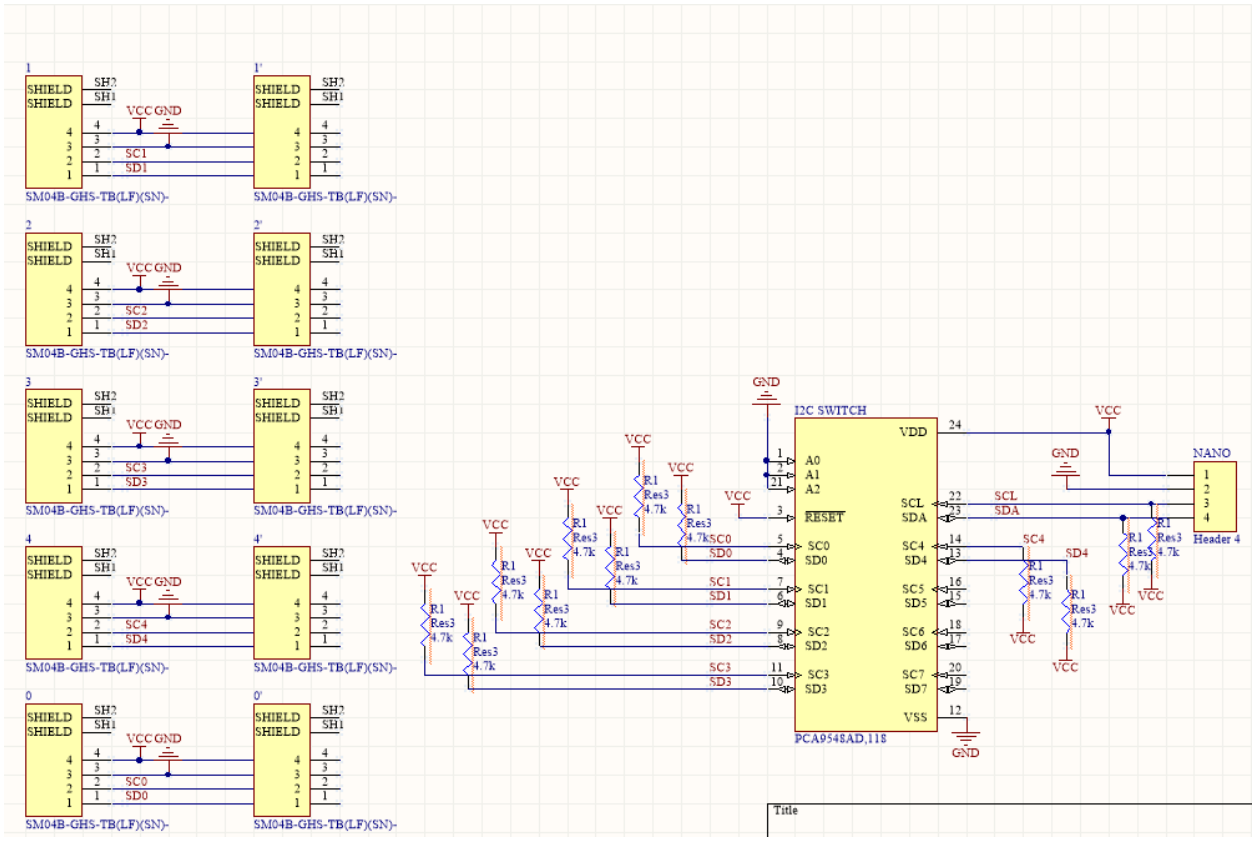


Fig. 9 - Custom I2C Switch Board Schematic

Figure 8 shows the circuit view of the custom I2C switch board that we designed and figure 9 shows the circuit schematic. The goal when designing this board was to create a way to connect all of the sensors on each of the fingers of the Touch Typer with the Arduino Nano microcontroller. This was accomplished by using an I2C multiplexer to take the singular I2C bus that comes from the Nano and multiplex that between each IMU board. We made two of these I2C boards, one for each hand. This meant that each board dealt with switching between five different channels.

Ideally, we would have only needed the one I2C bus coming from the Nano but unfortunately the IMU chips and the magnetometer chips had a set I2C address which meant that if we put more than one of those sensors on the same channel, they would interfere with each other. Therefore, we needed to isolate each sensor board to guarantee that they all functioned properly.

The components that went into each I2C board include the I2C switch, various I2C pull up resistors, and the connectors. Additionally, this board was designed to be reversible. That way we could use the exact same board on both the right and left hands while maintaining an ergonomic design. You can see in the schematic above that there are a total of 10 positions that connectors can be soldered on to each board. Five on the top side and five on the bottom.

One more aspect of the hardware design is the final assembly, which requires the sensors and Arduino to all be mounted securely to a pair of gloves. We achieved this by 3D printing brackets to receive the printed circuit boards, and we then secured the brackets to the gloves using velcro. We decided on this strategy because we wanted to be able to remove the brackets and circuit boards easily if we needed to make changes.

Software

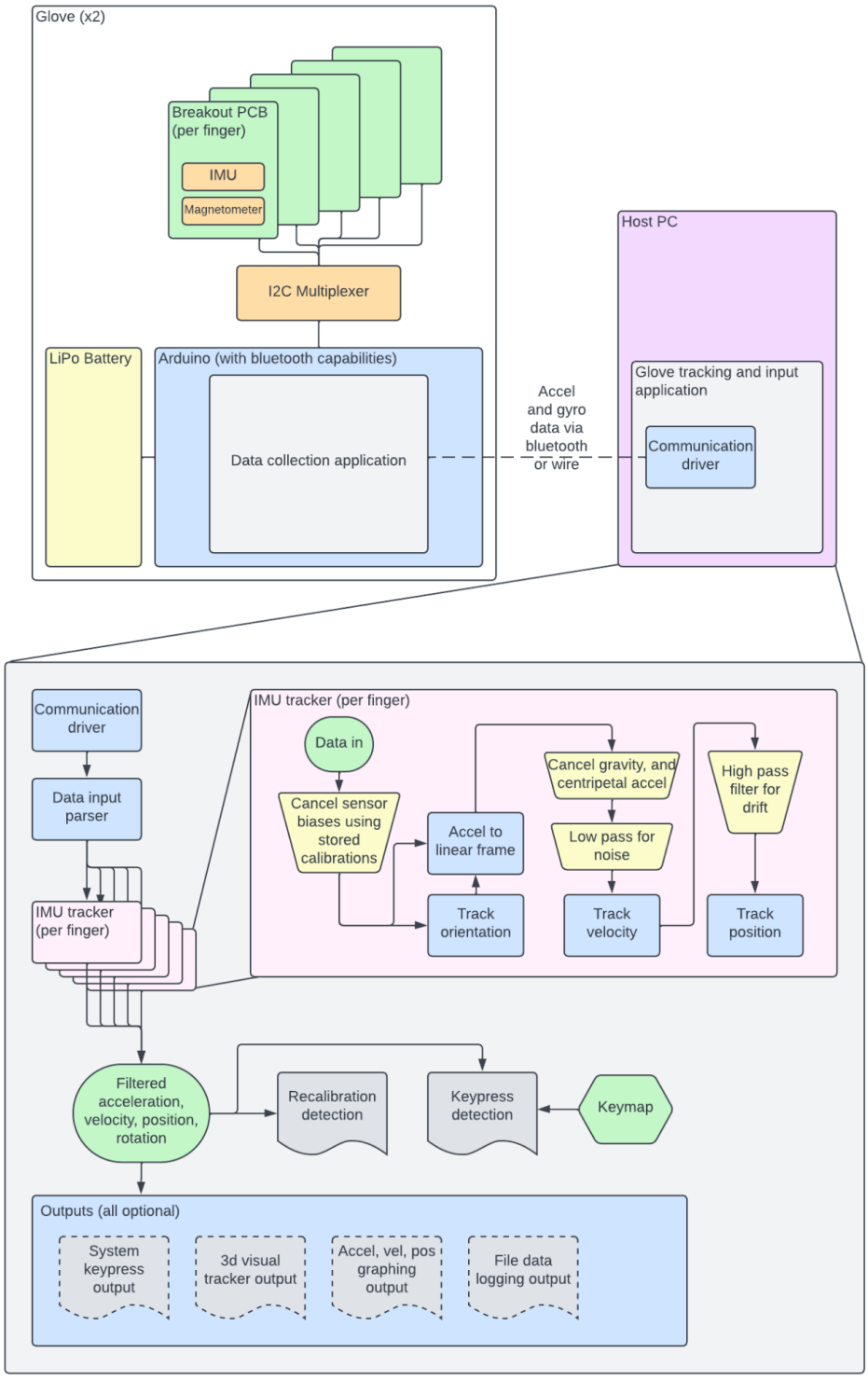


Fig. 10 - Software Block Diagram

There are several different software components in this project. The majority of the software was run on a laptop for testing purposes, but could be made to run on any computer or other device that needs a keyboard.

The only program that is run on the Arduinos is a program to serialize the sensor data into packets at each time step. The packet includes the time, so that the program can account for any missing data points, as well as the IMU acceleration and gyroscopic data at the time step. This can be serialized into a packet that is less than the maximum packet size in the Bluetooth Low Energy and sent to the Raspberry Pi. Shown in Equation 1 is the calculation of sensor data from a single IMU, with 3 4-byte float data points of linear acceleration, 3 4-byte float data points of rotational velocity, and 3 4-byte data points from the magnetometer.

$$3\ lin\ acc.\ \times\ 4\ bytes\ +\ 3\ rot.\ vel\ \times 4\ bytes\ +\ 3\ magnetometer\ \times\ 4\ bytes\ =\ 36\ bytes$$

Equation 1 - Sensor Data Size per IMU

The maximum packet size of the Bluetooth Low Energy protocol is 265 bytes, and as shown by Equation 2, the packet size that we use is 184 bytes [8].

$$5\ IMU\ \times\ (36\ bytes\ (data)\ +\ 1\ byte\ (IMU\ identifier))\ +\ 4\ bytes\ (time\ data)\ =\ 184\ bytes$$

Equation 2 - Data Size per Time Step

To determine the maximum throughput that could be reached, there are various Bluetooth protocol requirements which allow for the calculation of approximate data throughput. The Bluetooth Low Energy protocol requires acknowledgments, which are 10 byte packets, and has a minimum packet gap time of 150µs. Each data packet has a header of 14 bytes, at a transmit speed of 2 Megabits/second [8].

$$((159\ bytes\ +\ 14\ bytes\ +\ 10\ bytes)\ \times\ 8\ bit/byte)\ \div\ (2\ \times\ 10^{7}\ bit/sec)\ +\ (2\ \times\ 150\ \mu s)\ =\ 1032\ \mu s$$

Equation 3 - Time for a single packet to be transmitted and acknowledged

The time that the packet takes to transmit and be acknowledged is calculated to be 1032µs, which can then be turned into packets per second using the following equation:

$$1\ second\ \div\ 1032\mu s\ =\ 969\ Hz$$

Equation 4 - Number of Data Packets per second

This gives a calculated maximum data rate of 969 packets per second, which is enough to adequately track the IMUs. At the beginning of the project, we expected that this would be the limiting factor of data transmission, but through testing, we were not getting anywhere near this results in practice. While we were expecting to get at least 100 Hz, if not more, we were only able to send 20-30 Hz for either wired or Bluetooth transmission.

On the host computer, there are a number of different processes. The first process in the pipeline is a Bluetooth receiver and data parser. This receives the data from the Arduinos and separates it into each of the IMUs so that each of the IMUs can be detected individually. This data is then sent into the next process, the IMU Position Calculation step.

The IMUs return linear acceleration and rotational velocity. To determine the position from these values, the acceleration is first rotated based on the rotational velocity so that the linear accelerations are stable relative to the earth frame. The acceleration can be integrated twice to get the position (a process known as dead-reckoning). When the data is integrated, there is drift due to the constants of integration, so we utilize several filters and heuristics to decrease the drift that is experienced.

The recalibration procedure is one way drift is limited. If none of the fingers have moved for a set amount of time, which we set to 1 second, then the location is reset to the home keys. This is how the user starts a typing session, or resets if they are unable to type correctly. In addition, when a keypress is detected, there is no lateral movement, so the lateral velocities of the pressing fingers can be set to zero, which decreases drift from the first constant of integration. The position is also set to the center of the key. While this does decrease the accuracy of the tracking, it decreases the drift that is accrued from being incorrectly centered. Feedback from this procedure is then sent back to the position calculation to decrease future drift.

There is also a keypress detection process, which determines if a key, and which key, has been pressed at any given time. This is done by looking at acceleration and position. On a tap, the vertical acceleration is very high, as the finger suddenly stops moving downward. The position can also be used on the keypress detection to determine if the fingers are low enough to activate a key, as a sudden acceleration above the predefined keyboard location should not activate any key presses. If a keypress has been detected, the x and y positions are used to determine the key that has been pressed based on the keyboard mapping.

Position Tracking Methodology

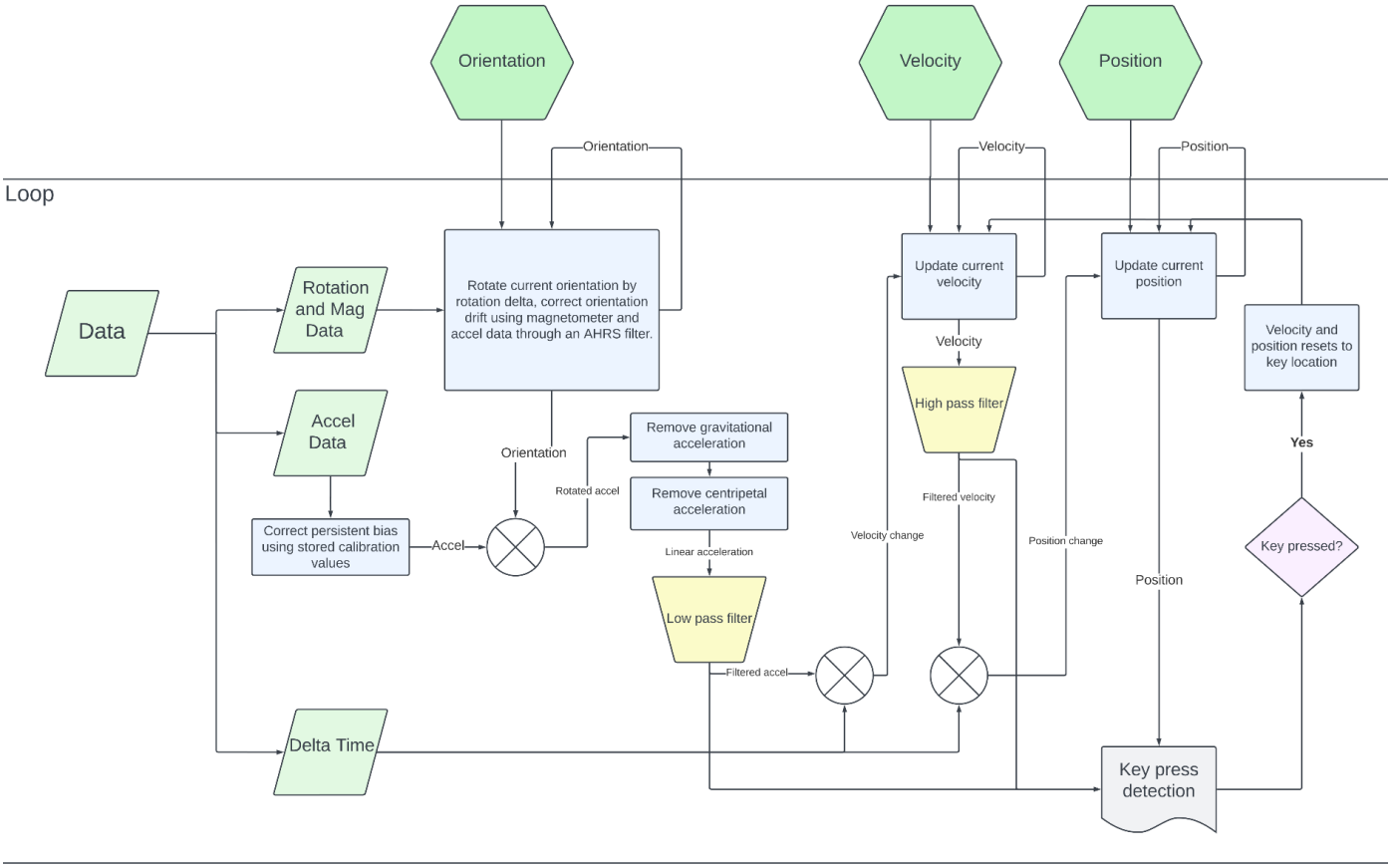


Fig. 11 - Per Finger Position Tracking Algorithm

In order to properly track the IMUs for each finger of the glove, keeping in mind their tendency to accumulate error over time, several techniques are used. The general process is outlined in the flow diagram above, and the following paragraphs go into detail on everything.

After the acceleration data is passed to the computer, the first thing that is applied is the bias. All IMUs have some consistent bias that can be approximated as linear, which we measured for each of the IMUs that are on the prototype. These corrected acceleration vectors are then rotated by the current rotation of the IMU, which was found using the magnetometer and gyroscope data. The rotated vector can then remove gravity, as it is then known what direction the gravity vector is.

After we have a zero-gravity acceleration vector, centripetal acceleration is removed from the vector. This is linear acceleration that will be incorrectly measured due to rotation, so can be

calculated using the gyroscopic data. This vector is then fed into a low-pass filter to remove as much noise as possible.

The final acceleration vector can then be integrated using the timestamp to determine the velocity, which is passed through a high-pass filter to counteract drift. This removes steady state error of the velocity, which would cause the position vector to consistently move away from zero. This filtered velocity vector can be then integrated a second time to determine the position vector relative to its starting location.

The position, velocity, and acceleration vectors are then passed into a keypress detection function, which uses all three of these to determine if a key is pressed, and the position vector to determine which key has been pressed, if one has. If a key has been pressed, the position and velocity vectors are reset to the key location to remove any persistent drift.

Parts and Implementation

Arduino Nano 33 BLE

When we were designing the TouchTyper, we knew that we needed a small board that could utilize bluetooth. The Arduino Nano 33 BLE fits this description perfectly, as it is only 45x18mm [9], which is small enough to fit on a person's hand. It also has inbuilt bluetooth capabilities, meaning that we did not have to have a separate bluetooth module to communicate with the central computer used.

MPU6050

The chosen Inertial Measurement Unit was the MPU6050. Because we would need 10 of these sensors, as well as an additional 2 for backups, cost was a major consideration when choosing these. Another major consideration for this was size, as the IMU would have to fit on a finger, so larger IMUs would restrict movement and typing ability. We performed some testing with the MPU6050 and other similarly priced IMUs, such as the LSM6DSOX, but determined that the MPU6050 was the most reliable IMU for its price point.

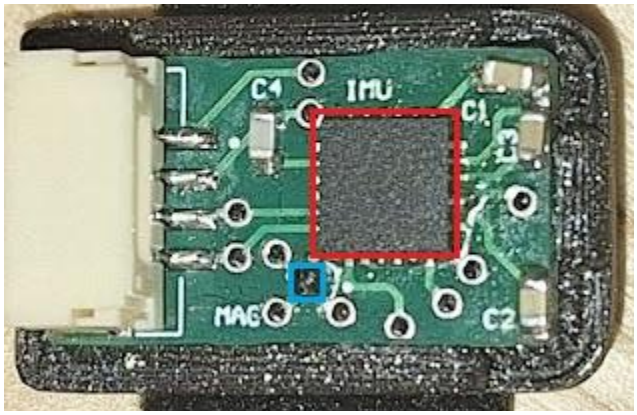


Fig 12. Finger Board with IMU outlined in red and Magnetometer outlined in blue

MMC5603

Because the MPU6050 does not have an inbuilt magnetometer, we needed to also get magnetometers. These had the same restrictions as the IMUs in their size and cost restrictions. Because the IMU took up a significant portion of the available space on the board, the magnetometer that we used, the MMC5603, was only 0.82x0.82 mm [10], and was also inexpensive so that we could get one for each finger.

Housings

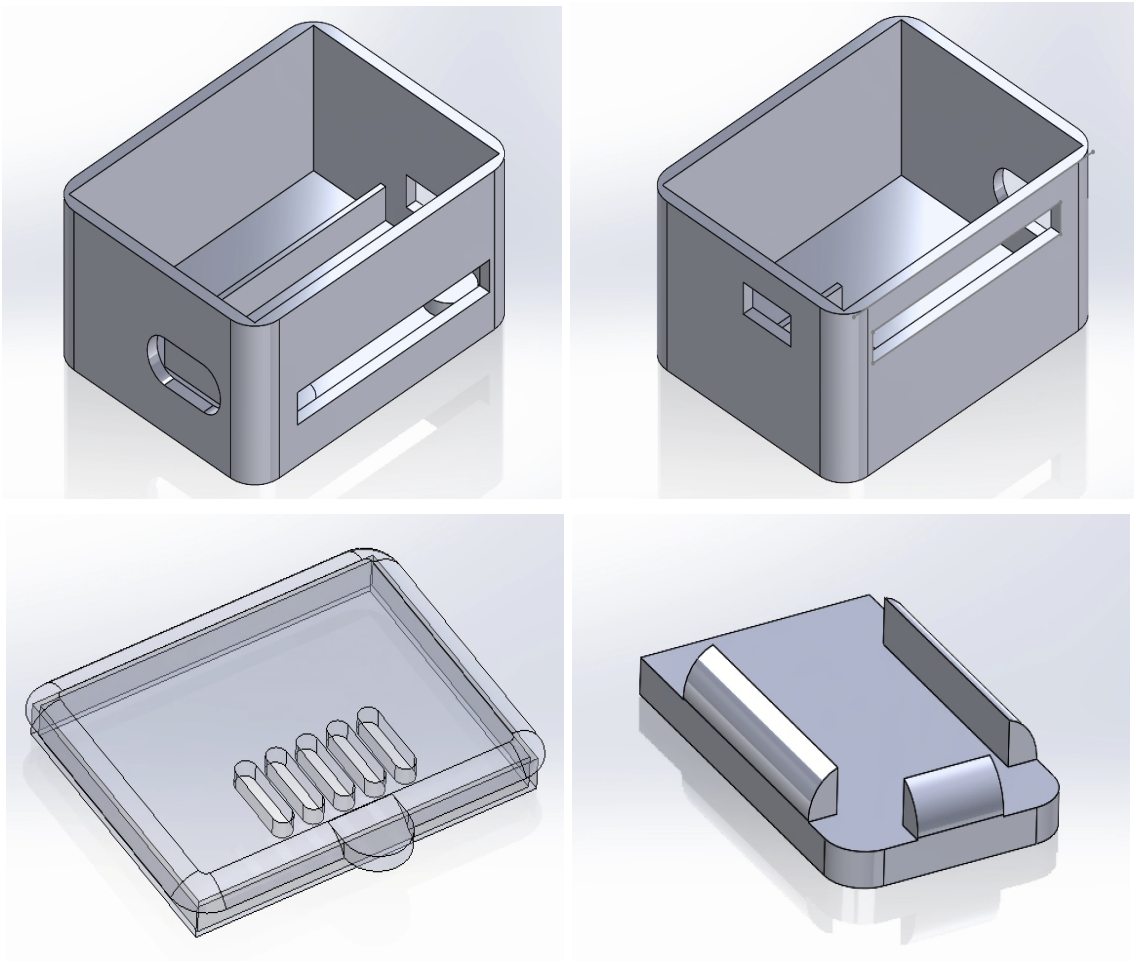


Fig. 13 - 3D Printed housings and brackets

For the TouchTyper's external design, size and flexibility were important factors to consider in order to make the gloves as comfortable and ergonomic as possible. The hardware enclosure and fingerboard mounts needed to be light and allow free movement of the user's fingers. The enclosure and fingerboard brackets were designed in SolidWorks and 3D printed in the capstone lab. Due to the mirrored I2C PCB design, the hardware stack for the left and right hands were assembled in a different order, so the port holes and internal braces in the respective enclosures were uniquely designed. The lid features ventilation holes as well as a tab for easy removal in case of adjustments or repair. The cables from the hardware stack to each IMU board were made an appropriate length to allow the user to make a full fist without putting stress on the cables.

Gloves

The base gloves are standard black compression gloves purchased from Amazon. Each IMU board bracket was glued onto a small, adjustable velcro loop to be wrapped around each of the

user's fingertips; we found that the IMU velcro loops over the gloved fingers felt awkward and harder to control smaller typing motions, so we removed the fingertips of the original gloves for a more comfortable experience. Velcro strips were glued onto the back of the compression gloves and on the bottom of each hardware stack enclosure, allowing the user to adjust the position of the enclosure on the back of their hands as needed.

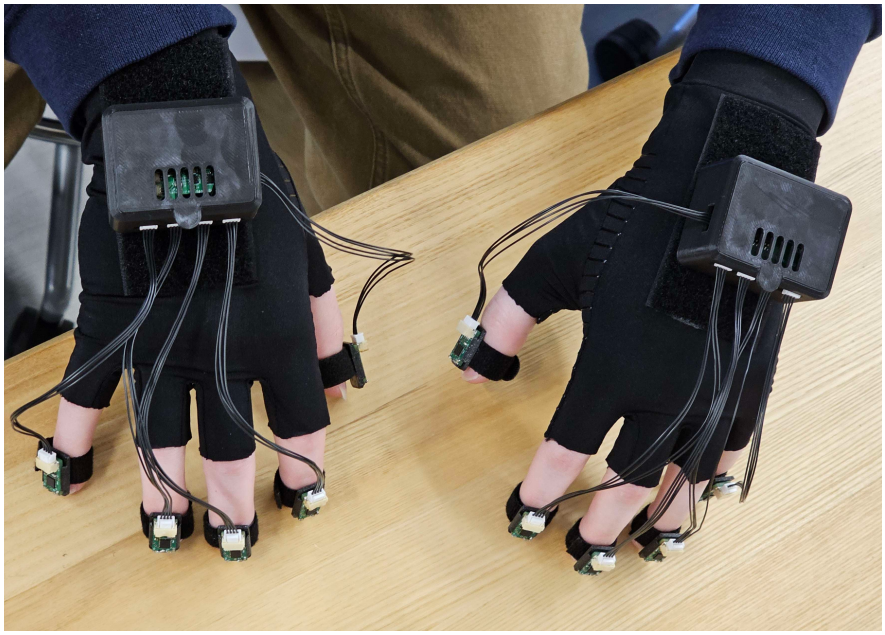


Fig. 14 - Fully-assembled TouchTyper

Design Validation

Many processes were implemented in parallel, so it was critical that there be a way to validate separate parts of the design. The hardware stack was the first thing that needed to be validated, so we were able to make an application on the Arduinos that allowed for the tracking of multiple IMUs at once and would then graph the accelerations without having to worry about data transfer directly to the host PC.

The Bluetooth data transmission also needed to be tested, so a separate Arduino program was written that would send sample data to the host PC repeatedly to determine the maximum data transmission speed. This allowed us to create subroutines and programs to understand the Bluetooth transmission process to ensure that it would be functional for when we had data from the IMUs.

The IMU data processing step was a large project in itself, so we needed to be able to determine if the computer was tracking the IMUs correctly. We had a number of steps that we took to section off different parts of the process, the first of which was the ability to save IMU data to a

file. With this ability, we could decrease the random variability between two tests by just being able to use the same movements repeatedly with different filter setups.

We also had visualizations of the IMUs with both graphs of the acceleration, velocity, and position as well as a mock-up of a virtual keyboard which would just show the position in 3D space. Each of these tools allowed us to move towards a filter and position tracking model that was ideal for our project.

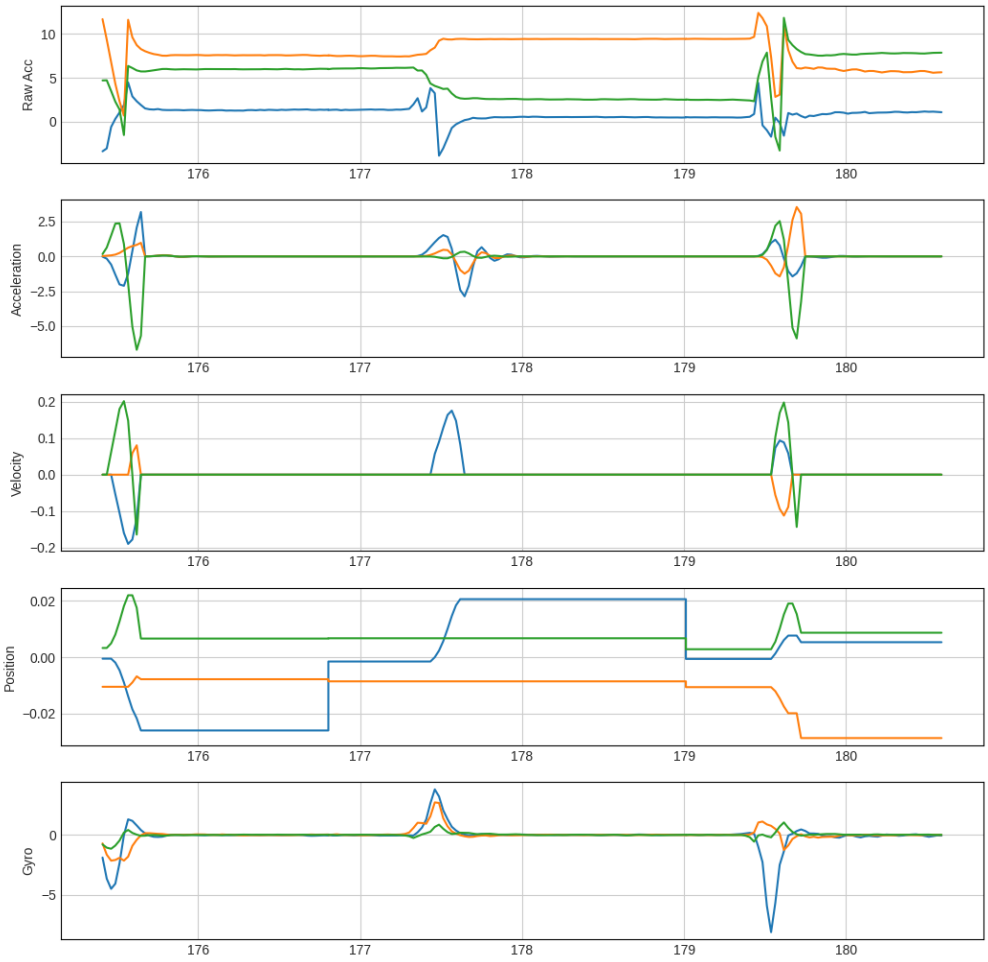


Fig. 15 - Live data graphing example

Filter Selection Iterative Design

From the initial design phase of the project, the design of the filters had to change a lot. Going into the implementation phase, we had an idea of how position tracking could be accomplished, and what sorts of filters we would probably use, but several assumptions were challenged.

Initially we thought that we would use a high pass filter on acceleration, velocity, and position. We had seen an example of IMU tracking in an oscillatory situation where a high pass was used

on both velocity and position [6]. This filtering setup worked well for the authors of that result. However, we had less success because our typing motions were not truly oscillatory, even if we did have zero mean displacement over a long time, so the high pass on position was just dragging the tracker locations back to their starting locations far too quickly. For this reason, we had to eliminate the high pass filter on position, meaning that getting the other filters correct was even more important.

We also initially thought that a high pass filter on acceleration would be useful to cut out any persistent sensor bias. However, with the calibration we performed as well as some tactical thresholding where we would not use acceleration below a certain threshold, we did not end up needing this high pass. Instead, it became obvious to us through our experiments with various filter setups that a low pass on acceleration was far more important to eliminate sensor noise.

In the end we used a combination of the Madgwick attitude heading and reference system along with our own implementation of direct gyro integration to track orientation over time. Then we used a 6th order low pass filter on acceleration with a cutoff frequency of 6Hz, and a 1st order high pass filter on velocity with a cutoff frequency of 0.09Hz. The values were all determined experimentally by looking at the debug graphs and finding filter constants that allowed the data to be critically damped.

Domain Specific Heuristics

In order to make the gloves function with any degree of consistency, we implemented some heuristics specific to the situation of typing on a keyboard. These were recalibration, which is addressed later in the paper, and centering on keystroke detection.

Essentially, with only the filters described previously, the tracked positions of the IMUs could still drift completely away from the location of the virtual keyboard. To combat this, we implemented the system wherein the tracked velocity and acceleration are zeroed on keypress detection, and the position is centered in the pressed key. This way, the tracked positions are almost always kept in a known state, and only leave that state for the short journey to another key. This was one of the crucial improvements to prevent drift over time.

Initial Calibration

All IMUs have some consistent bias that can be approximated as linear, which we measured for each of the IMUs that are on the prototype. This is important because if this bias is not counteracted, there will inevitably be a significant amount of drift using the IMUs, so counteracting this bias as the first step when getting the IMU data is critical.

To get the calibration of each of the IMUs, the sensors were placed on each of their faces, and, while completely motionless, the average error for each linear acceleration reading and each rotational velocity reading was captured and saved to the host computer. This error was calculated by knowing the correct acceleration, which should be exactly 9.81 m/s^2. Then, when using the gloves for input, the data from each IMU can be partially corrected by interpolating the calibration values based on the current orientation of the sensor, and then subtracting the offsets from the live data streams.

Recalibration & Keypress Detection

```
{
    "key_size" : 0.016,
    "key_width_multipliers": [ [1,1,1,1,1,1,1,1,1,1,1,1],
                               [1,1,1,1,1,1,1,1,1,1,1,1],
                               [1,1,1,1,1,1,1,1,1,1,1,1],
                               [1,1,1,1,1,1,1,1,1,1,1,1],
                               [1,1,1,2,2,1,1,1] ],
    "key_order":             [ ["esc","1","2","3","4","5","6","7","8","9","0","-"],
                               ["tab","q","w","f","p","g","j","l","u","y",";","-"],
                               ["lcontrol","a","r","s","t","d","h","n","e","i","o","'"],
                               ["lshift","z","x","c","v","b","k","m",",",".","/","rshift"],
                               ["lwin","lalt","lfn","bkspc","space","rfn","ralt","rwin"] ],
    "rows_offset" : [0, 0, 0, 0, 1],
    "starting_keys" : ["bkspc","t","s","r","a","space","n","e","i","o"]
}
```

Fig. 16 - JSON file for a Colemak keymap

Fig. 17 - Virtual Keyboard from JSON in Figure 16

Fig. 18 - Virtual QWERTY keyboard layout

The goal with determining keypresses was to map the positions of finger presses onto a ‘virtual keyboard’ and then determine which key was pressed on that keyboard. The first step of that process was to create a virtual keyboard that was built with a user-specified configuration. That configuration information was stored in a json file, like the one in Figure 16, with the following options:

key_size	The base size of the keys on the virtual keyboard.
key_order	A 2D array of strings that represents the order of all of the keys. This allows the user to use different keyboard layouts if they prefer that.
key_width_multiplier	A 2D array of numbers that scale the width of the keys, order of these numbers matches up with the key_order to determine what keys to scale
rows_offset	This controls the staggering of the keyboard rows. Normal keyboards have about a 1/3rd keywidth stagger from each row to the next.
starting_keys	The keys that the fingers will be calibrated to the center of.

Following the construction of the virtual keyboard, the program then initially calibrates the fingers with their starting_keys positions. This calibration operation occurs at the beginning of the program but also every time all of the fingers are still within a certain threshold for a certain amount of time. These thresholds are determined by user configuration values. The calibration algorithm involves determining a ‘distance offset vector’ that compares the real-life location of the finger to the center of its starting_key on the virtual keyboard. This is done for each finger so 10 total offset vectors are obtained. Performing calibration at frequent intervals can provide some benefits to the functionality of the keypress detection since it diminishes the impact of IMU drift.

The keypress detection algorithm occurs in 3 steps and requires the prerequisite of having at least one key recalibration cycle done since it relies on the distance offset vectors that were calculated in that process. The first step of the algorithm is to constantly poll for a ‘tapping motion.’ This is done on a per-finger basis, looking to see if any finger has a swift downwards acceleration followed by an upwards acceleration. If these conditions are met, the algorithm moves onto the second step which is assessing if this ‘tapping motion’ is also a ‘key press’. This was determined by examining the height of the finger at the downwards apex of the press. If that height is below a certain threshold, it could be said that the finger had made contact with the key, and a ‘key press’ has therefore occurred. Finally, the algorithm determines which key was actually pressed. To calculate the X and Y positions of the keypress on the virtual keyboard, the finger’s X and Y coordinates at the time of the key press were added to the calibration offset vectors that were calculated prior. This results in the final position on the virtual keyboard that the finger pressed. The actual key that was pressed is determined by rounding those positions up to a whole number, then indexing into the key_order array which outputs the corresponding key name.

Future Work

The biggest improvement going forward would be utilizing better inertial measurement units. Due to cost constraints, the IMUs that we decided to use in this project were very inexpensive, which also means that they were not the most reliable. If a similar project had been done with IMUs that were more expensive, we expect that the results would be much more impressive and the TouchTyper would be significantly more accurate than the current version. We also posit that more accurate IMUs would mean that the software would have to run fewer domain-specific heuristics, and make the gloves more extensible to other applications.

One processing option we could consider in the future is the use of machine learning and waveform matching. The idea behind this is that each distinct character a user types will result in a unique set of waveforms in the acceleration data from the fingertip sensors. A machine learning model could train on sample data that a user generates by typing out a particular calibration text, at which point the model could potentially recognize, or assist in recognizing, when keys are pressed based on the generated waveforms. This approach could either take the place of the direct position tracking, or act as another heuristic for where the fingers are that would be combined with the rest of the data for better accuracy.

Timeline

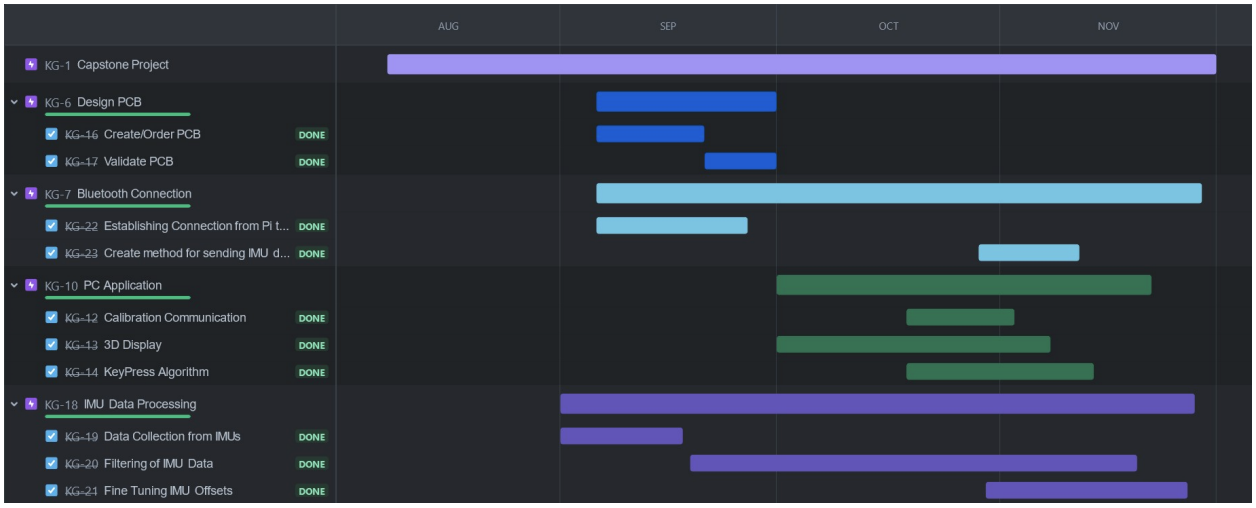


Fig. 19 - Project Gantt Chart

Shown above is the Gantt chart of the TouchTyper project. The first things that were done was the PCB design and validation and the validation of the Bluetooth connection. We were able to show that both of these were able to be used in our project. These are the most important to do early, as ensuring that the correct data gets from the IMUs to the PC is reliant on both of these steps. The next part of the project was the majority of the IMU data processing, which we started early because we knew that there would be a significant amount of trial and error. The IMU data processing feature was a major part of our design, so needed to be worked on continuously throughout the entire project. The next thing which was started was the calibration subroutine, which decreases the drift in the IMUs when the position is calculated. This was worked on in parallel with the rest of the PC application, which includes most of the various software components that run on the host PC, including the position estimation algorithms, which ties into the keypress detection, and the 3D keyboard display.

Cost

Item	Cost	Shipping	Total
Arduino Nano 33 BLE x2	52.60	7.99	60.59
LiPo Battery Packs x4	20.88	0	20.88
Magnetometers	9.88	0	9.88
MPU6050 IMUs	83.63	0	83.63
Other board components	168.45	20.97	95.91
Custom PCB Orders	8.20	18.47	26.67
IMU Boards (decision process)	42.85	10.22	53.07
Gloves/velcro	49.75	0	49.75
		Total Cost:	**400.38**

Table 1 - Bill of Materials

Conclusion

TouchTyper offers an innovative solution that bridges the gap between traditional typing on keyboards and emerging wearable technology. This product's ability to seamlessly integrate users' existing muscle memory with virtual keyboard input removes the need for physical keyboards. By utilizing IMUs to accurately capture finger movements and translate them into keystrokes, TouchTyper ensures a seamless and intuitive interaction, redefining the way people can engage with technology.

The benefits of TouchTyper extend far beyond its immediate applications. This innovative solution holds the potential to completely change human-computer interaction, opening doors to increased efficiency, adaptability, and accessibility. By switching to virtual keyboards, there is much more room for customization. Different keymaps can be made to suit different user's needs such as changing positions of keys to ensure ergonomic typing. This technology can also be extended in the future so that users can redefine the size and layout of the virtual keyboard, possibly not even restricting the user to a single plane, catering to their preferences and comfort. This adaptability not only enhances the typing experience but also creates a better sense of personalization which standard off the shelf physical keyboards do not provide.

Eventually, this technology can be pushed further and applied to many different fields. By removing the need to physically interact with devices and transitioning towards a more virtual environment, this device pushes the boundaries of our relationship with technology. In the future, this technology can be used to allow the gloves to connect to many different kinds of devices expanding beyond just computers. Devices such as phones and tablets would benefit from TouchTyper's capabilities. By giving users an alternative way of inputting text onto the screen, the problem of small devices having keyboards that are difficult to use would be resolved.

There are many components involved in creating TouchTyper including collecting acceleration and orientation data using IMUs, implementing data filtering techniques to counter sensor drift, transmitting the acquired data to the PC application, and displaying the intended keystrokes on the screen. TouchTyper integrates various facets of electrical and computer engineering which we believe showcased the skills within our group. This project contains a variety of hardware and software tasks that culminates for an overall presentation of our capablities. The promising results observed by the TouchTyper team show the proof of concept of the TouchTyper prototype and feasibility of IMU tracking technology.

References

[1] “TapXR,” Tap, https://www.tapwithus.com/ (accessed Aug. 10, 2023).

[2] “Quantum Precision XR,” Manus, https://www.manus-meta.com/products/quantum-precision-xr (accessed Aug. 7, 2023)

[3] T. Mańkowski, J. Tomcysński, & P. Kaczmarek, “CIE-DataGlove, A multi-IMU system for hand posture tracking,” *Advances in Intelligent Systems and Computing*, vol. 550, pp268-276, March 2017, doi: 10.1007/978-3-319-54042-9_24.

[4] M. Garcia, “Attitude Estimators,” AHRS: Attitude and Heading Reference Systems, https://ahrs.readthedocs.io/en/latest/filters.html (accessed Aug. 3, 2023).

[5] S. Madgwick, “Oscillatory motion tracking with x-IMU,” X-IO Technologies, https://x-io.co.uk/oscillatory-motion-tracking-with-x-imu/ (accessed Jul. 28, 2023).

[6] T. Klaus, “Digital DC blocker filters,” *Frequenz*, vol. 75. February 2021, doi: 10.1515/freq-2020-0177.

[7] M. Mizuno, “Nano 33 lipo adapter nano9jira,” Hackster.io, https://www.hackster.io/ebaera/nano-33-lipo-adapter-nano9jira-6e78a1 (accessed Aug. 7, 2023).

[8] *Bluetooth Core Specifications,* IEEE 802.15.1, 2016

[9] “Arduino Nano 33 BLE,” Arduino Official Store, https://store.arduino.cc/products/arduino-nano-33-ble (accessed Dec. 16, 2023).

[10] MMC5603NJ Memsic Inc. | Sensors, transducers | DigiKey, https://www.digikey.com/en/products/detail/memsic-inc/MMC5603NJ/10452796 (accessed Dec. 16, 2023).