VR Ping Pong

Henry Yi, William Wang, Logan Herman

Department of Electrical and Computer Engineering, Carnegie Mellon University

Abstract**—**Our goal is to create a virtual reality table tennis gaming system capable of resembling the real world game, using IMU sensors (which contain an accelerometer and gyroscope), basic kinematics calculations from a physics engine, subsystem to subsystem communications such as Wi-Fi and Bluetooth, to create a game experience that approximates real-time interaction with the opponent.

Index Terms**—** computer vision (CV), inertial measurement unit (IMU), virtual reality (VR)

I. Introduction

Our desired end product is a virtual reality table tennis setup, including the actual game software, "paddles" with sensors, VR headsets. The application/motivation is the prevalence of social distancing during the pandemic and the implications the isolation has on many peoples' lives and interactions. Both video games and sport are key avenues of socialization today and our game aims to users in either (or both) categories. We want to create a fun and interactive experience that approaches real-time interaction as closely as possible in order to boost engagement between players who may physically be far apart. There exist similar products, such as Eleven Table Tennis, an extremely realistic VR table tennis game. When it comes to the accuracy metrics discussed later, Eleven Table Tennis is undoubtedly superior. However, it is played on the Oculus Quest 2 which costs about $300. Our product is far cheaper and is able to accommodate those who cannot afford the Quest 2, assuming they have a smartphone that can run the game.

II. Use-Case Requirements

The most desired characteristic of our end product is a realistic feeling when playing. This is difficult to quantify but can be broken down into a handful of metrics that allow us to evaluate this feeling in a quantitative manner. The first and most important is latency, which consists of delay on two separate paths. We would like our latency between each player's paddle and headset to amount to less than 300 milliseconds. The other path is from a player to their opponent. We would like that to be less than 300 milliseconds. The reason for this latency is that for the opponent to actually see an action done by a player, the appropriate data must travel from the player's paddle to the server, from the server to the opponent's paddle, and finally from the opponent's paddle to the opponent's headset. The next major metric is the accuracy of the swing characteristics. We have decomposed the swing into three primary components: the type (lob, slice, smash), the power, and the direction. This differs from our previous representation of paddle state via position and orientation. Since each factor clearly has a major effect on the overall nature of the swing, we have set high thresholds as goals for accuracy: 90% for swing type and power, and 95% for swing direction. In other aspects, as before, we would like our game to have at least 30 frames per second, about the framerate of most video playback and a moderate resolution (360p). We believe that this is enough for the user to comfortably discern what is happening without risking the video quality (which is a lesser concern) costing us more latency time. Lastly we require our power supply in the paddle to last for 1 hour of continuous gameplay. This is because we do not want the paddle to have a wired connection to an external power supply. As for the size/weight of the paddle, most ping pong paddles fall in the range of 200-400 grams. We believe that a little bit of extra weight will not be detrimental as even real paddles deviate greatly in terms of weight. For now we will aim for 500 grams or less, and for the size to be roughly the same size as real ping pong paddles - handle about 4 in x 1 in x 1in, face about 6-7 inches diameter, ½ in thick. One key metric that was missing from the design report was the user's swing speed, and a benchmark on how fast of a swing our system would be able to handle. We have since decided on an upper limit of 5 meters per second. This was done by carefully reviewing videos of professional play. On offensive strokes (which have by far the fastest paddle speed), professional players consistently took between 0.2 and 0.25 seconds (or just outside of this range) to swing. These strokes have an estimated length of approximately 1 meter (or slightly more). This is how we arrived at our benchmark for swing speed. While of course it is possible that users of our product could achieve racket speeds higher than this, we feel that users who are playing seriously will be well in this range and that accuracy dropoffs outside of this range are reasonable.

III. Architecture and/or Principle of Operation

The system features a central server, virtual reality devices and paddles for each player. The paddle sensing systems track the paddle's motion using data from an IMU. The IMU provides the acceleration of the paddle as well as the orientation. Using this data, the paddle's Raspberry Pi Zero is able to calculate the paddle state which is sent to the server over Wi-Fi, where it will be forwarded to the opponent's headset/game client so the player's action and its subsequent effects can be rendered on the opponent's VR headset display. This process will repeat with the opponent's shot, and so on. Figure 1 shows the high level interaction of our subsystems. The intricacies of each connection between the subsystems and a breakdown of the operations required will be elaborated upon in future sections.

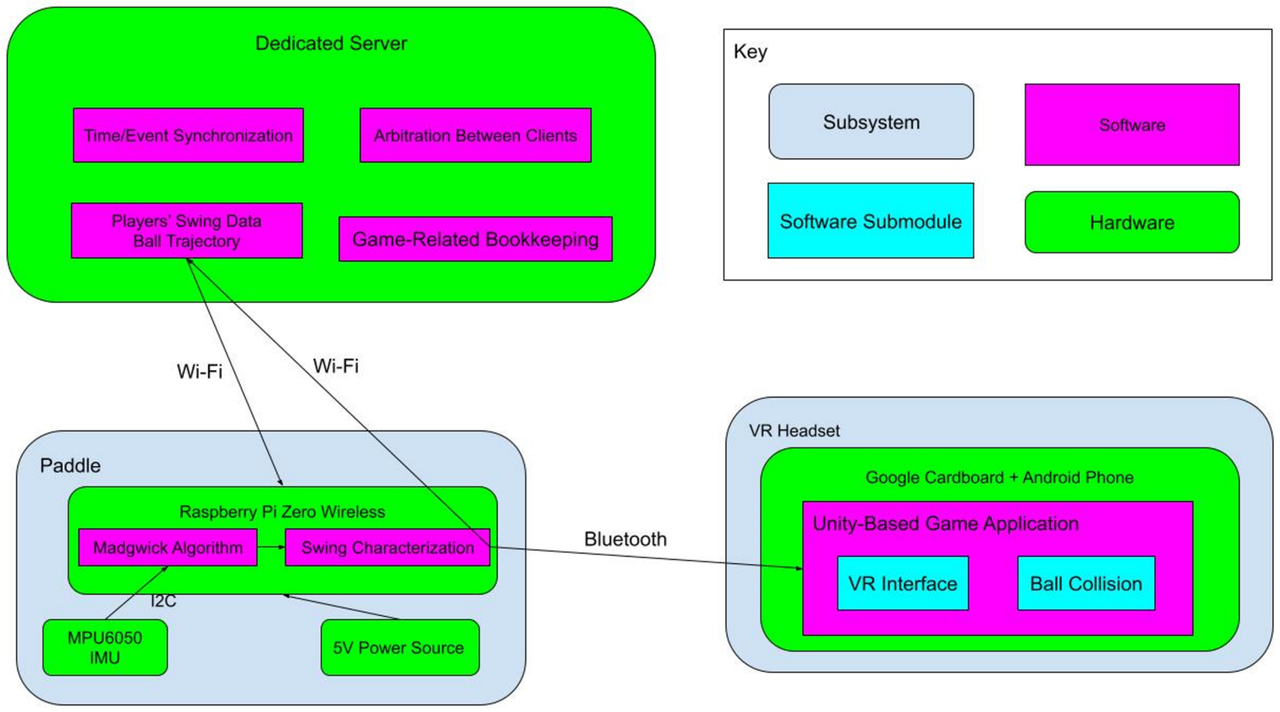


Fig. 1. High level block diagram of the overall system.

IV. Design Requirements

One of the most important metrics of our system is the latency. While it was difficult to nail down a quantitative value without actually knowing how it feels to play on specific latencies, we had settled on 100 milliseconds from user to user during the design phase of our project. Later on we decided to break down the latency into two segments, paddle-to-headset (within a user's scope), and user-user. We set a goal for each of these to be less than 300 milliseconds.

Another design requirement concerns the actual physical paddle, the overall requirement remained unchanged from our design report (although more detail to the allocation has been added). In Section II it is mentioned that we would like our paddle to be less than 500 grams. As for a breakdown of this, we first began by allocating weight to the smaller components first, knowing that the majority of the weight would arise from the power source and any casing used to encapsulate the subsystem. The Raspberry Pi Zero is about 20 grams, the MPU6050 IMU is about 2 grams. The other component outside of the actual power source would be a Buck converter, and only if we used batteries. Buck converters weigh about 11 grams. A normal table tennis blade (a paddle without the rubbers) weighs about 80 grams. This would leave us with over 350 grams to allocate towards batteries or a battery pack, as well as the subsystem's casing. AA batteries are about 30 grams each, 9V batteries are only slightly heavier. The Raspberry Pi Zero Wireless draws up to 370 mA. We would like our paddle to hold enough power for one hour without charging or replacing batteries. An average set of table tennis takes about somewhere between 5 and 10 minutes, and nearly half of this is because players need to pick up the ball. Since this is not necessary in a virtual game, a user can comfortably 10 normal sets within one hour. An AA battery holds 2850 mAh, enough energy for nearly 8 hours on the RPi. If higher voltages are needed, a 9V battery contains 580 mAh. This would leave over 300 grams for any necessary casing. PLA, a common material for 3D printers, has a density of 1.25 g/cm^3. This means we comfortably have weight for 240 cm^3 of PLA, which is approximately the volume of two entire table tennis blades. As for the size of the paddle, the requirements have largely remained the same, adjusting the thickness to accommodate for hardware. The face should be around 6-7 inches in diameter and between 0.5 and 1 inch thick. The handle should be about 4 inches long and no longer than 1 inch thick or wide.

Regarding the CV subsystem, our initial plan was for the tracking of the paddle to be accurate to within 7.5 cm. This distance was chosen because the diameter of a paddle face is approximately 15cm. With this level of accuracy, we would be able to detect any contact between the ball as long as it hit near the center. If the accuracy was very close to this level we would also probably introduce a small range of leeway in case the paddle was registered to be in a location such that the ball contacted it near the edge (when in reality it should be a successful shot). We would also like to display at least 30 frames per second. While this is a requirement of the ultimate output (the user's display), without interpolation/extrapolation, this would also require us to have at least 30 frames per second processed from the CV subsystem. As mentioned near the end of Section II, we were aiming to accurately track paddles moving up to 5 meters per second.

To make the determination of swing type more accurate, we need to make sure that the IMU performs enough samples where it can always classify a swing if swung at the right time. We also need to make sure that the IMU data is accurate and that noise in the sensor's samples does not send false positive swings signals back to the game. Another thing that needs to be accurate is the acceleration and orientation thresholds we set to determine swing types and swing directions. We require that IMU take at least 30 samples, which we derived by taking the average swing window for a player (0.3 seconds) and multiplying it by the sampling rate of the IMU (100 samples/second). To make sure that noise isn't affecting false positives, we are going to graph the IMU output of 20 swings and make sure that 95% of them do not contain false positives. We will also require that at least 80% of the time, our swing thresholds output the correct swing type and direction.

V. Design Trade Studies

In this section I will be highlighting the tradeoffs we made in our two main subsystems: the VR headset and the paddle.

A. VR Headset Subsystem

For further clarification, the VR headset consists of the Unity game, Google Cardboard, Android phones, and paddle-headset bluetooth communication. To start, we are going to talk about why we chose these specific components for this subsystem.

- We chose to use Unity over other game engines such as Unreal because it had features that made programming the game a lot easier. Unity had a much larger asset store and the active community in Unity forums made integrating new components a lot easier. When we were implementing the multiplayer component of the game, Unity offered a Mirror Networking asset that abstracted away a lot of complexity and saved time.

- We chose the Google Cardboard because it was much cheaper than the Oculus.

- Our game was built on an Android because the setup for the bluetooth communication with it was much simpler compared to iOS. For iOS, we would have had to program in complex, low-level code for bluetooth communication whereas for Android it used more intuitive socket libraries.

- Using Bluetooth communication was an idea we had from the beginning, and throughout our integration and testing process, the data transfer was fast enough where we did not think about any alternatives.

In our Unity-created game, we had a trade-off between how realistic the game felt versus latency of the paddle swing reflecting in the graphics. The reason for this is that we sample 200 data points from the IMU from the point the opponent hits the paddle, we analyze those data points, and have the paddle send a message to the VR headset. We saw that if the paddle was too close to the table, the message would not have enough time to reach the VR headset and trigger a swing animation before it made contact with the idle paddle, which is why we made the decision to push the paddle further back than we originally wanted.

We also had to trade-off the accuracy of how the game reflected a player's swing with latency. In the beginning of our project, we were sending a stream of orientation data from the paddle to the VR headset, which makes reflecting a player's swing in the graphics really accurate, but sending such a large volume of data would cause really high latencies. To counteract that and make the game more playable, we decided to try to extrapolate swing characteristics from a sample of the player's motion and just send swing characteristics to the headset. Less data means lower latencies, but our predictions were not always correct. We weren't registering swings that occurred outside of our sampling period and sometimes our swing characteristics were wrongly predicted. However, what we saw at the end of the project was that with this strategy, the game had much lower latency and it only lost 20% accuracy when it came to swing classification, which we thought was justifiable.

B. Paddle Subsystem

Our paddle consists of a 3D-printed box containing an IMU sensor, Raspberry Pi Zero, and 5V, 2.1A battery pack. For our user requirements, we wanted a lightweight paddle that had long battery life, which is why we chose the battery pack over alternative power supplies. The Raspberry Pi had bluetooth capabilities which is why we chose it over a Jetson. We did not think much about alternatives for the IMU, we just needed something that would track orientation and acceleration data.

Just like in the VR Headset subsystem, the paddle has tradeoffs between the latency and accuracy of the swing classification, however in the case of the paddle, it sacrifices latency for swing accuracy. When our paddle is done sampling data, it uses a Butterworth filter to smooth out the data, which prevents false positive signals that there was a swing. However, it takes extra time to analyze the data, thus increasing the latency. An alternative would have been to analyze the data in real time and send signals to the player when a swing is identified, which leads to possible false positives but better latency. Looking back, we should have taken the alternative method because our sampling method lowered accuracy in the sense that swings that are too early or late do not get registered. Also, we could have used a sliding window filter to quickly smooth the incoming data in real time, which would have been the best of both worlds: no false positives and no processing latency.

VI. System Implementation

A. *Paddle Motion Sensing Subsystem*

The paddle performs motion tracking using a MPU6050 6-degree-of-freedom inertial measurement unit (IMU) connected to a Raspberry Pi Zero Wireless, which performs transmission of data via bluetooth as well as mathematical calculations to analyze the swing data. It is powered by a battery pack that provides 2 Amperes and 5 volts to sufficiently power the Raspberry Pi, which requires 1.5 Amperes and 3.3 Volts to run properly. In addition, the Raspberry Pi uses bluetooth to communicate with the Virtual Reality application on the VR phone. The bluetooth was implemented using the python bluez library, which transmits bits using Linux RFCOMM.

The IMU gets data regarding the acceleration and rotational velocity regarding the X, Y, and Z axes of the IMU relative to the frame of reference of the IMU. The data is sampled at 100 samples per second, and is sent to the Raspberry Pi in real-time via an I2C connection. Once the data arrives at the Raspberry Pi, it is passed through the Madgwick Attitude-Heading Reference System algorithm, which converts the accelerometer and gyroscope data from the IMU to quaternions in order to determine the rotation of the paddle. Subsequently, the quaternions are converted to roll, pitch, and yaw to represent the orientation of the paddle relative to Earth's frame of reference.

This live processing of IMU data is leveraged so that we can determine how the swing interacts with a ball in the game in as close to real-time as possible. Our method of determining that interaction is twofold: first, we decide when the window in which a swing would be appropriate in order to "make contact" with the ball, and secondly, we analyze data during that time frame to determine if and how the swing of the paddle will interact with the ball.

To achieve the first part of deciding the valid "swing window", the game application sends a signal to the Raspberry Pi signifying the start of the swing window. When the window starts, the Raspberry Pi begins tracking and analyzing the IMU data. The window will gather N samples of IMU data in real time, with N being a parameter that was tuned to help the game have a smooth game play. While sampling, the gyroscope data is passed into the Madgwick Algorithm as stated earlier to get the roll, pitch, and yaw of the paddle over time. In addition, the accelerations over time are also stored in a list during the sampling window. These stored accelerations are in the frame of reference of the IMU, but are also adjusted using some quaternion arithmetic so that acceleration due to gravity is removed from all 3 axes of measurements, effectively "centering" the measurements around 0 when there is no acceleration in any direction relative to the frame of reference. When the window ends, the accelerations are passed through a butterworth filter to smooth out the noise in the signals, so that we have smoothed out acceleration over time data centered at 0.

The assembled orientation and acceleration data are then used to determine the swing of the player. Because the swing window is a relatively short window of time (<0.7 sec), the orientations are simply averaged to get an estimate of the general orientation over that time. This average, especially for the roll measurement of the paddle, is useful in determining the angle of the swing by the user. For instance, an average roll of 10 degrees would suggest a very shallow upward swing that resembles a lob, whereas an average roll of 90 degrees would suggest more of a straight angle shot such as a push. The acceleration of the paddle during the swing window is then used to determine if and how hard the paddle was swung, as well as whether the swing was forehand or backhand. This is primarily done checking the acceleration on the Z-axis of the paddle, which goes through the plane of the paddle. A positive spike in acceleration followed by a negative dip resembles the paddle being moved in the forehand direction, whereas a negative dip in the acceleration followed by a positive spike in the acceleration suggests a backhand swing.

Given the correlation between acceleration and swing direction, the swing direction can be determined by seeing if during the swing window, whether the maximum of the graph occurs before or after the minimum of the graph, which in essence determines if the upward spike precedes the downward dip, or vice versa.

We also can determine what direction the ball will go following a swing. To do this, we use a design similar to that used in Wii Sports, where if you swing a forehand early, it will send the ball left, and if you swing a forehand late, it will send the ball right. The inverse holds true for backhands: an early swing sends the ball right whereas a late swing sends the ball left. These parameters are assuming the user is right-handed.

Once the swing direction, orientation, and ball direction are determined, it is relayed onto the game application via bluetooth. This process repeats during the back and forth of the game.

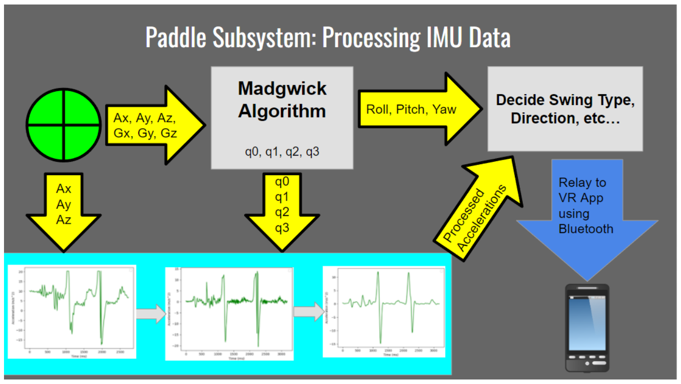


Fig 2. Diagram for paddle subsystem

B. *VR Headset Subsystem*

The VR Headset Subsystem of the product consists of the Google Cardboard, Android, and Unity-engine developed game. Unity builds an apk file that allows us to run our created game on our Android phone. The game should have stereoscopic imaging and head tracking so that when the player puts the phone into the Google Cardboard, they can experience the game in an immersive environment. In the diagram shown below, it shows how the software is organized and how it allows all of the hardware components to interact.

Starting with the bluetooth communication system, it sends and receives messages from the paddle, so when the ball hits the opponent's paddle, the game sends a message to the paddle telling it to start sampling data from the IMU. The moment the paddle is done sampling and analyzing the data, it sends the swing type and direction to the game.

From the game's point of view, the bluetooth communication is implemented using a PaddleConnect script that creates a Bluetooth socket to send and receive data. The script parses the message that the paddle sends to it and it triggers an animation that corresponds to the swing time. There are six possible animations that can be triggered (Lob, Spike, and Normal for forehand and backhand) and all of them are created by us for this project. The information that PaddleConnect receives is used to determine how the paddle to ball interactions work in the PaddleCollision script. In the latter script, when Unity Colliders recognize a collision between the ball and the paddle, it adds a force vector to the ball that is dependent on the swing type and direction. A paddle that is classified as a lob will add a high y-component to the force vector, a normal swing will have a lower y-component, and a spike swing will have the lowest. Swing direction determines the z (left and right) component of the force vector.

Originally, our game was meant to have a camera subsystem that kept track of the lateral position of the paddle. However we were unable to integrate that into our game, so as a contingency plan, we added the PaddleMovement script to the game. This script tracks the height and lateral position of the ball relative to the table (y and x coordinates) and uses them to set the same coordinates for the paddle's position. So whenever the ball reaches the player's end of the table, the paddle will always be in position for the ball to be hit. Essentially it makes the game timing based to see if a swing will trigger as the ball is hitting the paddle. The WallCollision and TableCollision scripts in the game are used to enforce the rules of ping pong. So whenever a player hits the ball and it lands on their own half of the table or if the ball lands out of bounds and hits the ground, the game will reset back to a serve state where the computer will start a new rally with the player.

The last two scripts involved in the game integrate the VR headset and add multiplayer capabilities into the game, both use external libraries that help us add functionality to the game. For the VR script, we import the Google XR library which displays the game with stereoscopic imaging so that it can be viewed in an immersive environment on the Google Cardboard as well as adding head tracking to the headset as well. We use the Mirror Networking library for multiplayer capabilities, which allows us to either set a dedicated server to run the game or allow a player to host a game for other people to join.

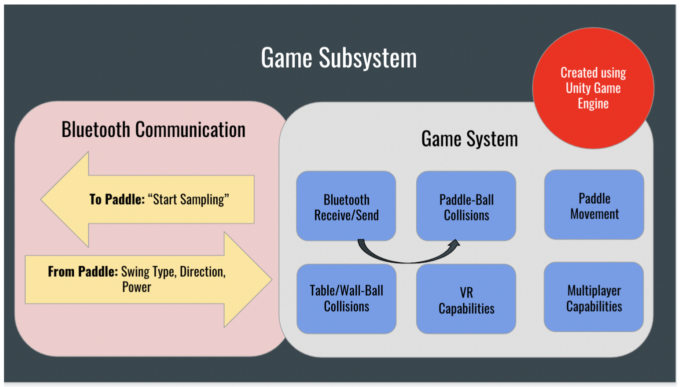


Fig 3. Diagram for game subsystem

VII. Test, Verification and Validation

For testing, the main components we tested were accuracy of the swing, effects of the latency of the swing transmission, and also the accuracy of the graphics and "physical" interactions on the VR application.

A. *Results for Swing Direction, Type, and Ball Direction*

Swing type refers to whether the swing was forehand or backhand, or whether there wasn't a viable swing. Swing Direction accuracy refers to whether the shot was a lob, straight shot, or smash. Ball direction refers to whether the ball goes left, down the middle, or right. To determine the accuracy for all of these metrics, we performed a large sample of swings of all the possible combinations of swing type, direction, and intended ball direction, and calculated the accuracy of the classification of those factors versus the expected type of the swing.

At the beginning, the swing directions were hitting at a 50 percent rate. This was because we were averaging the orientations throughout the entire swing window (from when the ball was bouncing to the ball passing the end of the table). We realized this was not the most reliable method, as if there was a range of motions they swung through, the average could not be indicative of the actual orientation of the paddle at the point of contact. We tried to fix this by detecting when the maximum velocity of the swing was, and then finding the orientation at that corresponding time. However, we realized that was much more prone to error, as we were getting some smashes being classified as lobs, which was a gap over two classes of swing directions, as the progression of swing directions goes from lob to straight shot to smash. We realized that this was because the accuracy of the orientation data from the Madgwick Algorithm was not accurate during the course of a swing, as the huge spikes in acceleration in short periods of time combined with the accelerometer noise would throw off the calculations. Thus, we realized that we probably should sample the orientation before the swing began, to eliminate the error coming from the duration of the swing where accelerations are high. This method was actually quite an improvement, as we were able to increase the accuracy to an 85 percent accuracy as is shown in the table below.

The swing type was another attribute that improved through testing. At the beginning, we faced two primary issues: false positives (registering a swing when the user didn't swing), and forehands being registered as backhands, and vice versa.

As for false positives, we had a ton of those in those to begin with as a chain reaction of the Madgwick algorithm taking time to converge to the initial orientation after each swing. This was an issue as during the process of eliminating acceleration from gravity from our accelerometer data to center our accelerations around 0, we used the quaternions from the Madgwick Algorithm. However, because the Madgwick orientations did not converge to the correct orientations yet, it would cause the adjustment of gravity to be inaccurate, and thus, our accelerations would not actually be centered at 0. This would propagate to the false positives, as we would not read a magnitude of 0 for the acceleration, but a large magnitude of acceleration even when the paddle wasn't moving. causing our classification to believe that the paddle was moving and thus registering a forehand/backhand instead of no swing. To remedy this, we increased our swing window so that we could have a few extra samples to allow for the orientations to properly converge before starting to use those orientations to adjust the accelerations. This method proved effective, as our data during the relevant time periods was more accurate and we eliminated almost all of the false positives.

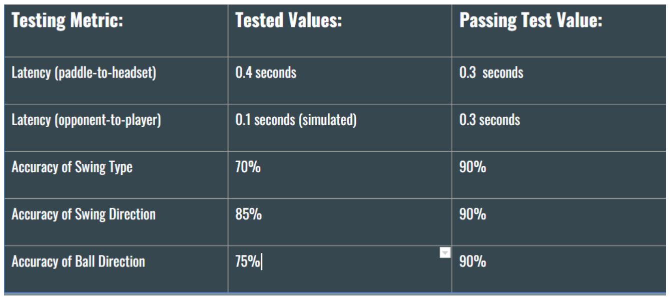
As for the forehands and backhands being mixed up, we realized that one cause was due to the fact that forehands would show a positive acceleration followed by negative, and vice versa. This was causing problems because sometimes, we would start sampling at the tail end of a swing (e.g. for a forehand swing, we would start sampling when the paddle is decelerating). Thus, the data would read that there was an initial negative dip instead of an initial positive peak, and read a backhand instead of a forehand. To remedy this, we realized that in addition to further expanding the data sampling window, we would add a buffer between the time the paddle orientation converges and the time that all swings are valid. This way, any swings that would have been valid would have been fully sampled, instead of being only sampled in the tail end. The buffer is helpful as if the tail end of a swing occurred during the buffer, it would be invalid anyways, so we wouldn't need to worry about it being misrepresented.

After tuning several parameters for the length of the swing window and the duration of the buffer, the issue of false positives and forehands and backhands being mistaken for each other diminished greatly to get us the metric of 70 percent accuracy on swing type. This still did not meet the requirements we set, as the issue causing the inaccuracy was now false negatives (the user swung but didn't register). The cause of this issue was that the swings sometimes didn't get registered well by the accelerometer, as the accelerations wouldn't cross the threshold for determining a swing. This was a tricky matter as we did not want to lower the thresholds too much such that noise would cause a false positive. This is something that would require more work in the future to analyze and tune.

Ball direction was a huge product of tuning parameters. As our methodology for determining the ball direction was the timing of the swing, it was all about tuning the thresholds for what would constitute an early or late swing. The accuracy metric fell short here, as it was hard to actually determine whether one truly felt like he or she hit the ball or late versus what the sensors recorded. Because the samples have such a high granularity, one that is hard for humans to process, it is actually quite logical that humans may not actually have swung the paddle at the time they expected. Thus, perhaps this metric is a little flawed, as it doesn't account for human error entering the accuracy formula.

B. Results for Latency

To find the latency between the paddle and the VR device, we took the Unix timestamp of the VR device and compared it to the arrival timestamp at the paddle. We got a metric of 0.4 seconds, which is a little higher than the metric of 0.3 seconds we desired. Because there is 0.15 second margin of error for getting the Unix timestamp and because there was not a noticeable latency between when a player swung and when the swing was reflected in the game, we decided that a 0.4 second latency was fine and our original goal of 0.3 seconds was too strict. We also tested for latency between players, or how long it takes for an opponent to see the other player's swing and movement. The best way we thought of testing this was measuring the latency when the opposing player was moving their paddle really quickly from side to side. So for our test, one player started on the right side of the table and moved as quickly as possible to the left side of the table. The moment the player reached the left side of the table, it would print out the Unix timestamp. We used ParallelSync to get the point of view of the opposing player. ParallelSync essentially allows us to run two versions of the game simultaneously and let them join the same game. So from the opposing player's point of view, when it sees the player's paddle reach the left side of the table, it also prints out the Unix timestamp. We subtracted the two timestamps to see to latency between the players and saw that it came out to an average of 0.1 seconds, which was a lot better than our goal, and qualitatively, the latency seemed just as low.



Testing Metric:	Tested Values:	Passing Test Value:
Latency (paddle-to-headset)	0.4 seconds	0.3 seconds
Latency (opponent-to-player)	0.1 seconds (simulated)	0.3 seconds
Accuracy of Swing Type	70%	90%
Accuracy of Swing Direction	85%	90%
Accuracy of Ball Direction	75%	90%

Fig 4. Obtained values and goals for testing metrics

VIII. Project Management

A. *Schedule*

Our schedule changed quite a bit from the schedule referenced in the design document. The computer vision aspect of the project was taking a lot longer than any of us were expecting, and we realized a lot of the features of the game that we had originally planned were not going to make it to the final product because the integration of the paddle took a while and swing identification was a newly added aspect to our project that took a lot of time as well. The updated Gantt chart is referenced on Figure 5 below.

B. *Team Member Responsibilities*

Henry's main responsibility in this project was creating the game in Unity. He programmed all of the ball-paddle interaction and movement scripts, created animations for each type of swing, and added bluetooth communication with the paddle. His secondary responsibilities included helping William with swing identification and helping Logan out with integrating the Jetson into the games.

William's main responsibility in this project was analyzing data from the IMU and using it to classify swing types and directions. He also was in charge of designing and fabricating the paddle as well as working with Henry to set up the Bluetooth connection between the paddle and VR headset.

Logan was in charge of the computer vision aspect of the product, which included setting up and integrating the Jetson with the VR headset and using OpenCV to determine what position the paddle was relative to the ping pong table.

C. *Bill of Materials and Budget*

We have included a table on Figure 6 that details what we spent our budget on. The items shaded in the light blue are materials used in our final product, every shaded in red are items that we bought but never used. We ended up spending a total of $154 on our project.

D. *Risk Management*

From the beginning of the project, we thought our biggest risk in our design of the project was the latency between the paddle and the VR headset. We ended up falling back on our contingency plan to have the paddle project what the swing will look like and send the paddle classification to the headset.

We had to also spend a lot of time coming up with contingency plans for our schedule in case certain parts of the project were unable to be completed or took too long to be completed. The product was separated into a group of mandatory components and non-mandatory components. We would first try to complete the mandatory components and if it took longer than expected, we took out a non-mandatory component. One example would be when we were doing swing classification. Completing this part of the project took way longer than we expected, so we decided it was smartest to scrap the spin capabilities that we wanted to add in the beginning. For the mandatory items, we came up with multiple ways to implement them so that if one way failed, we could always have a backup. This was the case with our paddle to VR headset connection. Our plan was to use Bluetooth, but if that failed, we would have transitioned to Wifi-direct. We also wanted to mention our contingency plan for computer vision. When our group saw that there was no way that the computer vision aspect of the project would be completed and integrated in time, we decided to simplify the game. So instead of the player needing to put their paddle in the correct lateral position, the game would do that for them.

We came up with a contingency plan for if any part of the paddle broke and actually had to use it during the final week. When we were creating our second paddle, we realized that our Raspberry Pi for one of the paddles was broken and we had no time to get a new one online and there were no available Raspberry Pi's in the inventory. We decided for the final demo to create an automated player that would always hit the ball back with either a lob or a normal swing. That way, the player could have a smooth experience with the gameplay and be able to test out the different kinds of swing types as well.

IX. Ethical Issues

A big ethical issue that arises from our product is safety. Because our game is a VR game, people playing do not have awareness of their physical surroundings. This poses a danger for the player and the people around them as a wayward swing could impose serious physical harm. One approach we are using to mitigate this risk is by creating warnings in the beginning of the game recommending that players play in an open area to protect themselves and the people around them. Other ethical issues that we envision include privacy risks that originate from the game being a multiplayer game and exclusion of players who cannot afford to buy this product and are thus left out of good social interactions. Ways that we can mitigate these risks is by including single player modes to the game and making the game as cheap as possible by using components like the Google Cardboard.

X. Related Work

We compared our game a lot to Wii Sports Tennis because the user experience was really similar. Players would swing with either a backhand or a forehand and depending on the timing of the swing, the direction of the ball would be changed. Our project was different in that players could swing with different swing types and our timing mechanism was different. In Wii Sports Tennis, all swings will be registered in the game, but in our game, only swings that are swung within a specific sampling period will be registered.

Another similar game we found was a VR ping pong game called ELEVEN table tennis. ELEVEN wasn't like our game in that it made players have to place the paddle in the correct lateral position. However, it was played on an Oculus, which abstracts away a lot of the paddle tracking and paddle orientation challenges that we had to deal with in our project.

XI. Summary

We were able to reach the design specifications for latency and IMU data accuracy. Qualitatively and quantitatively speaking, we were happy with the latency between a player's swing to the graphics reflecting it. And during our testing, we saw very little false positives and false negatives when players swung within the sampling period. However, our biggest issue was matching our sampling period with when a player would swing. From our experience and the experience of players who tested it, timing the swing so that the sampling period registered it was really difficult to get. If we had more time, we would have implemented a sliding window filter to smooth out the data in real time so that all swings at all times could be registered by the paddle and reflected on the graphics. We also had a bit of trouble with swing classification accuracy where sometimes a normal swing would be classified as a lob. If we had more time there, we would have worked on tuning thresholds and even adding a few more variables into our algorithm to make swing classification more accurate.

A. Lessons Learned

We all agree that the biggest lesson learned from this experience was focusing more on the integration aspect of the project. At the beginning of the project, we figured out how to partition the project into three chunks: paddle, VR headset, and camera. However, we did not spend nearly enough time talking about how these subsystems were going to be integrated with each other. We needed to talk more about what information needed to be sent and received between each subsystem and clearly define dependencies where progress could not move ahead without another subsystem. Potential problems and contingency plans were not discussed nearly enough from the beginning.

Also, we underestimated the difficulty of tasks too much, which caused us to bite off more than we could chew when we were creating a schedule for the project and talking about what we needed to do in future weeks. In planning, we needed to brainstorm potential problems that could have occurred and assign deadlines based on them.

Glossary of Acronyms

MQTT – Message Queuing Telemetry Transport
OBD – On-Board Diagnostics
RPi – Raspberry Pi
VR – Virtual Reality
CV - Computer Vision

References

[1] Boer, Jonas. "Morgil/madgwick_py/Madgwickahrs.py, Morgil/madgwick_py/Quaternion.py." GitHub, 2015, https://github.com/morgil/madgwick_py.

[2] mrrobles09. "Questions about MPU9250_MS5637_AHRS_t3.Ino Script... · Issue #223 · Kriswiner/MPU9250." GitHub, 2018, https://github.com/kriswiner/MPU9250/issues/223.

[3] "MPU6050 (Accelerometer+Gyroscope) Interfacing with Raspberry Pi:" ElectronicWings, https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-interfacing-with-raspberry-pi.

[4] "Raspberry Pi Bluetooth Setup and Running Rfcomm Server." 2017, https://www.youtube.com/watch?v=DmtJBc229Rg. Accessed 3 May 2022.

[5] Mallari, Jan. "How to Setup Bluetooth on the Raspberry Pi." Circuit Basics, 26 Nov. 2021, https://www.circuitbasics.com/how-to-use-bluetooth-with-raspberry-pi/.

[6] Google. (n.d.). *Quickstart for Google Cardboard for unity | google developers*. Google. Retrieved February 25, 2022, from https://developers.google.com/cardboard/develop/unity/quickstart

[7] YouTube. (2021, June 23). Build your first 3D game in Unity | Unity beginner tutorial. YouTube. Retrieved February 20, 2022, from https://www.youtube.com/watch?v=n0GQL5JgJcY&ab_channel=CodinginFlow

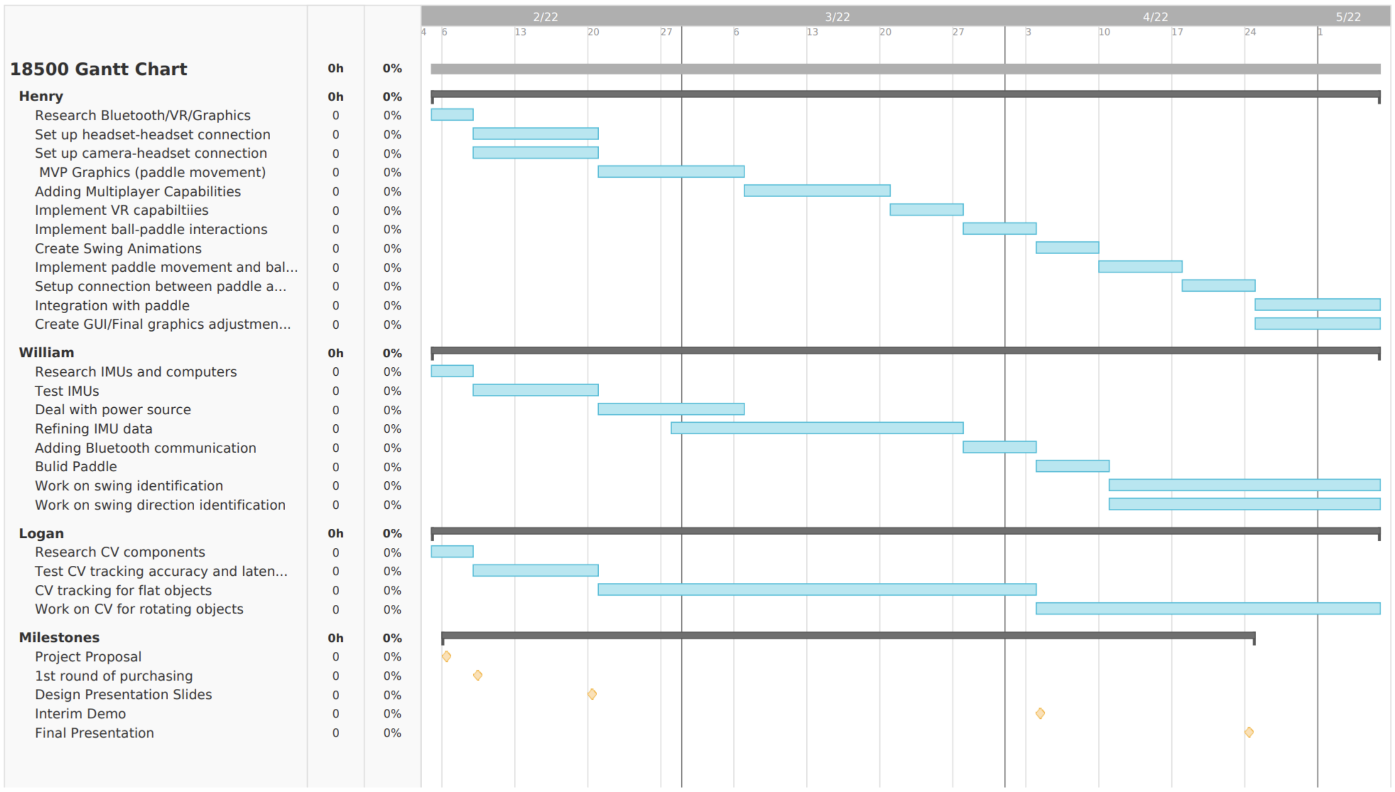
[8] Vis2k/mirror: #1 open source unity networking library. GitHub. (2018). Retrieved March 22, 2022, from https://github.com/vis2k/Mirror

[9] Technologies, U. (n.d.). Unity user manual 2021.3 (LTS). Unity. Retrieved May 7, 2022, from https://docs.unity3d.com/Manual/index.html

[10] YouTube. (2019, August 30). How to animate in unity 3D. YouTube. Retrieved April 7, 2022, from https://www.youtube.com/watch?v=sgHicuJAu3g&t=1283s&ab_channel=JonasTyroller

[11] Pounder, Les. "Raspberry Pi Zero 2 W Review: The Long Awaited Sequel." Tom's Hardware, 28 Oct. 2021, https://www.tomshardware.com/reviews/raspberry-pi-zero-2-w-review.

[12] "Ma Long vs. Fan Zhendong, Men's Singles Table Tennis Gold Medal Match Tokyo 2020." YouTube, Olympics, 13 Aug. 2021, https://youtu.be/VTCDQYYKA9o. Accessed 7 May 2022.



18500 Gantt Chart	0h	0%
Henry	0h	0%
Research Bluetooth/VR/Graphics	0	0%
Set up headset-headset connection	0	0%
Set up camera-headset connection	0	0%
MVP Graphics (paddle movement)	0	0%
Adding Multiplayer Capabilities	0	0%
Implement VR capabiltiies	0	0%
Implement ball-paddle interactions	0	0%
Create Swing Animations	0	0%
Implement paddle movement and bal...	0	0%
Setup connection between paddle a...	0	0%
Integration with paddle	0	0%
Create GUI/Final graphics adjustmen...	0	0%
William	0h	0%
Research IMUs and computers	0	0%
Test IMUs	0	0%
Deal with power source	0	0%
Refining IMU data	0	0%
Adding Bluetooth communication	0	0%
Bulid Paddle	0	0%
Work on swing identification	0	0%
Work on swing direction identification	0	0%
Logan	0h	0%
Research CV components	0	0%
Test CV tracking accuracy and laten...	0	0%
CV tracking for flat objects	0	0%
Work on CV for rotating objects	0	0%
Milestones	0h	0%
Project Proposal	0	0%
1st round of purchasing	0	0%
Design Presentation Slides	0	0%
Interim Demo	0	0%
Final Presentation	0	0%

Fig 5. Gantt Chart for our project

Description:	Model Number:	Manufacturer:	Cost:	Quantity:
Google Cardboard VR Kit	FBA_TT9LK	Topmaxions	$7	2
Inertial Measurement Unit	3886	Adafruit	$9	1
5V, 2.1A Battery Pack	A-10K05	Alongza	$19	2
Ping Pong Paddle	PN2	Champions Sports	$7	1
Raspberry Pi Zero	F21024 & F21025	Raspberry Pi	$0	2
Unity Bluetooth Plugin	N/A	Unity	$19	1
Ping Pong Graphics Pack	N/A	Unity	$5	1
3D printed sensor container	N/A	CMU Techspark	$30	1
Bluetooth Adapter	9V3K5WF64F326	hudiemm0B	$2	1
Jetson Wifi Card	B07X2NLL85	Maketronics	$22	1
Wifi Card Antennae	6484109	Chaohang	$12	1
Jetson Nano 2GB	F21034	Jetson	$0	1
Camera	F21043	Sainsmart	$0	1
Arduino Uno Rev3	EDE0070	Arduino	$0	1

Fig 6. Bill of Materials for our project (Shaded blue items are in our project, shaded red are not in our project)

Report Section Breakdown:

- Introduction: Logan
- Use Case Requirements: Logan
- Architecture and/or Principle of Operation: Logan
- Design Requirements: Logan
    - Henry added a paragraph at the end about design requirements for swing accuracy user requirement.
- Design Trade Studies: Henry
- System Implementation:
    - Part A: William
    - Part B: Henry
- Test, Verification, and Validation
    - Part A: William
    - Part B: Henry
- Project Management: Henry