ECE 4973

Tracking Baseball Pitches with CEI's iScan Phantom Radar

Kevin Calhoun, Jessica Vilarino, Joey Orr

Department of Electrical and Computer Engineering
Villanova University, 800 E Lancaster Ave, PA 19085

Advised by: Mark Jupina and George Simmons
Sponsored by: Colorado Engineering Inc.

February 21, 2020

ABSTRACT

This final report provides details and insight for tracking baseball pitches using an iScan Phantom Radar unit and Microsoft Azure Kinect Camera. It contains specifications of how the senior design project was approached including a project schedule, comparison to related projects, potential impact, and more. This report will demonstrate what the final project achieved and how it was designed. In short, a 77GHz Radar connected to a computer was used in conjunction with the Microsoft Azure Kinect Camera to set up a strike zone and track the path of baseball pitches. This information was then used to predict whether the pitch was a ball or a strike.

Acknowledgements

The group is very thankful for all the time and effort that Dr. Mark Jupina and Professor George Simmons have put into this project. Their extensive knowledge about the professional engineering world as well as the technology used in this project was an incredible help to the team. We would also like to thank CEI and its staff, especially Larry Scally, Mike Bonato, Sijia Qiu, and Brian Heflin, for providing us with the opportunity to work with the 77 GHz radar and for all the hours they dedicated to helping us understand the technology.

Table of Contents

1. Introduction

1.1 Overview - Problem, Opportunity, Need - Goals and Objectives

In this project, technology was developed to track the path of a baseball pitch and classify it as a ball or strike using a camera and a radar provided by Colorado Engineering, Inc. (CEI). This system addressed a problem that many baseball players face when practicing their hitting and pitching, which is that they do not know if they are swinging at or throwing a pitch that is a ball or strike. With this device, the player can make changes to their technique given the information about the pitch. The most practical use of this technology is in a practice setting, as a game setting requires a different design to account for other players and factors. Additionally, the system PITCHvr (Perceptual Image Trainer for the Complete Hitter in the virtual realm), developed by Dr. Mark Jupina, tracks a baseball pitch in the virtual realm and identifies if it is a strike or a ball. The system designed in this project allows the players to test if their pitch recognition skills got better in a real situation after using PITCHvr to train. Overall, there is a clear need for technology to help baseball players improve their overall performance.

In the end, the main goal of this system was to be able to accurately track the path of a baseball after it was thrown by a pitcher and classify it as a ball or strike. This was done using an iScan Phantom Radar and a Microsoft Azure Kinect camera. Then, the system was successfully tested in a contained environment in the Multidisciplinary Lab (MDL) at Villanova University.

1.2 Global, Societal, Cultural and Economic Impacts

Using a radar and camera system to track baseball pitches could have a global, societal, and cultural impact because it would change the way baseball is played. While the system is not ready to be implemented in a Major League Baseball setting, the proper layout is there for it to work on a larger scale in the future, impacting the game forever.

For instance, all over sports there are issues with referees making the wrong calls in tense situations that end up changing the outcome of the game and affecting millions of people across the world. The argument could be made that this is a part of the game and that referees have a grasp of the game that computers simply cannot understand. On the other hand, if the automatic umpiring system was used during an actual game of baseball, it could be used as a “perfect” way to call balls and strikes. This type of technology would change the way baseball is played in both a positive and negative way.

In addition, one of the biggest unsolved problems in baseball right now is shortening the length of a game. Using this system to track balls and strikes would eliminate the interactions and arguments that batters, pitchers, catchers, and managers all have with the home plate umpire. The new system would call the game so effectively and uniformly that it would be fruitless to argue with the computer after already knowing the strike zone has been being perfectly monitored all game.

Another significant impact that this system would have is that it would be excellent in helping to train pitchers and hitters during practice. It would allow the pitcher to pinpoint exactly where they need to fix their control or where the hitter can find his sweet spot. Baseball is a game of inches and the type of precision provided by this system could provide all the difference in getting a player to the next level.

Overall, it is clear that this system would have a global, societal, and cultural, and economic impact on the baseball community.

1.3 Ethical Issues related to Public Health, Safety and Welfare and Environmental Impacts

One of the major ethical issues associated with this project is that it may affect the welfare of the baseball umpire. Since the beginning of the sport, one of the main responsibilities of the umpires was to call balls and strikes during a game. While this system is designed to be used in a practice setting, it could be modified to be implemented in ballparks. This change would mean that it would no longer be necessary for the umpire to call balls and strikes. With their role diminished, umpires would likely lose their jobs. This technology would improve the accuracy of the ball and strike calls, but it would also have a negative impact on the welfare of umpires.

Another ethical issue that could impact public health, safety, and welfare is that the 77GHz is not approved by the FCC to be used for the specific purpose of this project. However, it was necessary to use this device in the design because the 77GHz radar is more developed than the 60GHz radar that CEI had also built. Since the application of the system is at longer distances, it was determined that using the 77GHz radar was safe.

If the proposed system malfunctioned while being used in a practice setting, it would not jeopardize public health and safety. The only possible consequence is that the batter or pitcher would receive inaccurate data about the pitch. While this is annoying and could hinder the development of a player, there are not any dire consequences to a system malfunction in a practice setting. Also, if this was being used in a game atmosphere as a replacement for umpires, a malfunction could impact the game because play would have to stop until the system was fixed or an umpire could be found. However, in both of these scenarios, no one's safety would be in jeopardy.

Finally, one great thing about this project is that it does not have a large impact on the environment. The only element that could cause an environmental issue is the 3D printed plastic case for the radar and the plastic shell of the baseball. If this system were to be mass produced, it would be in the best interest of the environment to use a different material for these two items. All in all, this is a very environmentally safe product.

1.4 Regulatory Issues

The radar unit for this project was designed to detect automobiles and pedestrians, so it is generally considered safe for most applications. The FCC's Office of Engineering & Technology (OET) has stated that power levels should be limited to under 1 mW/cm^2 when operating radar at a frequency of 77 GHz [1]. CEI's unit operates at 0.0997 mW/cm^2 which is well below this threshold. Additionally, the radar is giving off non-ionizing radiation, which humans are exposed to low levels of everyday. According to the CDC, the only risk of non-ionizing radiation is heat damage to tissue which only occurs when exposed to intense and prolonged amounts [2]. This will not be the case for this project.

1.5 Prior Work

As one of the most popular sports leagues in the world, Major League Baseball (MLB) has been at the forefront of many of the pitch tracking technologies that have been developed in the last decade. One of the first instances of the use of pitch tracking in the MLB was the PITCHf/x system, which was widely implemented beginning with the 2007 regular season [3]. PITCHf/x utilizes two high-speed cameras, in the stands above first base and home plate, which image the ball on its path between release and the plate. From this information, PITCHf/x calculates with reasonable accuracy the velocity, acceleration, and "movement" of the pitch

which it uses to classify the type of pitch and location in or out of the strike zone. In addition, it only uses the data starting from 40 feet away from the batter as opposed to data from the release [4]. This is done because there is less break over that distance, and it is more comparable to what the batter will see from home plate.

While PITCHf/x is still used by announcers and fans, recently, the MLB has been updating its ballparks by implementing TrackMan, which the league uses for their official data and to evaluate umpires. TrackMan is a radar system which sits high behind home plate and measures the speed, spin rate, and spin axis of a pitch [5]. In a way, TrackMan is the opposite of PITCHf/x as it directly measures these parameters at release and uses this data to determine the location, path, and outcome of the pitch. TrackMan is often seen as the more beneficial system as it gives a more complete picture of the details of the pitch and can also be used to track bat speed, batted balls, and player movement as well.

Additionally, there have been several private companies that have designed their own products that they market as practice tools that can benefit the players. One of these companies is Rapsodo. Rapsodo's device utilizes both a camera and a radar unit that they mount behind home plate. Although much of the specific information regarding the Rapsodo device is not public, it uses this setup to track the spin, trajectory, and final location of a pitched ball [6].

The system designed for this project was influenced by the techniques used in the TrackMan and Rapsodo technology because both of the systems use a radar to track the pitches.

1.6 Organization of the Proposal

The remainder of this report details the specifics of the design such as what the system does, how it works, and the results that were obtained. It also delves into the schedule of the project, what each member of the team contributed, and the total estimated budget. Lastly, it includes conclusions that were drawn about the final product and appendices that outline various programs used to implement the final design.

2. Specifications

2.1 Functional Specifications

Functional Specifications:

- Easy to use Microsoft Azure Kinect Camera set up across from batter
- 77 GHz iScan Phantom Radar fixed on a tripod in front of the pitcher
- Straightforward connection between devices
- Data from pitches accessible during session
- Clear display of pitch track and strike zone on computer

2.2 Performance Specifications

Performance Specifications:

- Radar [9]
    - 77 GHz Frequency
    - 2.0 GHz Bandwidth
    - Antenna gain: 16.3 dBi

    - Digital beam forming
    - 3 Transmission Antennae
        - 9x1
    - 4 Receiving Antennae
        - 9x1
    - 20 frames/second
    - Az: ±33°, El: ±5°
    - Connected to computer via Ethernet
- Microsoft Azure Kinect Camera [8]
    - Body Tracking SDK
    - Up to 30 fps
    - Coordinate System of Depth Camera
        - In relation to the camera:
            - +x direction is to the right
            - +y direction is down
            - +z direction is forward
                - All measurements captured in millimeters
    - Connected to computer via USB 3.0
- Baseball Being Used
    - Design increases the RCS of the ball
        - Trihedral reflector inside
        - Ball weighs the same as a baseball (5oz)
        - Leather wrapped around 3D-printed Nylon surface of ball
        - Inside was filled in order to provide enough stability to withstand multiple throws at high velocity without cracking
- Capture Zone
    - Approximately 6 meter pitch
    - Capture position to predict velocity and acceleration
- Strike Zone Metrics
    - Captured using Body Tracking SDK of Microsoft Azure Kinect Camera
    - Typical Measurements of Strike Zone:
        - Length: 19.94 inches
        - Height: Determined by Camera
        - Width: 16.97 inches
    - Key Joints Captured: Right and Left Knee, Spine - Naval, Spine - Chest, Right and Left Foot, and Head

3. Design Overview

3.1 Overview and Preliminary Ideas

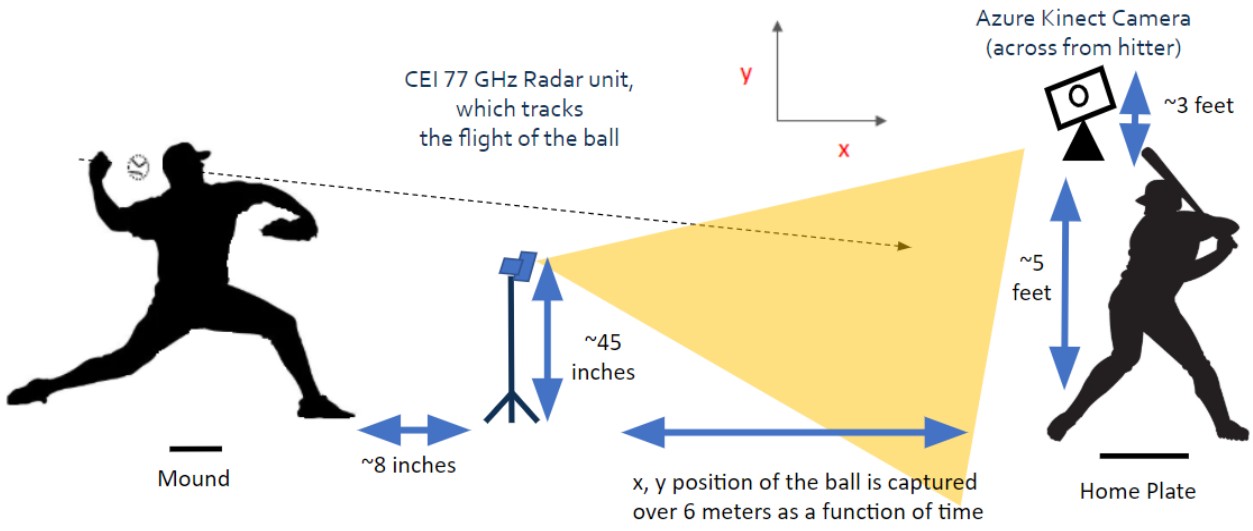


Figure 1: Visual Overview of System

The diagram above provides a visual overview of the baseball tracking system. The 77GHz radar was placed about 45 inches above the ground and about 8 inches in front of the pitcher on a tripod to capture the flight of the pitch. The radar captured the x and y position of the ball over 6 meters, which allowed the system to gather enough data to track the path of the pitch. Also, the Microsoft Azure Kinect Camera was set up about 5 feet across from the batter and about 3 feet off of the ground to accurately capture each batter's unique strike zone. Each aspect of the system will be discussed in more detail later in the report.

Before getting into the intricacies of the system, it is important to go over some of the ideas for the system that were eliminated earlier in the design process. First, classifying the type of pitch that was thrown by the pitcher was discussed. However, it was determined that tracking the type of pitch would require a high-speed camera to pick up the seams and spin rate of the ball as it was travelling to home plate. That type of camera was way too expensive, so the scope of the project was narrowed to just concentrate on tracking the path of the baseball and determining if it was a strike or if it was a ball.

Another idea that was considered during the design process was connecting the camera to a device called the TK-1 for GPU processing. This idea was considered because the TK-1, another device developed by CEI, could be used to process the information from the camera to create the strike zone. However, because integrating the device into the system proved to be a timely and complicated task, the team decided to connect the camera directly to the computer in the MDL to do the graphics processing because it contained a Nvidia 1080 Graphics card.

Next, a more detailed view of the final design will be discussed.

3.2 Final Design

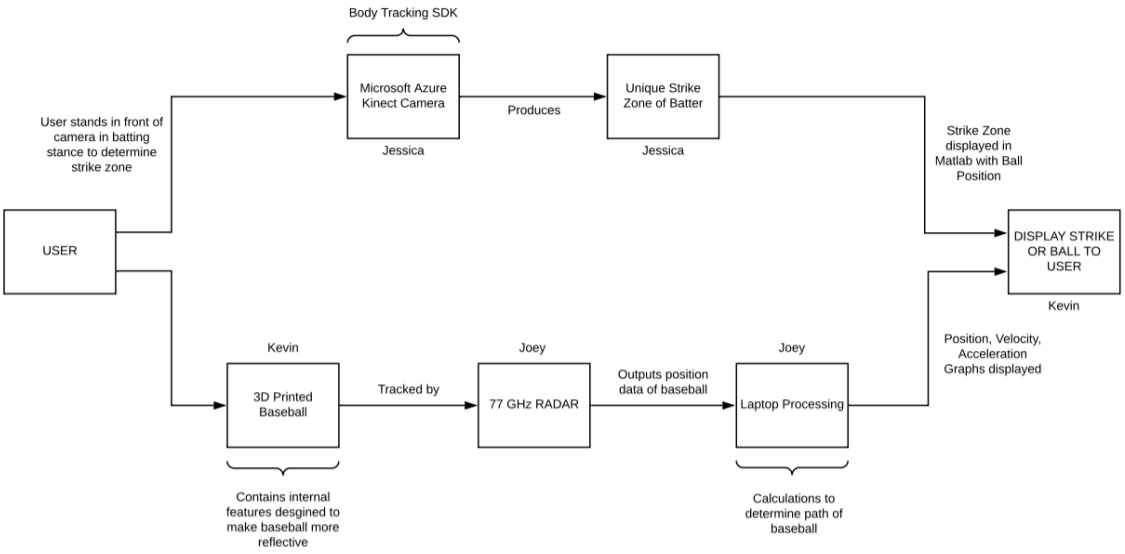


Figure 2: System Block Diagram

The system block diagram, which can be seen in the figure above, outlines the final design for the system. The upper path of the diagram shows the steps that were taken to produce the unique strike zone of the batter, while the lower path describes how the path of the pitch was tracked.

First, the strike zone was created. This was done using the Microsoft Azure Kinect Camera and its Body Tracking SDK, which identified the positions of the key joints of the batter. Analyzing the positional data produced the unique strike zone's height and its height off the ground. This was necessary to determine because a typical strike zone spans from the knees of the hitter to about halfway between the navel and chest of the player, which is different for everyone. The length and width of the strike zone remained constant at 19.94 inches and 16.97 inches, respectively, because these are the measurements of home plate. Using the data from the camera, the correct alignment of the strike zone was identified.

In order to track the pitch with the radar, a 3D printed baseball was created to make the ball more reflective for the radar. Next, the 77 GHz radar tracked the ball when it was thrown and output the positional data in the x and y directions over its flight [7]. After that, additional code on the laptop performed calculations to predict the flight path of the ball. These calculations included predicting the velocity and acceleration, which remained constant, of the ball based on the positional data [7]. Then, a line of best fit was created to show the path of the ball.

With the strike zone and the path of the ball determined, the pitch was classified as a ball or a strike and displayed to the user. A graphic of the strike zone, created in Matlab, was shown with the word BALL or STRIKE on the screen. If the pitch was a strike, the ball appeared in the box of the strike zone. If it was a ball, the ball appeared outside the strike zone.

While this system is intricate and complex, this design fulfilled all the objectives outlined previously.

4. Subsystem Designs

4.1 The Baseball

Subsystem Specifications:

- Improve reflective properties of baseball for radar
- Maintain same size and weight as typical baseball

The first part of the design to discuss is the design of the baseball. The radar was very sensitive and also sometimes struggled to detect the baseball due to the very small size and strange surface. In order to help the baseball show up better on the radar, a new baseball was designed with a trihedral reflector inside. Thanks to the help of Professor Simmons, the group was able to 3D print a "baseball" that was the identical size (9-inch circumference) and weight (5 oz). Inside the baseball tin sheets were used to create a trihedral reflector so as the ball spun it would be able to reflect the waves sent by the radar. This was created by tracing a circle around a paper cup and then cutting those pieces with tin cutters. Finally, the skin of a real baseball was cut off and placed around this 3D printed ball and super-glued closed. Below are the images of the process to create this unique baseball.

Image 1: Materials used for the Baseball

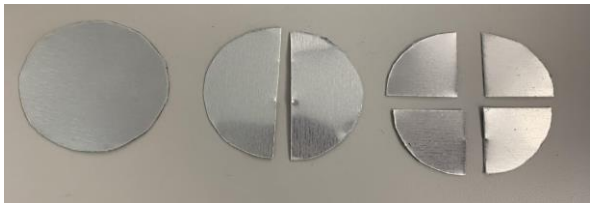


Image 2: Tin cut into full circle, semicircle, and quadrants

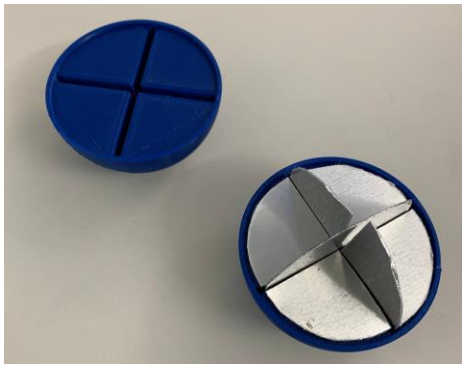


Image 3: Tin sheets placed in trihedral reflector

4.2 Capturing the Strike Zone with the Microsoft Azure Kinect Camera

The next part of the system focused on capturing a batter's unique strike zone with the Microsoft Azure Kinect Camera. Before beginning the design phase of this subsystem, the following specifications were identified.

Subsystem Specifications:

- Easy-to-use camera with the capability of tracking the joints of a batter
- Batter's unique strike zone created based on the positional measurements of their knees, chest, and naval
- Ability to capture a catcher simultaneously without affecting the batter's measurements
- Strike zone measurements triggered when the foot of the batter is lifted
- Positional data capable of being written to Matlab for a 3D display the strike zone

In order to meet all of the subsystem's specifications, it was essential to begin with designing the hardware setup of the system. First, the Microsoft Azure Kinect Camera was selected to capture the measurements of the batter because it included a Body Tracking SDK, which was capable of easily tracking the position of a person's joints. Then, it was determined that the camera should be positioned about 3 feet off the ground and about 5 feet away from the batter in order to collect the most accurate positional measurements. Additionally, the camera was connected via USB 3.0 to the computer in the MDL at Villanova because it contained a Nvidia 1080 Graphics Card, which was sufficient for the graphics processing. Setting up the subsystem in this manner was essential to obtaining reproducible and accurate results.

Before settling on the final design of the hardware setup, alternative setups were considered. In particular, using OpenPose, a software that also has the ability to track the joint positions of a person with a digital camera, was discussed at the beginning of the design phase. However, this idea was abandoned due to the ease of use of the Microsoft Azure Kinect since the tracking software was already compatible with the camera.

After settling on the design of the hardware setup, a design for the software component of the subsystem was created. It was decided that preexisting code from the Microsoft Azure Kinect website would be used to control when the camera was turned on and off in the code. Then, the code needed to process the frames was added. This included a trigger that began the collection of the joint measurements for the strike zone when the batter lifted their foot, which indicated that the ball is on its way to the plate. Once this was picked up by the camera, the code then collected 30 frames (1 second) worth of the x, y, and z positional data for the following joints: the right and left knee, spine - naval, spine - chest, right and left foot, and head of both the batter and catcher. The team considered having capability to track only a batter, but it was decided that it was necessary to include the catcher because it mimicked the circumstances of the practice situation that this system would be used in. Next, the averages of these measurements over the 30 frames were calculated for the batter only. Finally, the averages were used to calculate the height of the unique strike zone and the height of the strike zone off the ground.

After settling on the design, the implementation phase began. The first piece of code that was written was the trigger, which can be seen in the figure below.

```
if (headyinitial < 0)  // look only at batter foot
{
    float lfootx = skeleton.joints[15].position.xyz.x;
    float lfooty = skeleton.joints[15].position.xyz.y;
    //track foot height values with array
    lfoot[frame_count] = lfooty;
    //activate the trigger to collect other values if plant foot is coming down
    if (lfoot[frame_count] > (abs(lfoot[frame_count - 5] + 200)))
    {
        trigger++;
    }
    float lfootz = skeleton.joints[15].position.xyz.z;
    printf("Left Foot: \n");
    printf("X: %f\n", lfootx);
    printf("Y: %f\n", lfooty);
    printf("Y: array value %f\n", lfoot[frame_count]);
    printf("Y: array value -5 %f\n", lfoot[frame_count - 5]);
    printf("Z: %f\n", lfootz);
    printf("Trigger = %f\n", trigger);
}

if (trigger > 1)
```

Figure 3: Implementation of the Trigger

The trigger for the camera was implemented using an array called lfoot, which held the height values of the left foot for each frame. Then, the code compared the height value of the left foot in the current frame to the value of the left foot in a previous frame. If this value showed a significant change in height, the trigger was incremented. Otherwise, the code loops back to get another frame of data.

Once the code was triggered, the joint measurements were collected. In order to find an average of these measurements over 30 frames, a running total for the batter and catcher was calculated during the data collection using the method seen below. Then, after the data collection was complete, the average was computed over the 30 frames just for the batter's data. The key formula used to obtain the height of the strike zone was subtracting the chest average from the knee average, since the positive y axis of the camera was oriented downwards. Also, the formula used to find the height of the zone off the ground was footaverage - kneeaverage. Using this code, the camera was able to accurately determine the measurements for the batter's unique strike zone.

```
if (heady < 0)   // above camera --> batter
{
    totallfooty = totallfooty + lfoot[frame_count];
    totalrfooty = totalrfooty + rfooty;
    totallkneey = totallkneey + lkneey;
    totalrkneey = totalrkneey + rkneey;
    totalnavaly = totalnavaly + navaly;
    totalchesty = totalchesty + chesty;
    //able to track catcher
    totalheady = totalheady + heady;
}


if (heady > 0)    //below camera --> catcher
{
    totalchesty2 = totalchesty2 + chesty;
    totalnavaly2 = totalnavaly2 + navaly;
    totalrkneey2 = totalrkneey2 + rkneey;
    totallkneey2 = totallkneey2 + lkneey;
    totalrfooty2 = totalrfooty2 + rfooty;
    totallfooty2 = totallfooty2 + lfoot[frame_count];
    totalheady2 = totalheady2 + heady;
```

Figure 4: Running Totals

```
lfootaverage = totallfooty / 30;
rfootaverage = totalrfooty / 30;
lkneeaverage = totallkneey / 30;
rkneeaverage = totalrkneey / 30;
navalaverage = totalnavaly / 30;
chestaverage = totalchesty / 30;
headaverage = totalheady / 30;
kneeaverage = (rkneeaverage + lkneeaverage) / 2;
upperaverage = (chestaverage + navalaverage) / 2;
footaverage = (rfootaverage + lfootaverage) / 2;
zone_height = (kneeaverage) - (chestaverage);
```

Figure 5: Joint Averages

The testing that was done to ensure that the strike zone measurements were accurate was a fairly straightforward process. First, actual measurements for the joint positions and strike zone were collected. For instance, Jessica Vilarino's zone height was measured to be 610 mm and the height of her zone off the ground was measured to be about 400 mm. Then, the code was run while the camera captured Jessica as the batter and Kevin as the catcher. Finally, simple comparisons were made between the expected values and the generated values that were written to a text file. Because the generated measurements, 673 mm for the height and 343 mm for the

height off the ground, were extremely close to Jessica's actual measurements, it was clear that the zone measurements were correctly collected and calculated for the batter.

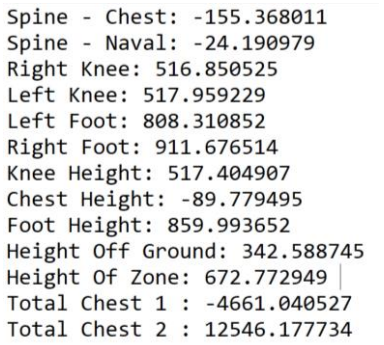


Figure 6: Camera Measurements

The last phase of testing required checking that the calculations for the strike zone began when the batter raised their foot. This consisted of looking at the print statements before the trigger and after the trigger and running the code. When running it was easy to see that when the foot was lifted, the code switched from printing the trigger information to printing the measurements for the joints. Because the measurements were still consistent, it was clear that the entire subsystem was working as expected.

After testing the camera subsystem, it was clear that it was able to accurately capture the measurements of the batter's unique strike zone with a catcher present as the batter lifted their foot. One way to expand the capabilities of the subsystem would be adding additional code to allow the tracking of an umpire as well, so this could be used in a real game situation. Also, offloading the graphics processing to another smaller device would aid in mobility. Overall, designing, implementing, and testing this subsystem was achieved.

4.3 Tracking the Pitch with the 77GHz Radar

Subsystem Specifications:

- Easy to use radar unit capable of isolating and tracking the baseball
- Captures the x and y positional data of a pitch over 6 meters

While CEI provided the group with some of their basic example code to get us started, the general purpose of this code was only to showcase the most basic functionalities of the radar. However, the team did use these Matlab scripts as the starting point for the detection of the ball. Specifically, a script was used that converted range and angle data to rectangular coordinates using an FFT. Starting with this script, all static objects in the radar's scene were zeroed out so that it would make it easier to pick up the moving ball. This was done by subtracting the data in the first frame that the radar picked up from every other frame during detection. This essentially provided a blank slate that would pick up any new objects that appeared in the scene while the radar was receiving data.

From there, the ball was picked up by searching for intensity values within each frame. Each frame would return three 44 x 256 arrays, representing x-coordinates, y-coordinates, and intensity of the image. If the radar picked up an object above a certain intensity threshold tested for the ball, then it would recognize that the ball was present in the current frame. To get the x-

and y-coordinates for the position of the ball, an average was taken for each point that exceeded the intensity threshold in that particular frame. Multiple coordinates were recorded, giving a relatively accurate approximation of the ball's trajectory by providing a singular coordinate in the x-y plane for each point. Time data was also recorded to use in future calculations.

After these points were recorded, steps were taken to extend the maximum range of the radar without sacrificing too much velocity resolution. This was done by manipulating equations for the range, velocity, and angle measurements.

$Max\ Range = \frac{f_{IF} T_{ramp} c}{2B}$	$Range\ Res = \frac{c}{2B}$
$Max\ Velocity = \frac{\lambda}{4T_{chrip}}$	$Velocity\ Res = \frac{\lambda}{2T_{frame}}$
$Max\ Angle = \sin^{-1}\frac{\lambda}{2d}$	$Angular\ Res = \frac{\lambda}{Nd\cos\theta}$

Equation 1: Radar Equations

All the variables in these equations were parameters that could be manipulated in Matlab. Most significantly, the bandwidth and the chirp profile were modified to increase the maximum range. However, given the RCS of the ball being tracked, the range was limited to around 6.5 m. The parameters are listed below.

B = 2e9 Hz
f_{IF} = 6.25e6 Hz
T_{ramp} = 1.1392e-4 s
T_{chirp} = 3.9043e-4 s
T_{frame} = 3.9043e-4 s
Figure 7: Radar Parameters

Another part of this process was determining where to position the radar. Initially, it was proposed that the radar be set up behind the pitcher. However, given the range limitations, it was determined that it would be better to place the radar in front of the pitcher to pick up the maximum amount of points. Also, capturing pitches at higher speeds around 80 mph was proposed. Unfortunately, because the frame rate of the radar was determined to be 20 fps, only slower pitches were tracked.

Using this setup, approximately 4-5 points along the path of a 6-meter toss traveling at approximately 10 m/s could be picked up consistently.

4.4 Determining the Path and Location of the Pitch

Subsystem Specifications:

- Simple display of the path of the pitch
- Predicts velocity and acceleration of pitch from the positional data
- Estimates the final position of the pitch as it crosses home plate
- Straightforward representation of strike zone and pitch location

- Clear “STRIKE” or “BALL” call on display

In order to analyze the data from the pitch, a separate file was made to visually demonstrate what was happening with the ball. This file showed not only positional data but also velocity and acceleration. The end goal of all these graphs was to find the final projected position of the ball and map it onto the strike zone captured by the Azure Kinect Camera.

The first step taken in getting data from the radar was to store all the x, y, and time values in an array. Since the radar only works in two dimensions, unfortunately, the horizontal position across the plate could not be retrieved, only the vertical and the distance from the radar were obtained. Once these values were stored in the array, they were then put into a function written in another file to create all of the graphs.

To start the function, the x and y values are displayed to show the position from the radar. From those points, a line of best fit was also displayed in order to have a clearer image of the true path of the ball. Getting a clear path was very difficult because of the importance of the set of positional values. Since velocity and acceleration are found through taking derivatives, it is critical to make sure that the first set of data points is as accurate as possible. The problem arose from the radar detecting objects outside of just the baseball. If there was a data point in there that detected the floor or someone’s arm, it would skew the data of the baseball. Therefore, to prevent this inaccurate data, multiple rules were put into the radar file to filter out all the incorrect data points and find the points that were almost certainly the baseball. This was a very difficult process to find all the different scenarios of the flight of the ball and eliminate the outliers. Finally, though, a consistent set of rules allowed for the correct data to be sent into the function. An accurate graph of these positional values looks like a steadily declining curve, as shown in the image below.

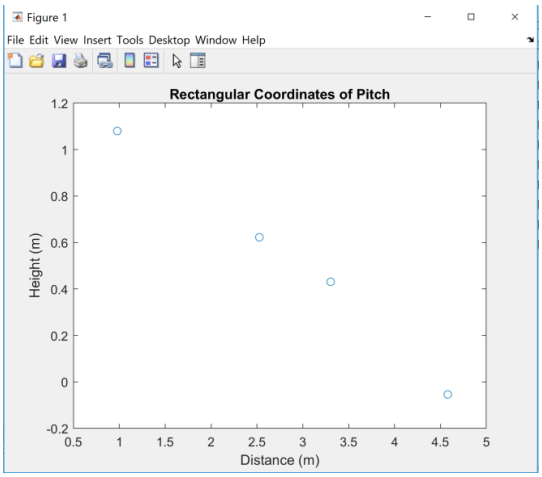


Figure 8: Rectangular coordinates of pitch

Next, the x and y positions over time show the steady increase in distance from the radar in the x direction and the decline due to gravity in the y.

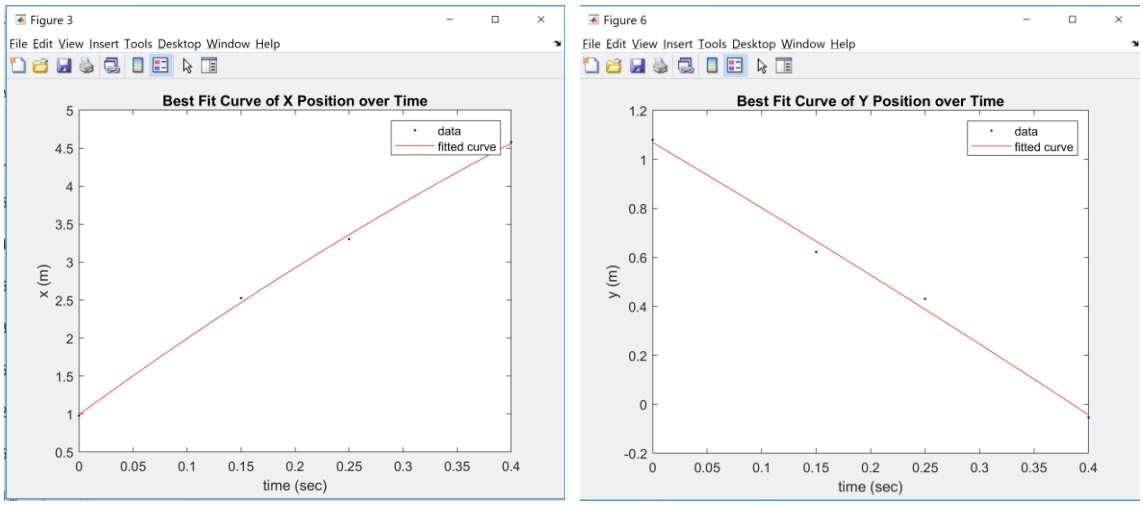


Figure 9: X and Y position over time

The next graphs that were made were the x and y velocity, which were taken using the derivative function in Matlab and each of the positional arrays with the time array. First, the x velocity (velocity moving away from camera) looked like a steady decline, which is exactly what was expected. There is drag on the ball through the air which reduces its velocity in the x direction. In fact, in the MLB a pitch starts out around five mph faster at release than when it finally reaches home plate. This system's method of pitch tracking was very different though, with a different material of the ball, a much slower speed at release, and a much shorter distance to track. Due to this, it didn't create as much of a drag effect as expected.

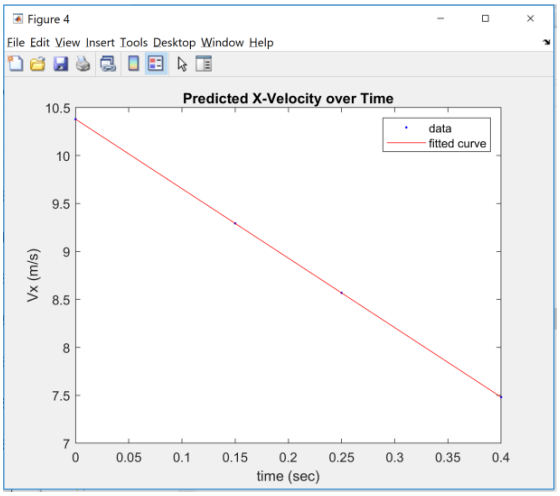


Figure 10: Velocity of pitch in the X direction

After this, the y velocity was found through the same manner of taking the derivative of the positional data. This graph was similar to x velocity in that it steadily declined but for a different reason. The y velocity was more affected by gravity than by drag. It is for that reason that it sped up in the negative y direction.

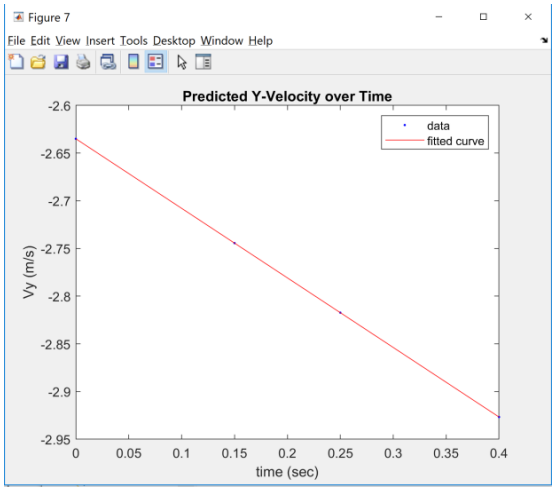


Figure 11: Y velocity over time

The final step of the process consisted of calculating the final position of the ball as it crossed home plate. This was determined by calculating the time the ball crossed home plate with the kinematic quadratic equation seen below using the velocity and acceleration estimates.

$$t = \frac{-v_o \pm \sqrt{v_o^2 - 4\left(\frac{1}{2}a\right)(-d)}}{2\left(\frac{1}{2}a\right)}$$

Equation 2: Calculating Final Time of Pitch [10]

Once this number was determined, the time was inserted into the line of best fit equation for the path of the ball to obtain its final position.

Using all of this data as well as the data from the Azure Kinect Camera, the group was able to map the projected position of the ball at six meters away to the unique strike zone of the batter.

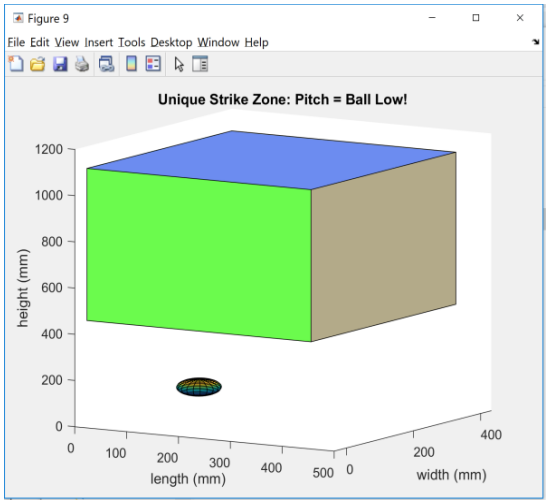


Figure 12: Pitch in Relation to Strike Zone

5. Project Management

5.1 Schedule and Milestones

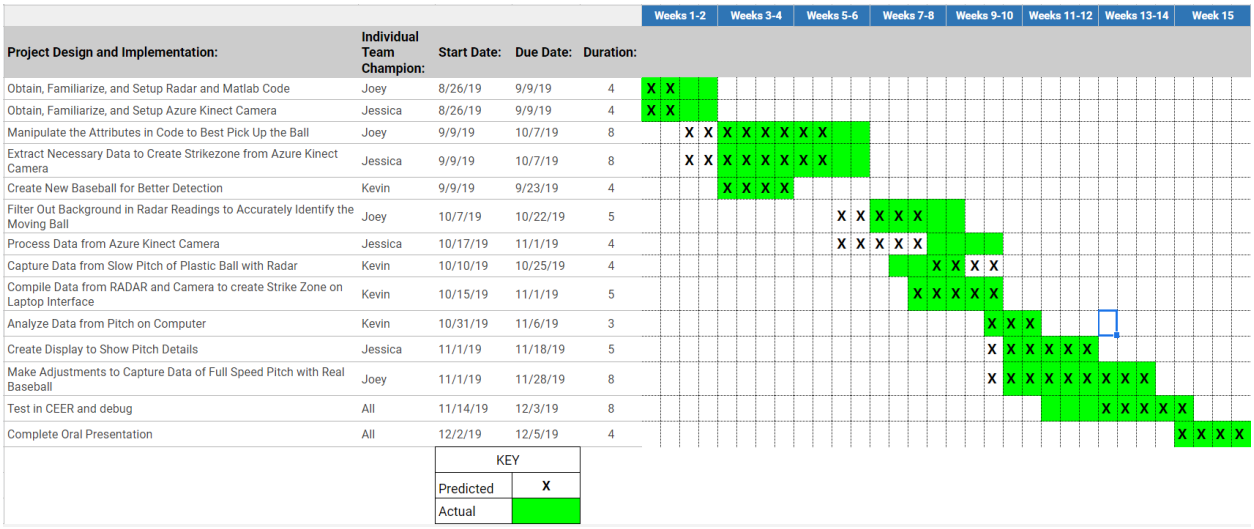


Chart 1: Full Timeline

Project Design and Implementation:	Individual Team Champion:	Start Date:	Due Date:	Duration:
Obtain, Familiarize, and Setup Radar and Matlab Code	Joey	8/26/19	9/9/19	4
Obtain, Familiarize, and Setup Azure Kinect Camera	Jessica	8/26/19	9/9/19	4
Manipulate the Attributes in Code to Best Pick Up the Ball	Joey	9/9/19	10/7/19	8
Extract Necessary Data to Create Strikezone from Azure Kinect Camera	Jessica	9/9/19	10/7/19	8
Create New Baseball for Better Detection	Kevin	9/9/19	9/23/19	4
Filter Out Background in Radar Readings to Accurately Identify the Moving Ball	Joey	10/7/19	10/22/19	5
Process Data from Azure Kinect Camera	Jessica	10/17/19	11/1/19	4
Capture Data from Slow Pitch of Plastic Ball with Radar	Kevin	10/10/19	10/25/19	4
Compile Data from RADAR and Camera to create Strike Zone on Laptop Interface	Kevin	10/15/19	11/1/19	5
Analyze Data from Pitch on Computer	Kevin	10/31/19	11/6/19	3
Create Display to Show Pitch Details	Jessica	11/1/19	11/18/19	5
Make Adjustments to Capture Data of Full Speed Pitch with Real Baseball	Joey	11/1/19	11/28/19	8
Test in CEER and debug	All	11/14/19	12/3/19	8
Complete Oral Presentation	All	12/2/19	12/5/19	4

Chart 2: Task List with Schedule

Above was the schedule for the project last semester. The first task was to obtain and become familiar with the radar and camera, which took about 2 weeks. Joey worked with the radar first because some students over the summer had experience with it. However, they were not able to provide much valuable information for this project's specific needs. During this time,

Jessica worked with the Microsoft Azure Kinect camera and figured out how to control the camera with the code. Both tasks were predicted to initially take about 1 week, but an additional week was needed due to the team's unfamiliarity with the devices.

Then, the next 4 weeks were spent manipulating the radar attributes to pick up the baseball and extracting positional data from the camera. Both tasks were initially allotted 4 weeks, however, work began 1 week late due to the previous setback when setting up the devices. As Joey and Jessica were working on these tasks, Kevin created the baseball on schedule.

Next, Jessica used the camera data to create a strike zone in Matlab for the batter. During that time, Joey worked on filtering out the background the radar picks up so that it only captured the data from the baseball and Kevin captured a slow pitch with the radar. The first two tasks were completed in the 2 weeks allotted, but were, once again, started later than anticipated. However, Kevin was able to begin his task earlier than originally scheduled. This was because he was able to work on this at the same time as combining Joey's radar code and Jessica's Microsoft Visual Studio code to detect if the pitch was a strike.

Finally, Joey and Jessica began their last tasks, which consisted of creating the pitch path display and increasing the accuracy and range of the radar. Both were completed on time. This final system was tested in CEER until the group was able to get a consistent pattern of data over multiple attempts. Finally, the group presented the project to multiple advisers.

5.2 Personnel

Team Member	Responsibilities
Jessica Vilarino	● Captured joint data from Microsoft Azure Kinect Camera ● Modeled lines of best fit to ball trajectory ● Created 3D model of strike zone in Matlab ● Designed filters to get rid of unwanted radar data ● Timeline Management
Joey Orr	● Controlled radar parameters to get best possible range and velocity data ● Extracted x vs. y positional data for ball from radar ● Designed filters to get rid of unwanted radar data
Kevin Calhoun	● Created 3D printed baseball with reflectors and leather covering ● Mapped location of ball onto strike zone model ● Designed filters to get rid of unwanted radar data

Table 1: Bulleted lists of roles for each team member

Jessica was assigned the task of capturing the joint data from Microsoft Azure Kinect Camera due to her extensive coding knowledge and history of coding with C in Microsoft Visual

Studio. In addition, she worked on modelling the lines of best fit to the trajectory of the ball and creating a 3D model of strike zone in Matlab because of her familiarity with creating graphs and figures in Matlab. Also, she aided in designing the filters for the radar data. Lastly, she managed the timeline, making sure the team met all deadlines, due to her organizational skills.

Joey was assigned duties pertaining to capturing data with the radar and analyzing it due to his hardware background. To achieve this, he adjusted the radar parameters to get the best possible range and velocity data and extracted the x and y positional data for ball from radar. Finally, he also helped design the filters needed to obtain accurate radar data in Matlab.

Kevin was assigned the task of creating the new 3D printed baseball with a trihedral reflector due to his past experiences with 3D printing. Also, he took on the role of mapping the location of baseball obtained from the radar onto the strike zone model due to his experience with both Matlab and C. Lastly, he also contributed to designing the filters that were used to obtain accurate radar data in Matlab.

5.3 Key Parts & Components

- 77GHz iScan Phantom Radar Provided by CEI
    - 3D Printed Case
    - Matlab Code (See Appendix)
- Azure Kinect Camera
    - Microsoft Visual Studio Code (See Appendix)
- Computer with proper GPU (provided in MDL)

5.4 Skills and Tools

- Baseball
    - 3D printed shell and trihedral reflector
- Radar
    - 3D printed case
    - Matlab
        - Radar code provided by CEI
- Camera
    - Body Tracking SDK
        - Microsoft Visual Studio
        - Code in C
- Final Display
    - Matlab

Both software and hardware skills were necessary to complete this project. On the hardware side, a 3D printer was used to create a shell and trihedral reflector when designing the baseball. A case for the radar was also produced.

In terms of the software used for the project, all of the code controlling the radar operation was done in Matlab. This was because CEI provided the team with preexisting code for the radar in Matlab.

For the Microsoft Azure Kinect Camera software, the Body Tracking SDK was used to track the position of the joints. This required the use of Visual Studio and coding in C to control the camera.

Lastly, the final display of the pitch track and strike zone was created in Matlab.

5.5 Budget

Equipment	Quantity	Cost
TK-1 SOM 4GB	1	Donated by CEI ($429.00)
77 GHz Radar	1	Donated by CEI ($5,490.00)
Microsoft Azure Kinect Camera	1	$399.00
Tripod 72”	2	$26.99
Rawlings Official League Baseball	3	$8.94
3D Print Material	500 g	$8.00
Labor - Joseph Orr ($25/hr)	100 hours	$2,500
Labor - Jessica Vilarino ($25/hr)	100 hours	$2,500
Labor - Kevin Calhoun ($25/hr)	100 hours	$2,500
TOTAL	N/A	$7,942.93

Table 2: Detailed budget with quantities and cost of each item

Because of the CEI sponsorship of this project, the budget was kept reasonably in check. The expenses for the other equipment that were not provided by CEI were all within the budget. The most significant parts of the budget were the Microsoft Azure Kinect Camera, which was very useful, and the labor hours that were spent working.

Overall, it is estimated that each member of the group spent a total of 100 hours working on this project. This includes preliminary work done during the spring of 2019, technical work done in the fall of 2019, and reporting done in the spring of 2020. The majority of these hours came during the fall of 2019. Each individual spent approximately 5 hours a week on the project which totals 75 hours for the entire 15-week semester. Adding 10 hours from the spring of 2019 and 15 hours from the spring of 2020 brings the total time spent on the project to 100 hours.

6. Executive Summary of Achievements and Deliverables

Creating a unique strike zone for each batter with the Microsoft Azure Kinect Camera worked just as expected. The calculations done to create the strike zone were accurate and consistent with the actual measurements collected. Additionally, triggering the calculations based on a batter's leg kick and tracking both a batter and a catcher at the same time worked just as expected without impacting the measurements and calculations.

In the initial design of the system, a device called the TK-1 was supposed to be used to offload the processing of the camera code since the TK-1 contained a GPU. Unfortunately, this was not accomplished due to a myriad of hardware issues discovered throughout the semester. Due to continuous issues stemming from re-flashing the device, the team went through three different TK-1 devices from CEI before finally finishing the setup of the device on a computer. However, this was only a week prior to the end of the semester. Therefore, the team was unable to accomplish the original goal of integrating the TK-1 with the Microsoft Azure Kinect Camera.

Lastly, many aspects of the radar performed well despite the various constraints that were found. First, the original CEI code was modified to successfully filter out existing still objects to only look at moving ball. Also, the x, y, and time arrays were efficiently passed into an "LSFit" function, which returned the projected final height of the pitch. Next, the team successfully used the radar data to project lines of best fit for the x and y position as well as the velocity and acceleration. Then, outliers that didn't fit the pattern of the ball flight were successfully filtered out of the graphs. Unfortunately, the ball could only be tracked in two dimensions. Also, increasing the range of the radar affected the velocity resolution, which is why higher speed pitches were not tracked. Another limit discovered was that the radar could not obtain the actual velocity and positional data simultaneously. Finally, the team was not able to increase the sampling rate of the radar by making executables and C code copies.

7. Conclusions

All in all, the system that was designed met the main goal of the project, which was to accurately track the path of a baseball pitch and classify whether it was a ball or strike. Therefore, the team believes that this system could be used as an effective way for baseball players to train and improve their strike zone recognition. Although a few issues arose during the design process, including not being able to track high velocity pitches as well as not being able to implement the TK-1 to process the data from the camera, the team was still able to create a system that worked as initially intended. In the future, increasing the frame rate of the radar would allow for tracking faster pitches with increased accuracy. Also, given more time, the TK-1 could be added to the system to process both the camera and radar. Overall, this project was a success because each team member worked effectively and divided the work evenly. The group was able to meet multiple times every week and make progress to bring this project to its full potential.

8. References

[1] *Evaluating Compliance with FCC Guidelines for Human Exposure to Radiofrequency Electromagnetic Fields*, OET Bulletin 65 Edition 97-01, 1997.
[2] CDC, *The Electromagnetic Spectrum: Non-Ionizing Radiation*, December 7, 2015. Accessed on: January 26, 2020. [Online]. Available: https://www.cdc.gov/nceh/radiation/nonionizing_radiation.html
[3] M. Fast, "What the Heck is PITCHf/x" in *The Hardball Times Baseball Annual 2010*, Chicago: ACTA Publications, 2010.
[4] Cross, Rodney. *Physics of Baseball & Softball*. New York: Springer, 2011.
[5] A. Nathan, "Tracking the Baseball with the TrackMan Doppler RADAR," in *Saberseminar*, Boston, 2012.
[6] Rapsodo, "Pitching 2.0," Rapsodo Pte Ltd. [Online]. Available: https://rapsodo.com/baseball/pitching. [Accessed April 29, 2019].
[7] Nathan, Alan. *Determining Pitch Movement from PITCHf/x Data*. Oct. 18, 2012.
[8] *Azure Kinect DK Documentation | Microsoft Docs*. https://docs.microsoft.com/en-us/azure/Kinect-dk/. Accessed 30 Jan. 2020.
[9] *IScan-Phantom-with-Turbo-1-Rev-1.Pdf*. https://coloradoengineering.com/wp-content/uploads/2018/10/iScan-Phantom-with-Turbo-1-Rev-1.pdf. Accessed 20 Feb. 2020.
[10] *Displacement, Acceleration Algebra | Zona Land Education*. http://zonalandeducation.com/mstm/physics/mechanics/kinematics/EquationsForAcceleratedMotion/AlgebraRearrangements/Displacement/DisplacementAccelerationAlgebra.htm. Accessed 20 Feb. 2020.
[11] 3D Polygon - Draw a Box with Matlab. http://www.matrixlab-examples.com/3d-polygon.html. Accessed 20 Feb. 2020.

9. Appendices

9.1 77GHz Radar Matlab Code

Detecting Ball with Radar:

```
if (max(JNew(:))>-17)
    [i,j] = find(JNew>-17);
    row = round(mean(i));
    col = round(mean(j));
    tidx(count) = idx/20;
    t(count) = idx/20 - tidx(1);
    x(count) = mX(row,col);
    y(count) = mY(row,col);
    count = count + 1;
end
```

Capture Positional Data of Pitch and Filter out Inaccurate Points:

```
radar_height = 44; %inches
rad_h = radar_height/39.37; %meters

len = length(x);

if(x(1) > -0.6)  %First Point Negative
    x = x(2:len);
    y = y(2:len);
    t = t(2:len);
end

len = length(x);

% if(x(1) > x(2))  %First LEss Than Second
%    x(1:(len-1)) = x(2:len);
%    y(1:(len-1)) = y(2:len);
%    t(1:(len-1)) = t(2:len);
%    len = len-1;
% end

le = len;

for(k = 1:(le-1))  %Any Point Lower than Ground
    if(x(k) > rad_h)
      x(k:len-1) = x(k+1:len);
      y(k:len-1) = y(k+1:len);
      t(k:len-1) = t(k+1:len);
      len= len-1;
    end
end

if(x(len) > rad_h)  %Last Point Lower than Ground
   x = x(1:len-1);
   y = y(1:len-1);
   t = t(1:len-1);
   len = len-1;
end

if(x(len) < x(1))  %Last Point Higher than First
   x = x(1:len-1);
   y = y(1:len-1);
   t = t(1:len-1);
   len = len-1;
end

le = len;
```

```
for(k = 2:(le-1))   %Any Point Higher than First
   if(x(k) < x(1))
      x(k:len-1) = x(k+1:len);
      y(k:len-1) = y(k+1:len);
      t(k:len-1) = t(k+1:len);
      len= len-1;
   end
end

le = len;

for(k = 2:(le-1))   %Points Past 1 m Similar to First
   if(x(k) < (x(1)+0.15) && y(k) > (y(1)+1))
      x(k:len-1) = x(k+1:len);
      y(k:len-1) = y(k+1:len);
      t(k:len-1) = t(k+1:len);
      len= len-1;
   end
end

le = len;

for(k = 2:(le-1))   %Middle Lower than Two Adjacent
   if(x(k) > x(k-1) && x(k) > (x(k+1) + 0.1))
      x(k:len-1) = x(k+1:len);
      y(k:len-1) = y(k+1:len);
      t(k:len-1) = t(k+1:len);
      len= len-1;
   end
end

for(k = 2:(le-1))   %Middle Higher than Two Adjacent
   if(x(k) < x(k-1) && x(k) < (x(k+1) - 0.1))
      x(k:len-1) = x(k+1:len);
      y(k:len-1) = y(k+1:len);
      t(k:len-1) = t(k+1:len);
      len= len-1;
   end
end

if(x(len) < x(len - 1))  %Last Point Higher than Second to Last
   x = x(1:(len-1));
   y = y(1:(len-1));
   t = t(1:(len-1));
   len = len-1;
end
```

```
    x = x(1:len);
    y = y(1:len);
    t = t(1:len);
    t = t - t(1);

plot(y,-x,'o');
title('Scatterplot of x and y Positional Values');
xlabel('Horizontal distance from radar');
ylabel('Vertical distance from radar');

fileID = fopen('Positional_Values.txt','w');
fprintf(fileID,'%f ',y);
fprintf(fileID,'\n');
fprintf(fileID,'%f ',-x);
fprintf(fileID,'\n');
fprintf(fileID,'%f ',t);
fclose(fileID);
```

Strike Zone Display: [11]

```
finy = LSFit(y,-x,t);
fprintf('\n Final Height is: %g',finy);


fileID = fopen('zonemat_height.txt','r');
formatSpec = '%f';
A = fscanf(fileID,formatSpec);
fclose(fileID);

height = A;

fileID_2 = fopen('heightmat_off_ground.txt','r');
formatSpec = '%f';
B = fscanf(fileID_2,formatSpec);
fclose(fileID_2);

height_off_ground = B;

% Creates 3D Representation of Zone

p1 = [0 0 0+B];
p2 = [431.8 0 0+B];
p3 = [431.8 431.8 0+B];
p4 = [0 431.8 0+B];

p5 = [0 0 height+B];
p6 = [431.8 0 height+B];
```

```
p7 = [431.8 431.8 height+B];
p8 = [0 431.8 height+B];

poly_rectangle_mat(p1, p2, p3, p4)
poly_rectangle_mat(p5, p6, p7, p8)
poly_rectangle_mat(p1, p2, p6, p5)
poly_rectangle_mat(p2, p3, p7, p6)
poly_rectangle_mat(p4, p3, p7, p8)
poly_rectangle_mat(p1, p4, p8, p5)

view(120, 43)
%Get pitch height
vert_pitch_dist_from_radar = 1000*(finy);
height_of_radar = rad_h*1000;
pitch_height_from_ground = height_of_radar+vert_pitch_dist_from_radar;

[x,y,z] = sphere;
surf(x*36.5+215.9,y*36.5,z*36.5+pitch_height_from_ground)
xlabel('length (mm)')
ylabel('width (mm)')
zlabel('height (mm)')

top_of_zone = height_off_ground+height;

if (pitch_height_from_ground >= height_off_ground) &&
(pitch_height_from_ground <= top_of_zone)
    title('Unique Strike Zone: Pitch = Strike!')
end
if(pitch_height_from_ground < height_off_ground)
       title('Unique Strike Zone: Pitch = Ball Low!')
end
if(pitch_height_from_ground > top_of_zone)
    title('Unique Strike Zone: Pitch = Ball High!')
end

fprintf('\n Final Height off Ground (mm): %g
\n',pitch_height_from_ground);

figure
plot(t1,-x1,'o')
title('Original Y Position v Time')
xlabel('Time (s)')
ylabel('Y (m)')
```

LSFit Function: Display Path of Pitch and Final Location

```
function h = LSFit(xarr,yarr,time)
```

```
%% Change polar coordinates to Regular
%A=load('Positional_Values.txt');

figure(1)
x = xarr'; %A(1,:)';
y = yarr'; %A(2,:)';
plot(x,y,'o');
title('Rectangular Coordinates of Pitch')
xlabel('Distance (m)')
ylabel('Height (m)')
%% Position - x vs time
XX = xarr';
t =  time';

figure(2)
plot(t,XX,'o')
title('X Position over Time')
xlabel('time (sec)')
ylabel('x (m)')

%[B,TF] = rmoutliers(XX,'movmedian',0.05,'SamplePoints',t);
%plot(t,XX,'b.-',t(~TF),B,'r.-')
%legend('Input Data','Output Data')

figure(3)
curvefit_x = fit(t,XX,'poly2','normalize','off');
plot(curvefit_x,t,XX);
title('Best Fit Curve of X Position over Time')
xlabel('time (sec)')
ylabel('x (m)')

time_final = 7;
curvefit_x(time_final);

formula(curvefit_x);
p1 = curvefit_x.p1;
p2 = curvefit_x.p2;
p3 = curvefit_x.p3;
%p4 = curvefit_x.p4;
equation_x = p1*(t.^2) + p2*t + p3; % + p4;
fprintf('X Equation: %ft^2 + %ft + %f \n', p1,p2,p3);%,p4);

% Differentiate to get the predicted velocity in x direction
figure(4)
fx = differentiate(curvefit_x,t);
plot(t,fx,'o');
```

```
% Get equation of best fit line of velocity
curvefit_x_velo = fit(t,fx,'poly1','normalize','off');
plot(curvefit_x_velo,t,fx);
title('Predicted X-Velocity over Time')
xlabel('time (sec)')
ylabel('Vx (m/s)')

% time_final = 7;
% curvefit_x_velo(time_final);

formula(curvefit_x_velo);
p5 = curvefit_x_velo.p1;
p6 = curvefit_x_velo.p2;
%p7 = curvefit_x_velo.p3;
equation_velo_x = p5*t + p6;
fprintf('X Velocity Equation: %ft + %f \n', p5,p6);

%% Position - y vs time
YY = yarr';
t = time';
figure(5)
plot(t,YY,'o')
title('Y Position over Time')
xlabel('time (sec)')
ylabel('y (m)')

figure(6)
curvefit_y = fit(t,YY,'poly2','normalize','off');
plot(curvefit_y,t,YY);
title('Best Fit Curve of Y Position over Time')
xlabel('time (sec)')
ylabel('y (m)')

time_final = 7;
curvefit_y(time_final);

formula(curvefit_y);
p8 = curvefit_y.p1;
p9 = curvefit_y.p2;
p10 = curvefit_y.p3;
%p11 = curvefit_y.p4;
equation_y = p8*(t.^2) + p9*t + p10;
fprintf('Y Equation: %ft^2 + %ft + %f \n', p8,p9,p10);

% Differentiate to get the predicted velocity in y direction
figure(7)
fy = differentiate(curvefit_y,t);
```

```
plot(t,fy,'o');

% Get equation of least squares fit line
curvefit_y_velo = fit(t,fy,'poly1','normalize','off');
plot(curvefit_y_velo,t,fy);
title('Predicted Y-Velocity over Time')
xlabel('time (sec)')
ylabel('Vy (m/s)')

time_final = 7;
curvefit_y_velo(time_final);

formula(curvefit_y_velo);
p12 = curvefit_y_velo.p1;
p13 = curvefit_y_velo.p2;
%p14 = curvefit_y_velo.p3;
equation_velo_y = p12*t + p13;
fprintf('Y Velocity Equation: %ft + %f \n', p12,p13);

%% Predicted Velocity without Magnitude
velocity = sqrt(((fx).^2) + ((fy).^2));
figure(8)
plot(t,velocity, 'o')
title('Velocity over Time')
xlabel('time (sec)')
ylabel('Velocity')

% Get equation of least squares fit line
curvefit_velo = fit(t,velocity,'poly1','normalize','off');
plot(curvefit_velo,t,velocity);
title('Predicted Velocity over Time')
xlabel('time (sec)')
ylabel('Velocity (m/s)')

time_final = 7;
curvefit_velo(time_final);

formula(curvefit_velo);
p15 = curvefit_velo.p1;
p16 = curvefit_velo.p2;
%p17 = curvefit_velo.p3;
equation_velo = p15*t + p16;
fprintf('Velocity Equation: %ft + %f \n', p15,p16);

%% Predicted Acceleration
% Differentiate to get the predicted acceleration
figure(9)
```

```
facc = differentiate(curvefit_velo,t);
plot(t,facc,'o');

% Get equation of least squares fit line
curvefit_acc = fit(t,facc,'poly1','normalize','off');
plot(curvefit_acc,t,facc);
title('Predicted Acceleration over Time')
xlabel('time (sec)')
ylabel('Acceleration (m/s^2)')

time_final = 7;
curvefit_acc(time_final);

formula(curvefit_acc);
p18 = curvefit_acc.p1;
p19 = curvefit_acc.p2;
%p20 = curvefit_acc.p3;
equation_acc = p19;
fprintf('Acceleration Equation: %f \n', p19);


%% Final Time

xf = 252; %inches
xfm = (xf/39.37) - x(1); %meters
c = -xfm;
v0 = p6;
a0 = p5;
a = a0/2;
tf1 = (-v0+sqrt((v0^2)-(4*a*c)))/(2*a);
tf2 = (-v0-sqrt((v0^2)-(4*a*c)))/(2*a);

tf = min(abs(tf1),abs(tf2));

h = p8*(tf^2) + p9*tf + p10;
fprintf('%g %g %g \n',tf1,tf2,h)
```

9.2 Camera Visual Studio Code

// Note: Code from Microsoft Azure Kinect Sample Code indicated in LIGHT GREY [8]
// Code written by team indicated in BLACK

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include <k4a/k4a.h>
#include <k4abt.h>

#define VERIFY(result, error)                                                                        \
    if(result != K4A_RESULT_SUCCEEDED)                                                               \
    {                                                                                                \
        printf("%s \n - (File: %s, Function: %s, Line: %d)\n", error, __FILE__, __FUNCTION__, __LINE__); \
        exit(1);                                                                                     \
    }                                                                                                \

int main()
{
        float totalchesty = 0;
        float totalnavaly = 0;
        float totalrkneey = 0;
        float totallkneey = 0;
        float totalrfooty = 0;
        float totallfooty = 0;
        //able to track catcher
        float totalheady = 0;

        //able to track catcher
        float totalchesty2 = 0;
        float totalnavaly2 = 0;
        float totalrkneey2 = 0;
        float totallkneey2 = 0;
        float totalrfooty2 = 0;
        float totallfooty2 = 0;
        float totalheady2 = 0;

        k4a_device_t device = NULL;
        VERIFY(k4a_device_open(0, &device), "Open K4A Device failed!");

        // Start camera. Make sure depth camera is enabled.
        k4a_device_configuration_t deviceConfig = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
        deviceConfig.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;
        deviceConfig.color_resolution = K4A_COLOR_RESOLUTION_OFF;
        VERIFY(k4a_device_start_cameras(device, &deviceConfig), "Start K4A cameras failed!");

        k4a_calibration_t sensor_calibration;
        VERIFY(k4a_device_get_calibration(device, deviceConfig.depth_mode, deviceConfig.color_resolution,
&sensor_calibration),
                "Get depth camera calibration failed!");

        k4abt_tracker_t tracker = NULL;
        k4abt_tracker_configuration_t tracker_config = K4ABT_TRACKER_CONFIG_DEFAULT;
        VERIFY(k4abt_tracker_create(&sensor_calibration, tracker_config, &tracker), "Body tracker initialization
failed!");
```

```
	int frame_count = 0;
	float trigger = 0;
	int trigcount = 0;
	float lfoot[900];
	do
	{
		k4a_capture_t sensor_capture;
		k4a_wait_result_t get_capture_result = k4a_device_get_capture(device, &sensor_capture,
K4A_WAIT_INFINITE);
		if (get_capture_result == K4A_WAIT_RESULT_SUCCEEDED)
		{
			frame_count++;
			k4a_wait_result_t queue_capture_result = k4abt_tracker_enqueue_capture(tracker,
sensor_capture, K4A_WAIT_INFINITE);
			k4a_capture_release(sensor_capture); // Remember to release the sensor capture once you
finish using it
			if (queue_capture_result == K4A_WAIT_RESULT_TIMEOUT)
			{
				// It should never hit timeout when K4A_WAIT_INFINITE is set.
				printf("Error! Add capture to tracker process queue timeout!\n");
				break;
			}
			else if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
			{
				printf("Error! Add capture to tracker process queue failed!\n");
				break;
			}

			k4abt_frame_t body_frame = NULL;
			k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker, &body_frame,
K4A_WAIT_INFINITE);
			if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
			{
				// Successfully popped the body tracking result. Start your processing

				size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);
				printf("%zu bodies are detected!\n", num_bodies);

				for (size_t i = 0; i < num_bodies; i++)
				{
					k4abt_skeleton_t skeleton;
					k4abt_frame_get_body_skeleton(body_frame, i, &skeleton);
					uint32_t id = k4abt_frame_get_body_id(body_frame, i);

					// look only at batter foot
					float headxinitial = skeleton.joints[20].position.xyz.x;
					float headyinitial = skeleton.joints[20].position.xyz.y;
					float headzinitial = skeleton.joints[20].position.xyz.z;
					printf("Head Initial: \n");
```

```
printf("X: %f\n", headxinitial);
printf("Y: %f\n", headyinitial);
printf("Z: %f\n\n", headzinitial);

if (headyinitial < 0)  // look only at batter foot
{
	float lfootx = skeleton.joints[15].position.xyz.x;
	float lfooty = skeleton.joints[15].position.xyz.y;
	//track foot height values with array
	lfoot[frame_count] = lfooty;
	//activate trigger to collect other values if foot is coming down
	if (lfoot[frame_count] > (abs(lfoot[frame_count - 5] + 200)))
	{
		trigger++;
	}
	float lfootz = skeleton.joints[15].position.xyz.z;
	printf("Left Foot: \n");
	printf("X: %f\n", lfootx);
	printf("Y: %f\n", lfooty);
	printf("Y: array value %f\n", lfoot[frame_count]);
	printf("Y: array value -5 %f\n", lfoot[frame_count - 5]);
	printf("Z: %f\n", lfootz);
	printf("Trigger = %f\n", trigger);
}

if (trigger > 1)
{
	float rfootx = skeleton.joints[19].position.xyz.x;
	float rfooty = skeleton.joints[19].position.xyz.y;
	float rfootz = skeleton.joints[19].position.xyz.z;
	printf("Right Foot: \n");
	printf("X: %f\n", rfootx);
	printf("Y: %f\n", rfooty);
	printf("Z: %f\n", rfootz);

	float lkneex = skeleton.joints[13].position.xyz.x;
	float lkneey = skeleton.joints[13].position.xyz.y;
	float lkneez = skeleton.joints[13].position.xyz.z;
	printf("Left Knee: \n");
	printf("X: %f\n", lkneex);
	printf("Y: %f\n", lkneey);
	printf("Z: %f\n", lkneez);

	float rkneex = skeleton.joints[17].position.xyz.x;
	float rkneey = skeleton.joints[17].position.xyz.y;
	float rkneez = skeleton.joints[17].position.xyz.z;
	printf("Right Knee: \n");
	printf("X: %f\n", rkneex);
	printf("Y: %f\n", rkneey);
```

```
printf("Z: %f\n", rkneez);

float navalx = skeleton.joints[1].position.xyz.x;
float navaly = skeleton.joints[1].position.xyz.y;
float navalz = skeleton.joints[1].position.xyz.z;
printf("Spine - Naval: \n");
printf("X: %f\n", navalx);
printf("Y: %f\n", navaly);
printf("Z: %f\n", navalz);

float chestx = skeleton.joints[2].position.xyz.x;
float chesty = skeleton.joints[2].position.xyz.y;
float chestz = skeleton.joints[2].position.xyz.z;
printf("Spine - Chest: \n");
printf("X: %f\n", chestx);
printf("Y: %f\n", chesty);
printf("Z: %f\n", chestz);


//able to track catcher
float headx = skeleton.joints[20].position.xyz.x;
float heady = skeleton.joints[20].position.xyz.y;
float headz = skeleton.joints[20].position.xyz.z;
printf("Head: \n");
printf("X: %f\n", headx);
printf("Y: %f\n", heady);
printf("Z: %f\n\n", headz);


//able to track catcher
if (heady < 0)   // above camera --> batter
{
        totallfooty = totallfooty + lfoot[frame_count];
        totalrfooty = totalrfooty + rfooty;
        totallkneey = totallkneey + lkneey;
        totalrkneey = totalrkneey + rkneey;
        totalnavaly = totalnavaly + navaly;
        totalchesty = totalchesty + chesty;
        //able to track catcher
        totalheady = totalheady + heady;
}


if (heady > 0)    //below camera --> catcher
{
        totalchesty2 = totalchesty2 + chesty;
        totalnavaly2 = totalnavaly2 + navaly;
        totalrkneey2 = totalrkneey2 + rkneey;
        totallkneey2 = totallkneey2 + lkneey;
```

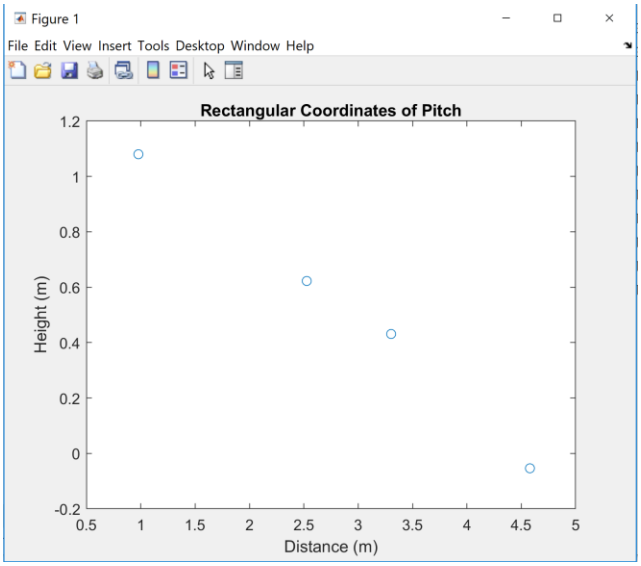
```
                                        totalrfooty2 = totalrfooty2 + rfooty;
                                        totallfooty2 = totallfooty2 + lfoot[frame_count];
                                        totalheady2 = totalheady2 + heady;
                                }

                                trigcount++;
                        }
                }

                        k4a_image_t body_index_map =
k4abt_frame_get_body_index_map(body_frame);

                        k4a_image_release(body_index_map);

                        k4abt_frame_release(body_frame); // Remember release frame once done with it
                }
                else if (pop_frame_result == K4A_WAIT_RESULT_TIMEOUT)
                {
                        //  It should never hit timeout when K4A_WAIT_INFINITE is set.
                        printf("Error! Pop body frame result timeout!\n");
                        break;
                }
                else
                {
                        printf("Pop body frame result failed!\n");
                        break;
                }
        }
        else if (get_capture_result == K4A_WAIT_RESULT_TIMEOUT)
        {
                // It should never hit time out when K4A_WAIT_INFINITE is set.
                printf("Error! Get depth frame time out!\n");
                break;
        }
        else
        {
                printf("Get depth capture returned error: %d\n", get_capture_result);
                break;
        }

} while (trigcount < 60); //collect data for about 1 second

//able to track catcher
float chestaverage;
float navalaverage;
float rkneeaverage;
float lkneeaverage;
float rfootaverage;
float lfootaverage;
```

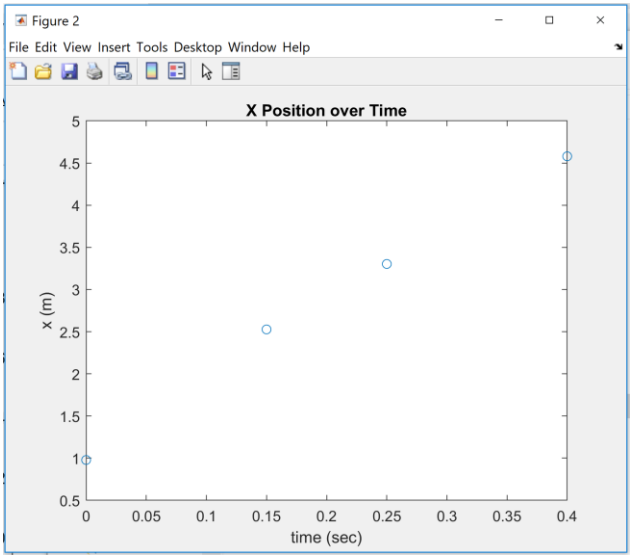
```
	float headaverage;
	float kneeaverage;
	float upperaverage;
	float footaverage;
	float zone_height;

	lfootaverage = totallfooty / 30;
	rfootaverage = totalrfooty / 30;
	lkneeaverage = totallkneey / 30;
	rkneeaverage = totalrkneey / 30;
	navalaverage = totalnavaly / 30;
	chestaverage = totalchesty / 30;
	headaverage = totalheady / 30;
	kneeaverage = (rkneeaverage + lkneeaverage) / 2;
	upperaverage = (chestaverage + navalaverage) / 2;
	footaverage = (rfootaverage + lfootaverage) / 2;
	zone_height = (kneeaverage) - (chestaverage);

	FILE* fp;
	errno_t err;
	err = fopen_s(&fp,
"C:\\CEI\\CEI_Matlab_files_07_19_2019\\CEI_Matlab_files_07_19_2019\\Camera\\Matlab_Capstone\\camera_test
.txt", "w");

	if (err != NULL)
	{
		printf("Error Opening File!\n");
		return 0;
	}

	fprintf(fp,"Spine - Chest: %f \n", chestaverage);
	fprintf(fp,"Spine - Naval: %f \n", navalaverage);
	fprintf(fp,"Right Knee: %f \n", rkneeaverage);
	fprintf(fp,"Left Knee: %f \n", lkneeaverage);
	fprintf(fp, "Left Foot: %f \n", rfootaverage);
	fprintf(fp, "Right Foot: %f \n", lfootaverage);
	fprintf(fp,"Knee Height: %f \n", kneeaverage);
	fprintf(fp,"Chest Height: %f \n", upperaverage);
	fprintf(fp,"Foot Height: %f \n", footaverage);
	fprintf(fp,"Height Off Ground: %f \n", footaverage-kneeaverage);


	fclose(fp);

	FILE* fmat;
	errno_t errmat;
	errmat = fopen_s(&fmat,
"C:\\Users\\jorr5\\Documents\\MATLAB\\CEI_Matlab_files_07_19_2019\\cameramat_test.txt", "w");
```

```
	if (errmat != NULL)
	{
		printf("Error Opening File!\n");
		return 0;
	}

	fprintf(fmat, "Spine - Chest: %f \n", chestaverage);
	fprintf(fmat, "Spine - Naval: %f \n", navalaverage);
	fprintf(fmat, "Right Knee: %f \n", rkneeaverage);
	fprintf(fmat, "Left Knee: %f \n", lkneeaverage);
	fprintf(fmat, "Left Foot: %f \n", rfootaverage);
	fprintf(fmat, "Right Foot: %f \n", lfootaverage);
	fprintf(fmat, "Knee Height: %f \n", kneeaverage);
	fprintf(fmat, "Chest Height: %f \n", upperaverage);
	fprintf(fmat, "Foot Height: %f \n", footaverage);
	fprintf(fmat, "Height Off Ground: %f \n", footaverage - kneeaverage);
	fprintf(fmat, "Height Of Zone: %f \n", zone_height);
	fprintf(fp, "Total Chest 1 : %f \n", totalchesty);
	fprintf(fp, "Total Chest 2 : %f \n", totalchesty2);

	fclose(fmat);

	FILE* f;
	errno_t err2;
	err2 = fopen_s(&f,
"C:\\CEI\\CEI_Matlab_files_07_19_2019\\CEI_Matlab_files_07_19_2019\\Camera\\Matlab_Capstone\\zone_height
.txt", "w");

	if (err2 != NULL)
	{
		printf("Error Opening File!\n");
		return 0;
	}

	fprintf(f, "%f", zone_height);

	fclose(f);

	FILE* fmatl;
	errno_t errmat2;
	errmat2 = fopen_s(&fmatl,
"C:\\Users\\jorr5\\Documents\\MATLAB\\CEI_Matlab_files_07_19_2019\\zonemat_height.txt", "w");

	if (errmat2 != NULL)
	{
		printf("Error Opening File!\n");
		return 0;
	}
```

```
		fprintf(fmatl, "%f", zone_height);

		fclose(fmatl);

		FILE* ff;
		errno_t err3;
		err3 = fopen_s(&ff,
"C:\\CEI\\CEI_Matlab_files_07_19_2019\\CEI_Matlab_files_07_19_2019\\Camera\\Matlab_Capstone\\height_off_
ground.txt", "w");

		if (err3 != NULL)
		{
			printf("Error Opening File!\n");
			return 0;
		}

		fprintf(ff, "%f", footaverage - kneeaverage);

		fclose(ff);

		FILE* fmatla;
		errno_t errmat3;
		errmat3 = fopen_s(&fmatla,
"C:\\Users\\jorr5\\Documents\\MATLAB\\CEI_Matlab_files_07_19_2019\\heightmat_off_ground.txt", "w");

		if (errmat3 != NULL)
		{
			printf("Error Opening File!\n");
			return 0;
		}

		fprintf(fmatla, "%f", footaverage - kneeaverage);

		fclose(fmatla);

		printf("Finished body tracking processing!\n");

		k4abt_tracker_shutdown(tracker);
		k4abt_tracker_destroy(tracker);
		k4a_device_stop_cameras(device);
		k4a_device_close(device);

		return 0;
}
```

9.3 Tracking Baseball Results and Figures





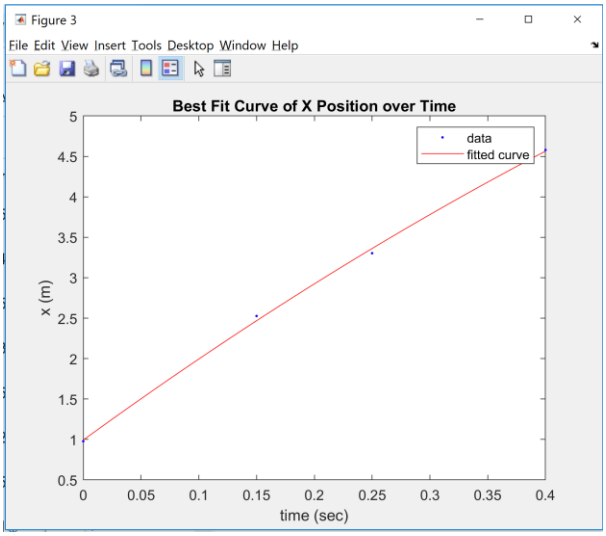


Figure 3
File Edit View Insert Tools Desktop Window Help
Best Fit Curve of X Position over Time
data
fitted curve
x (m)
time (sec)

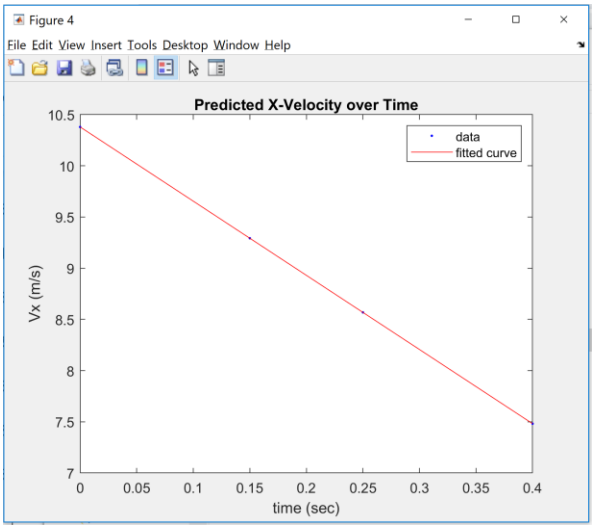


Figure 4
File Edit View Insert Tools Desktop Window Help
Predicted X-Velocity over Time
data
fitted curve
Vx (m/s)
time (sec)

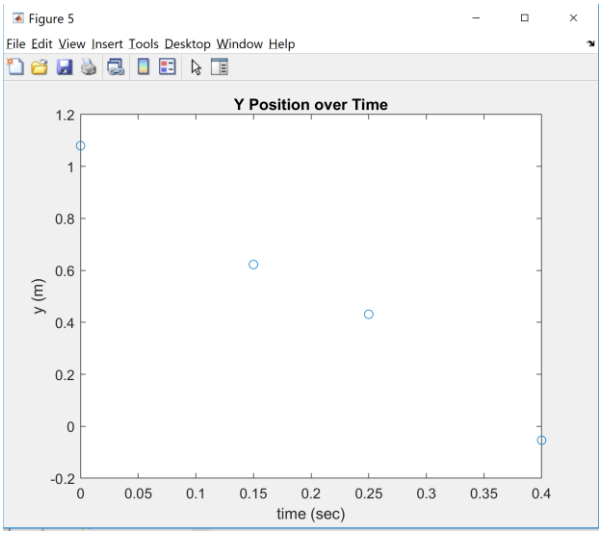


Figure 5
File Edit View Insert Tools Desktop Window Help
Y Position over Time
y (m)
time (sec)

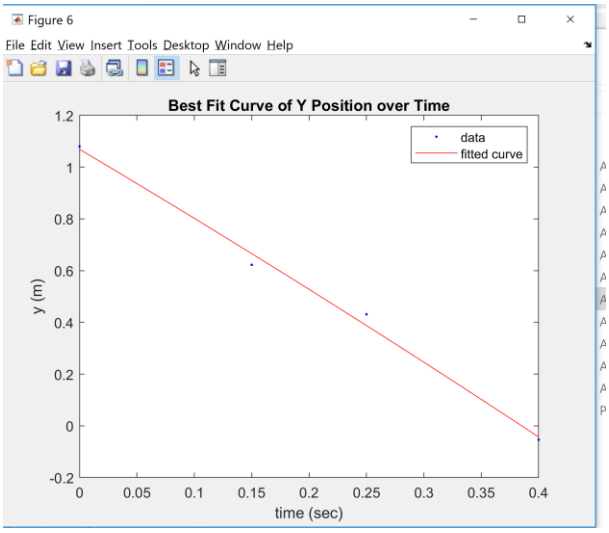


Figure 6
File Edit View Insert Tools Desktop Window Help
Best Fit Curve of Y Position over Time
data
fitted curve
y (m)
time (sec)

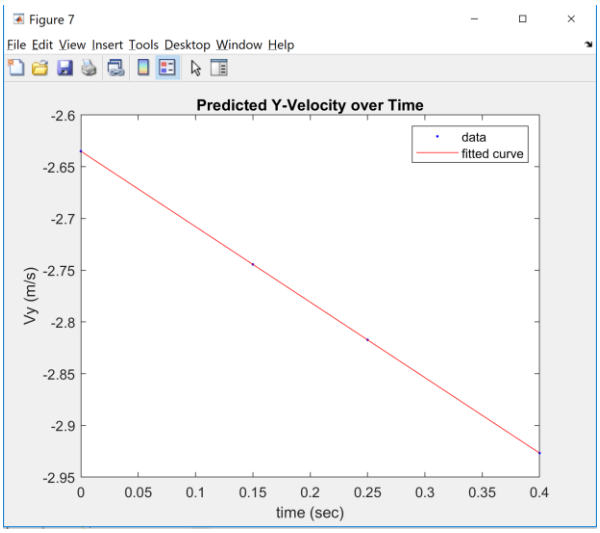


Figure 7
File Edit View Insert Tools Desktop Window Help
Predicted Y-Velocity over Time
data
fitted curve
Vy (m/s)
time (sec)

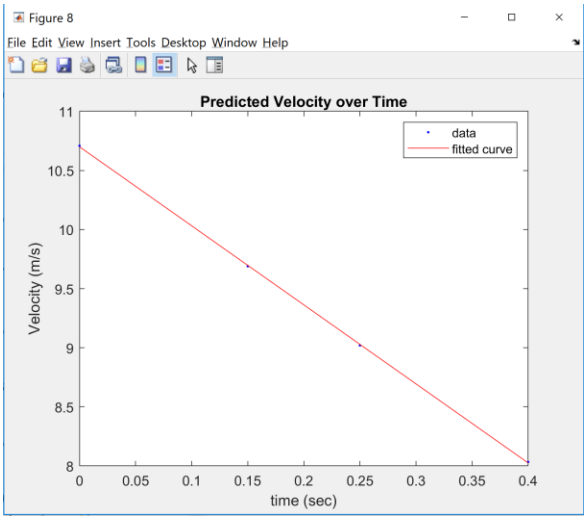
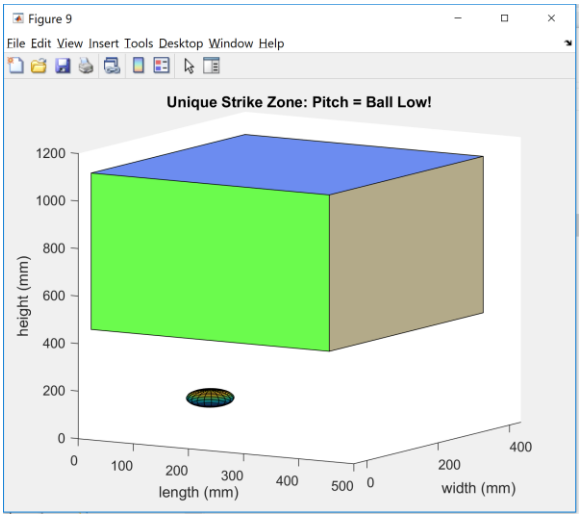
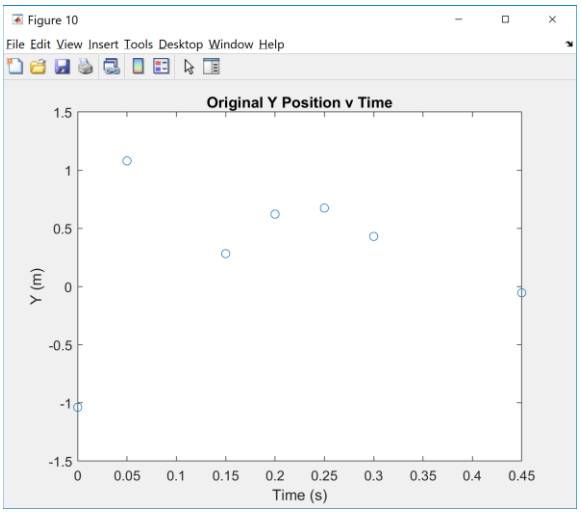


Figure 8
File Edit View Insert Tools Desktop Window Help
Predicted Velocity over Time
data
fitted curve
Velocity (m/s)
time (sec)






```
X Equation: -3.622114t^2 + 10.379466t + 0.992908
X Velocity Equation: -7.244227t + 10.379466
Y Equation: -0.364665t^2 + -2.634984t + 1.069286
Y Velocity Equation: -0.729330t + -2.634984
Velocity Equation: -6.687866t + 10.700535
Acceleration Equation: -6.687866
0.687249 2.17833 -0.913839
```

```
Final Height is: -0.913839Final Height off Ground (mm): 203.763
```