W.R.I.S.T.

Edward Lu, Joanne Park, Anushka Saxena
Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—Immersive 3D modeling interfaces, such as hologram pyramids or head-mounted AR displays, demand new and novel input devices for high-fidelity interaction. Current trackpad technology on laptops constrain the user to their displays, restricting mobility. In this paper, we plan to solve this issue by building a system that will allow users to use their forearm as a trackpad to manipulate the view of a 3D model. Our system, called W.R.I.S.T., contains a wearable device with a small footprint, includes a 3D hologram pyramid, and has a wireless interface with low latency communication. We integrated one and two finger swiping gestures to respectively rotate and scale a 3D model displayed on a custom-build hologram pyramid.**

***Index Terms*—3D Modeling, Hologram Pyramid, MQTT, Sensing, Wearables, Wireless**

I. Introduction

The rise of digital 3D environments will improve the way professionals give presentations. With 3D models, presenters are able to interact with their audience while also effectively engaging with the content on their screens. However, current technology is limited in how we display and interact with these 3D models, especially in terms of mobility. For instance, when university professors give lectures, they typically walk around the room while projecting their 3D content onto a 2D display. In order to interact with the content on the screen, they ultimately have to walk back-and-forth from their computers and audience members, disrupting the flow of the lecture. This simple solution to this is to use devices like wireless clickers. If the content was more complex, like a 3D model that requires more than just a click to interact with, clickers are not the best tool to use. There needs to be a new input device that can be used to interact with 3D models while a presenter is mobile.

We present W.R.I.S.T.[1], a system aimed to enable a user to use their forearm as a touch interface to control the view of a 3D model. W.R.I.S.T. allows for mobile, wearable sensing using a surface that is nearly ubiquitous to all human beings: skin. Our bodies are always with us, no matter where we go, so it is natural to use it as an interface for computing. W.R.I.S.T. includes two main components: a wearable device with a compact form factor and a 3D hologram pyramid to view a 3D model. Most of the graphical interfaces of W.R.I.S.T. are Web-based, able to run in the browser of nearly all devices that have one. W.R.I.S.T. is an accessible, small, and intuitive interface for scenarios that require interacting with 3D models in a remote manner.

II. Use-Case Requirements

When a user puts on the W.R.I.S.T. device, their forearm should transform into a surface capable of recognizing two gestures: one finger swiping and two finger swiping. Each gesture is respectively mapped to two types of manipulations of a 3D model: rotations and scaling.

For a one finger swipe, a user should be able to point their finger and tap the skin on the forearm of the arm wearing the W.R.I.S.T. wearable and swipe across in any direction. This gesture should cause a 3D model to rotate in proportion to the direction and displacement of the user's finger swipe. For instance, if a user displaces his or her finger by 10mm, we expect the model to rotate approximately 5 times less than if the user displaced their finger by 50mm.

For a two finger swipe, a user should be able to point two fingers (thumb and index or index and middle) on their forearm and drag their fingers towards or away from the device, which correspond to growing and shrinking the 3D model, respectively. Again, the direction and displacement of the two finger drag should proportionally scale the 3D model.

All of the sensor data will be transferred wirelessly to ensure full mobility. Processing gestures will happen on a user's laptop.

The 3D model will be displayed on a 3D hologram pyramid made using four acrylic sheets and a standard screen. The model should appear to be holographically rendered onto the acrylic sheets. A user should not be able to notice a large delay between their gestures and the actions done to the 3D model. Additionally, the pyramid should be able to be displayed on any standard Web browser.

III. Architecture

With these requirements in mind, our solution is to build a small device that is slightly larger than a smart watch. The user will be able to move their fingers in front of the device on their arm as if they are moving their fingers on a trackpad. These gestures will be translated into a geometric transformation and sent to the scene via a Web application and be projected onto the hologram pyramid.

There are five main blocks: PCB, microcontroller (MCU), Edge Server, Web Application, and Hologram Pyramid. You can refer to Figure 1 for a block diagram of our application. The PCB and MCU are attached to a wearable wristband. We may sometimes refer to this device as the "wearable" or "Tony." The PCB contains sensors used for finger detection. The MCU gathers all the sensor data and sends it to the Edge Server. The Edge Server performs finger recognition and gesture classification. The Web application takes in finger coordinates associated with a gesture and applies filtering and noise reduction. Then, it will apply the gesture to a 3D model. Finally, the hologram pyramid projects the 3D model.

[1] WeaRable Immersive Sensing Technology

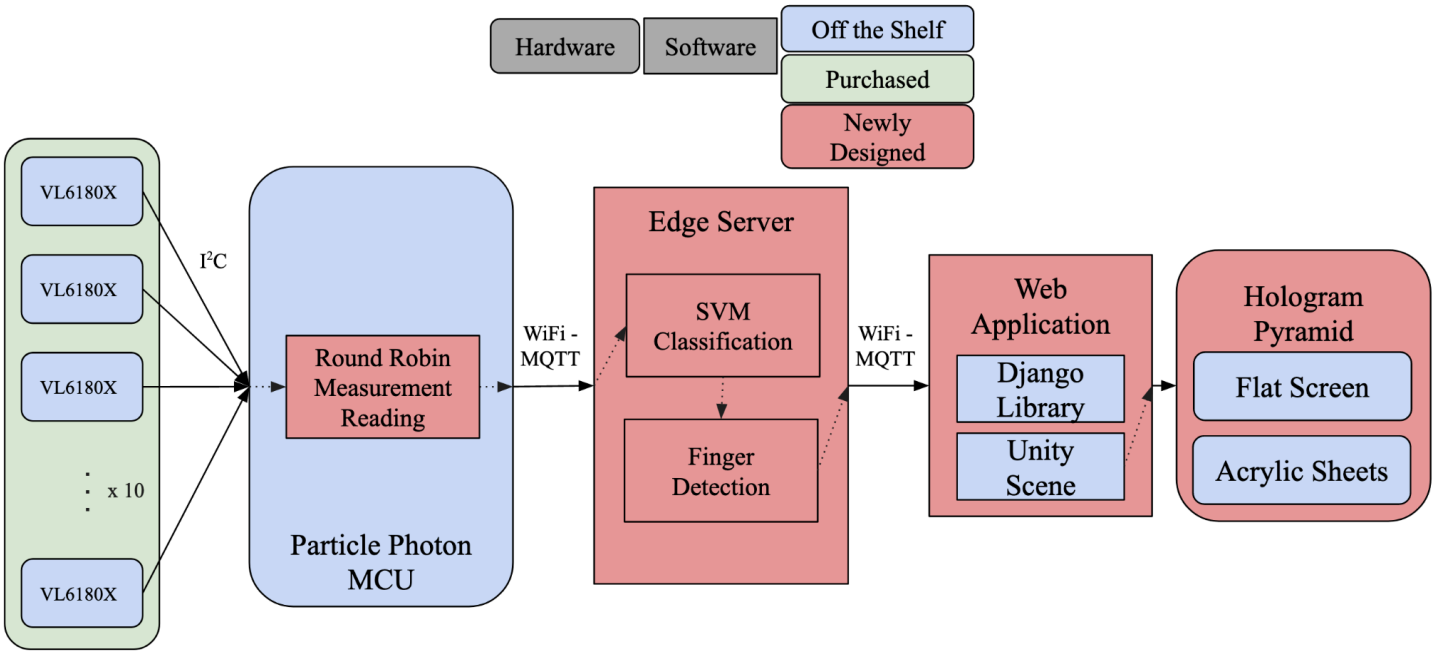


Fig. 1: High-Level Block Diagram.

We will now go through each component in more detail. For the PCB, we initially wanted to build it off of an Adafruit VL6180X breakout board. We wanted to make two boards, one for 10 VL6180X sensors and another one for all the "support components," such as voltage regulators, resistors, capacitors, etc. However, when we finished designing and received the boards, the support components board did not seem to work. To debug, we manually attached 10 resistors to the sensor board and it ended up working.

Once the Photon reads and sends all the data to our edge server, the data needs to be processed to determine the gesture and eventually the finger locations. To do this, we used a Support Vector Machine (SVM) to classify whether or not there is one or two fingers in front of the sensors. Once we classify, we fit a parabolic curve to the data with one finger and take the minimum point. For two fingers, we simply take the minimum point and send it to the webapp.

Once the finger locations are streamed to the webapp, the webapp sends the positions to a Unity process. Unity filters out noise and incorrect data points as well as filters and queues finger locations to be processed in its event loop. The event loop performs the actual rotations and scalings of the model. The magnitude of these actions is proportional to the elapsed time of when data was collected to ensure a realistic translation of gestures to actions.

IV. Design Requirements

A. *Engineering*

The most important requirement for us is portability. Therefore, we aimed to make the weight and dimensions of the W.R.I.S.T. wearable about the same as those of a typical Apple Watch. We wanted the wearable to be less than 100g, less than 65mm by 65mm, and cost less than $150. The heaviest Apple Watch available is about 41.7g, so we aim to be about double that weight at 100g so our device would feel light and comfortable on a user's wrist. We purposefully aim much higher (more than double the weight), since we are not building custom chips like Apple, and want a lot of slack for our weight requirement to account for the off-the-shelf batteries and boards we used.

With the evolution of visual technologies, we anticipate the next wave will be ubiquitous hologram technology [1]. However, due to cost and lack of availability, holographic projectors have a long way to go before being more common. Instead, we propose the use of a holographic pyramid that will be able to deliver the same effects of real-life 3D models. To design this pyramid, we followed the designs of Pepper's Ghost, a popular optical illusion technique to project objects that aren't in direct view [2]. Each side of the pyramid is an isosceles trapezoid that is connected to each other by the two slanted sides. The two main geometric considerations are (a) the smaller angle in the trapezoid is 45 degrees, and (b) the ratio of the top and bottom side of the trapezoid is 1:9. The pyramid will be placed on top of a monitor or flat screen that will lie on a table and will be provided by the user. 22" monitors are popular, which means the monitor is approximately 20" by 10". If we divide the shorter side by 3, we get the size of the smallest size of the trapezoid. Therefore, we estimate that the pyramid size should be 3 ⅓" and the longer size be 30". The material of the pyramid is acrylic, which is extremely cheap, so we could test out different sizes of the pyramid to achieve the optimal effect.

B. *Gesture Classification Accuracy*

The intended gesture of the user and the gesture that our classifier identifies should match about 90% of the time for

one finger swiping and 75% of the time for two finger swiping. This means we can tolerate 1 failure out of every 10 tries for a one finger swipe and 1 failure out of every 4 tries for a two finger swipe. The reason why we have different accuracies for one and two finger swiping is because we predict, based on our initial algorithms and testing plans, that there might be two finger actions that may be detected as one finger ones, especially as the hand gets farther away from the device's sensors.

C. *Gesture Translation Correctness*

Furthermore, we want the scaling of our transformations to be correct. Due to potential overlap of the sensor readings when a user's finger is farther away from the sensors, we predict that the accuracy of our sensor readings may drop as the finger moves away from the sensor array. We plan for an 85% difference between the true displacement of the finger paths when compared to the measured displacement predicted by our algorithm. We expect a somewhat low accuracy here since it will not take much away from user experience if the 3D model does not transform *exactly* proportional to their finger displacement.

D. *Latency*

We also want data to be translated to almost instantaneously, and we are aiming for a benchmark set by past projects that used similar technologies. We plan for a 100ms delay between when a user finishes his or her intended gesture to when the 3D model is transformed. This will give about a 10FPS (frames per second) update rate of our 3D scene. While 10FPS is somewhat low for rendering, it is not low enough to distract away from user experience [3].

V. System Implementation

A. *Wearable*

The wearable device consists of two main components: our custom-designed PCB for housing ten VL6180X distance sensors connected through a single I²C bus and a WiFi-capable microcontroller board.

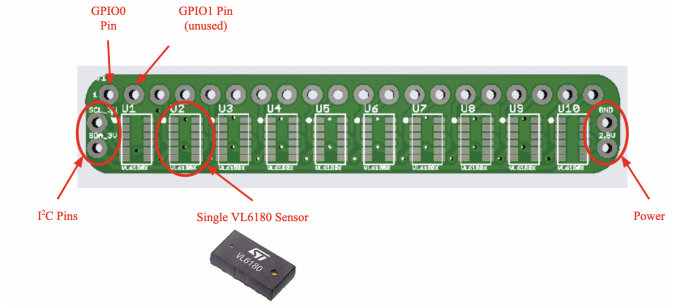


Fig. 2. Sensor Array PCB Diagram.

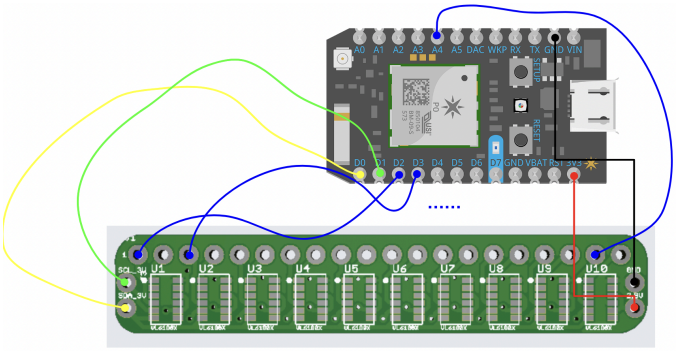


Fig. 3. Photon Wiring Diagram.

Our PCB contains a sensor array of ten VL6180X sensors arranged in a straight line. You can refer to Figures 2 and 3 for a visualization of our custom PCB design and its wiring. The VL6180X uses I²C for communication and can measure up to about 200mm (20cm). Additionally, they contain two GPIO pins: one for turning on and off the device and another for generating an interrupt when sensor data is available. A custom PCB allows us to place the ten sensors in a way so that the sensor array is compact, since we need the sensors to be as close as possible for our algorithm to correctly classify swipes with as much granularity as possible. We chose ten sensors because ten was the largest number of sensors we could fit into an array before the PCB became longer than the length of one of our wrists.

We choose a Particle Photon microcontroller for sensor data collection and transfer. The Photon has 72MHz clock speed, a Real Time Clock, and a Cypress Wi-Fi chip. To collect sensor data, the Photon turns on a single VL6180X sensor and reads data from it and repeats this in a round-robin fashion, storing sensor data into a 10 element byte array. Since the VL6180X reads up to 200mm, its measurements can easily fit into 8 bits. We choose to read measurements in a round-robin fashion due to IR interference across sensors. The VL6180X sensors read the distance to the closest object within a 25° cone in front of it, meaning the VL6180X's lasers and IR sensors can easily clash with each other.

The PCB, Photon, and battery are all soldered together on a board attached to a wristband. The sensor array will stick out perpendicular to the forearm so that the sensors' lasers are parallel to the arm, while the Photon and battery will lay flat on the wrist. When a finger is swiped across the forearm, some of the sensors will detect an object and send the distance to the Photon. The Photon will read all the sensor data, pack it into an array, and send through MQTT for processing.

B. *Middleware and Gesture Recognition*

We have two edge servers on our system: a Mosquitto-based MQTT broker [5] and a custom Python script running an MQTT client that receives and processes sensor data for finger detection. The Mosquitto MQTT broker will be used to shuffle around data through MQTT topics. Our topic structure is detailed in Figure 4. Both the broker and the data processing Python script are run on a M1 MacBook. Since the Python

MQTT client and the broker are run on the same device, network traffic between them should be faster. All data is sent through a local WiFi network to ensure that communication is as fast as possible.

We are using a coordinate plane paradigm where the x-axis represents the direction along the user's arm and the y-axis represents the direction along the sensors. We can place coordinates where fingers are and compare the relationship between the points every 100ms in order to (a) accurately predict which gesture is being performed and (b) quantify the translation. These gestures are displayed in Figures 5, 6, and 7.

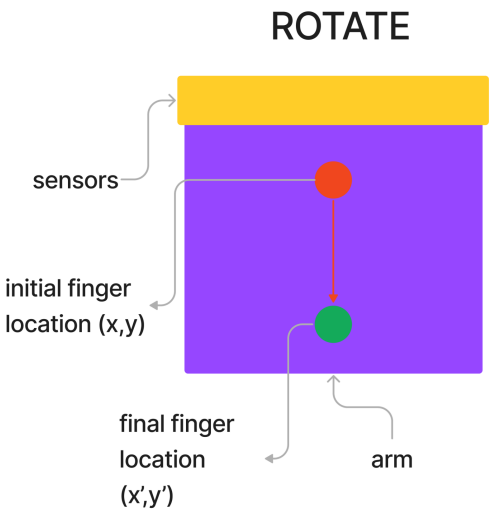


Fig. 5. Gesture for rotation. Rotation occurs in all directions based on x,y coordinates of the finger. Top-Left is (0,0) Bottom Right is (175, 55). The x-axis is perpendicular to sensors and the y-axis is parallel to sensors.

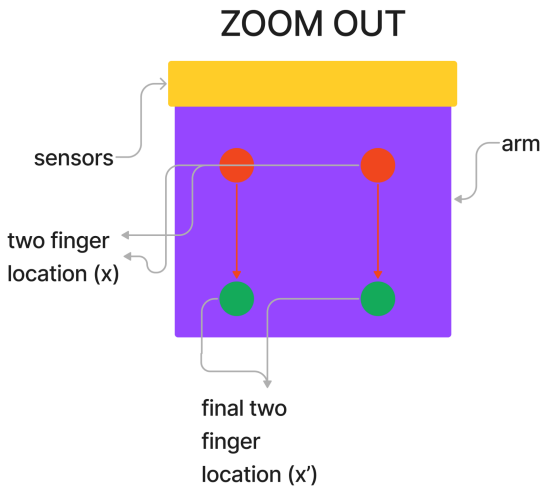


Fig. 6. Gesture diagram for zooming out. Represented as two finger swipes. Top-Left is (0,0) Bottom Right is (175, 55). The x-axis is perpendicular to the sensors and the y-axis is parallel to sensors.

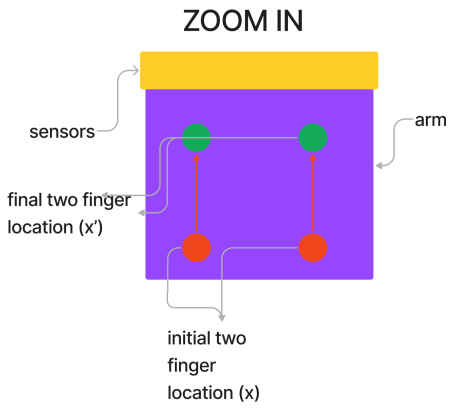


Fig. 7. Gesture diagram for zooming in. Represented as two finger swipes. Top-Left is (0,0) Bottom Right is (175, 55). The x-axis is perpendicular to the sensors and the y-axis is parallel to the sensors.

Our edge server runs an MQTT client to receive sensor data. Once data arrives, the Python script first says that all values over 150mm are invalid and makes them equal to 255. This means we actually have a max detection distance of 150mm, which is still a significant amount for a finger to move. We ignore values over 150mm because sensor readings over 150mm seemed to be unstable and a gesture at such a distance would be difficult to detect due to measurement instability.

Once data is preprocessed, we applied a SVM classifier to the 10 data points to detect if there is one or two fingers present in front of the sensors. We chose a SVM since training is somewhat fast and applying the model to classify data is extremely fast. We wanted to make sure our classifier did not take too much time to ensure low latency processing. The SVM model is custom-trained on data we collected ourselves. On test data, it has a 95% accuracy. See the Design Trade Studies section for more information on how we trained the SVM. Figure 8 for examples of how the data looks like when there is a one finger swipe gesture happening and when there is a two finger swipe gesture happening. The SVM is able to learn and distinguish the shapes of the two gestures when a radial kernel is applied.

Once a frame of data (10 sensor values after a full round-robin reading) is classified as either one or two fingers, we fit a parabolic curve to the sensor data. If the curve fitting algorithm fails, we ignore the data frame and return that it is noise. If curve fitting succeeds, we case off the number of fingers to find the x and y-value for the finger.

If the number of fingers is classified as one, we grab the x-value of the minimum point of the fitted parabolic curve and the index of the sensor that is closest to the minimum value. If the number of fingers is classified as two, we simply grab the x-value and index of the sensor with the minimum sensor reading.

Once we have (minimum sensor value, index of sensor with minimum value), we apply a weighted average of the y-values of the two adjacent sensors like so:

$$y = D_s \frac{\sum_{i=-1}^{1} w_i(y_i + c + i)}{\sum_{i=-1}^{1} w_i}$$

$$w_i = \frac{1}{|x_i - x_c| + 3} \; if \; 0 < i < 10, \; else \; \frac{1}{3}$$

Where **y** is the y-value of the finger, **c** is the index of the sensor that is closest to the minimum value of the fitted parabola, $\mathbf{x_c}$ is the sensor reading in mm of sensor at index c, $\mathbf{x_i}$ is the sensor reading of sensor at index i, $\mathbf{w_i}$ is the weight at sensor index i, and $\mathbf{D_s}$ is the distance the sensors are from each other in mm. The expression above finds the weighted average of the y-values of the sensor at index c, c - 1, and c + 1 to get the y-value of the finger. Note: for two finger gestures, the y-value is not used, but still calculated.

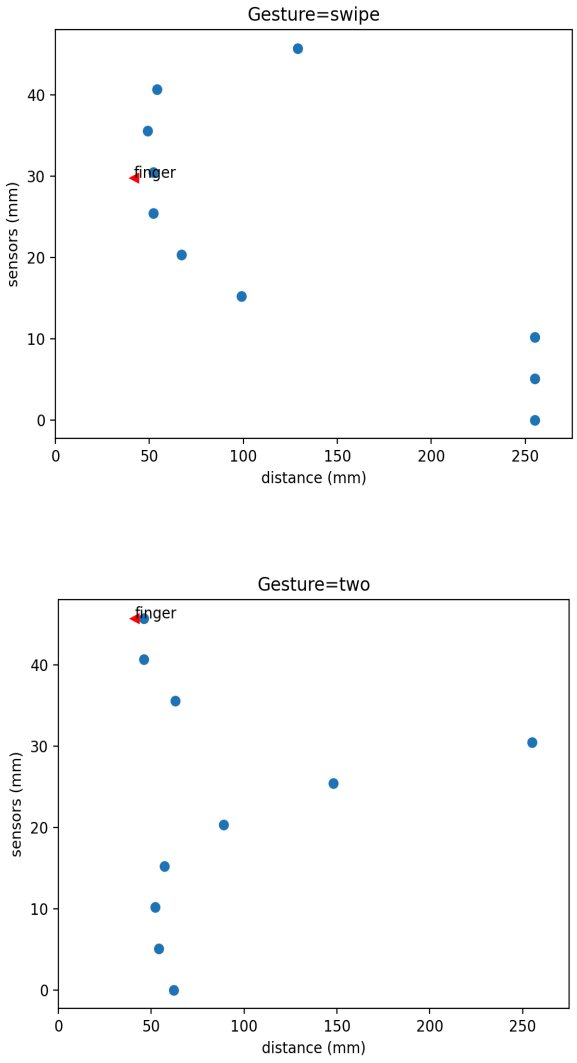


Fig. 8. Curve fitting on sensor data. The top plot is for one finger swipes and the bottom plot is for two finger swipes. The blue dots are sensor readings gathered from the wearable.

Once we have the predicted finger x and y-values, we package the finger location, classified number of fingers, and sensor reading timestamp to send to the Web application for further processing.

C. *Web Application*

The 3D hologram will be presented via a Web application and a hologram pyramid. We use Django for the back-end framework of our Web application and are using the MQTT protocol for data transfer between the edge server and the Web application. We are going to use Unity for manipulation of the 3D object and formatting. Unity has a WebGL option that will allow us to build our content as JavaScript and WebAssembly programs that will run our Unity application on a Web browser.

The sensor, battery, and gesture data will then be parsed by a JavaScript program on the Web application. It is possible to call Unity functions via JavaScript function calls using a Unity instance. The JavaScript on the Web application parses MQTT data to update battery status, update sensor information on a data visualizer, and structure gesture/finger information to be sent to the Unity build.

We created Unity functions that will handle the incoming stream of information. Unity will receive the x and y coordinates of the finger location, timestamp, and number of fingers for each data frame sent. The Unity functions queue finger displacements and timestamps for each gesture. The finger displacements are filtered for noise using an averaging filter with a window of three data points. Then depending on which gesture appears the most for a single user gesture input, a flag will be set to start processing. A single user gesture input is defined as the group of data points that are either rotation or zoom between two none data points. Once a rotation or zoom flag is set for processing, the Unity update function will start performing model translations based on the queued timestamp data and finger displacements. See Figure 9 at the end of the report for a detailed block diagram of the entire Unity pipeline.

Our Web application has two main tabs, one that shows the battery information, connection status, and sensor data visualization and one that shows the hologram pyramid display. See Figures 10 and 11 for an image of the tabs of the Web application.

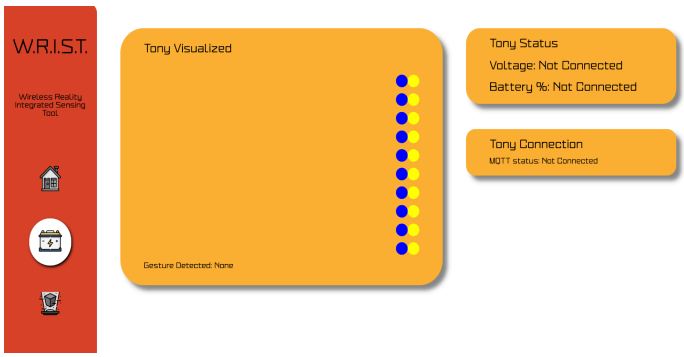


Fig. 10. Battery status page of web application. Shows a visualization of sensor readings, battery status, and the MQTT connection.

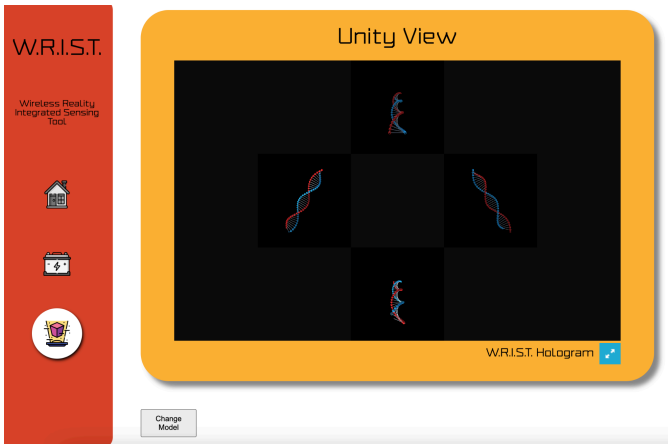


Fig. 11. Unity view to project onto a hologram pyramid.

D. Hologram Pyramid

To go more in depth about the hologram pyramid, we will be using four acrylic trapezoids zip tied together to create our pyramid. We based our trapezoid dimensions based off the angles from Pepper's Ghost Experiment, which is a popular design mechanism for optical illusions. Unity will display four perspective views of the 3D model (Figure 12*)*. The white lines are not actually displayed, and are drawn in Figure 12 to show where the pyramid base will be in respect to this image. The shorter base edge of the trapezoids will go along the white square, and the lines coming out of the corners of the square represent the edge between two trapezoids. We used a monitor to produce a larger view of the model.

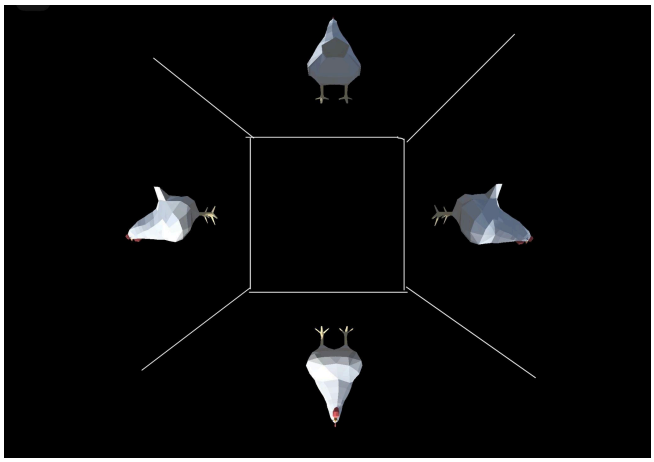


Fig. 12. Unity View of 3D model

VI. Test, Verification, and Validation

To test our implementation, we ran two experiments to record the accuracy of gesture translation and the accuracy of finger location detection.

A. Experiment 1: Classification Accuracy

The first experiment involved a user performing six gestures: swiping with one finger up/down/left/right, and swiping with two fingers left and right. The hologram pyramid was in front of the user when the experiment was performed. The user performed each gesture 50 times and recorded what action was applied to the 3D model. The results are shown in Figure 13.

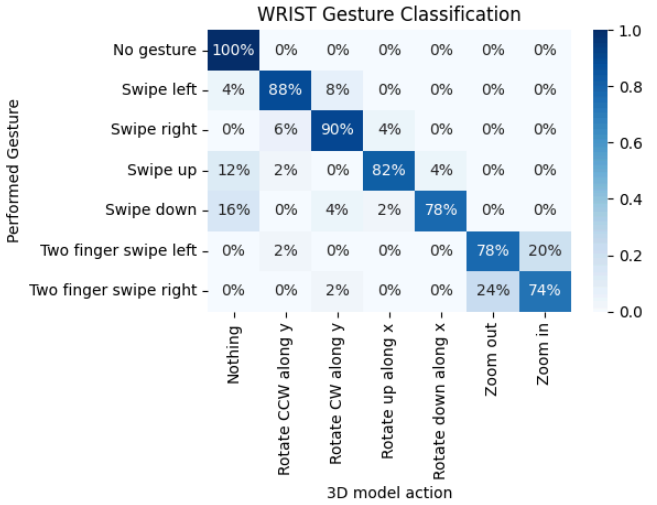


Fig. 13. Gesture-to-Action Confusion Matrix.

The confusion matrix shows that our model is very good at detection when tAhe user does not perform any gesture. Also, our model is pretty good at detecting horizontal one finger swipes. The model's 4-8% confusion of left and right one finger swipes possibly comes from the user not fully lifting up their finger when performing the gesture. Our model seems slightly worse at detecting up and down one finger swipes, with 12-16% of vertical one finger being detected as no gesture. This is because the area to swipe vertically along the sensor is smaller than the area to swipe horizontally (15cm along the wrist vs 25cm length forearm), so a somewhat fast swipe across the sensors might not pick up enough data frames to qualify as a full gesture. Two finger swipes have the worst accuracy, due to the fact that the two fingers need to be a significant distance apart for our model to classify it as two fingers. If the two fingers are too close, then the sensor data frame looks almost indistinguishable from that of one finger. Also, if the hand is slanted at a large enough angle, the sensor array will not be able to see one of the fingers or the fingers may blend together as one when picked up by the sensors. Two finger swipes significantly suffer from the user not fully lifting up their finger when performing the next gesture, as left and right swipes get confused for each other ~20% of the time.

These results were expected, as one can reason about the shortcomings of a 1D sensor array (can only see fingers in 1D, not lifting up a finger when performing another gesture can mess up the gesture classification, etc.).

B. Experiment 2: Finger Displacement Accuracy

To measure if we can properly measure how much a finger moves, we ran another experiment. In this experiment, a user placed a 150mm by 50mm grid on their forearm, marked with intervals of 25mm. The user performed 50mm horizontal and vertical one and two finger swipes 10 times and we recorded how much our model thought the finger moved. We ran this experiment to make sure our model translations were

proportional to finger displacements. A high error rate on this experiment would mean we are unable to distinguish between a long swipe vs a short swipe, which means our actions will not scale proportionally to finger movements. The error rate was calculated using:

$$\frac{|D_M - D_A|}{D_A} \times 100$$

Where D_M is the measured displacement of the finger(s) by our model and D_A is the actual displacement of the fingers, which was always 50mm in this test.

Gesture	**Distance Error (%)**
Swiping across sensors	20.29
Swiping along sensors	19.40
Two finger swiping across sensors	13.42

Table 1. Finger tracking distance error table.

According to Table 1, we have a maximum of 20% distance error, which means our measurements are at most 20% off of the actual finger displacements. We aimed for 15%, but 20% error is tolerable. Errors in sensor accuracy, approximations, and sensor update speed all factor into this ~20% error.

C. Experiment 3: Latency

The latency of our system was measured using timestamps collected at every step of the pipeline. Every 100 sensor data frames when the wearable was in use (user was swiping), we found the average of the differences of the timestamps between each step of the pipeline and recorded it in Table 2 below. The data was collected with all devices connected with the same WiFi network.

Pipeline Step	**Latency (ms)**
Time to collect all sensor datas	40
Wearable → Edge Server	29
Edge Server → WebApp	34
Total	**103**

Table 2. Latency table.

According to Table 2, we have a total latency of 103ms, which gives us an approximate 10 Hz update rate. This was just shy of meeting our requirements. It appears that collecting all 10 sensor values in a round robin fashion takes the most time. This is because when there is nothing in front of a VL6180X sensor, it takes almost twice as long to collect data. So for gestures where not all sensors need to fire to detect a finger, it takes much longer to collect sensor data than it really should.

D. Engineering

As mentioned before, portability is an important aspect of our project. We are able to attach Tony to an exercise band. The band does not obstruct data input. Although it is slightly bigger than a smart watch, it does not exceed the width of the exercise band, which is something that people are used to wearing comfortably.

As part of our demonstration, we wanted to show that we could use the device anywhere in a room. We visited the lecture hall in Wean 7500 and were able to demonstrate that the device could send data to the computer from the farthest corner in the room.

E. User Experience

Overall, we received positive feedback from users on our project. For the wearable device, when asked to rate how comfortable they are wearing the device from 1-5, the average rating was 3.75. Furthermore, each user was asked to perform each gesture 5 times. The accuracies are listed below. Due to time constraints from the users, the correctness and latency were not recorded. As shown, the rotation gestures had a higher average accuracy than the zooming gestures.

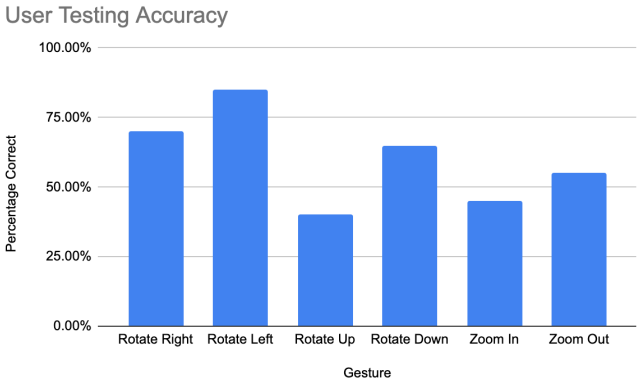


Fig. 14. Average user accuracies from each gesture.

F. Live Demonstration

During our demonstration, we qualitatively observed new user's enjoyment levels when using the wearable and viewing the 3D hologram. Many users seemed to be able to use our project with some explanation. We had to tell them that the gestures had to be extremely deliberate, but once we demonstrated, all users were able to understand how to use it.

One finger swipes were the most successfully detectable and enjoyable for the users. Almost all the time and for all users, one finger swipes were able to rotate the 3D model.

Two finger swipes were not always successful. This gesture was much less intuitive for users to understand. Many users had to swipe multiple times in order to successfully scale the model, but after a couple tries and false detections, the model would perform a zoom in or out. The success rate of two

finger swipes was highly variable. However, when it did work, the user was almost always impressed.

Many users experienced almost no noticeable delay and we received many compliments on how fast our system was. The choice of an SVM, finger detection, and local WiFi connections seemed to successfully give us low latency.

Users also positively took to the hologram tool. As quoted, they were "surprised it works in the light", as the area we were testing in was generally lighted. This was a concern of ours as the tool works best when the only light reflections are coming from the monitor. When seeing the entire system come together, a majority of users described our project as "cool."

Overall, we received many glowing reviews and praise on our system. Some users even spread the word about our project to their colleagues and many people came to our booth saying that their friend or colleague said to check us out. Lastly, we ended up winning second place in Apple's competition!

VII. Design Trade Studies

A. Sensors

We considered many different types of sensors for our project. For our purposes, we chose the VL6180X, an IR-based ToF distance sensor made by STMicroElectronics, amongst all the other sensors because of multiple factors. Some of the other sensors we were considering were pressure sensors, gyroscopes, and camera-based hand tracking sensors. We deemed that these other sensors all affected user experience in the scope of our project. Gyroscopes require extreme hand gestures which would cause discomfort for the user. Pressure sensors would take away the portability aspect of our design and would make our device's form factor be quite large as it would have to be more of sleve than a wristband. Since pressure sensors rely on a defined medium to be attached to, it mimics a traditional trackpad. Finally camera-based hand tracking sensors need to have the subject in front of a camera or similar device in order to detect hand motion. This would prevent the user from moving outside a fixed range of space, which is not what we wanted. We ultimately decided on distance sensors, since they are lightweight and can be leveraged to detect finger movements.

Amongst distance sensors, we chose the VL6180X [4] because of its short range of 20cm. We did not need a sensor with a longer range than 40cm distance, and we thought that having a shorter range would provide more accurate readings at short distances and give us a smaller data payload. We decided that 20cm would be a good size since it gives enough space for a user to do a gesture, but also not too large since the human forearm is only about 25cm on average. We included a chart (Figure 15) that lists out the different distance sensors we looked at and their feature comparison.

Fig. 15. Comparison chart of different distance sensors taken from ST [4].

B. Communication

We decided to not use a wired approach like USB cables for data communication, since it would defeat the purpose of our project and its goals of mobility. Initially, we were deciding on using Bluetooth for communication, but the BLE device we had was a bit hard to get working to prototype, so we shifted to WiFi. Since we want to integrate with Web technologies, WiFi was the better option due to its ubiquity and long range. We decided on the Particle Photon for our microcontroller, since it is easy to program and contains a built-in WiFi chip. For the WiFi communication protocol, we were deciding between the Particle Cloud's pub-sub system, HTTP requests, or MQTT. For our case, MQTT seemed like the best fit. The Particle Cloud is mostly a black box, and we wanted to have some more control over deployment. Additionally, HTTP requests are not as fast as MQTT and are typically used for more document-based data rather than raw sensor data. MQTT, which is popular for IoT applications, seemed to allow high throughput communication and excels at sending small quantities of data at a high rate. MQTT also allows for high scalability, as any device can simply subscribe to an MQTT topic and receive either a stream of our sensor array measurements or our detected gestures and finger coordinates.

C. Web Application Backend

We thought about multiple web application frameworks, such as Symfony, Express, Ruby on Rails, and Django. However, amongst these, we decided to use Django. We mainly chose Django because our members were most familiar with it and because Django is used using Python. It also provided all of the basic functionalities we needed. We only needed an interface where we could store user data and serve our Unity application, which we could do with Django.

D. Hologram

For the hologram pyramid, we noticed online that there were many designs that had trapezoid panels with angles of 53-53-127-127. We initially started with that design to fit on an iPad, as shown below.

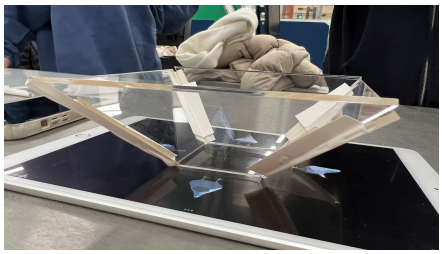


Fig. 16. First prototyping of the hologram pyramid.

However we realized that if we wanted the projection be more visible, we needed to a) increase the monitor size so that the images themselves are more spread apart, b) increase the size of the pyramid to have more slant to show the image, and c) increase the smaller angle of the pyramid to increase the height of the pyramid. We ultimately ended with the design below to fit the size of a normal computer monitor, something that most presenters have access to.

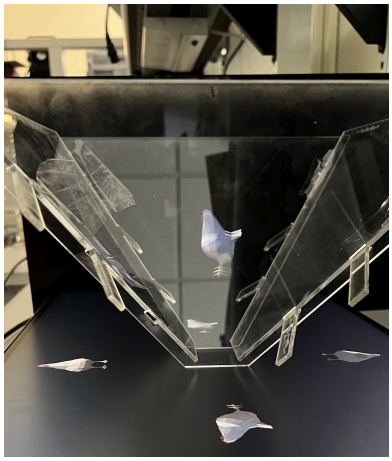


Fig. 17. Second to last prototyping of hologram pyramid.

E. Finger Detection Model

We went through several iterations of the data pipeline after the sensors collected data. We had two schools of thought on what the model can do after a user performs a gesture in front of the sensors:

a) The model can directly determine the gesture that the user just performed, or
b) The model can determine how many fingers are present.

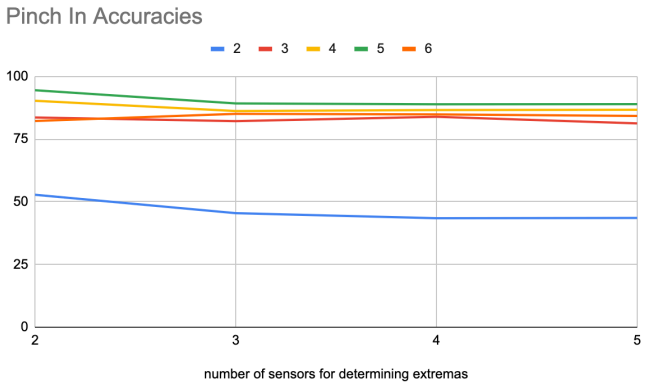
Even though we didn't end up going through the models described below, we did take features from each model and incorporated them into the final model. In the following sections, we will detail each approach we tried and the trade-offs for each of them.

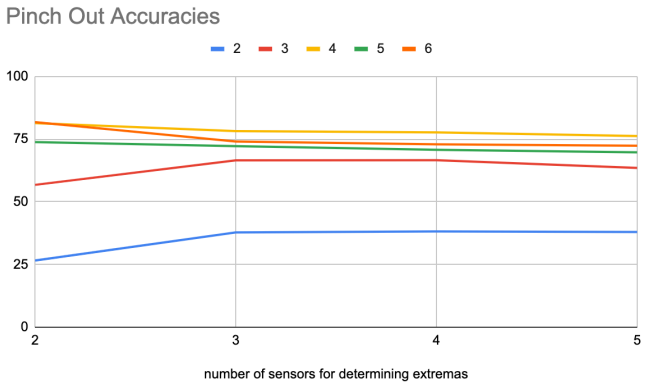
1. Curve Fitting

Before we looked into using machine learning, we wanted to find a way to determine finger location based on mathematically properties of the data. When there isn't an object in front of the sensors or if an object is more than 255 millimeters away, the sensors default to a reading of 255. Otherwise, the sensors detect the distance of the closest object to the nearest millimeters. Therefore, a differentiable equation can be fit to the data. After that occurs, we determine the minimums and maximums of the equation. When there is one finger present, a U-shaped line is created, and the minimum of the equation estimates where that finger is present. When there are two fingers presented, a W-shaped line is created, and the two local minimums represent where each finger is present, and the intersection of the two U-shaped lines created by the fingers intersect at the maximum of the equation. The curve fitting [6], minimums [7], and maxes [8] were computed using libraries from SciPy.

Accuracy

There is an argument *n* to the function where we can dictate how many points from each side of a point to determine the extrema. There is an argument *polynomial* to the function that allows us to dictate the order of the polynomial that the data is being fitted to. To determine which n and which polynomial would yield the highest accuracy, we collected data from the wearable placed on a table and performed three gestures six times: pinch in, pinch out, and swiping perpendicular to the sensors. After data collection, we ran the curve fitting model on each frame from each instance of the gesture, which guessed what gesture was occurring based on that frame. Our results are shown below.





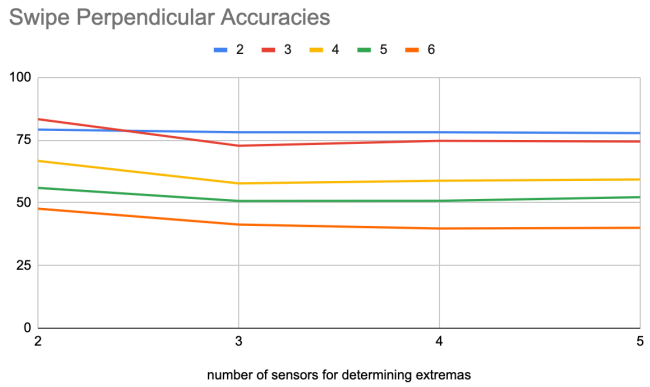


Fig. 18. Accuracies for each gesture. Top is for pinch in, middle for pinch out, and bottom for swipe perpendicular.

The highest accuracy for each gesture is detailed in the table below.

Gesture	**n**	**Polynomial degree**	**Accuracy (%)**
pinch in	2	5	94.4
pinch out	2	6	81.7
perpendicular swipe	2	3	83.3

Table 3. Highest accuracy of each gesture for polynomials of degree 3, 5, and 6.

Even though the highest n was the samest for each gesture, there was no consistency in the degree of the polynomial. The highest average of all gestures was 79.4% with n=2 and polynomial=4, so there wasn't a specific polynomial that we could use. Therefore, we decided to not use the curve fitting for determining the gesture, but we did use curve fitting to determine the finger position when there is only one finger present.

Latency

The curve fitting model is almost instantaneous, as it doesn't need to be processed by a complex machine learning model. This was one of the biggest appeals of using a model that didn't require machine learning.

Correctness

At this stage, we weren't concerned with correctness, just whether we could identify which gesture was occurring. However, we did look at the general correctness of the model. Identifying where the finger is located when a user was performing a swipe was the easiest, since only one point needed to be determined out of all the points. However, whenever a pinch was supposed to be detected, the model would sometimes think that there was too much noise to be classified as either gesture, the beginning of the pinch out was a swipe, or the end of the pinch in was a swipe. Therefore, in the duration of a pinch, it would be harder to determine where the fingers were present since the accuracy of detecting the gesture itself was low. Therefore, we decided not to use curve fitting to determine finger location of the zoom gestures.

2. Support Vector Machines with Pinch and Swipe Data

Since the sensors were extremely close to each other, there would be some overlap from the sensors due to the cone shape of the lasers. From the naked eye, we could guess what gesture was occuring, but something simple like curve fitting would not be able to determine the differences between swipes and pinches. Therefore, we decided to look into different machine learning options.

A support vector machine (SVM) is a type of machine learning that uses supervised learning mainly used for classification of data. We decided to use the support vector machine [9] and the grid search library [10] from scikit-learn , a Python machine learning tool. The model will attempt to classify which gesture is occurring based on each data frame.

Accuracy

We collected new data from Tony after it was attached to the exercise band and performed the gesture on our arms. We randomly allocate ⅔ of the data for training and ⅓ of the data for testing. The model tunes three different parameters: the type of *kernel* that the SVM uses, the regularization parameter *C*, and the kernel coefficient *gamma* if the SVM uses a radial kernel. The training accuracy of the gamma kernel is shown below.

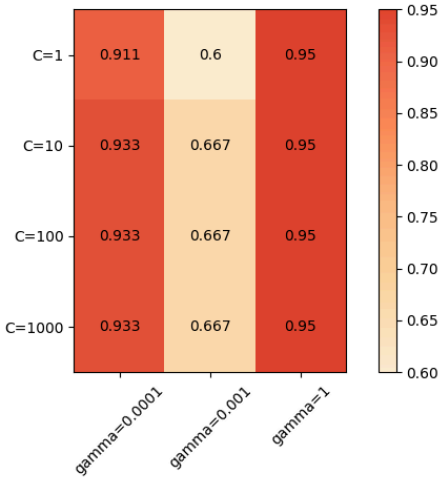


Fig. 19. Colormap of the SVM parameters.

After determining which parameters yield the highest training accuracy, the model uses those parameters for testing. The testing accuracy is shown below.

	precision	**recall**	**f1-score**	**support**
-1	1	1	1	53
0	0.85	0.81	0.83	48
1	0.82	0.86	0.84	49

accuracy			0.89	150
macro avg	0.89	0.89	0.89	150
weighted avg	0.89	0.89	0.89	150

Table 4. Accuracy of SVM on collected pinch and swipe data.

The green cell represents the average accuracy of the model. -1 represents noise, 0 represents swipes, and 1 represents pinches. Overall, the accuracy of the model was better than curve fitting, although even after we determined the gesture, we had no way to detect where the fingers were located. At this point, we decided to use curve fitting to determine finger location after we determine what gesture the user is attempting to perform. Even though the accuracy on collected data was decent, using this model in real-time yielded a lower accuracy.

Latency

The training for the model sometimes took hours, but that would not be a part of the latency of the system. To determine what gesture is occurring, the model took around 30 milliseconds to run. Even though this was a small time, this was around ⅓ of the total latency we were aiming for the system.

Correctness

Similar to the problem with curve fitting, since the real-time accuracy was low, it was hard to detect the finger location when most of the pinches were being classified as noise from our SVM, so it was hard to determine the correctness of the gesture.

3. Time Series Classification

We wanted to look into more accurate machine learning models to determine what gesture was occurring on our collected data. Time series classification is a type of machine learning that uses supervised learning to classify time series data. This way, we would be able to classify multiple data frames instead of one data frame. We predicted this would help with detecting pinches as pinches are easier to classify once the data is observed over time rather than during the duration of the gesture. There were five different classifications of data we could now get: pinch in, pinch out, swipe left, swipe right, and noise. We used the time series classification library from the `sktime` [11] Python machine learning framework.

Accuracy

One parameter that we could control is how many data frames the model should look at. The drawback to this model is that it cannot look at sets of data with unequal amounts of data frames, so our algorithm divides each instance into equal-sized windows to classify, and we tested the accuracy of different size windows. The accuracy of the different windows is shown below.

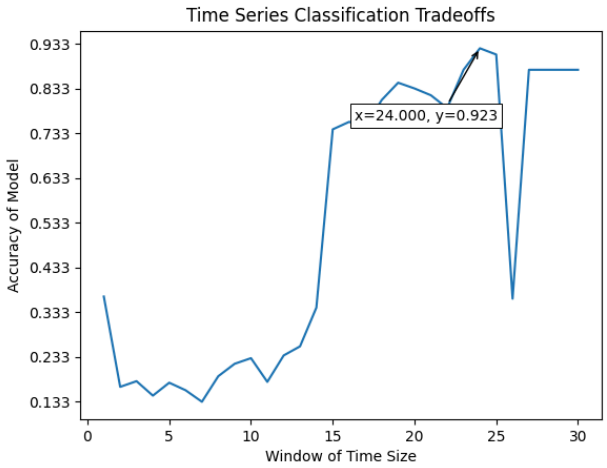


Fig. 20. Accuracies of time series classification on collected data compared to different time window sizes.

The highest average accuracy was 92.3% when the window size was 24 data points. This was the highest accuracy so far.

Latency

To achieve that accuracy, there would need to be 24 data frames before a gesture can be determined. This was mainly because pinches on average require more data frames to determine what gesture is occurring than swipes. When running on collected data, it took around 280 milliseconds to classify the data, and the latency would only increase in real time as the model would need to wait 24 data frames on top of the classification period.

Correctness

Because we can accurately determine what gesture is occurring but not where the fingers are located, we began theorizing other ways to map real life transformation to the model transformation. We hypothesized that averaging each data point in each data frame and comparing the averages overtime would tell us what direction the gestures would need to move. Although we didn't test this directly with time series, when running the data visualizations, we saw on average the data increased as a user swiped away from the sensors and the data decreased as a user swiped towards the sensors. After seeing the results of the time series classification, we came to the conclusion that any two finger gestures would either be confused as noise or would need more data to classify as a pinch, so we decided to switch the zooming gesture.

4. Support Vector Machines with One and Two Finger Data

We decided to revisit the support vector machine model. We found it was easy for the model to detect the end of a pinch out or the beginning of a pinch in since it was only during those times that two fingers were obviously present. Consequently, we changed the zooming gesture to be two finger swipes so that we would be able to better recognize the differences between the two types of gestures. Therefore, we changed the type of data we collected to be one and two finger

placements along the arm, with some data being actual swipes in different areas of the arm. However, we used the same machine learning library as before.

Accuracy

We tested in a similar fashion to when our data was swipes and pinches. The data is shown below.

	precision	**recall**	**f1-score**	**support**
0	0.93	0.91	0.92	138
1	0.95	0.97	0.96	264
accuracy			0.95	402
macro avg	0.94	0.94	0.94	402
weighted avg	0.95	0.95	0.95	402

Table 5. Accuracy of SVM on collected one and two finger data.

The green cell indicates that this model has an average of 95% accuracy. Instead of classifying one, two, and noise, we only classified whether the data saw one or two fingers, with 0 and 1 respectively. We decided to filter noise on the web application when it received the finger detection data, which gives us more power to determine what to do with the model output. This was definitely a bigger increase from our previous SVM model, and as shown in the testing section, performed reatime at around the metrics we initially aimed for.

Latency

The latency of the system was around 30 milliseconds.

Correctness

Since determining the gesture was a lot more accurate than before, it was easier to test for correctness. We decided to take features from our previous models to compute the mathematical translations.

If there was only one finger present, we know the user is performing a rotation, so we used curve fitting and found the minimum to determine the finger position. After that, we found the deltas of the fingers and determined to move the model that much. Because of the higher accuracy with one finger, we had more liberties with rotations, which is why we introduced rotations in two more directions. If there were two fingers present, we took the idea of taking the average of the data points and applied that to the zooms.

Overall, the translations were more proportional to the actions.

Model	Accuracy (%)	Latency (ms)	Correctness
Curve Fitting	79	0	
SVM of swipes vs pinches	89	~30	
Time Series Classification	92	~280 (ms)	
SVM of one vs two fingers	94	~30	

Table 6. Summary of Finger Detection Model.

VIII. Project Management

A. Schedule

Our tentative schedule is attached on page 12 (Figure 21).

B. Team Member Responsibilities

Even though there are five blocks, there are six main parts of the capstone. Edward worked on PCB creation, programming the wearable, gesture and finger detection, and communication between components. Joanne worked on the Unity code, Web application, and hologram. Anushka worked on gesture and finger detection and prototyping the wearable and hologram. Although each member on our team is specialized in at least one part, we all worked together on each block since there are a lot of dependencies from one block to another. A more extensive breakdown of responsibilities is documented in our schedule (Figure 21).

C. Bill of Materials and Budget

Our Bill of Materials is shown in Table 7.

D. Risk Mitigation Plans

We are planning to create a custom PCB to accommodate our wristband size. However due to the term of our project, the shipping of the PCB after design is one huge risk factor we had to consider. It would be hard to have multiple iterations of our PCB because of time constraints. We planned to mitigate this risk by first modeling our PCB off an already existing PCB. We also got a professor to check our PCB before we ship it off.

One issue we ran into the day before demonstrations was that our battery broke. Although we had a spare battery with us, we realized that this could be an issue for users. Fortunately, since Tony can be charged while being wired to the computer, we were able to still use Tony while we didn't have access to the battery before the demonstration. We have learned that in the future, we have to be careful with wiring but also should order spare parts if our budget allows us to.

E. Ethical Issues

Our project does not pose any significant risks to the user, but there are very extreme cases that could occur.

1. Physical harm

Tony was attached to an exercise band and wrapped around another layer of fabric so that there are no electronic parts exposed to the skin. However, all the wiring sautered to the board, and the battery is also in close proximity to the body. If there were to be any electrical problems, the users'

hand would be at the greatest risk. The person wearing Tony would be affected adversely, but one mitigation technique would be to build a better encasing system for Tony so that there would be more protection. We haven't tested our product's long term effects on users, so it would be interesting to see if the product has any effect on arm or finger movement.

2. Data communication channels

There is data being transferred from the sensors to the edge server to the web application, which means there are several areas where a hacker could send their own data and somehow overpower the system. This would be more critical if the sensor data were affecting more parts of the screen rather than an isolated web application. A hacker could potentially take control of the users' computer. The person who would be at the greatest risk is the person running the edge server and running the web application. This could be mitigated by using secure servers from trusted companies.

3. Web security

The web application does not have security, and this might be a security risk in two main ways. One, the user has to sign up to use the web application. It is important to make sure whoever does sign up knows their data is secure and is only being used for authentication purposes. The person at the greatest risk is the user accessing the web application. This could also be solved from the previous solution - by running the servers on more secure platforms.

IX. Related Work

There are many portable trackpads out there, such as the Apple Magic Trackpad, Logitech Wireless Touchpad, which could provide a similar gesture recognition function as our wearable watch. However none of the existing trackpads out in the market right now has a design where there is no actual physical medium for a user to perform gestures on. Our wristband makes use of distance sensors to detect gestures made on our arm, thus adding a level of mobility and portability that differs our product from the existing trackpads.

We are designing a hologram pyramid to visualize 3D objects and manipulate them using gestures taken by our wristband. When compared to the other existing 3D model viewing technologies such as Autocad, Unity, three.js, many of them require the user to view from a display. They are limited to where they can view this 3d model. We provide this hologram pyramid as a way for users to view their model from any space and also be able to walk around it and manipulate its view using our wristband. Thus also adding another level of uniqueness from the existing trackpad technologies out there.

X. Summary

Originally, we had planned to only rotate and zoom in two directions using swipes and pinches respectively, which guided our original accuracy prediction of 82.5% and correctness error of 15%. However, since then, we have included an all degree rotation while altering the user experience to increase the accuracy and correctness. Furthermore, we achieved our latency requirement. Since our numbers are close, we consider our system a success.

We are hoping that this project will change the immersive technology field in the future and to improve how we interact with computer graphics and modeling. We are changing the field of digital environments that will improve how professionals give presentations. Presenters want to be able to interact with their audience while also effectively engaging with the content on their screens. Professors who prefer moving around the classroom, engineers who want to provide a more interactive experience to stakeholders, and doctors who want to inspect MRI imaging in a more practical sense will be able to use our product with ease and add value to their daily work.

A. Future Work

In the future, we hope to see this project adapt to more OS-level interface functionalities such as cursor movement and text. We expect our project to become more widely needed in the future as immersive technologies become more ubiquitous, just as Iron Man envisioned.

B. Lessons Learned

Our system was either able to successfully meet or get close to the design specifications. It was definitely able to pass the use-case requirements, as we did create a system that allows people to control a 3D hologram with their fingers and forearm.

A large limit to our project was the communication latency. We believe if we have chosen Bluetooth over WiFi we would be able to reach much lower than 100ms latency. We ultimately chose WiFi because it was the easiest to use for the hardware we all on-hand, but WiFi is highly susceptible to interference from other devices and campus WiFi is not always the fastest. If we had no choice but to use WiFi, we would probably have used UDP instead of MQTT, which uses TCP. UDP is much faster than TCP and it would be ok to drop some sensor data frames.

Another limitation of our system was the VL6180X's slow update rate when there is no object in front of it. Since the round robin algorithm needs to wait for each sensor to finish before reading from the next one, it has to wait for sensors whose measurements might not even be important in finger detection. For instance, when a user is swiping on the lower part of their forearm towards the sensors, only half the sensors will have valid readings and the other half will read nothing. However, the round robin algorithm needs to read *everything* before sending data to the edger server. So, it spends most of the time waiting for empty data. If we were to do it again, we would possibly pick a single sensor that can read a whole area instead of 10 individual sensors.

Since we have a 1D distance sensor array, it is highly susceptible to interference. If anything is in front of the sensors, our system will say there is a finger. So, if there are other fingers blocking a finger swiping on the forearm, there may be a false two finger detection, when only one finger is touching the skin. This is unfortunately impossible to mitigate

given our implementation, but maybe adding a heat signature sensor can at least distance between random objects blocking the finger like clothes. This means users' sleeves need to be removed out of the way when using our device, as the system may classify cloth as fingers or sleeves may block the sensors entirely. It was a bit annoying for users to roll up their sleeves during the demo.

If a student reading this report is planning to remake this project in the future or tackle this application, we would suggest changing the use case to an OS-interface rather than a 3D hologram. If you are to stick with a 3D interface, perhaps use an AR headset since it would make more sense to use your forearm as a touch interface when wearing a headset that is always on your body. Overall, this was a very interesting challenge to explore and a super fun project to make!

Acknowledgements

We would like to thank Professor Sullivan and Funmbi Jaiyeola, our TA, for all the support over the semester! We would also like to thank everyone who was excited about our project during the demo and those who spread the word of our project to their friends. We would also like to thank Apple for the prizes and all the monetary support for 18-500 over the years.

Glossary of Acronyms

PCB – Printed Circuit Board
MQTT – Message Queuing Telemetry Transport
ToF – Time of Flight
IR – Infrared

References

[1] Team ISTE. "8 Classroom Uses for Holographic Technology," ISTE, 2015, https://www.iste.org/explore/ISTE-blog/8-classroom-uses-for-holographic-technology.
[2] "Pepper's Ghost: Holgoram Illusion," Science World, https://www.scienceworld.ca/resource/peppers-ghost-hologram-illusion/.
[3] Deber, Jonathan, et al. "How Much Faster Is Fast Enough?: User Perception of Latency & Latency Improvements in Direct and Indirect Touch." Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, 2015, pp. 1827–1836.
[4] "Time of Flight Sensors," ST, https://www.st.com/en/imaging-and-photonics-solutions/time-of-flight-sensors.html.
[5] https://mosquitto.org/
[6] "Curve Fitting." https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html
[7] "Argrelmin." https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.argrelmin.html#scipy.signal.argrelmin
[8] "Argrelmax." https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.argrelmax.html#scipy.signal.argrelmax
[9] "Support Vector Classification." https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[10] "Grid Search CV." https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[11] "Time Series Classification." https://www.sktime.org/en/stable/api_reference/classification.html

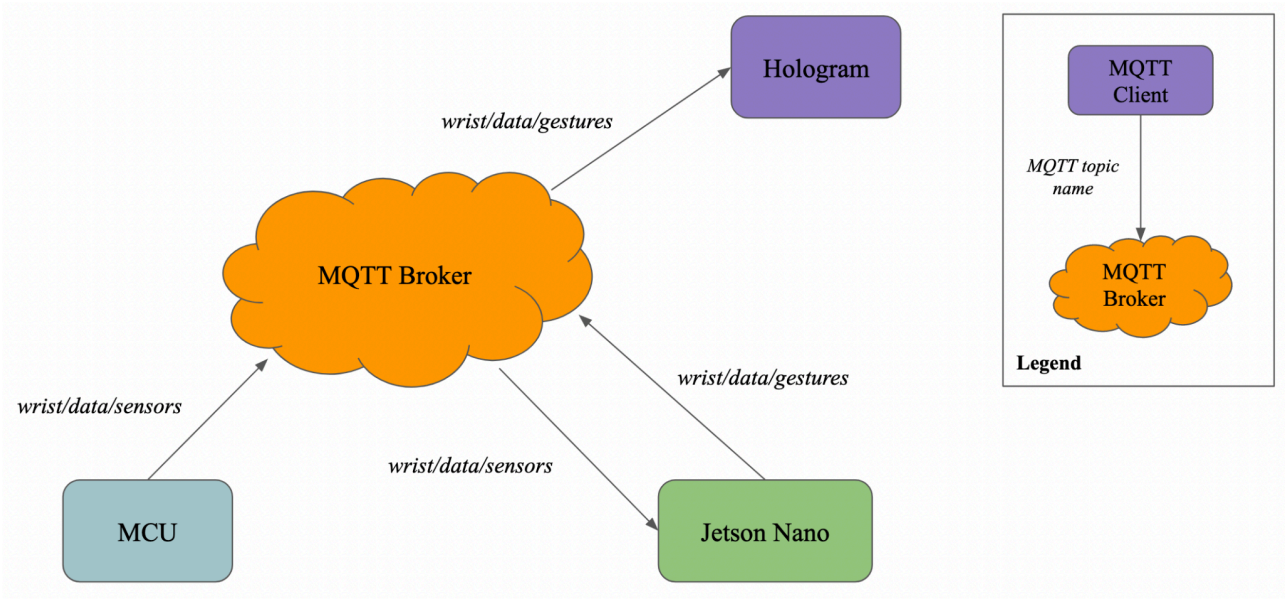


Fig. 4. MQTT Data Flow.

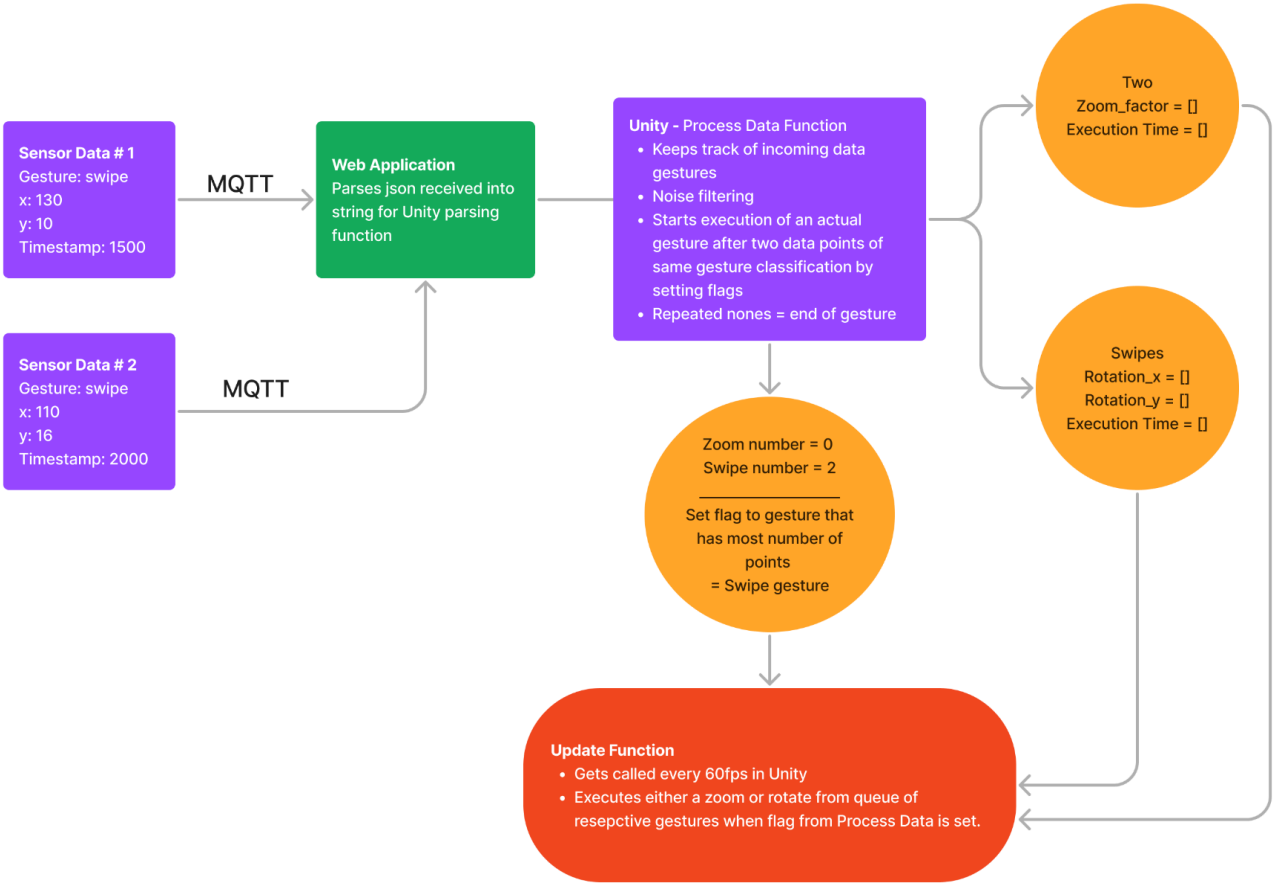


Fig. 9. Unity-Webapp Flow Chart.

Item Name	Part #	Manufacturer	Quantity	Price	Description	Total
Custom PCB	n/a	JLCPCB	1	$4.00	Custom-designed Printed Circuit Board	$4.00
Distance Sensors	VL6180X Time-of-Flight distance sensors	STMicroelectronics	10	$2.95	Proximity sensor and ambient light sensing (ALS) module	$29.50
Microcontroller Board	Particle Photon	Particle	1	$19.00	WiFi-enabled board with STM32 ARM Cortex M3 microcontroller (personal device)	$19.00
Wristband	n/a		1	$2.50	Exercise Wristband to attach electronics	$2.50
Resistors and Wires	10K Ohm	n/a	12	$0.00	10K Ohm Resistors and Wires (scrounged from lab spaces)	$0.00
Total						**$55.00**
Acrylic Sheets	n/a		6	$7.50	Clear plastic sheets to cut trapezoids out of for hologram pyramids	$45.00
Monitor			1	$0.00		$0.00
Total						$45.00

Table 7. Bill of Materials and Budget.

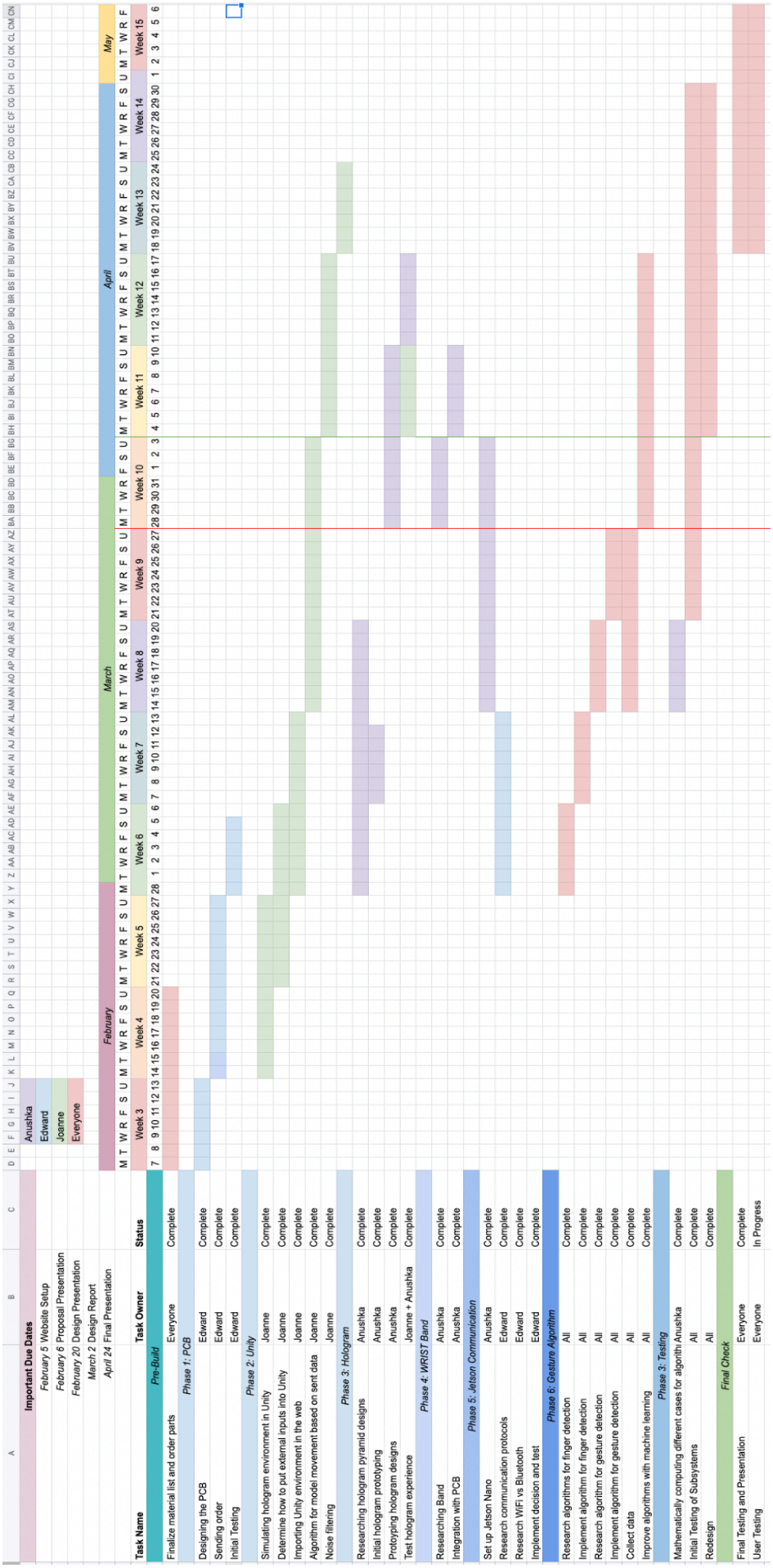


Fig. 21. Schedule and Task Breakdown.