That's so Fetch

Luca Amblard, Dan Barychev, Hana Frluckaj
Electrical and Computer Engineering,
Carnegie Mellon University
{lamblard, dbaryche, hfrlucka}@andrew.cmu.edu

Abstract

Our project is to develop a motorized device simulation that plays fetch with data received from Inertial Measurement Units (IMUs) on the user's hand. Our previous project involved building an omnidirectional robot to anticipate, catch, and return the object thrown, but due to constraints related to COVID-19 and remote instruction, we have since moved to a simulation based design for the robot. The data is fed into the inputs of a simulation to represent an omnidirectional motorized base. The result is a simulation that predicts the thrown object's landing location from the IMU data and then provides a visual representation of the object's projectile motion in three dimensions and of whether or not the robot may reasonably catch it.

Keywords

Inertial Measurement Unit (IMU), Particle Photon, I2C, Serial Communication, Magdwick's Filter, Dead Reckoning, Ball Release Detection, Equations of Motion, Target Estimation, Simulation, PID

Introduction

Our project is aimed towards children who would like to play fetch with a real dog but are unable to because of allergies, housing constraints, and other external factors. While the only current function of our device is the ability to play fetch (i.e. anticipate throw and catch the ball, or in our case a cornhole bag) virtually, most commercial robot dog products do not have this option in real life. As far as other motorized devices capable of catching items, there have been similar projects in the past, e.g. Smart Trashbox from Minoru Kurata and Smart Trash Can from F19 Team B4. Kurata's device had a success rate of about 20% but was highly adaptable and motorized, while Team B4's device had a success rate of about 50% with small movements and low motorization (i.e. small range of ~30 cm). Our goal was to create a highly mobile device (i.e. range of 1m) with a success rate over 50%. While previous projects had different approaches, to our knowledge, this is the only project to use IMUs to predict the trajectory of the object thrown. Our original project involved building an omnidirectional motorized base. This would be moved by motors on a PID control system based on data first sent over WiFi from a Particle Photon to a Jetson Nano. Given the sudden necessity to work remotely due to COVID-19 concerns, our project has now moved to a simulation based design that incorporates the same data from the hand IMU sensors, a portion of our project that we've been able to maintain. Our project now consists of having a Particle Photon connected to two IMUs on a user's glove, which sends the necessary input data for the simulation when an object is thrown, from which the simulation predicts a landing location that is then compared on-screen to the real one.

Design Requirements

The design requirements for our catch-and-retrieve system center around the need for an intuitive 'dog' virtual simulation that can catch the ball thrown in different locations with a success rate greater than 50%. In order to achieve greater than a 50% success rate and properly mimic the experience of playing catch, we are enforcing a starting distance of 1m away from the user and limiting the catch radius to 1m around this starting position. Thus, if the user decides to throw outside of this range, the simulator will refuse to catch the object thrown and a manual reset will be necessary.

In order to create a viable catch scenario, we are also enforcing the use of an action we term the "prethrow." The purpose of the prethrow is largely to collect data on the thrown object's approximate starting direction and speed before it is actually thrown. Multiple prethrows over several trials allow for a much more stable and accurate final trajectory prediction. From this data, we aim to compute accurate measurements on the bag's height at launch and the X,Y, and Z components of its velocity. In addition, due to changes made from our original dead reckoning system, we now require the user to input the resting height of their arm from the ground and the starting horizontal angle they wish to throw from.

To give the robot a good chance of actually catching the ball, a small-enough total computation time for predicting landing position and transmitting data is necessary. Deeming that the robot needs about 0.5 seconds out of an average 0.8 second time-of-flight to catch the ball, the total computation time must be less than 0.3 seconds.

Architecture and/or Principle of Operation

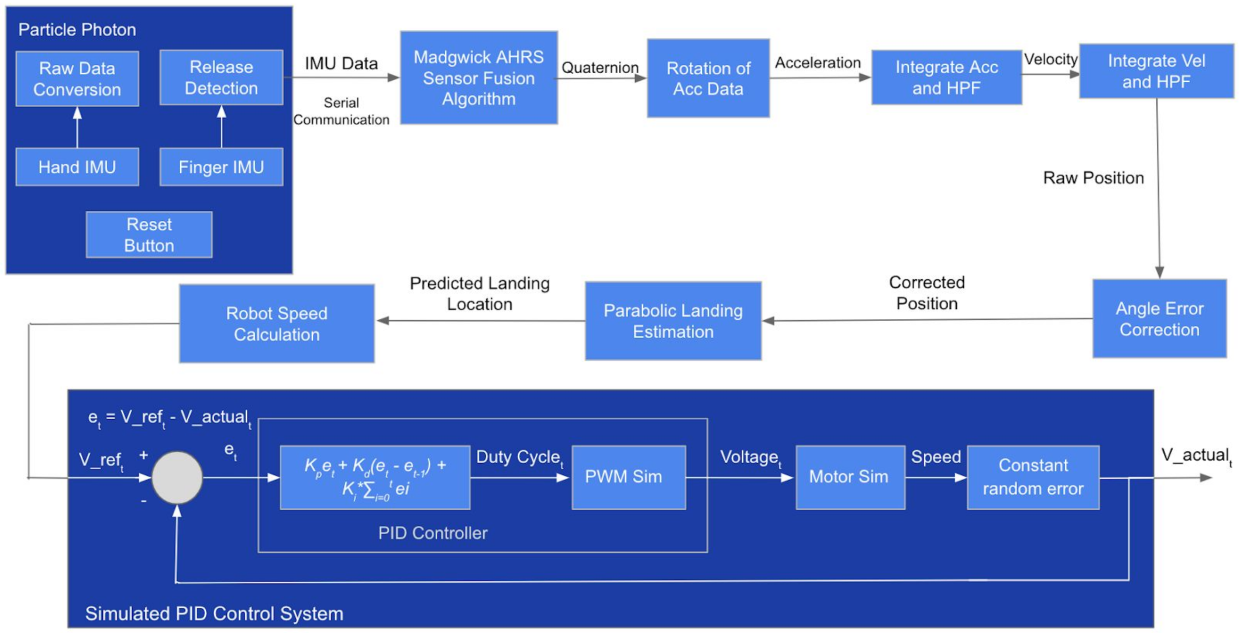


Figure 1 displaying Architectural Layout [11]

Our design, shown in Figure 1, consists of three main parts: the IMU data collection and transmission from the user's arm to a laptop, the computation to map raw hand IMU data to 3D velocity and throw height when the ball is released, and the robot simulation based on aforementioned data.

We are using the MPU 9250 (IMU) for motion sensing. It is small, light, and contains an accelerometer as well as a gyroscope which we both need to estimate the location where the ball will land. The raw data of the devices is read by the Particle Photon which communicates with the IMUs through the I2C serial communication protocol [7]. One IMU is located on the user's hand and the other is located on one of the user's fingers. Once the accelerometer and gyroscope data are received by the Particle Photon, they are converted to $m\ s^{-2}$ and $deg\ s^{-1}$ respectively. Because wireless communication from the Particle Photon ended up being too slow, we are now sending data from the Photon to a laptop through serial communication..

The hand IMU data is passed into Madgwick's AHRS Sensor Fusion Algorithm, which maps the hand IMU raw data to the IMU's velocity in 3D and throw height when the ball is released. The height of the IMU from the ground when the user's arm is straight pointing down is measured and fed into the software. The increase in vertical distance from the AHRS algorithm is added to the starting height to determine the throw height of the ball.

The throw is detected using the *y* gyroscope value of the finger IMU. We determined a threshold through experimentation. We made sure that the threshold is low enough to consistently determine when the ball is thrown but not too low, otherwise a throw would be falsely detected during a swing.

The data at the throw are applied to equations of motion in 3D to predict the landing location of the ball. The data is fed into the simulation, which displays the ball moving to its actual landing location, indicated by a yellow circle, and the robot moving to the predicted landing location, indicated by a blue circle. To simulate the ball moving to its actual landing location, we measure the ball's actual landing location after it lands using a tape measure and also its time of flight using a slow-motion camera and input these data into the simulation. If the user wants to throw the ball at an angle rather than straight, the user must also input the horizontal angle into the simulation. After the ball is caught, the robot returns the ball to the user and then moves back to its starting position. The simulation includes a bird's eye view and a side view. The motion of the objects in both views are synchronized and all speeds and distances are to scale.

We model real-world behavior in the simulation. Since we are simulating the robot we included a simulated PID control system to control its speed. Another example of real-world behavior is the reaction time of the robot. The AHRS computation time is fed into the simulation so that the robot is delayed by at least this amount of time before it moves.

The Particle Photon is programmed using C++. We use Python to run the AHRS algorithm and use Processing for our simulation. Processing makes it easier to work with graphics than a language like Python and it is very compatible with serial outputs.

System Description

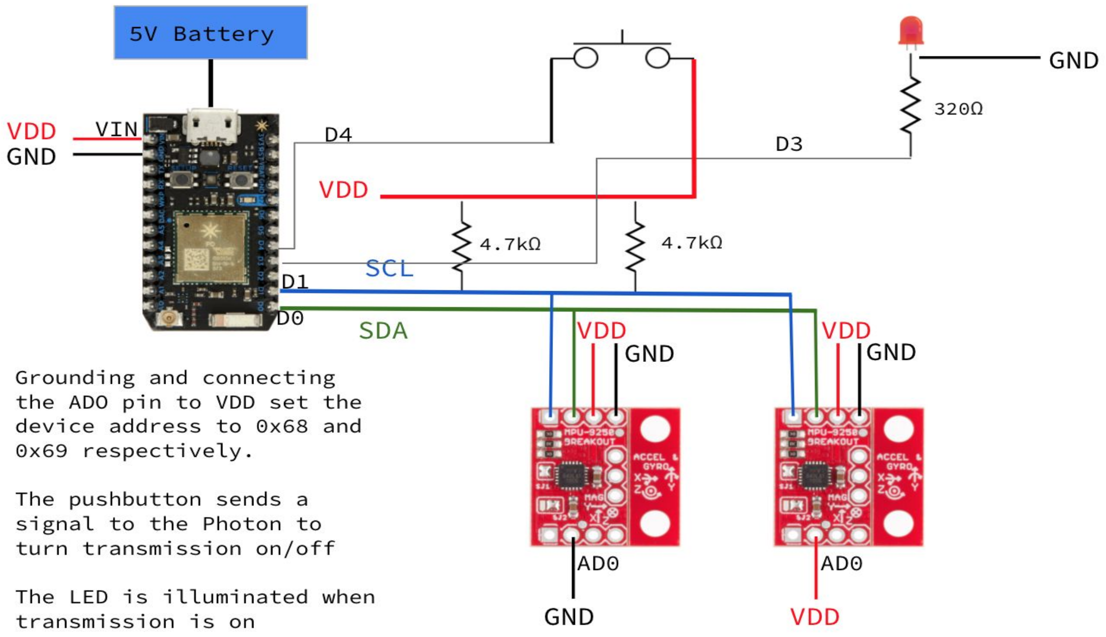
IMUs to Photon

The gyroscope data and accelerometer data are read from the two MPU 9250 devices (IMUs) by the Particle Photon using I2C. The data is transmitted from IMUs to the Photon through the circuit connections shown in Figure 2 below, and is later used to compute the ball's trajectory data.

We write to the gyroscope configuration and accelerometer configuration registers select the full-scale range for each device. The gyroscope can be set to have a full-scale range of either ± 250dps (degrees per second), ± 500dps, ± 1000dps and ± 2000dps. The accelerometer can be set to have a full-scale range of either ± 2g (g = 9.81$m\ s^{-2}$), ± 4g, ± 8g or 16g. We want the full-scale range for each device to be as low as possible to get greater sensitivity. Through experimentation, we determined that ± 500dps for the gyroscope and ± 2g for the accelerometer are ideal. After reading the data, the Particle Photon sends the data to a laptop through serial communication.

Each IMU contains an AD0 pin, which is used to set the device address of the IMU. This pin is connected to ground by solder, which sets the device address to 0x68. To address the second IMU, we removed the solder from the AD0 pin of the second IMU and connected the pin to VDD, setting the IMU's device address to 0x69.

Desiring an IMU sampling rate that would give us an acceptable granularity and wishing to avoid processing times of more than 0.5 seconds, we decided to sample data at 50Hz. We noticed similar sampling rates among individuals using the MPU9250 for granular tasks and decided 50Hz should suffice in terms of accuracy and speed.



Grounding and connecting the AD0 pin to VDD set the device address to 0x68 and 0x69 respectively.

The pushbutton sends a signal to the Photon to turn transmission on/off

The LED is illuminated when transmission is on

Figure 2 of connections between MPUs and Particle Photon

IMU raw data conversion

Formulas to convert gyroscope and accelerometer data to angular velocity($deg\ s^{-1}$) and acceleration (in g) values respectively:

For the gyroscope:

angular velocity (*dps*) = *raw data / sensitivity scale factor*

For the accelerometer:

acceleration (g) = *raw data / sensitivity scale factor*

g: gravitational field strength in $m\ s^{-2}$

The raw data can take values between 0 and 32768. Sensitivity scale factor (S.F.) is determined by the full-scale range used by the corresponding device. The tables below show the sensitivity scale factor that corresponds to each full-scale range for each device (accelerometer and gyroscope) [5].

Gyroscope :

Full Scale Range ($deg\ s^{-1}$)	Sensitivity S.F. ($deg^{-1}\ s$)
± 250	131
± 500	65.5
± 1000	32.8
± 2000	16.4

Accelerometer:

Full Scale Range (g)	Sensitivity S.F. (g^{-1})
± 2	16384
± 4	8192
± 6	4096
± 8	2048

Madgwick's AHRS

At the core of our prediction system, we have Sebastian Madgwick's Attitude and Heading Reference System (AHRS) algorithm. Madgwick's algorithm is an orientation filter that allows for an IMU's gyroscope and accelerometer readings to work in tandem. While the gyroscope provides angular velocities in the sensor plane, the accelerometer provides information about the Earth's gravitational field. By integrating quaternion derivatives and performing gradient descent on a large amount of possible rotations, Madgwick's algorithm provides an estimate for the sensor's rotation relative to the Earth's plane. This then allows us to rotate our acceleration data accordingly and subtract the effect of gravity from the z axis. As a result, we achieve accurate acceleration vectors in all three dimensions [1].

After calculating the proper acceleration vectors in all three dimensions, we can then integrate the acceleration to achieve velocity and subsequently integrate the velocity to achieve position. After each integration step, however, we must apply a high pass filter (HPF) to eliminate an appropriate amount of drift [2]. This allows us to significantly simplify our dead reckoning solution. Due to the double integration, this approach works best with cyclic motion, where mean velocity and displacement are 0. Since our throwing motion resembles a cyclic pendulum, the integration/HPF approach works quite well and our resulting 3D position data is quite true to form.

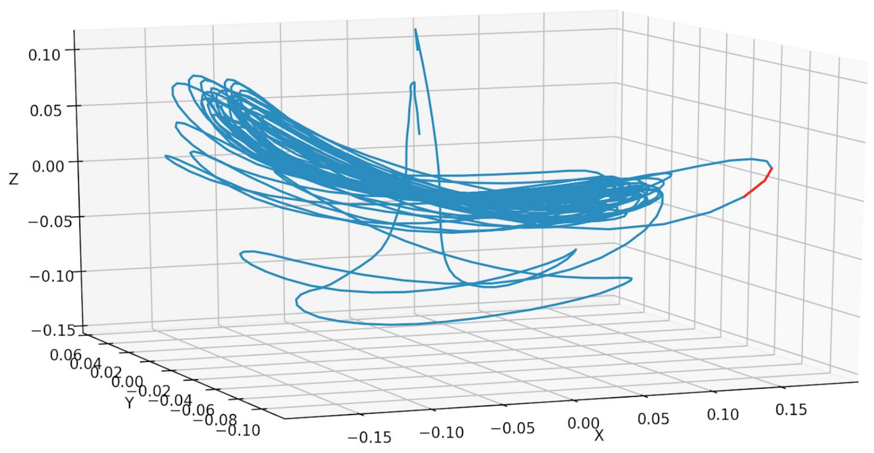


Figure 3 showing the 3D position results of a set of ~20 pre-throws. Release point is in red

Simulated PID Control System

The simulated PID controller takes in the error, which is the difference between the target speed and actual speed, and outputs the duty cycle, which would be used in a PWM driver. The duty cycle is used to calculate corresponding voltage of the signal, which is then used to determine the actual speed. A random error within the 10% error of the motors is generated at the start of the simulation and is used throughout to simulate the actual speed of the motors, which would be determined by encoders [11].

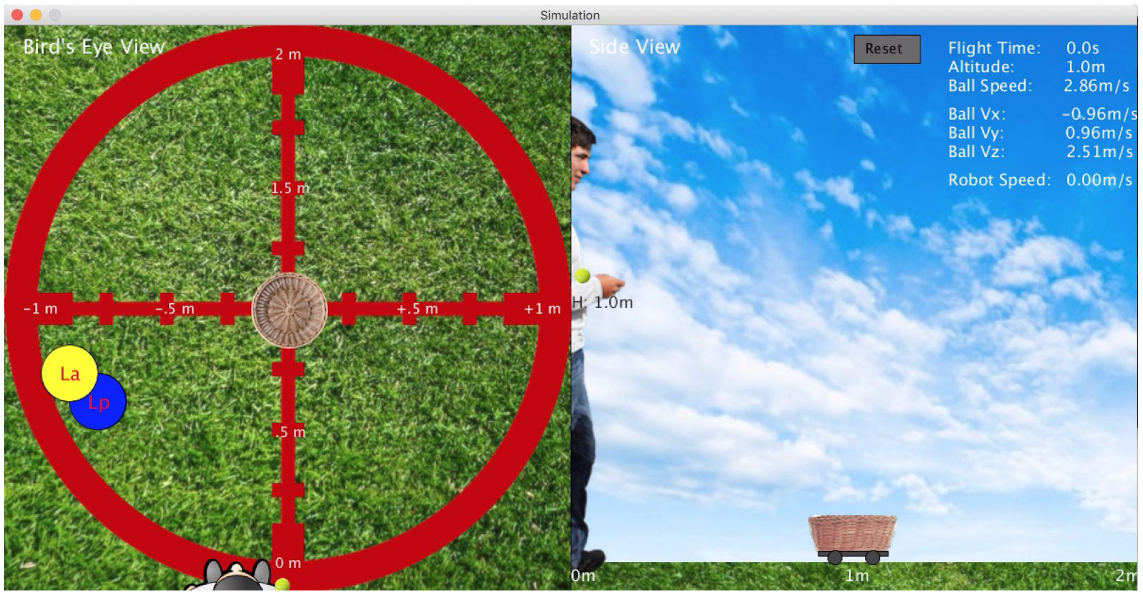
Parabolic Landing Estimation

Once the 3D velocity, throw height and horizontal angle of the ball at release are determined, we predict the ball's landing location using equations of motion in 3D and trigonometry. The Z-axis is the vertical axis, the Y-axis a horizontal axis (forward/backward) and the X-axis is the second horizontal axis (left/right). We start by calculating the predicted landing location in 2D using Vz (vertical) and Vy (forward direction) as well as the throw height. We then add the third dimension by introducing Vx (left/right) to calculate where the ball lands to the side. Finally, we add the horizontal angle component by rotating the predicted landing location around the user by this angle. We obtain (x, y) coordinates for the ball's predicted landing location, where the positive Y-axis is the forward direction relative to the user with the user positioned at the origin.

The actual landing location and actual time of flight of the ball are measured and fed into the simulation. Using these data, along with the ball's throw height, we can calculate the ball's actual velocity in 3D throughout its flight and simulate the actual throw and display the ball's trajectory.
Vz (vertical) is constantly updated due the Earth's gravitational field strength and the Vy and Vx components (horizontal) remain constant.

The ball's actual time of flight is also important to determine whether the robot catches the ball. In order for the robot to catch the ball, the predicted landing location must be very close to the actual landing location. Additionally, the AHRS algorithm execution time, the landing location prediction time as well as the robot's reaction time and time to move to the target location must all fall below the actual time of flight of the ball.

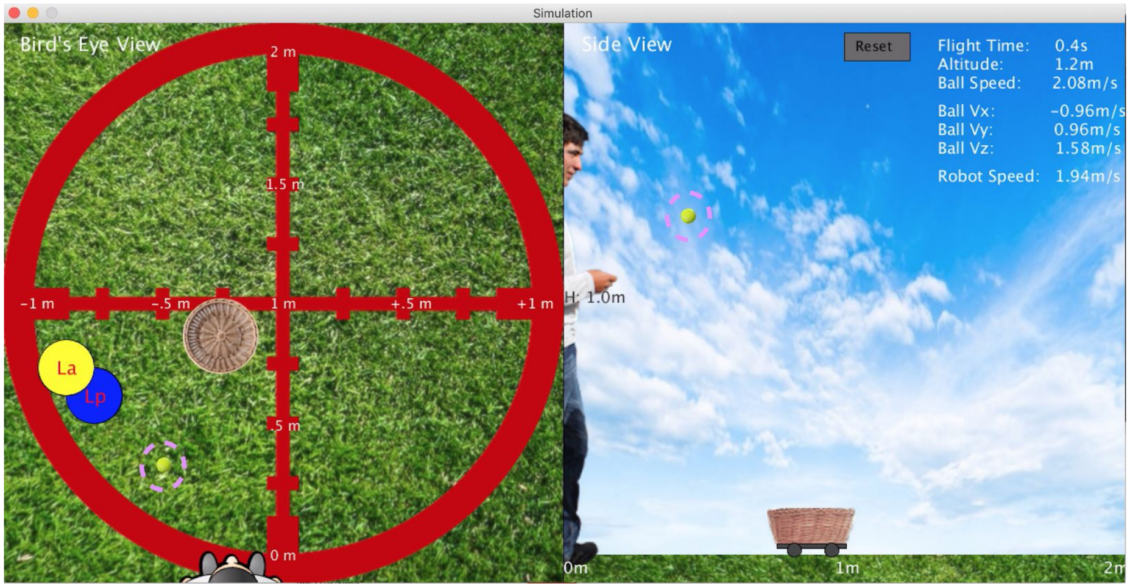
Figure 4 of Initial Frame (See [10] for image citations):



Flight Time: 0.0s
Altitude: 1.0m
Ball Speed: 2.86m/s
Ball Vx: -0.96m/s
Ball Vy: 0.96m/s
Ball Vz: 2.51m/s
Robot Speed: 0.00m/s

In Figure 4, we can see the initial state of the simulation. On the left hand side, in the Bird's Eye View, we can see the initial positions of the user at the bottom of the screen and the robot waiting 1m directly in front of them. We see several distance markers in the robot's movement radius that indicate displacement relative to the user. The user has not yet thrown the ball at this point. On the right hand side, we see the side view, which displays a user standing to the left of that screen, poised to throw the ball. This view also displays the side view of the robot with implied omnidirectional wheels, distance markers and several parameters including time and the initial velocities for the robot and ball.

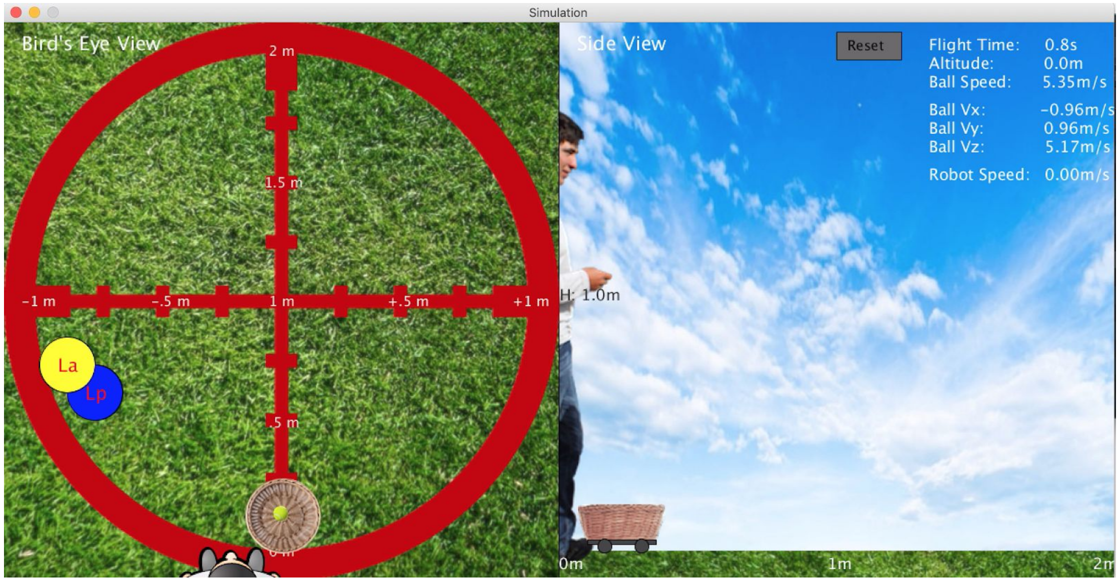
Figure 5 of Movement Frame:



Flight Time: 0.4s
Altitude: 1.2m
Ball Speed: 2.08m/s
Ball Vx: -0.96m/s
Ball Vy: 0.96m/s
Ball Vz: 1.58m/s
Robot Speed: 1.94m/s

In Figure 5 above, we may observe the simulation 0.4 seconds into flight, as indicated by the time on screen. In the Bird's Eye view, we may observe that the ball has moved forward and to the left of the user mid-flight, while the robot has moved about 0.25m leftward to the predicted landing position. In the Side View, we're able to see the ball following a parabolic motion, the robot moving slightly forward towards the user, with the Altitude, Ball Speed, and Robot Speed constantly updating on the side. From these views, it appears that the robot is on track to catch the ball.

Figure 6 of Catch and Return Frame:



Flight Time: 0.8s
Altitude: 0.0m
Ball Speed: 5.35m/s
Ball Vx: -0.96m/s
Ball Vy: 0.96m/s
Ball Vz: 5.17m/s
Robot Speed: 0.00m/s

In the final frame in Figure 6, we can see that the ball has been caught and returned to the user. The total time of flight was 0.8 seconds as shown on the screen, and we can see from both views that the robot has returned to the feet of the user. In the Bird's Eye view we may note that the ball is caught in the center of the robot's basket. In the Side View, we may see the ball's final velocity when it landed, as well as the altitude showing that the ball is currently on the ground or in the basket.

Design Trade Studies

IMU placement (inside ball v. on hand)

In discussing possible IMU placement configurations, we considered the possibility of placing an IMU inside the ball. The benefit to doing this would be the ability to track the trajectory of the ball as it moves through the air, thus mimicking the functionality of a CV setup. An issue with this setup, however, is that we would not know the ball's release point from an internal IMU alone. In order to avoid beginning landing prediction once the ball fell from the air, we would still need to have an IMU on the user's hand to determine release. Furthermore, we would either need to connect the IMU within the ball over WiFi which could result in an unstable connection as the ball flies through the air, or connect the IMU through an extremely long wire to the rest of the circuit. Either way, both of these options would mean that the IMU could withstand potential damage from being thrown around, and any potential incremental changes we'd like to make to the circuit would be extremely inconvenient. We decided that having IMUs in three different locations (hand, ball, and robot) would be too excessive and would take away from the uniqueness of the technical challenge. In addition, we would like for our system to work with multiple types of balls. We thus decided on having a knuckle IMU for detecting the hand opening at ball release and a hand IMU for trajectory information upon release.

Kalman Filter v. Madgwick's AHRS

When first brainstorming ways to track the motion of a hand while swinging, we settled on using a Kalman filter due to its widespread use in similar dead reckoning systems, as shown in Figure 1. We reasoned that after calibrating the hand IMU with the robot IMU, we could reliably track the position of both entities as they interacted with one another. Unfortunately, we severely underestimated the contribution of IMU drift to our measurement error. In addition, our sampling rate of 50Hz proved to be well outside of the ideal Kalman filter range (512 Hz - 30 kHz) [1] so detecting that an IMU was stationary would take around 5 seconds. These measurement errors and drift accumulations made it difficult to build a Kalman filter for both the gyroscope and accelerometer readings (necessary to remove gravitational influence). To solve these issues, we turned to Madgwick's AHRS orientation filter which could reliably tell us the hand IMU's orientation, allowing us to remove effects of the Earth's gravitational pull from our readings. By following this algorithm with double integration and use of high pass filters for drift, we were able to get reliable velocity and position readings since our swinging motion was cyclic. Unfortunately, however, we lost the ability to tell the user's starting height and horizontal throwing angle as relative swinging motion does not provide this. Since use of a reliable Kalman filter would require a dramatically different sampling rate and would likely fail to sufficiently eliminate the influence of drift, accurately tracking starting angle and height simply became an impossibility with our available hardware.

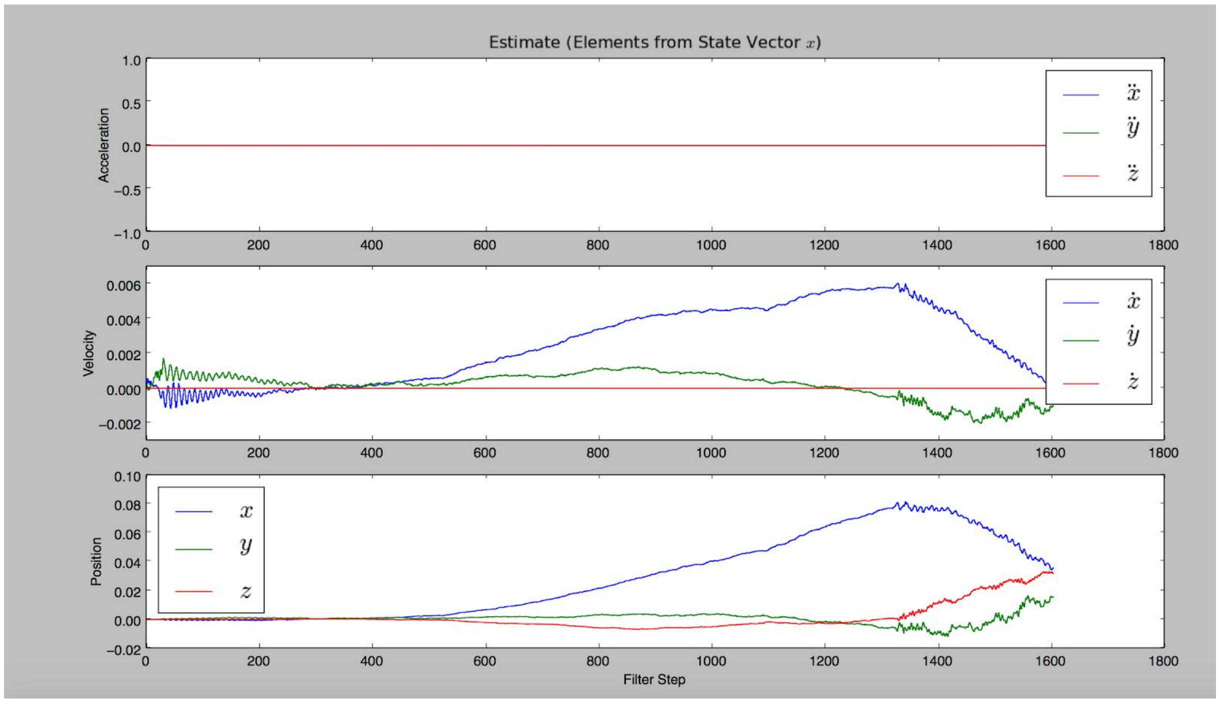


Figure 7 showing velocity and position estimation from a Kalman filter for a stationary IMU. 1600 samples in this case is equivalent to 32 seconds.

Wireless v. Serial

In our original design, we strived to communicate with the robot wirelessly using the Particle Photon's WiFi capabilities. We unfortunately discovered during implementation, however, that the Particle Photon's speed of transmission for wireless data was simply too slow. When trying to transmit each data point, we noticed that any delay of less than a full second between transmissions would result in data corruption. With 10 prethrows and 800 corresponding data points, transmitting this data wirelessly would take a full 13.33 minutes. In contrast, sending this data continuously through serial communication is nearly instantaneous. Even in a scenario where the Particle Photon would run Madgwick's orientation filter and output the data at time of object release, we would still have to wait a full second for this data to transmit, which is certainly too long for a successful catch. Thus, having to unfortunately subtract from the robustness of the project, we decided to rely upon the existing serial communication pipeline that we were using for testing due to it's fast speed of transmission.

Cornhole bag v. Hackysack

While there is no longer a physical robot that actually retrieves the projectile, we still need to throw something in order to gage the actual landing location and simulate the ball's actual trajectory. A standard rubber or plastic ball would've bounced everywhere and it would be difficult to accurately determine the ball's actual landing location. Therefore we devised to either use a cornhole bag or a hacky sack. Combined with the gloved hand, the hacky sack is much easier to throw as it doesn't restrict the user's arm movement like the cornhole bag, which is large and hard to grip properly. However, the hacky sack is more prone to rolling about 10 cm off from the actual landing, and would therefore be less likely to provide accurate landing data. As such, we decided to use the cornhole bag.

One IMU v. Two IMUs

Throughout the development process, our team discussed the possibility of simply using one IMU (likely positioned on the finger) to interpret both the arm's motion and its release point. The advantage of doing this option would be an added simplicity and a clear benefit to computational load when only having to continuously address one I2C device instead of two. When testing this strategy, we discovered that although it was feasible to use such an approach, the results were noticeably worse than those obtained using two IMUs. As shown in Figure 8, using the finger IMU for the hand motion at time of release creates a significant position error (clear if compared to Figure 3) when the user's finger opens to release their object. Thus, despite the advantages in computation time and constructional simplicity this method would allow, we resolved to stick to our design of a two-IMU system.

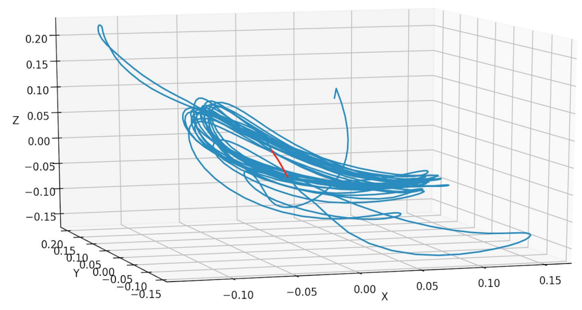


Figure 8 showing throw position / release data all from one IMU

Project Management

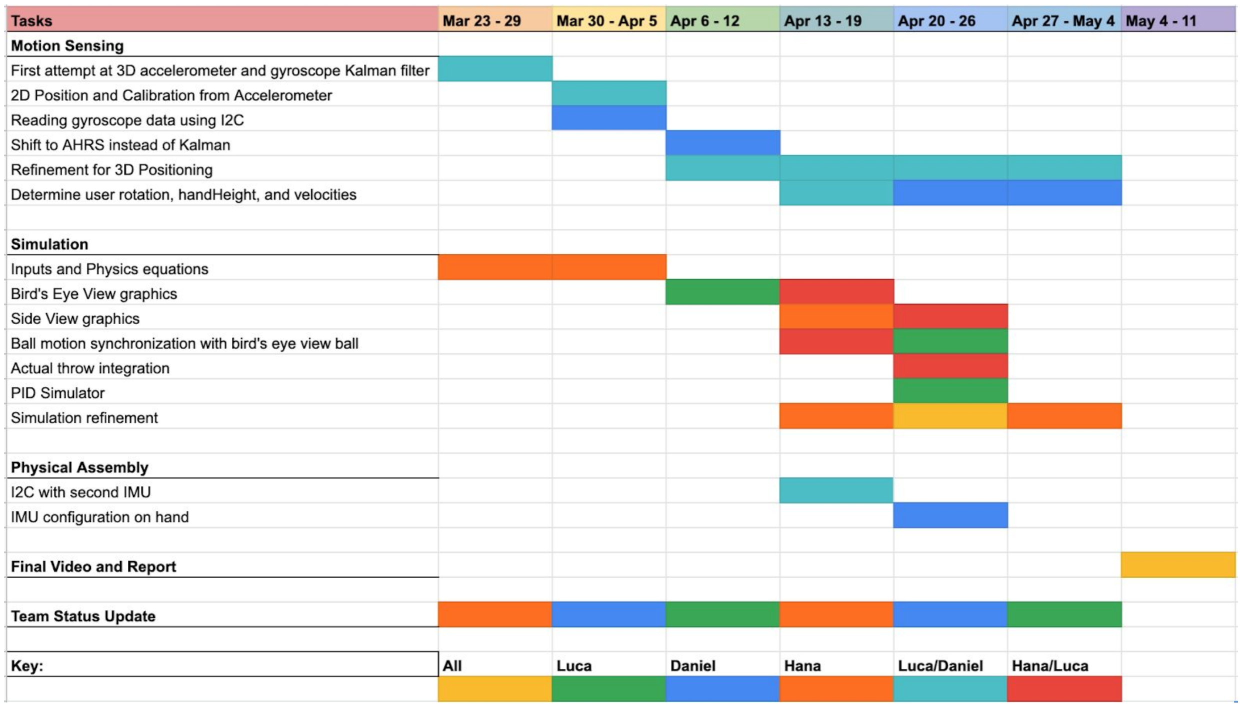
After re-working our design because of the new constraints of COVID-19, we also had to rework our schedule. This is an updated schedule which shows several changes during that time, most notably the removal of several hardware components and communication between said components, with the addition of a simulation to represent the removed modules. Overall, Dan worked primarily on the motion sensing and hardware components, Hana worked primarily on the simulation, and Luca worked on both aspects in conjunction with both members. Our schedule indicating the changes in roles due to COVID-19 and redesign are shown in Figure 9 on the following page.

Related Work

There are two existing similar projects: Smart Trashbox from Minoru Kurata and Smart Trash Can from F19 Team B4. The success rate for both projects were relatively low, with Kurata's device at about 20% and Team B4's at about 50%. We hope to accomplish a success rate over 50% within our simulation, as we do not have a physical product.

Kurata's device is a much faster and mobilized version than Team B4's, and so it could cover greater distances. This probably contributed to the low success rate there was a lot of distance to cover. From Team B4's project demo, it appears that their trash can could not move very far (~30cm), and so the team had a high success rate from aiming the object thrown very close to the device's rim.

We no longer have the same physical constraints as these two previous projects due to the fact that our robot is now a virtual one. Any latency that would have resulted from the many modules communicating between one another is now obsolete, just as potential difficulties with the control system managing the encoders, communication over WiFi, and other unforeseen hardware obstacles are no longer problems. As a result, our comparison is now one of a more tangential nature as there are currently no known projects like our hardware input-based simulation.



Tasks	Mar 23 - 29	Mar 30 - Apr 5	Apr 6 - 12	Apr 13 - 19	Apr 20 - 26	Apr 27 - May 4	May 4 - 11
Motion Sensing							
First attempt at 3D accelerometer and gyroscope Kalman filter	▨						
2D Position and Calibration from Accelerometer		▨					
Reading gyroscope data using I2C		🟦					
Shift to AHRS instead of Kalman			🟦				
Refinement for 3D Positioning			▨	▨	▨	▨	
Determine user rotation, handHeight, and velocities				▨	🟦	🟦	
Simulation							
Inputs and Physics equations	🟧	🟧					
Bird's Eye View graphics			🟩	🟥			
Side View graphics				🟧	🟥		
Ball motion synchronization with bird's eye view ball				🟥	🟩		
Actual throw integration					🟥		
PID Simulator					🟩		
Simulation refinement				🟧	🟨	🟧	
Physical Assembly							
I2C with second IMU				▨			
IMU configuration on hand					🟦		
Final Video and Report							🟨
Team Status Update	🟧	🟦	🟩	🟧	🟦	🟩	
Key:	**All**	**Luca**	**Daniel**	**Hana**	**Luca/Daniel**	**Hana/Luca**	
	🟨	🟩	🟦	🟧	▨	🟥	

Figure 9: Our updated Schedule

Summary

System Validation

IMU Data Verification

Through the MPUs on the user's hand, we are able to determine the velocity and angles that the object is thrown at in three dimension (i.e. X, Y, Z directions). These values are all relative to the starting position of when the glove is turned on, and so several prethrows may be necessary to get an accurate trajectory of the hand's motion. Once the perceived hand motion measurements have stabilized, we are able to obtain the aforementioned values. These data points can be roughly checked through recorded video from the side and above, especially for the angles. It's much easier to estimate and verify the angles than it is to do the same for the velocities from the video as it can be unclear how much distance was travelled during a fixed period of time based on the camera angle.

Madgwick's AHRS algorithm prefers cyclic motion since the mean displacement and velocity of such motion are zero. Deviations from this in the throw pattern may lead to angle and position errors that may be corrected using basic trigonometry. We are able to estimate such errors by throwing the bag as straight in front of the user as possible, and comparing the perceived IMU angle offset to that of the real-life one that should be relatively close to zero.

Simulation Verification

It is a challenge for our simulation to perfectly mimic how our object (a cornhole bag) would fall and how the robot would move in real life. For the bag's projectile motion, it is difficult to take variables such as air resistance into account. For the robot's motion, it is similarly difficult to represent the robot's acceleration and directional changes of the omnidirectional wheels, along with any wheel surface friction, especially since we do not have a real life robot to compare it to. However, despite these difficulties, we are still able to verify the simulation in a very simple way.

With the glove containing the IMUs, the user may throw the cornhole bag onto a marked floor, where the thrower can then measure the approximate landing location on the floor. These (x,y) coordinates will be inputted into the simulation along with the perceived projectile inputs from the final throw. Using the kinematic equations of motion, these inputs should lead to a predicted landing location, given the laws of physics. The predicted location and actual location are represented on-screen, and the robot will move to the predicted location. If the bag has been thrown somewhere a overlap between the two locations which is also within the bounds of the robot net's circumference, and the robot is able to get there in time given the fixed speed we have assigned to it, the simulation will show the robot catching the bag and bringing it back to its starting position. If not, and the robot wasn't able to catch the bag properly, then the bag will remain stationary in the predicted landing spot and the robot will return to its starting spot empty-handed.

Performance Results

In order to describe the system's accuracy within each of several forward distance regions (forward displacements from user), we have divided the three regions into the following ranges: 0m to 0.5m as Range 1, 0.6m to 1.4m as Range 2, and 1.5 to 2m as Range 3.

In Range 1, these distances are often achieved by tossing the bag lightly on the ground. This registers very slight measurements in the IMU so the resulting prediction is often somewhere around 0.1m to 0.2m from the user. This leads to scenarios where the prediction is accurate for throws around 0.1m to 0.2m but the robot doesn't have time to get there since the bag will have almost no vertical velocity. In contrast, the robot has time to get to throws around 0.4m even though the estimate is lower. It'll catch the ball along its path to the lower estimate. It is also hypothetically possible to achieve distances in this range by tossing directly up (giving the robot more time to catch) but our attempts to do this usually landed the thrown object into Range 2.

In Range 2, we have our ideal combination of accurate estimates and time-friendly landing locations. Since the robot is initially positioned at a 1m distance from the user, anything less than 0.4m away from the robot will be easy to catch. In addition, the AHRS prediction system exhibits the greatest amount of accuracy for this location range (roughly 60% success rate) as the IMU measurements respond best when the bag is thrown more vertically than horizontally (often landing in the 0.8m-1.2m range - where most samples come from).

In Range 3, we unfortunately have a set of poor predictions and often time restrictive landing points as we approach 2m. Due to the awkward weight distribution of the cornhole bag, throwing into this region required a rather flat forward throw. In these throws, the IMU measurements tend to respond poorly and fail to increase their estimates in proportion to the additional velocity in the y direction. Throwing closer to the 2m range also makes it harder for the simulated robot to catch the bag on time. Fortunately, most throws with the glove rarely enter this area since it feels much more natural to toss the bag lightly up into the air (often landing in Range 2).

In terms of angular variety, the results line up in terms of accuracy with each aforementioned range. In Range 1, variety in angle doesn't change much in terms of the accuracy as our predictions simply give readings that are too small to ensure a successful catch. In Range 2, our system comfortably handles throw angles from -45 degrees to +45 degrees. This was determined due to the fact that a throw of 1.4m at a 45 degree angle would require the robot to travel ~0.99 m in ~0.8 seconds (the limit of retrieval within the throw range). Practically, throws up to 1.2m at a 45 degree angle (0.85m) performed the best. Finally, in Range 3, our predictions are often too small to reflect throwing that far, but when we are able to get a good prediction, we have seen that variations of -10 degrees to +10 degrees seem to give similar results.

Overall, even though our prediction system struggles within especially short and long distances, our excellent performance in Range 2 from a wide variety of angles allows us to term the fundamental inquiries of this project a success. We have achieved a > 50% success rate in this region along with a ~0.85m horizontal range. Seeking to discover whether IMUs could drive a throw prediction and catch system, we discovered that, with aforementioned limitations, they certainly could.

References

[1] Madgwick, Sebastian, “An efficient orientation filter for inertial and inertial/magnetic sensor arrays,” University of Bristol, April 2010
[2] xioTechnologies. “Oscillatory-Motion-Tracking-With-x-IMU.” GitHub, 1 October 2017, https://github.com/xioTechnologies/Oscillatory-Motion-Tracking-With-x-IMU
[3] InvenSense, “MPU-9250 Register Map and Descriptions Revision 1.4,” RM-MPU-9250A-00 datasheet, September 2013
[4] InvenSense, “MPU-9250 Product Specification Revision 1.1,” PS-MPU-9250A-01 datasheet, June 2016
[5] Free Tutorials. "Understanding MPU6050 ACC full scale range," YouTube, Oct. 8, 2019 [Video file]. Available: https://www.youtube.com/watch?v=e28SHRiJBQY. [Accessed: Feb. 28, 2020].
[6] AddOhms. "Picking Pull-Up Resistor Values | AO #25," YouTube, May. 3, 2018 [Video file]. Available: https://www.youtube.com/watch?v=u3Xiy2DVnI4. [Accessed: March 2, 2020].
[7] Particle, “Photon Datasheet” V016 datasheet
[8] Balzer82. “Kalman.” *GitHub*, 12 March 2018, https://github.com/balzer82/Kalman
[9] bolderflight. “MPU9250.” *GitHub*, 1 May 2020, https://github.com/bolderflight/MPU9250
[10] DanBarychev. “ECE-Capstone.” *GitHub*, 3 May 2020, https://github.com/DanBarychev/ECE-Capstone
[11] A. Rowe, G. Kesden, Class Lecture, Topic: "Lecture 21: Implementing PID Control" 18-349, College of Engineering, Carnegie Mellon University, Pittsburgh, PA, Nov., 2019.