Smart Net, Kicking into the Future

Patent Pending
No.: 63/460,826 'Automated Kicking Evaluator for Sports Media'

A Major Qualifying Project Report Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
degree of Bachelor of Science

by
Joseph Durocher
Noah Litzinger
Aidan Lynn
Noah Skinner
Michael Sposato

Advisor:
Alireza Ebadi

Date
April 27, 2023

This report represents the work of four WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see: http://www.wpi.edu/Academics/Projects

Acknowledgements

The authors would like to recognize Professor Alireza Ebadi, who served as the project advisor for his guidance and mentorship. The authors would also like to thank the faculty and our student peers from the Mechanical Engineering, Robotic Engineering, Computer Science departments and the WPI Football team for their support.

The efforts of the authors and Professor Ebadi have culminated into a rewarding ending. We are happy to acknowledge that the current project has resulted in the provisional patent application No.: 63/460,826 ‘Automated Kicking Evaluator for Sports Media’. Throughout the next year, the team plans to work towards attaining the official patent.

Abstract

The average National Football League (NFL) game features about four field goal attempts. Since 2001, around 20% of NFL games have been decided by three points or fewer, a margin that can be achieved by a successful field goal. These statistics suggest that kicks in American football can be crucial in determining the outcome of games.

Currently, the kickers practice by kicking into a net on the sidelines with no feedback. The goal of this project is to develop a setup to collect data, run analysis and display the trajectory of the kick, so the kickers could adjust their kicks while practicing on the sidelines.

The project was divided into four academic terms, with background research done in A term, data acquisition and analysis in B term, and setup development in C and D term. The design, execution, analysis, simulation results are presented and the recommendations for future improvements are provided.

Table of Contents

Acknowledgements 2
Abstract 3
Table of Contents 4
List of Figures 6
List of Tables 7
1. Introduction 8
2. Background 9
2.1 Trajectory Simulators 9
2.1.1 TrackMan Technology 9
2.1.2 Gaining an Understanding of the Common Golf Simulator 10
2.2 Trajectory Calculations within Different Scenarios 11
2.2.1 Analyzing the forces on an American Football and its Flight Path 11
2.2.2 Effects of Lift and Drag on an American Football 12
2.3 MATLAB and Its Capabilities 12
2.3.1 Image Processing and Segmentation 13
3. Methodology 14
3.1 Mission Statement 14
3.2 Objectives 14
3.3 Objective 1: Define the general formulas for the flight path of an American football in a real world scenario 14
3.3.1. Identifying Variables Used in Trajectory Equations 15
3.3.2 Formulation of Excel Sheets for Drag, Lift, and Surface Area 16
3.3.3 Finding Components of Trajectory Equations 19
3.4 Objective 2: Gain understanding of MATLAB image processing in order to more accurately predict ball trajectory 21
3.4.1 Image Recognition and MATLAB Color Thresholds 21
3.4.2 MATLAB Video Processing 25
3.4.3 Locating and Graphing the Centroid of the Ball 26
3.4.4 Calculating Initial Conditions: Velocity, Launch Angle, & Spin Rate 30
3.5 Objective 3: Analyze the data collected through image processing and implement general formulas previously defined to predict the trajectory of the kick 31
3.5.1 Combining Calculated Variables Through Image Processing With Trajectory Calculations 32
3.5.2 Experimental Setup 32
4. Results and Discussion 34
4.1 Plotting Trajectory Equation and Final Messaging 34
4.2 Test Kick Results 36

4.3 Broader Impacts 37
4.3.1 Engineering Ethics 37
4.3.2 Societal and Global Impact 37
4.3.3 Environmental Impact 38
4.3.4 Codes and Standards 38
4.3.5 Economic factors 38
4.4 Future Additions 39
5. Conclusion 40
Bibliography 41
Appendix A: Trajectory Equations 43
Appendix B: Gantt Chart 44
Appendix C: Decision Matrices 45
Appendix D: Final MATLAB Code 46

List of Figures

Figure 1: Angle of the Football 16

Figure 2: Lift Coefficient Graph from Excel (0°-360°) 17

Figure 3: Drag Coefficient Graph from Excel (0°-360°) 17

Figure 4: SolidWorks Model of Football 18

Figure 5: Surface Areas for Each Full Degree 18

Figure 6: Force Diagram on the Football 19

Figure 7: Matlab Code for Components of Trajectory 20

Figure 8: MATLAB Script to Identify the American Football and its Centroid 22

Figure 9: The Result of the First Image Processing Script which Isolates the Football and Locates the Centroid with a Blue Indicator 22

Figure 10: The Labspace of the Color Thresholder within MATLAB Displaying the Four Different Filters Available 24

Figure 11: Workspace of the HSV Intensity Filter, Featuring the Color Wheel and Intensity Scales 24

Figure 12: Script where the Three Dimensional HSV Matrix is Transformed to Two Dimensional Black and White 25

Figure 13: MATLAB Video Processing Code Sourced from Previous Classes with Few Modifications 26

Figure 14: Second Iteration of the Centroid Code used to Plot Data Afterwards Featuring both Filters 28

Figure 15: Both Function Calculated (Blue) and Mathematically Calculated (Red) Centroids on the Identified Ball Area 29

Figure 16: Graph Plotting the Centroid and PostFilt Matrices. Centroid is Denoted by the Blue Circles and PostFilt in the Orange Asterisks. X-Axis is the X-Position and Y-Axis is the Y-Position. 29

Figure 17: Lines 28 and 29 Featuring the Conversion Factor Calculations 31

Figure 18: The Initial Condition Calculation Section 31

Figure 19: Experimental Setup on a Sideline 33

Figure 20: Code for Messaging 35

Figure 21: Final Result 35

Figure 22: Absolute Difference Between Measured and Calculated Distance of Each Kick in Yards 36

Figure 23: RMSE Equation 37

List of Tables

Table 1: List of Variables 15

Table 2: Bill of Materials for Experimental Setup 33

1. Introduction

American football features many different components that contribute to the score of the game. These components can be grouped into two major categories: touchdowns and field goals. Each team spends the week before a game crafting ways and training in order to either score or prevent touchdowns. The 32 teams of the NFL currently have many training techniques, facilities, and simulators to help increase performance for touchdown production, but lack such access to field goal production. One of the technologies being used today in the NFL is the TrackMan radar. The purpose of this technology is to use its sonar capabilities to track the ball's flight path; this is mainly used for field goals in the NFL in order to collect data afterwards on the kick attempt. Using this technology from a training stand point presents itself as *reactive* rather than *proactive*. It presents the data and analytics after the kick is over.

A current method of warming up for kickers in the NFL and across the United States is to kick into a free standing net on the sidelines. The kickers do this in order to warm-up their muscles and gain an understanding of how their body feels before going out onto the field and attempting a field goal. Although helpful for warm-up, this practice provides no feedback to the kicker about the attempt they are trying to simulate. Essentially, the kickers are going into the situation blind and possibly less confident than would be expected about their ability to make the field goal.

This project aimed to create a simulation technology that would be able to give the kickers data in real time. Inspired by the way golf simulators work, the team has created a setup that will capture the initial flight of the ball as it is being kicked and later predict the trajectory of the kick. The team has used knowledge of MATLAB and trajectory physics to innovate the kicking setup that is commonly seen on the sidelines. Kickers using this simulator are able to receive data directly following the practice kick informing them of how far their kick would have been good from. Knowing this information – along with the initial velocity, launch angle, and spin rate – kickers can be confident in themselves and their ability to have a successful attempt during the game.

2. Background

Before beginning work on this project the team did extensive research on the current technology that is available today. To begin, this project first acknowledges the existence of other trajectory simulators that the project will be based upon. Golf simulators are a popular product in the United States and provide a great example of what the team aims to accomplish in the sport of American Football. Most golf simulators utilize a series of cameras or radars setup around a room which track the initial velocity and angle of the golf ball. Aside from golf simulators, an example of a powerful trajectory simulator is the TrackMan. TrackMan is a tracking software that was developed using a radar system in 2003 and has grown into a widely used tool for professional sports teams. Both simulations require a fundamental understanding of projectile physics in order to calculate an accurate trajectory.

When considering the trajectory of an American football, many fundamental physics equations are insufficient. Many physics examples involving trajectory often assume the object has a uniform surface area, such as with a sphere or particle. In other cases, these examples may neglect drag and lift forces to create a simpler problem as well. The scenarios progress and become more complicated as the shape of the projectile changes and drag and lift forces are added. Our project, which uses an American football with a more ovular shape, will require an alternating surface with each point in time. As a result, the lift and drag forces acting on the football will also become variable with each instance in time. Understanding these factors contributed to the research the team needed to conduct.

2.1 Trajectory Simulators

2.1.1 TrackMan Technology

TrackMan is a current technology that utilizes radar systems in order to gather and analyze a multitude of information for swing mechanics and trajectory. The company was started in 2003 when Dr. Klaus Eldrup-Jorgensen developed the idea of using radar to track golf balls (TrackMan, n.d.). Dr. Klaus' idea blossomed from noticing the unchanged techniques of teaching golf throughout his years of play. TrackMan takes advantage of the radar technology that was available, allowing the game of golf to progress at a faster rate. According to Johansson (2015), TrackMan records nine values related to club head movement and 14 values for ball flight. These metrics are each used as variables to help trace the flight path of the golf ball through contact. Once gathered, they are processed through a series of formulas that will then show the predictive trajectory of the golf ball. The ability of the software has developed to reach across multiple platforms including stationary simulators and any portable systems.

Recently the software has been adapted into other sports such as baseball and football. The software maintains the same core purpose - to trace the trajectory of a sports ball, while using slightly modified formulas to account for the different ball sizes and shapes. Major League Baseball (MLB) started testing TrackMan as a “robot ump” in 2019 through a loose relationship with the Atlantic League (Acquavella, 2019). The tracking system is placed behind the backstop and analyzes the different pitches thrown during the game. They have also developed the radar system to generate a strike zone that is unique to each batter as they are in the batter’s box; this allows for proportional strike zones (Acquavella, 2019). The system reads the pitches and relays the information to the umpire which can then make the call.

In addition to the MLB, The National Broadcasting Company (NBC) plans to work with the NFL in order to continue using TrackMan. The software is only used for replays as of 2018, however the broadcasting company hopes to expand the use of the system to live punts, kickoffs, and passes (Lemire, 2018). This software will be able to help NBC and other companies gather data throughout the season in order to show more meaningful statistics.

2.1.2 Gaining an Understanding of the Common Golf Simulator

The purpose of a golf simulator is to capture, evaluate, and predict the trajectory of a golf ball from the first few seconds of flight. There are multiple ways to capture movement such as infrared detection, sonar (radar) detection, and stereo detection (cameras). Both infrared and radar detection present their abilities well but fall short in the application of capturing the varying speed and rotation of a golf ball (Wang et al., 2019) Infrared and Radar are both more indirect ways of measurement which introduce the possibility of a higher percent error. The stereo system provides a more direct approach through high speed cameras that are able to relay the proper information to a written algorithm. From the high speed or high frame-rate cameras, the system is able to capture and process a greater amount of points in flight and rotation that make the trajectory more precise.

The systems are first triggered by a hitting detection; simulators can be triggered by noise, line scan cameras, or coded image recognition within the algorithm (Wang et al., 2019). These triggers are what start the capture and analysis of the hitting instance. From there, the cameras (usually one to two) capture the flight of the golf ball until it hits a screen. Golf balls used for simulators normally feature a set of perpendicular lines on them which help the system capture and factor in the rotation. These lines along with the multitude of frames capturing the upward flight of the ball provide the necessary information for the system to then produce initial conditions. The computer processes the information captured by the cameras and runs it through an algorithm that is able to produce a three dimensional reconstruction of the flight path (Wang et al., 2019). Afterwards, the algorithm inserts the gathered information into predetermined equations which produce the predicted trajectory and display it upon the screen.

2.2 Trajectory Calculations within Different Scenarios

2.2.1 Analyzing the forces on an American Football and its Flight Path

When accurately calculating the trajectory of a football, many forces must be accounted for. These forces include the gravitational force on the ball, the drag force which is opposite of the velocity of the ball, and the lift force also known as a Magnus force, which is perpendicular to the drag force and acts at a right angle to the direction of motion through the air.

A Magnus force helps describe the curvature of trajectory on a spinning ball and is created by differences in air pressure (National Air and Space Museum, n.d.). This effect is common in many sports including soccer, baseball, golf, tennis, and American football. The Magnus effect is responsible for golf hooks, tennis slices, and baseball curveballs. The equation to calculate the lift force of trajectory can be written as $F_L = C_L \frac{1}{2} \rho A V^2$, where $C_L$is the lift coefficient, ρ is the density of the fluid the object is traveling through, in this case is air, A is the cross-sectional surface area of the object, and V is the total velocity of the ball (Guzman et al., 2015).

Drag is the aerodynamic force that limits an object's motion through the air (NASA, n.d.). Similar to the lift force, the drag force an be written as $F_D = C_D \frac{1}{2} \rho A V^2$ (Guzman et al., 2015). Each variable is the same as in the lift force equation, however in this case the drag coefficient is identified in the equation by C_D. Due to the shape and features of a football the team must also consider the spin rate of the ball as it comes off of the kicker's foot. Calculating the spin rate will allow the team to accurately determine the lift and drag forces acting on the ball throughout its predicted trajectory.

2.2.2 Effects of Lift and Drag on an American Football

The effects of lift and drag largely affect the trajectory of the football. When analyzing the football through flight, the lift and drag are constantly changing as the ball rotates. The article, *Aerodynamics effect on the accuracy of an end-over-end kick of an American football* (Lee et. al, 2013), performed a case study to analyze their effects in an end-over-end manner in a dynamic simulation. The authors developed a set of equations using Euler's method and a MATLAB code to compile a correlation between angle, lift, and drag from 0º to 360º. From there, they showed their data in graphs showing the change from angle to angle.

In addition to finding the lift and drag coefficients, the case study also analyzes two kicks, one from Super Bowl XXV and the 2009 Big XII Championship (indoor closed stadiums). Using these specific kicks, they were able to prove their simulated data for the trend of a football in a real game situation.

2.3 MATLAB and Its Capabilities

MATLAB is a programming software created for engineers and scientists that allows for the processing of natural mathematics through matrix based programming (MathWorks, n.d). The software has been developed to be compatible with many instruments; most of which only require installing a toolbox to be properly used. Since the purpose of the software is to be universal and easy to use, there are many forums on the MathWorks website that provide help to those in need. MathWorks also provides miniature crash courses for their MATLAB software which lays down a basic understanding of simple functions along with what the software is capable of doing.

2.3.1 Image Processing and Segmentation

The Image Processing toolbox within MATLAB provides users with a variety of comprehensive algorithms and applications which allow the user to complete a number of tasks such as segmentation, enhancement, noise reduction, geometric transformations, and registration amongst both two and three dimensional figures (MathWorks, n.d.). The specific toolbox featured in MATLAB also helps process videos by separating each frame into its own figure.

MATLAB's image processing robustness is justified by the sheer quantity and capabilities of its individual algorithms and applications. Image segmentation itself comes with three applications within the software as well as over 20 different functions. The purpose of image segmentation is to identify boundaries within the image using thresholds, edges, and morphology (MathWorks, n.d.). This means that the user is able to identify and analyze certain features within an image by silencing any features in the surrounding area. Applications that allow for the user to segment an image include the Color Thresholder, Image Segmenter, and Volume Segmenter. The Volume Segmenter is used mainly for three dimensional images which can be useful for medical scans or analysis. Image Segmenter is used on two dimensional images and creates a mask over certain regions of interest. This specific application allows for saving and recalling binary masks according to what the user wants. The Image Segmenter application features functions that provide options from manual to automatic analysis which makes it versatile for scientific usage (MathWorks Support, n.d.). Color Thresholder is best used with color two dimensional images. This application filters the color intensity of an image using one of four intensity labs and produces a binary segmentation mask (MathWorks Support, n.d.). The four intensity labs include RGB, HSV, YCbCr, and L*a*b*. Each lab space can be utilized by the user for different images; meaning some spaces are better to use than others in order to isolate the desired regions. Lab spaces feature a series of three sliders representing the corresponding intensity filter. For example, RGB has one slider that represents the red values, one slider for green values, and one slider for blue values. Moving the corresponding sliders will filter the same values found in the image being segmented; those values which are within the sliders will remain apparent. The image processing capabilities that the team utilized, specifically Color Thresholder, will be further explained in the methodology section.

3. Methodology

3.1 Mission Statement

The objective of this project is to design and build a kicking cage that simulates the trajectory of a football once it is kicked into the cage. The prototype will gather data such as initial velocity and angle of the kick, then perform trajectory calculations and inform the kicker of how far away the kick was good from.

3.2 Objectives

The goal of this project is to accurately calculate and predict the trajectory of a football within two dimensions and relay a message back to the kicker, informing them of the total distance of their kick, and how far away they would have made a field goal. To accomplish this goal our team has developed the following objectives:

1. Formulate an equation capable of calculating the trajectory of an American football.
2. Use MATLAB image processing in order to analyze a video of a football kick and calculate the initial velocity, launch angle, and spin rate.
3. Analyze the data collected through image processing and implement trajectory equations previously defined to predict the trajectory of the kick.

3.3 Objective 1: Define the general formulas for the flight path of an American football in a real world scenario

The purpose of this objective is to provide the necessary trajectory calculations which will help describe the movement of the football. Our team began by using trajectory examples of a perfectly spherical object with no air resistance or drag. This allowed us to get a general understanding of an object's trajectory. The next step was to use these examples of spherical objects and apply air resistance to them. Once these calculations were finished the team began using examples of non spherical objects such as a rugby ball and football. The equations used in these final examples were then inserted into our MATLAB code.

3.3.1. Identifying Variables Used in Trajectory Equations

The first step of finding the trajectory of the ball is understanding the necessary variables that need to be included. The team examined several place kicks that could occur in a football game to give us initial data to start calculations before the MATLAB code was able to produce the initial conditions. In this scenario, the ball is set at the 10 yard line making the total distance to the uprights 20 yards. Based on the field position, the angle for this kick is about 45° from the ground. The average velocity of an NFL kicker is 30 m/s. There was no height as the ball was kicked from the ground. From here, the team looked into the parameters of the football and other variables used in calculating trajectory. A comprehensive list of variables used can be found in Table 1.

Table 1: List of Variables

Variable	**Name**	**Number or Source**
DragCo	Drag coefficient	Excel Sheet Reference
LiftCo	Lift coefficient	Excel Sheet Reference
SA	Surface area	Excel Sheet Reference
theta (θ)	Launch angle	Calculated through image processing
v0	Initial velocity	Calculated through image processing
vx0	Initial velocity x direction	v0*cos(theta) (meters/second)
vy0	Initial velocity y direction	v0*sin(theta) (meters/second)
m	Mass	0.4 (kilogram)
rho (ρ)	Density of air	1.2 (kilogram/meter^3)
g	Gravity	9.81 (meter/second)
t_tot	Total time of kick	10 (seconds)
t	Range of time	[0,dt,t_tot] (seconds)
GPH	Goal post height	3 * 1.09361
TD	Kick distance	[0,80]

3.3.2 Formulation of Excel Sheets for Drag, Lift, and Surface Area

In the article, *Aerodynamic Effect of the end-over-end kick of an American Football*, the drag and lift coefficients were found for one rotation from 0° to 360° (Lee et. al, 2013). Where the angle is defined as the relationship between the football's long diameter and the ground as seen in Figure 1.

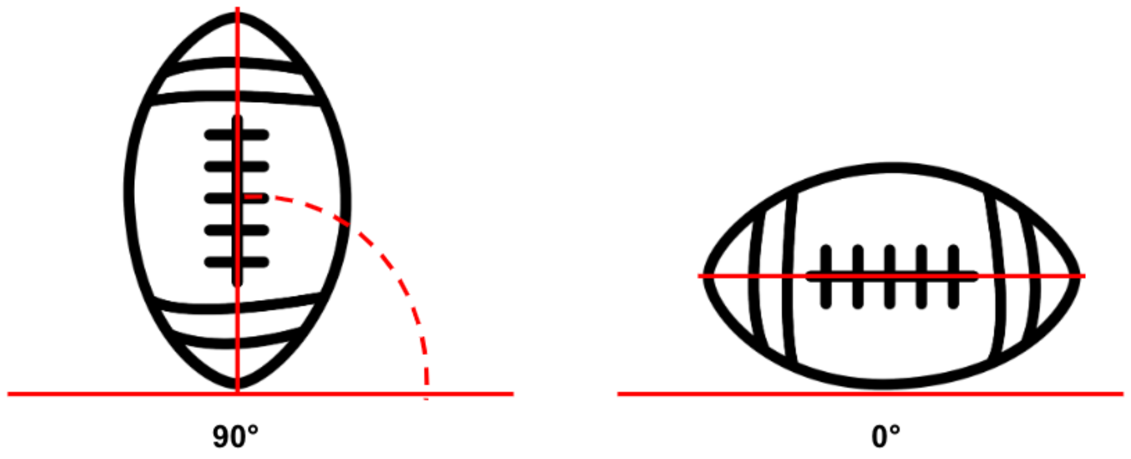


Figure 1: Angle of the Football

This data was shown in a chart in the article with no data tables to reference. The team took the data from the graphs by importing them into PlotDigitializer.com and then adding points to their slopes. PlotDigitalizer then exported the data of individual points into Excel sheets, one for lift and one for drag. Once in excel, the points were connected and the team ensured that they matched the table from the article. These excel sheets were imported into MATLAB using the 'xlsread' function. The matrices pair lift and drag coefficients to an angle in degrees from 0° to 360° as seen in Figure 2 and Figure 3 respectively. We then used the 'interp1' function to estimate the values of lift and drag for each angle not covered in the excel sheet.

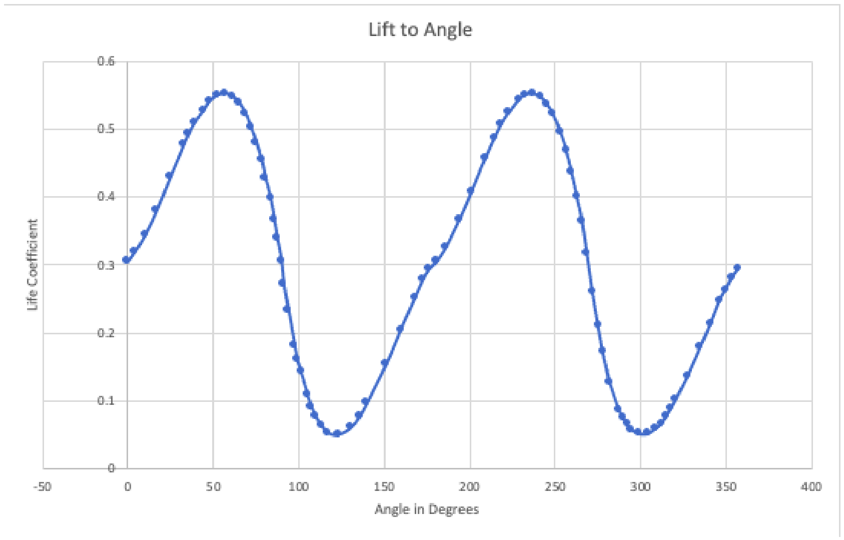


Figure 2: Lift Coefficient Graph from Excel (0°-360°)

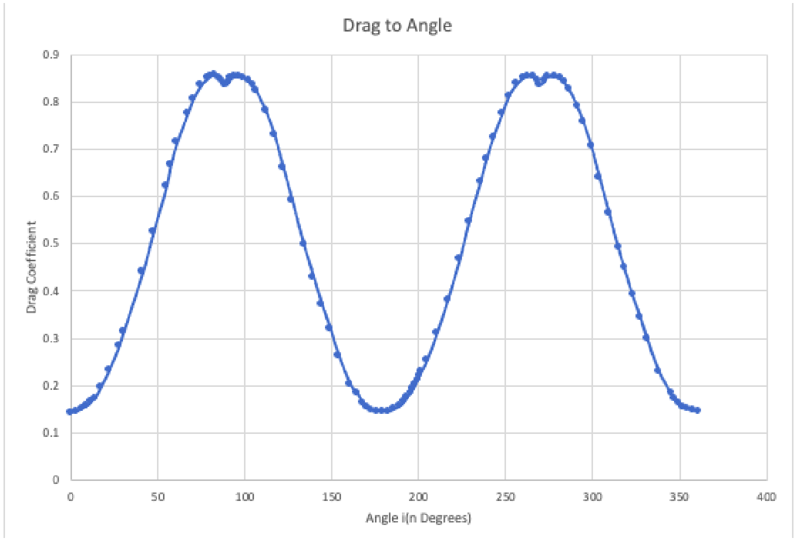


Figure 3: Drag Coefficient Graph from Excel (0°-360°)

Due to the unique shape of an American football, the surface area of the ball is constantly changing as the ball rotates in flight. To solve this problem the team used a three dimensional model of a football created in solidworks to help us calculate the surface area of a football at every degree of an angle from zero to 180 degrees. This solidworks model can be seen below in Figure 4. The ball dimensions were sized to accurately reflect the dimensions of a professional football, and then section viewed to show the total surface area of the ball at a certain degree. An angle of zero degrees can be reflected by the surface area of the short axis of the football, and an angle of ninety degrees is reflected by the long axis of the football. The angle of the ball was adjusted using a reference plane which rotated around the short axis of the ball. Our team calculated a minimum surface area of approximately 0.02234 m² at zero degrees and a maximum surface area of approximately 0.03446 m² meters squared at ninety two degrees as seen below in Figure 1. .

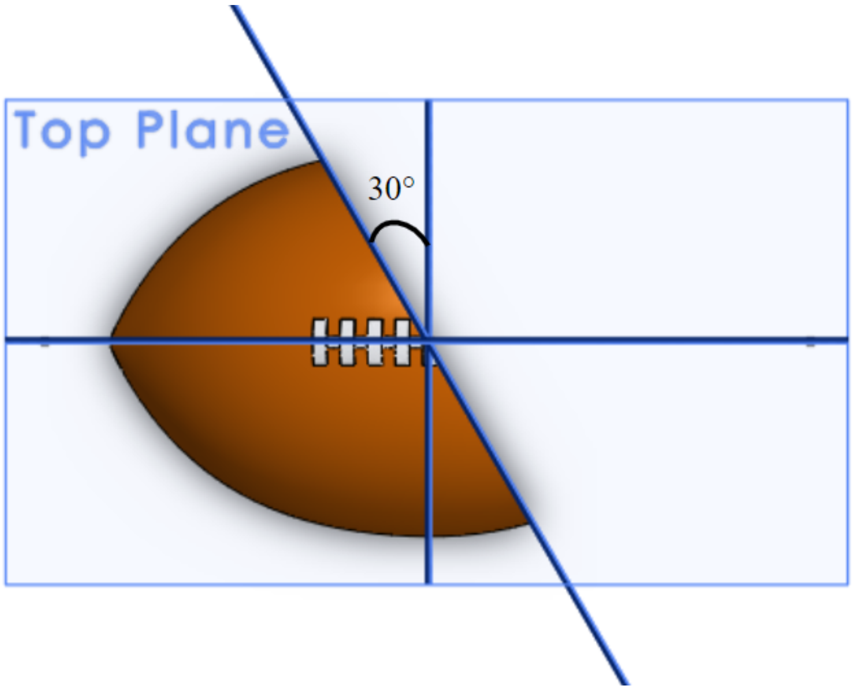


Figure 4: SolidWorks Model of Football

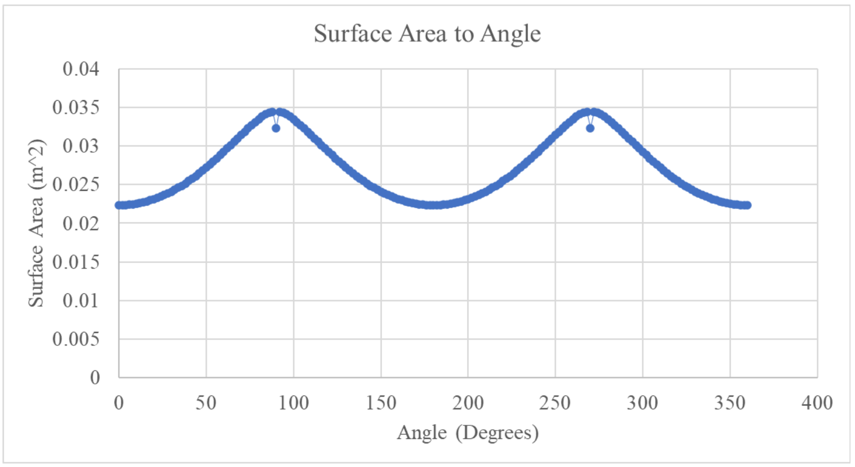


Figure 5: Surface Areas for Each Full Degree

3.3.3 Finding Components of Trajectory Equations

In order to simplify the equation of the trajectory, our team decided to separate the equation into the x and y components. First, the initial velocities in the x and y directions must be found by multiplying the total initial velocity by the cosine and sine of the initial launch angle of the ball as seen in Appendix A. The initial velocities of each component are applied to their respective component velocity equation. Each equation for the x and y velocities has been derived from existing trajectory equations, and has been slightly modified to include the necessary conditions and variables of a rotating football. The various forces acting on the football that our team considered in the velocity equations can be seen in Figure 6 shown below. A 2D X-Y plane is placed on the ball in an upright position where Y is perpendicular to the normal (ground) and X is parallel to the ground. 'θ' (theta) is the initial launch angle from the ground. 'V' represents the initial launch angle. For the forces acting on the ball, gravity is labeled as 'mg', 'F_D' is the drag force and 'F_L' is the lift force acting on the ball.

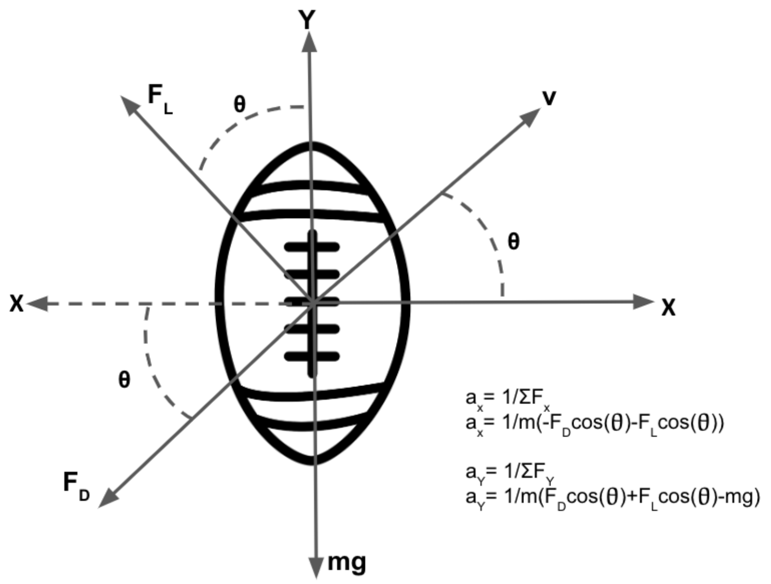

Y
F_L
θ
v
θ
X
X
θ
F_D
mg
$a_x = 1/\Sigma F_x$
$a_x = 1/m(-F_D\cos(\theta)-F_L\cos(\theta))$
$a_Y = 1/\Sigma F_Y$
$a_Y = 1/m(F_D\cos(\theta)+F_L\cos(\theta)-mg)$

Figure 6: Force Diagram on the Football

Within MATLAB, the 'numel' function allows our team to take a defined total amount of time and a desired change in time, and count by that change in time starting at zero until the total time is achieved (Matlab, n.d.). As a result of the team defining the forces on the ball in Figure 6, we were able to develop the following equation for the velocity of the ball in the X direction (1). The team was able to approximate $\frac{dV_x}{dt}$ (acceleration) using Euler's method shown in equation (2) below, where 'i' is the iteration index. We then arrived at the final trajectory equation for the X direction (3). Similarly, the equation for the velocity in the Y direction is shown in equation (4), where the *dt* remains constant at 0.001 seconds (5).

$$\frac{dV_x}{dt} = \frac{-1}{2m} \cdot \rho \cdot A \cdot (C_D \cdot V_x + C_L \cdot V_y) \cdot \sqrt{V_x^2 + V_y^2} \qquad (1)$$

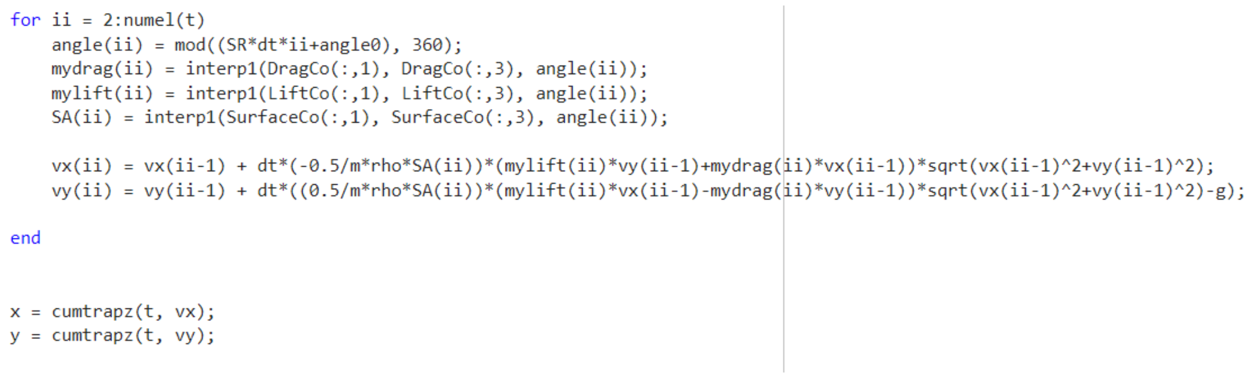
$$\frac{dV_{xi}}{dt} = \frac{(V_{xi+1} - V_{xi})}{(t_{i+1} - t_i)} \qquad (2)$$

$$V_{xi+1} = V_{xi} + dt \cdot (\frac{-1}{2m} \cdot \rho \cdot A_i \cdot (C_{Di} \cdot V_{xi} + C_{Li} \cdot V_{yi})) \qquad (3)$$

$$V_{yi+1} = V_{yi} + dt \cdot (\frac{-1}{2m} \cdot \rho \cdot A_i \cdot (C_{Di} \cdot V_{xi} - C_{Li} \cdot V_{yi}) - g) \qquad (4)$$

$$dt = t_{i+1} - t_i \qquad (5)$$

The initial components of the ball are captured at launch by the camera, and are represented by (V_{xi}, V_{yi}). The remaining flight is calculated using the equation above. C_{Di} and C_{Li} are the drag and lift forces, and A_i is the cross sectional area at each iteration, which are calculated based on the orientation of the ball. The ball orientation is calculated assuming a constant spin rate is detected at launch.



```
for ii = 2:numel(t)
    angle(ii) = mod((SR*dt*ii+angle0), 360);
    mydrag(ii) = interp1(DragCo(:,1), DragCo(:,3), angle(ii));
    mylift(ii) = interp1(LiftCo(:,1), LiftCo(:,3), angle(ii));
    SA(ii) = interp1(SurfaceCo(:,1), SurfaceCo(:,3), angle(ii));

    vx(ii) = vx(ii-1) + dt*(-0.5/m*rho*SA(ii))*(mylift(ii)*vy(ii-1)+mydrag(ii)*vx(ii-1))*sqrt(vx(ii-1)^2+vy(ii-1)^2);
    vy(ii) = vy(ii-1) + dt*((0.5/m*rho*SA(ii))*(mylift(ii)*vx(ii-1)-mydrag(ii)*vy(ii-1))*sqrt(vx(ii-1)^2+vy(ii-1)^2)-g);

end


x = cumtrapz(t, vx);
y = cumtrapz(t, vy);
```

Figure 7: MATLAB Code for Components of Trajectory

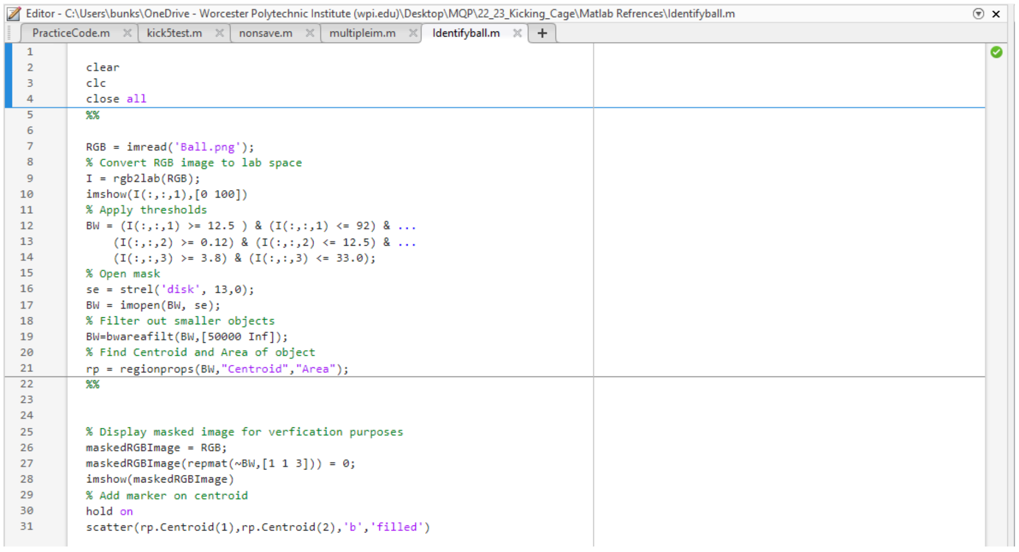
3.4 Objective 2: Gain understanding of MATLAB image processing in order to more accurately predict ball trajectory

This project is based on the team's capability to use MATLAB and utilize the versatility it has as an information processor. Before the project began, the team possessed only basic skills when it came to the MATLAB coding language. This is why the team followed a guided step-by-step approach to building the image processing script that is in use today. Resources such as the MATLAB forums, online instructional videos, the Robotics Department, the Mechanical and Materials Department, and fellow students were utilized to gather crucial information that would aid the team in completing the step-by-step approach.

3.4.1 Image Recognition and MATLAB Color Thresholds

The first step was to understand how to process a still image within MATLAB. During the football season, members of the team were able to capture still pictures of american footballs setup for kicking. The goal of such a step was to be able to upload, isolate, and calculate a centroid on a still frame within the MATLAB script. Forums were utilized for this first step as the team had little to no experience creating such a script that could do what was needed. Figures 8 and 9 show the first iteration of the image processing script and results that would become the basis for the final iteration.

The script begins on line seven which is where the software searches for the file specified (here it is "Ball.png") and is instructed to read it as an RGB matrix. This matrix is then stored in the workspace to be referenced later in the script. Next, the matrix is processed through MATLAB's Color Threshold Lab which can be seen in Figure 10. Each pixel within the image is assigned a certain value based on the RGB values and their intensities that the software can identify. The Color Threshold Lab allows for the user to isolate certain values and intensities; meaning, certain pixels maintain their values while others are assigned the value of zero. Since the current image is being processed as a three dimensional matrix, the team then shifts it into a two dimensional matrix using the command in line seventeen. This means that all the values that were above zero have now been assigned the value of one which creates the two dimensional matrix. Line twenty-one calls for the software to calculate the properties of the region made up of the pixels with the value of one; the properties specifically called for are the centroid and the area. The last of the script (lines 25 to 31) opens and displays the processed image seen in Figure 9.


Editor - C:\Users\bunks\OneDrive - Worcester Polytechnic Institute (wpi.edu)\Desktop\MQP\22_23_Kicking_Cage\Matlab Refrences\Identifyball.m
PracticeCode.m
kick5test.m
nonsave.m
multipleim.m
Identifyball.m

```
1
2   clear
3   clc
4   close all
5   %%
6
7   RGB = imread('Ball.png');
8   % Convert RGB image to lab space
9   I = rgb2lab(RGB);
10  imshow(I(:,:,1),[0 100])
11  % Apply thresholds
12  BW = (I(:,:,1) >= 12.5 ) & (I(:,:,1) <= 92) & ...
13      (I(:,:,2) >= 0.12) & (I(:,:,2) <= 12.5) & ...
14      (I(:,:,3) >= 3.8) & (I(:,:,3) <= 33.0);
15  % Open mask
16  se = strel('disk', 13,0);
17  BW = imopen(BW, se);
18  % Filter out smaller objects
19  BW=bwareafilt(BW,[50000 Inf]);
20  % Find Centroid and Area of object
21  rp = regionprops(BW,"Centroid","Area");
22  %%
23
24
25  % Display masked image for verfication purposes
26  maskedRGBImage = RGB;
27  maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
28  imshow(maskedRGBImage)
29  % Add marker on centroid
30  hold on
31  scatter(rp.Centroid(1),rp.Centroid(2),'b','filled')
```

Figure 8: MATLAB Script to Identify the American Football and its Centroid

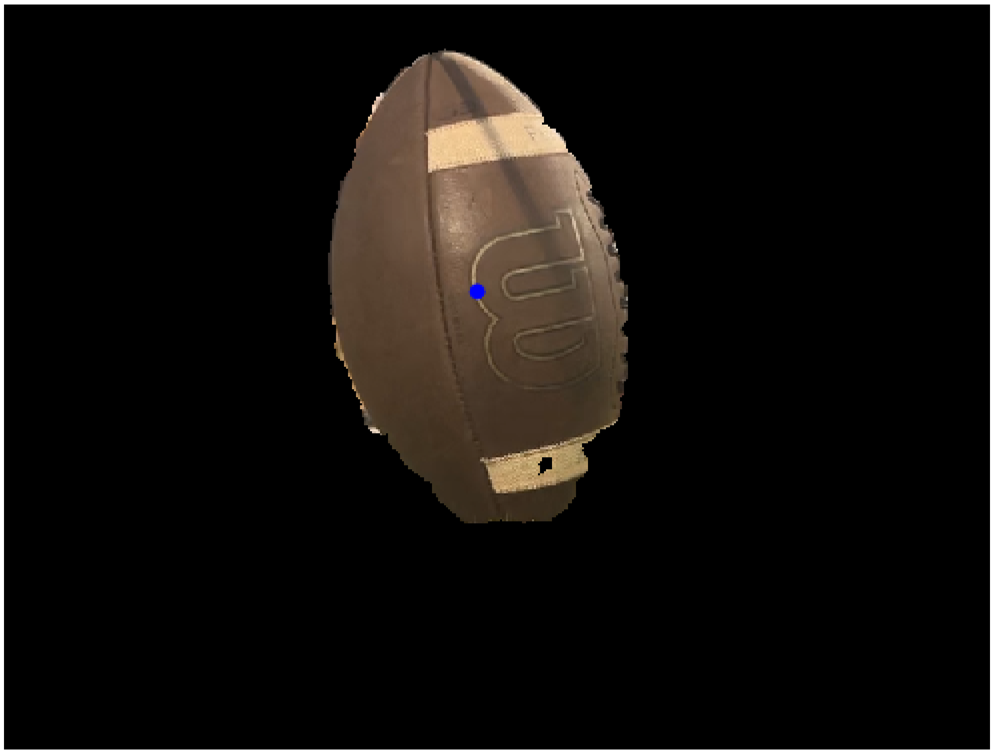
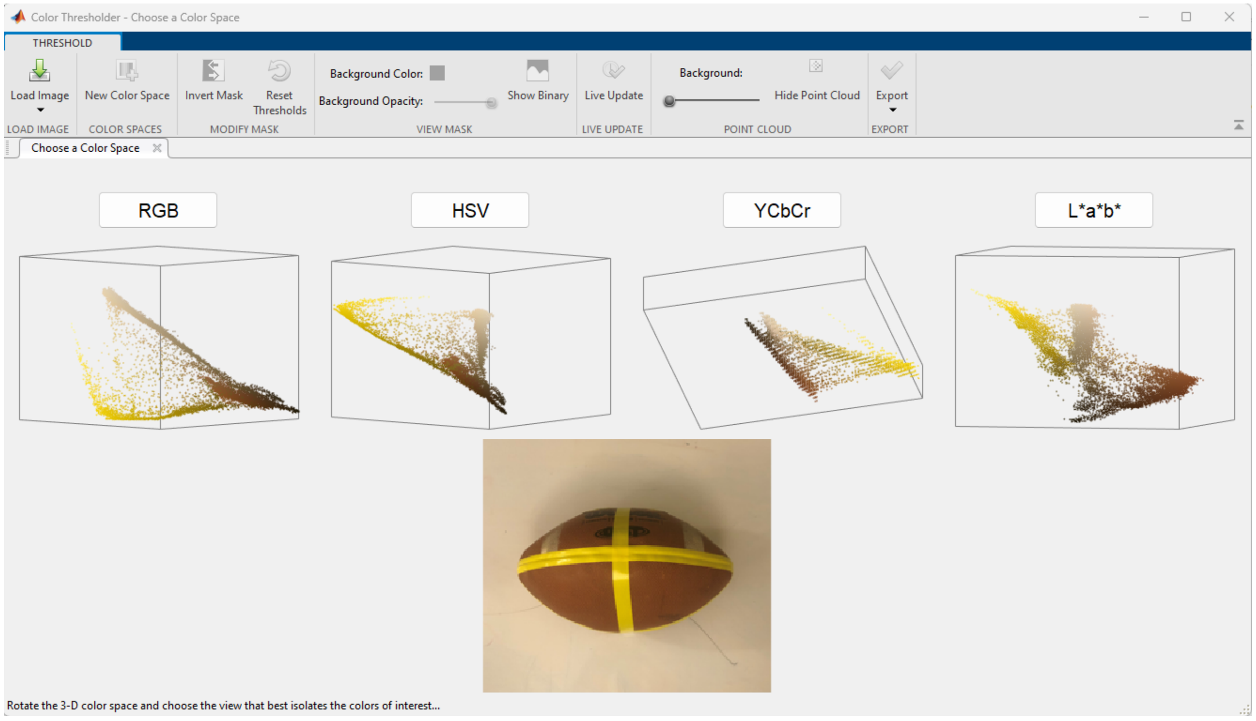


Figure 9: The Result of the First Image Processing Script which Isolates the Football and Locates the Centroid with a Blue Indicator

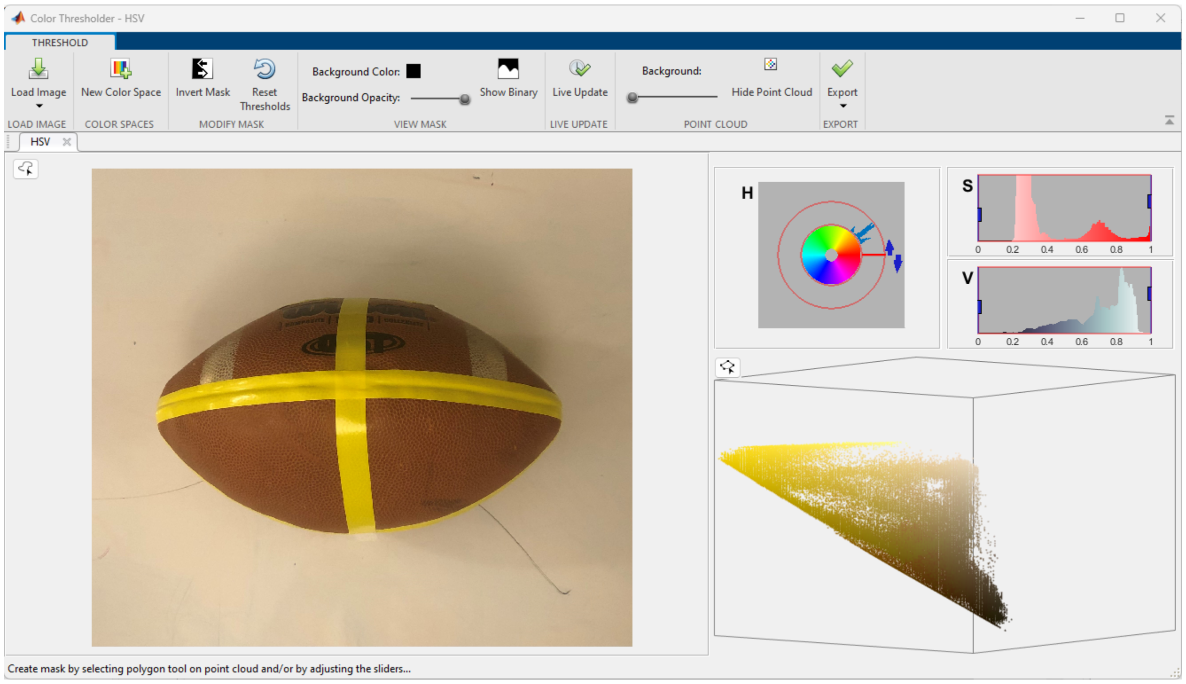
Once this first iteration was completed, the next step was to experiment with other still frames in order to verify if the script could be repeated. The team found that the current thresholds would not work for filtering other still frames of the football. Knowing this, the team started to investigate the complete capabilities of the MATLAB Color Thresholder (Figure 10). Each set of intensity filters can be utilized in different situations based on what someone is trying to isolate within an image. While using the still frames with an unmodified football, the team was unable to successfully isolate the ball which led us to look into other options. Options that were reviewed by the team were placing neon electric tape along the seams of the ball and painting the entirety of the ball. Both options would allow the team to place a distinctive color that could be easily isolated using the Color Threshold Lab. Using the decision matrix featured in Appendix C, the team decided to use the neon electric tape along the seams of the ball. This would allow for minimum modification and negligible changes to properties of the ball.

Through experimentation and recommendations by peers, the team found that the HSV intensity filter would work best for isolating the neon yellow electric tape. The HSV intensity filter features a color wheel and two intensity scalers that allow for specific color isolation which can be seen in Figure 11. The team has acknowledged that isolating the tape does not mean that the entire ball will be identified, but due to the specific placement along the seams, the critical dimensions of the football are able to be recorded. Thresholds produced by the HSV filter were then tested with multiple still frames to ensure their use in the next iteration of the MATLAB script. After verifying the new thresholds, the team decided to convert the three dimensional matrix into a two dimensional matrix. This would shift the frame from color to just black and white which makes later processing much simpler. Instead of trying to process three different numbers, now the software could process just ones and zeroes. The script lines in which this occurs can be seen in Figure 12.



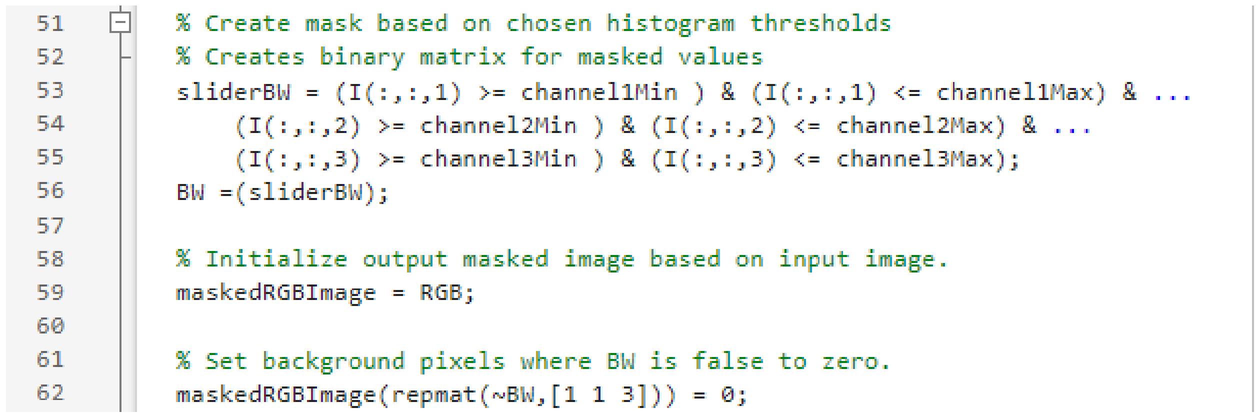
Rotate the 3-D color space and choose the view that best isolates the colors of interest...

Figure 10: The Labspace of the Color Thresholder within MATLAB Displaying the Four Different Filters Available



Create mask by selecting polygon tool on point cloud and/or by adjusting the sliders...

Figure 11: Workspace of the HSV Intensity Filter, Featuring the Color Wheel and Intensity Scales



```
% Create mask based on chosen histogram thresholds
% Creates binary matrix for masked values
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
BW =(sliderBW);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
```

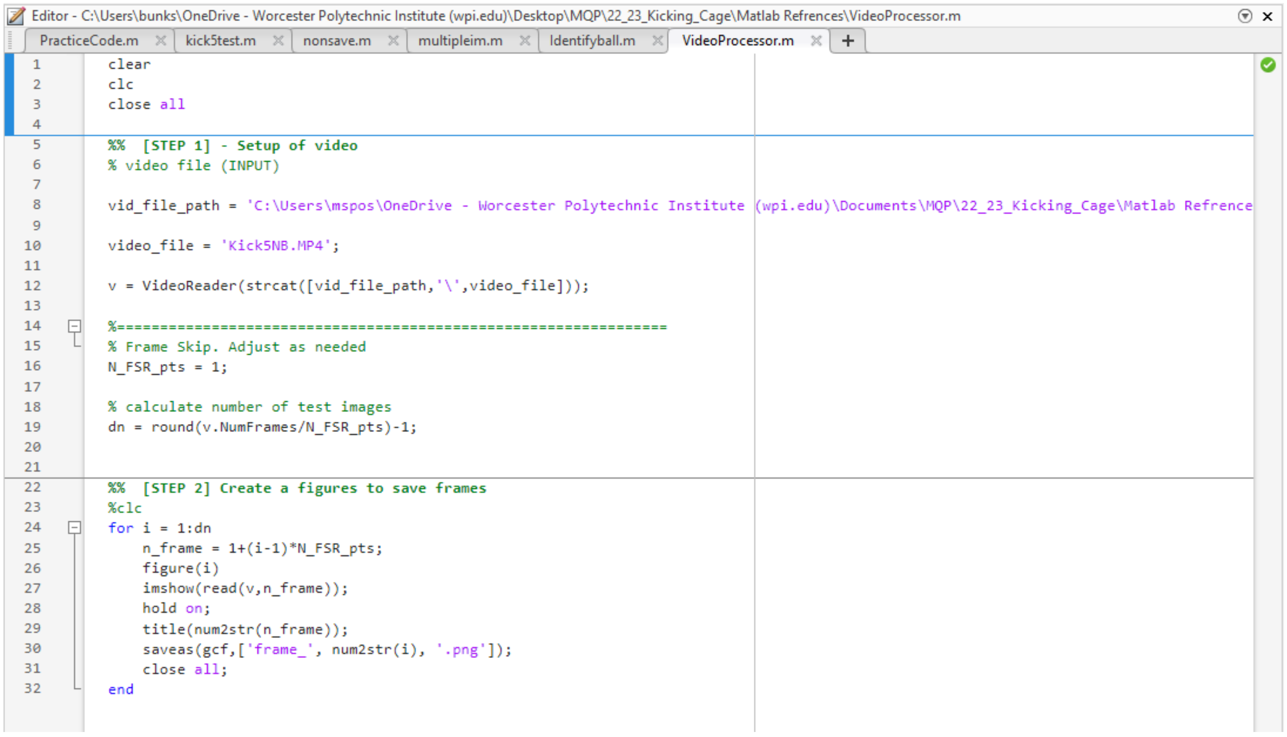
Figure 12: Script where the Three Dimensional HSV Matrix is Transformed to Two Dimensional Black and White

3.4.2 MATLAB Video Processing

Video processing deals with breaking down the uploaded video into individual frames. These individual frames are what the team uses to evaluate the ball's location during the kick. Fortunately, the team was able to source a video processing code from a team member's previous class. The code – seen in Figure 13 – is an original creation along with lines 30 and 31 which were added by the team. These lines allow MATLAB to save each figure created to a certain folder with an assigned name on the user's device which the team was able to recall later. The script is unable to run if lines 8 through 12 do not have the correct pathway; meaning the file path must be defined correctly so MATLAB knows which folder to access and what individual file to extract. Through the use of this script, the team found that it works best when recalling an MP4 or an MOV file.

This script also allows for the user to specify the amount of frames they want to capture. Lines 16 and 19 include a frame skip and frame count respectively. Utilizing these lines, the team was able to shorten the time of processing by skipping any specified amount of frames throughout the video. Skipping amounts used by the team were often 1, 3, 5, or 10 frames at a time; one at a time is best for accurate data points while ten is best for troubleshooting scripts where processing comes before the problem. Later, the script follows a for-loop which is where each frame is actually processed. Each frame that is created is then transformed into a figure that can be saved and named for recall afterwards using the "saveas" function.

The final iteration follows the basis founded by the first iteration. The video is broken down into each individual frame and then stored in the workspace under the variable, TotalFrames. Storing the information as a variable allows for the team to avoid saving multiple hundreds of frames to the computer for each video. This in turn cuts down on processing time and makes the code much more efficient. Once stored, the team is able to recall the variable and analyze each frame for the centroid in the following for-loop.


Editor - C:\Users\bunks\OneDrive - Worcester Polytechnic Institute (wpi.edu)\Desktop\MQP\22_23_Kicking_Cage\Matlab Refrences\VideoProcessor.m
PracticeCode.m
kick5test.m
nonsave.m
multipleim.m
Identifyball.m
VideoProcessor.m

```
clear
clc
close all

%%  [STEP 1] - Setup of video
% video file (INPUT)

vid_file_path = 'C:\Users\mspos\OneDrive - Worcester Polytechnic Institute (wpi.edu)\Documents\MQP\22_23_Kicking_Cage\Matlab Refrence

video_file = 'Kick5NB.MP4';

v = VideoReader(strcat([vid_file_path,'\',video_file]));

%================================================================
% Frame Skip. Adjust as needed
N_FSR_pts = 1;

% calculate number of test images
dn = round(v.NumFrames/N_FSR_pts)-1;


%%  [STEP 2] Create a figures to save frames
%clc
for i = 1:dn
    n_frame = 1+(i-1)*N_FSR_pts;
    figure(i)
    imshow(read(v,n_frame));
    hold on;
    title(num2str(n_frame));
    saveas(gcf,['frame_', num2str(i), '.png']);
    close all;
end
```

Figure 13: MATLAB Video Processing Code Sourced from Previous Classes with Few Modifications

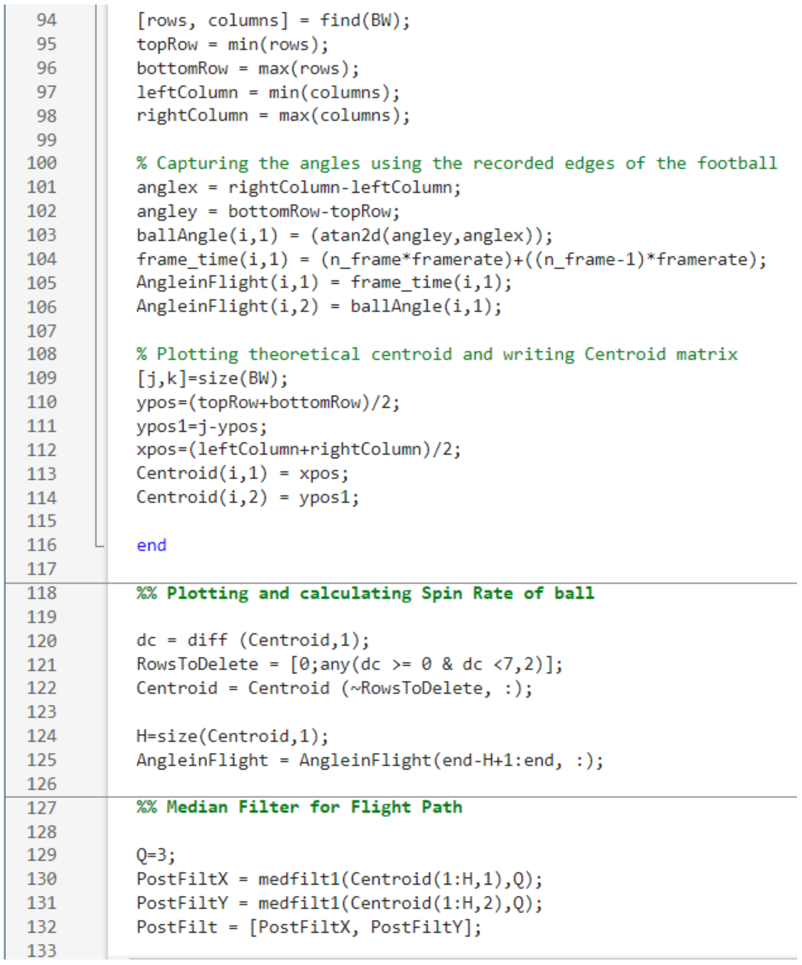
3.4.3 Locating and Graphing the Centroid of the Ball

Tracking the centroid of the ball allows for the team to calculate initial velocity, launch angle, and spin rate which are needed in predicting the trajectory. Many iterations of the centroid defining portion were tried until deciding on the final iteration seen in Figure 14. The accuracy of the centroid is dependent upon the range provided by the color thresholds. Through experimentation, the team was able to determine that tighter thresholds would produce less varying numbers for each of the specified variables: initial velocity, launch angle, and spin rate.

Tightening the thresholds ensures that the program will not experience rogue pixels that would inevitably distort the centroid calculations. MATLAB already has an embedded centroid calculating function which was used in the team's earliest iterations. Although more convenient, the centroid function proved to be inaccurate for a larger portion of the recorded frames than was desired. This is when the team decided to move to another option that utilizes basic functions and mathematics within MATLAB. Lines 93 to 98 record the boundaries presented by the recognition of the color thresholds. These lines allow for MATLAB to search the binary matrix for the first change in value in each direction. For example, "leftColumn" measures the first change in value from 0 to 1 and records this column number as the left side of the boundary. Once the boundaries are set, lines 108 to 114 use rudimentary mathematics in order to produce a synthetic centroid. In order to verify the legitimacy of the synthetic centroid, the team plotted both the MATLAB computed centroid and the synthetic centroid on its corresponding frames that were known to be the correct computed centroid (Figure 15). From experimentation, the team found that the synthetic centroid was consistently within a sufficient deviation of about five pixels. The centroid's x- and y-position is then recorded into the Centroid matrix.

Looking deeper into lines 108 to 114, the team decided to create their own matrix – [j,k] – for the synthetic centroid of the same size as the BW matrix. Originally when the centroid would be calculated and graphed, it was shown that the ball was consistently moving negatively in the y-direction as it moved positively in the x-direction. This was due to MATLAB placing the origin at which the matrix was based to the top-left of the frame. Therefore lines 109 and 111 were created to counteract the effects. Line 109 creates the new matrix, the size of the BW matrix and line 111 essentially flips the matrix so that the centroid is calculated from the bottom-left as the origin. Flipping the matrix allowed for the team to produce positive changes in both the x- and y-directions which in turn made calculations easier and more accurate afterwards.

Following the calculation of the synthetic centroid, the team added two filters, lines 120 to 122 and lines 127 to 132. Through the first iterations, the team still experienced the occasional outliers from the centroid calculations and plotting. These outliers would ultimately interrupt calculations of the initial velocity and launch angle. The median filter – lines 127 to 132 – is able to filter out these outliers and create a more accurate matrix of the centroid named PostFilt. The filter seen in lines 120 to 122 has been implemented in the latest iteration in order to address the issue of the ball not moving in the beginning frames. The videos are recorded in 240 fps which means the beginning frames produce no real change in the centroid position; this can interrupt calculations the same as the outliers. Eliminating the frames that produce no change in position allows for the team to more accurately compute the initial velocity and launch angle. The filtered graph of the centroids is shown below in figure 16



```
    [rows, columns] = find(BW);
    topRow = min(rows);
    bottomRow = max(rows);
    leftColumn = min(columns);
    rightColumn = max(columns);

    % Capturing the angles using the recorded edges of the football
    anglex = rightColumn-leftColumn;
    angley = bottomRow-topRow;
    ballAngle(i,1) = (atan2d(angley,anglex));
    frame_time(i,1) = (n_frame*framerate)+((n_frame-1)*framerate);
    AngleinFlight(i,1) = frame_time(i,1);
    AngleinFlight(i,2) = ballAngle(i,1);

    % Plotting theoretical centroid and writing Centroid matrix
    [j,k]=size(BW);
    ypos=(topRow+bottomRow)/2;
    ypos1=j-ypos;
    xpos=(leftColumn+rightColumn)/2;
    Centroid(i,1) = xpos;
    Centroid(i,2) = ypos1;

    end

%% Plotting and calculating Spin Rate of ball

dc = diff (Centroid,1);
RowsToDelete = [0;any(dc >= 0 & dc <7,2)];
Centroid = Centroid (~RowsToDelete, :);

H=size(Centroid,1);
AngleinFlight = AngleinFlight(end-H+1:end, :);

%% Median Filter for Flight Path

Q=3;
PostFiltX = medfilt1(Centroid(1:H,1),Q);
PostFiltY = medfilt1(Centroid(1:H,2),Q);
PostFilt = [PostFiltX, PostFiltY];
```

Figure 14: Final Iteration of the Centroid Code used to Plot Data Afterwards Featuring both Filters

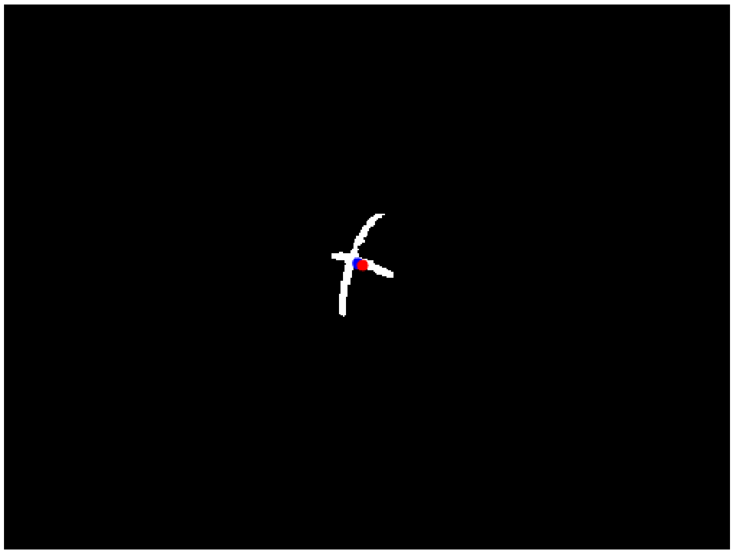


Figure 15: Both Function Calculated (Blue) and Synthetically Calculated (Red) Centroids on the Identified Ball Area

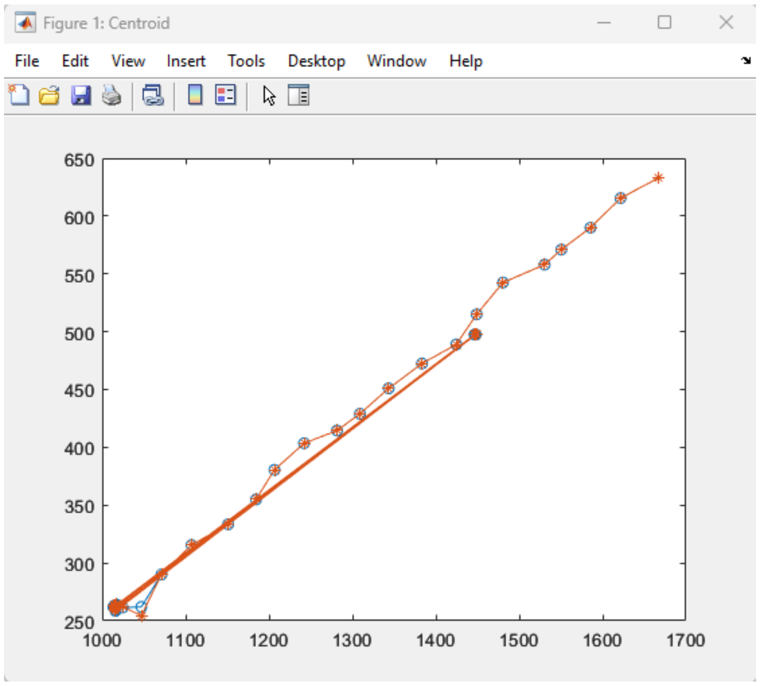


Figure 16: Graph Plotting the Centroid and PostFilt Matrices. Centroid is Denoted by the Blue Circles and PostFilt in the Orange Asterisks. X-Axis is the X-Position and Y-Axis is the Y-Position.

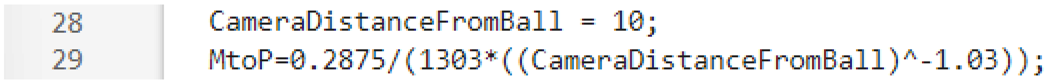
3.4.4 Calculating Initial Conditions: Velocity, Launch Angle, & Spin Rate

In order for the team to accurately predict the trajectory of the football the initial conditions – velocity, launch angle, and spin rate – need to be calculated. The first step of the process was to create an accurate conversion factor for pixels to meters. Creating this conversion (Figure 17, lines 28 and 29) factor ensures that each result of each condition is realistic to what was measured. The equation was derived from the known distance between the camera and the ball, as well as the portrayed yard lines in each frame of reference. On average, each frame would portray five yards when the camera was ten feet away from the ball; this average can change as the distance between the camera and ball changes.

Once the conversion factor was defined, the team was able to move on to calculating the initial conditions starting with velocity. Calculating velocity was based on the PostFilt matrix. The difference between each recorded point through flight is taken, divided by the frame rate, and then multiplied by the conversion factor (lines 136 and 137); this gives us the velocity in both x- and y-directions at each point of the ball's flight. Afterwards, the pythagoras theorem is utilized to attain the instantaneous velocity of the football (lines 144 to 148, and 158). The matrix, "V", of x- and y-direction velocities was used in the calculation of the initial launch angle. The matrix "Angle" was created by taking the arc-tangent of each column within the velocity matrix. Afterwards the mean of "Angle" is calculated to produce a singular number in degrees; this number is then converted to radians for accurate calculations and saved as a new variable, theta.

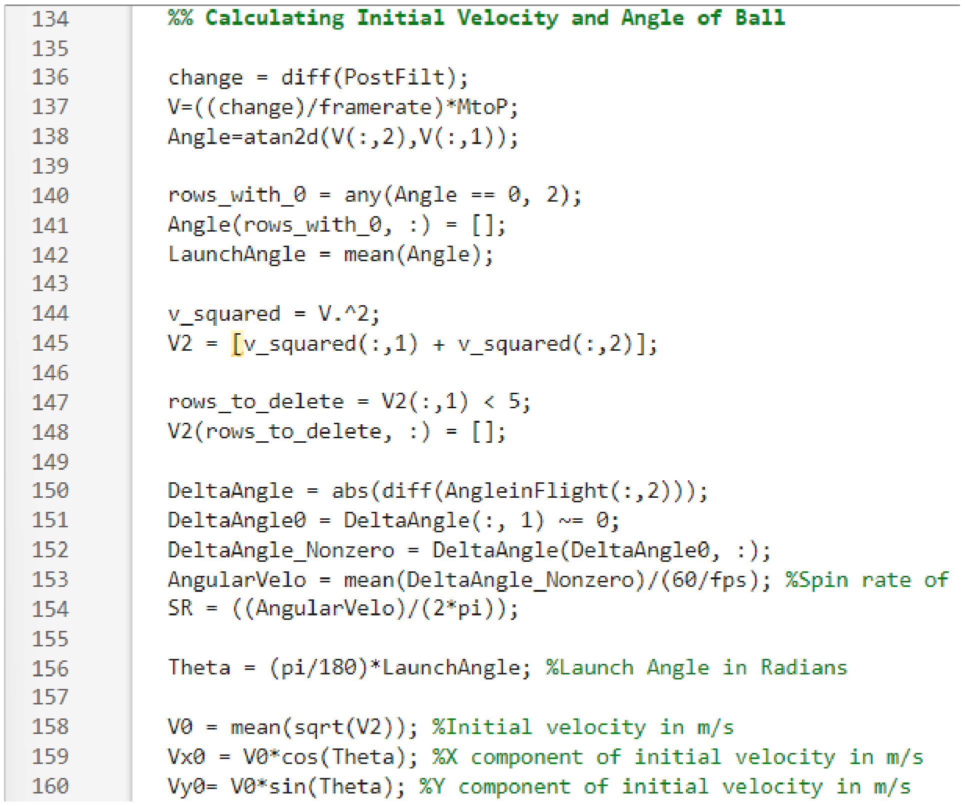
Spin rate is crucial when calculating the experimental trajectory of a football as it affects the corresponding lift and drag coefficients. As the ball spins, the surface area changes and in turn changes the lift and drag coefficients. First the angle of the ball itself is calculated for each frame and stored in the matrix AngleinFlight (lines 101 to 106). This matrix features the time of the frame in column one and the angle of the ball at that time in column two. In lines 150 to 154 the difference between each angle in column two of the AngleinFlight matrix is taken. This delta angle is then used to find the angular velocity of the football during flight which translates to the spin rate found in line 154.

Converting each of these initial conditions – velocity, launch angle, and spin rate – into their magnitudes, rather than their components, allowed for the team to simplify the later calculations. This conversation also allows for a simpler user interface in future iterations of the project. Our list of initial conditions are shown below in figure 18.



```
28    CameraDistanceFromBall = 10;
29    MtoP=0.2875/(1303*((CameraDistanceFromBall)^-1.03));
```

Figure 17: Lines 28 and 29 Featuring the Conversion Factor Calculations



```
%% Calculating Initial Velocity and Angle of Ball

change = diff(PostFilt);
V=((change)/framerate)*MtoP;
Angle=atan2d(V(:,2),V(:,1));

rows_with_0 = any(Angle == 0, 2);
Angle(rows_with_0, :) = [];
LaunchAngle = mean(Angle);

v_squared = V.^2;
V2 = [v_squared(:,1) + v_squared(:,2)];

rows_to_delete = V2(:,1) < 5;
V2(rows_to_delete, :) = [];

DeltaAngle = abs(diff(AngleinFlight(:,2)));
DeltaAngle0 = DeltaAngle(:, 1) ~= 0;
DeltaAngle_Nonzero = DeltaAngle(DeltaAngle0, :);
AngularVelo = mean(DeltaAngle_Nonzero)/(60/fps); %Spin rate of
SR = ((AngularVelo)/(2*pi));

Theta = (pi/180)*LaunchAngle; %Launch Angle in Radians

V0 = mean(sqrt(V2)); %Initial velocity in m/s
Vx0 = V0*cos(Theta); %X component of initial velocity in m/s
Vy0= V0*sin(Theta); %Y component of initial velocity in m/s
```

Figure 18: The Initial Condition Calculation Section

3.5 Objective 3: Analyze the data collected through image processing and implement general formulas previously defined to predict the trajectory of the kick

A large step to complete this project is using MATLAB and the code from Objective 2 to then give values to the motion of the ball. These values are crucial to then plug into the equations that the team found earlier to map the flight of the ball through the air. These equations are still being refined and will ultimately be revised enough and then typed into MATLAB so that the program can interpret and calculate all in one script. Overall, this objective is very important to tie the entire project together and have the program run seamlessly.

3.5.1 Combining Calculated Variables Through Image Processing With Trajectory Calculations

The final step to get the team's code to work fluidly and efficiently is to combine the image processing code results with the trajectory calculation code. The integration of both of these codes requires the image processing code to provide the three main metrics discussed above; initial velocity, launch angle, and spin rate. Once these three variables are successfully calculated, the trajectory code receives the information as the initial conditions and predicts the flight of the ball. The combination of these two codes is a crucial step in the success of the smart kicking net.

The initial velocity of the kick is identified in the code by the variable V0. This velocity is then multiplied by the cosine of the launch angle and the sine of the launch angle to get the initial components of each velocity in the x-direction and the y-direction. The components of each initial velocity are represented by the variables Vx(ii-1) and Vy(ii-1) at the beginning of each velocity equation.

Lastly, the team inserts the average spin rate of the ball in the air into an equation which calculates the angle in degrees at each instant of time. That angle is then associated with a surface area in the spreadsheet created from the SolidWorks model of a football.

3.5.2 Experimental Setup

Shown below in Figure 19 is the experimental setup. Within Table 2 all of the components are listed. The one component that is not pictured is the laptop running MATLAB off to the side.

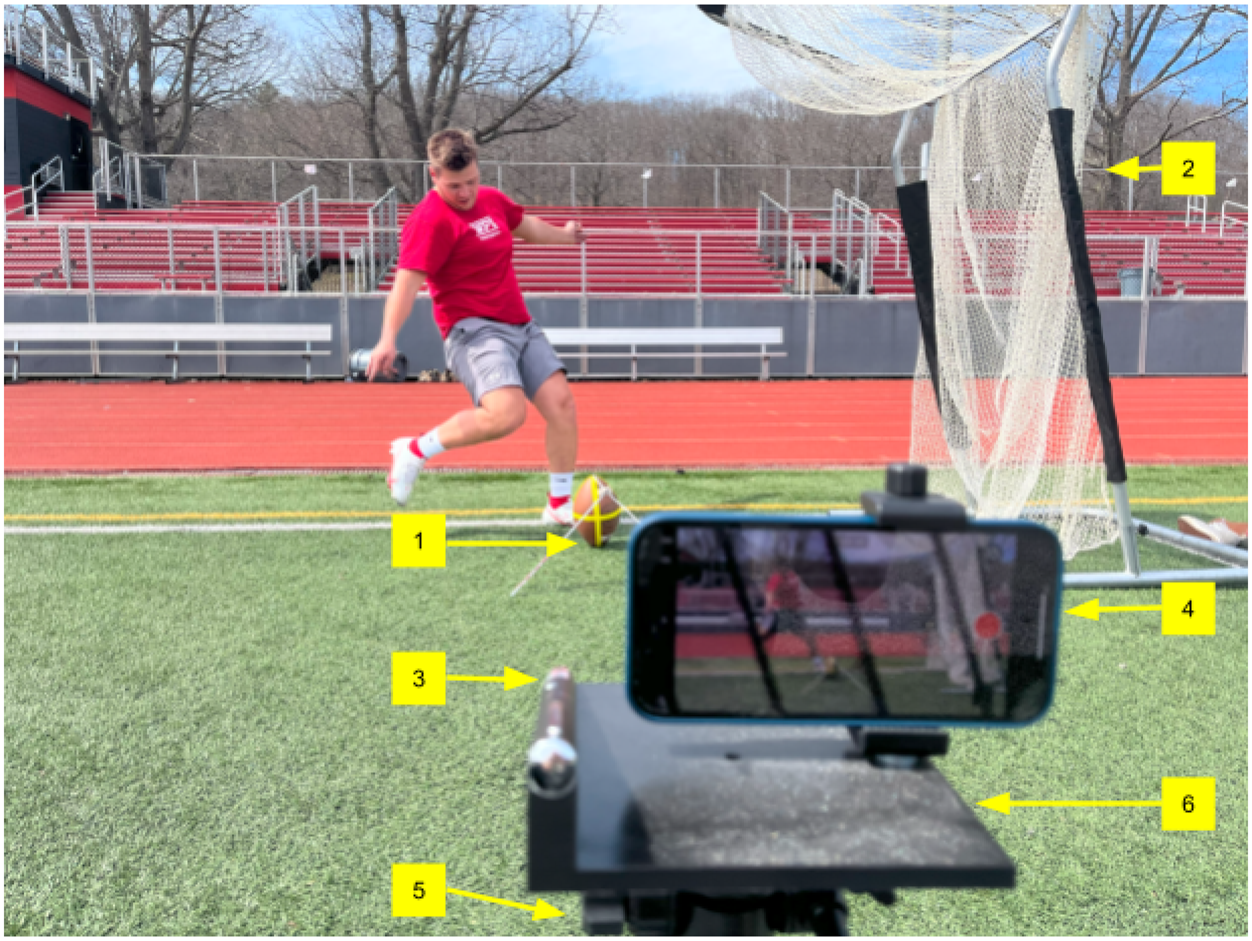


Figure 19: Experimental Setup on a Sideline

Table 2: Bill of Materials for Experimental Setup

Item Number	Name
1	Modified Football
2	Kicking Net
3	Laser for Distance Reference
4	iPhone for Video Capture
5	Tripod
6	3D Printed Plate to Hold iPhone and Laser
Not Pictured	Laptop Running MATLAB Code

This experimental setup allows data to be collected and then for the video to be manually edited (editing is optional). Once the video is captured the video is uploaded to OneDrive so that the video can be referenced and analyzed by MATLAB. The purpose of the laser is to give the user a reference point to place the football exactly 10ft away from the camera as mentioned above.

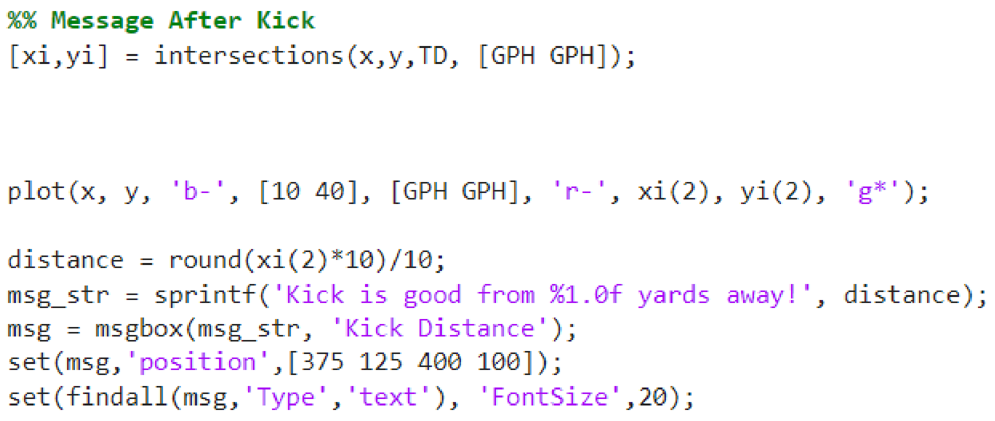
4. Results and Discussion

4.1 Plotting Trajectory Equation and Final Messaging

To create the final trajectory, the team first plotted the positions of the x and y components of the football for each instance of time. This allowed us to create a final trajectory arc which includes the initial launch angle of the ball, the spin rate, lift coefficient, drag coefficient, and surface area at each instance of time.

A horizontal line was then created on the same plot, 3.28 yards above the axis to represent the height of an American football goal post crossbar. The height of this line is represented by the variable GPH. The variable TD creates a sufficient range for the trajectory arc created by the kick to fall within. As a result, an intersection point is created between the calculated trajectory arc, and the height of the average american goal post. For the purposes of the project, the team allowed this intersection point to represent the first distance the kick would have entered the field goal post.

Downloaded from the MATLAB file exchange, the team installed the MATLAB intersections function as seen below in Figure 20. The intersections function allowed the team to locate the point of intersection between two lines, and plot that point on the same plot as the trajectory line and the cross bar line. The intersection point can be identified by a small green asterisk on the plot. The x component of the intersection point is inserted into a message string which will display the message: “Kick is good from X yards away!” The distance of the intersection point will change according to the variables calculated through the image processing of a video recording of a kick. This will allow American football kickers to identify how far away they could have made a field goal, as well as the total distance their kicks travel before hitting the ground. Figure 21 shows an example of the final message and trajectory prediction.



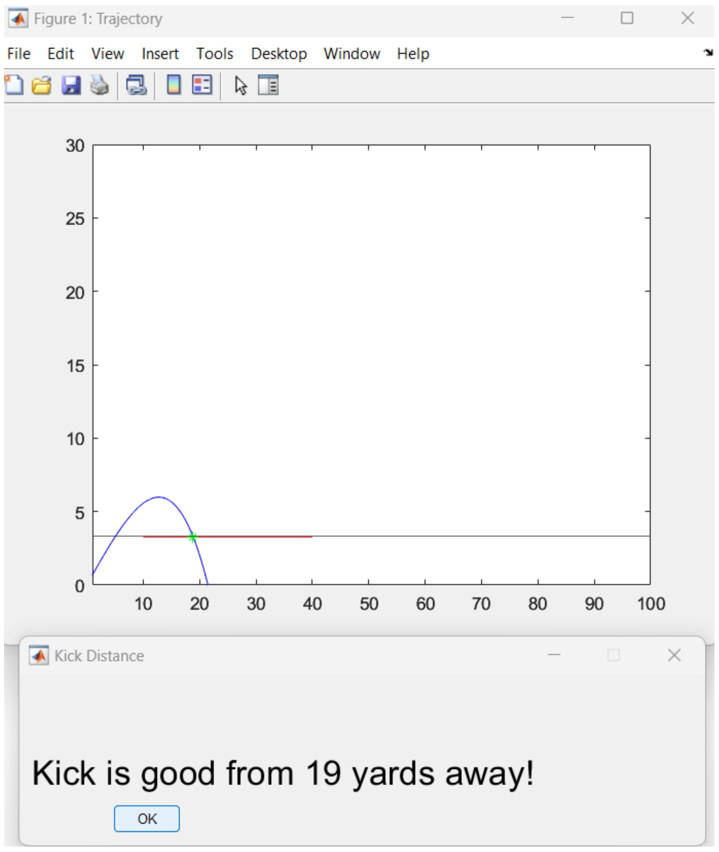
```
%% Message After Kick
[xi,yi] = intersections(x,y,TD, [GPH GPH]);



plot(x, y, 'b-', [10 40], [GPH GPH], 'r-', xi(2), yi(2), 'g*');

distance = round(xi(2)*10)/10;
msg_str = sprintf('Kick is good from %1.0f yards away!', distance);
msg = msgbox(msg_str, 'Kick Distance');
set(msg,'position',[375 125 400 100]);
set(findall(msg,'Type','text'), 'FontSize',20);
```

Figure 20: Code for Messaging



Kick is good from 19 yards away!

Figure 21: Final Result

4.2 Test Kick Results

Our team conducted a confidence test of the code by recording sixty different kicks on a football field, and recording which yard line the kick started on, and which yard line the kick ended on. Each video would then be processed in MATLAB and the initial velocity, launch angle, and spin rate calculated from the first half of the code, would produce the final trajectory plot relayed back to the kicker. Our team decided that the accuracy of the code would be best determined by recording the difference in distances of each kick recorded.

The observed distances, calculated distances, and differences were then used to create a histogram to better observe how accurate the code could predict the trajectory. The absolute differences are shown in Figure 22. As shown below, fifty percent of kicks were within four yards of the observed kick distance, and eighty eight percent of kicks were within ten yards of the observed kick distance.

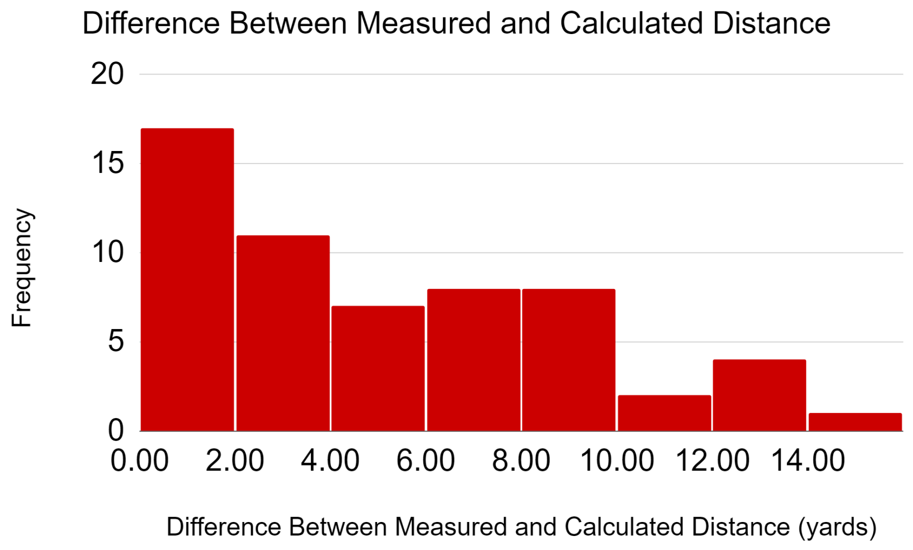


Figure 22: Absolute Difference Between Measured and Calculated Distance of Each Kick in Yards

In addition to this test, the team also found the RSME (Root-Mean-Square Error) of the sixty kicks. The RMSE is a statistical measure to determine the difference between predicted and actual values. The equation to find can be seen below in Figure 23. In this test, the team evaluated the difference between the recorded values and the values from the MATLAB code. The RMSE for these data sets was 6.53 yards. On average, the predicted kicks were 6.53 yards from the actual kicks.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Figure 23: RMSE Equation

4.3 Broader Impacts

4.3.1 Engineering Ethics

Throughout the project we wanted to create a product that can better human life. Some people kick field goals for fun and some kick for a job. The team worked to produce a project that would be as accurate as possible inorder to leave out any doubt of its capabilities. We want everyone to be able to use the product as they see fit which is the reasoning behind our price point. This product should be affordable for all parties instead of major corporations.

4.3.2 Societal and Global Impact

This project was completed in the mindset that it would be available to anyone that took interest in the product. SmartNet can be advertised to anyone from an individual level to the corporate level. Those who would like to use this as an at-home simulator and NFL teams alike would be able to afford this product all the same. Having a lower price point allows for everyone to be included in the venture to be a better athlete and to just have fun. Looking at a global scale, many countries have an american football league that would also be able to afford this product. The product mostly impacts the athlete culture and society which can be participated in by anyone of any ethnicity and culture.

Considering health impacts, there are also little to none. Those who choose to use this product are using it at their own discretion. Our intentions are not to create a product that may cause an injury most likely to the leg or foot. There is a risk that when training one may find themselves in the midst of an injury as a result of their bodily mechanics, but this result is not from the product.

4.3.3 Environmental Impact

This project is mainly a MATLAB code based with few physical components. The code component does not impact the environment in any manner. The only possible environmental impact of this project is where the kicking net is set up. If the kicking net is placed on grass, it may slightly impact the grass it sits on. But overall, the core of this project, the MATLAB code, does not make any environmental impacts.

4.3.4 Codes and Standards

Within this project there are no codes or standards that we have to abide by. There is no impact to the environment or the users body so there would be no regulations around our product. The only standard that the team strives for is accuracy and reliability. The team aimed to make this product reliable and accurate so that the end user is getting factual information not just a number that may be incorrect. Overall, the team did not conform to any codes or standards since the project did not require them.

4.3.5 Economic factors

This product in its current state would be sold for 120 to 150 dollars per unit. This includes a tripod, a specialized phone holder (for video acquisition), a laser, a ball holder, and a kicking net. Currently, there are no other competitors with our type of system except for TrackMan, which could develop a product that has the same capabilities. The cost of all the materials is around 100 dollars making it easily marketable even with our price increase to allow a profit. The software would be distributed through a subscription with our basic package costing around 10 dollars a month, there are future plans to add more packages for a higher price. This product can be sold to professional, college, and high school teams but has the ability to be purchased by a single user for practice in their own space.

This product could also affect the sports gambling arena. Using our product can allow for the kickers and coaches of a team to be more confident in their decisions to make a field goal attempt. The messages from the product have the capability to be passed on to sports reporting networks and sports books which would impact the odds that are set for the current bets. People may be more or less enticed to make a bet when the odds change based on the new information provided by our product. In turn, our product has the capability to add an entirely new dimension to the sports betting scene.

4.4 Future Additions

Although the team is extremely proud of the progress that has been made, there is still much that needs to be accomplished in order for this project to be complete. The current prototype uses one camera which is directed at the kicker from a side angle. However, the team would like to add an additional elevated camera behind the kicker. This would allow the team to capture an additional angle and would allow the creation of a three-dimensional plot rather than a two-dimensional plot which could alert the kicker if their kick was too far left or too far right. Similarly, the addition of a wind sensor would assist in the accuracy of how far left or right the kick would go within this additional dimension.

In addition, adjusting the way the football is processed in the video would result in more consistent trajectory plots. The color thresholds within MATLAB have granted the team a great understanding of the variables that need to be calculated from the video, as well as what aspects of the ball must be identified. Unfortunately, these color threshold sliders must be adjusted for each video to properly account for changes in lighting. If a threshold is found for one video, and used for another, it is highly probable that MATLAB will recognize additional colors within the new video that may not have been present in the previous video. These additional colors may cause errors in the calculations such as a negative launch angle, or a high initial velocity. Ultimately, the team would like to use a higher powered camera, capable of object tracking with minimum motion blur. This would allow the team to eliminate the need to consistently adjust the color threshold sliders, and would allow us to create a more efficient system.

Following the addition of a second camera, wind sensor, and an improved camera, the team would like to conduct further field tests to increase the sample size of the data collection, and refine the histogram results. With the improved features and capabilities the team expects the results to be much more accurate, with approximately eighty percent of kicks falling within four yards of the observed distance.

5. Conclusion

The National Football League has taken tremendous strides in developing new technologies to improve player safety and increase viewership. Many companies have worked on developing new helmets and padding for players. The league has also created many ways to keep viewers interested and engaged at all points throughout the games. The work that has been done around the sport has inspired many people, including this team, to create new and better technologies to help athletes and fans.

Although the project is in an early development stage, the team has made great strides in creating a strong foundation for creating a smart kicking net to help improve the performance of kickers. This idea has been filed and accepted for a provisional patent. In the future the team will aim to develop the first full prototype with some of the proposed future additions. The net will utilize the MATLAB code the team has created, and a camera video recording of a kick going into a net to supply the initial angle and velocity for the code. Our hope is that the team will take that next step in developing a new technology to help the National Football League and the sport in general.

Bibliography

Acquavella Aug 27, K. (2019, August 30). *Robot umpires: How it works and its effect on players and managers in the Atlantic League, plus what's to come*. CBSSports.com. Retrieved April 26, 2023, from https://www.cbssports.com/mlb/news/robot-umpires-how-it-works-and-its-effect-on-players-and-managers-in-the-atlantic-league-plus-whats-to-come/

Connor, N. (2019, May 22). *What is Lift Force - Definition*. Thermal Engineering. https://www.thermal-engineering.org/what-is-lift-force-definition/

GeeksforGeeks (20 Jul. 2020.). Root-Mean-Square Error in R Programming - GeeksforGeeks. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/root-mean-square-error-in-r-programming/

Guzman, C., Brownell, C., & Kommer, E. (2015, September 25). *The Magnus Effect and the American Football*. Springer. https://www.usna.edu/MechEngDept/_files/documents/brownell_files/JSE%2016.pdf

Matlab. (n.d.). *Cumtrapz*. Cumulative trapezoidal numerical integration - MATLAB. Retrieved December 12, 2022, from https://www.mathworks.com/help/matlab/ref/cumtrapz.html#d124e285436

Matlab. MathWorks. (n.d.). Retrieved December 13, 2022, from https://www.mathworks.com/products/matlab.html

Matlab. (n.d.). *Numel*. Number of array elements - MATLAB. Retrieved December 12, 2022, from https://www.mathworks.com/help/matlab/ref/double.numel.html

Lee, W. M., Mazzoleni, A. P., & Zikry, M. A. (2013). Aerodynamic effects on the accuracy of an end-over-end kick of an American football. *Sports Engineering*, *16*(2), 99–113. https://doi.org/10.1007/s12283-012-0110-y

National Air and Space Museum. (n.d.). *The Four Forces*. How Things Fly. Retrieved December 13, 2022, from https://howthingsfly.si.edu/forces-flight/four-forces#:~:text=Lift%20is%20the%20force%20that,Engines%20produce%20thrust.

Price, R., Moss, W., & Gay, T. J. (2020, August 19). *The Paradox of the Tight Spiral Pass In American Football: A Simple Resolution*. American Journal of Physics. https://aapt-scitation-org.ezpv7-web-p-u01.wpi.edu/doi/full/10.1119/10.000138

Profile, My Community (n.d.). Image Processing Toolbox. Mathworks.com. Retrieved from https://www.mathworks.com/products/image.html8

The Physics Of Football. (n.d.). Real World Physics Problems. Retrieved October 16, 2022, from https://www.real-world-physics-problems.com/physics-of-football.html

Wang, S., Xu, Y., Zheng, Y., Zhu, M., Yao, H., & Xiao, Z. (2019, August). Tracking a Golf Ball With High-Speed Stereo Vision System. IEEE Transactions on Instrumentation and Measurement.

Appendix A: Trajectory Equations

Initial velocity in the x direction

$$vx0 = v0 \cdot cos(\theta)$$

Initial velocity in the y direction

$$vy0 = v0 \cdot sin(\theta)$$

Velocity function x direction (matlab code)

$$Vx = -\frac{1}{2m}\rho A(C_L V_y + C_D V_x)\sqrt{V_x^2 + V_y^2}$$

Velocity function y direction (matlab code)

$$Vy = -\frac{1}{2m}\rho A(C_L V_y - C_D V_x)\sqrt{V_x^2 + V_y^2} - g$$

Theoretical Match Equation

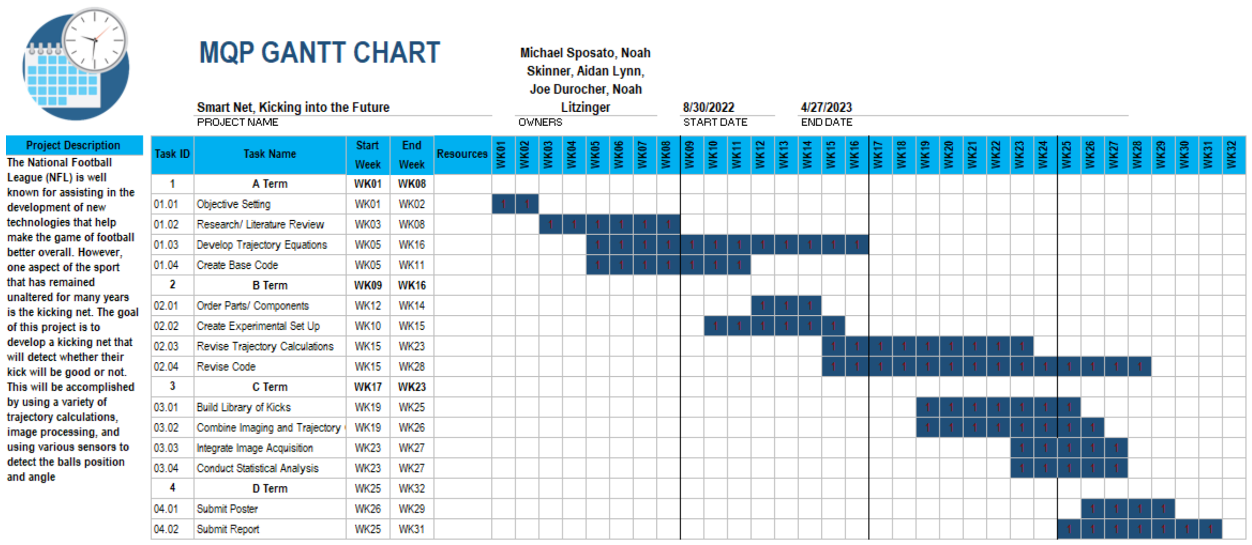
$$x = [0:1:100]$$

$$y = \frac{tan(\theta)\cdot x - g\cdot x^2}{2\cdot vx0^2}$$

*Trajectory of the ball for graphing:

$$y = x \cdot tan\theta - g \cdot \frac{x^2}{2\cdot Vo^2\cdot cos^2\theta}$$

Appendix B: Gantt Chart



MQP GANTT CHART

PROJECT NAME	Smart Net, Kicking into the Future
OWNERS	Michael Sposato, Noah Skinner, Aidan Lynn, Joe Durocher, Noah Litzinger
START DATE	8/30/2022
END DATE	4/27/2023

Project Description

The National Football League (NFL) is well known for assisting in the development of new technologies that help make the game of football better overall. However, one aspect of the sport that has remained unaltered for many years is the kicking net. The goal of this project is to develop a kicking net that will detect whether their kick will be good or not. This will be accomplished by using a variety of trajectory calculations, image processing, and using various sensors to detect the balls position and angle

Task ID	Task Name	Start Week	End Week	Resources	WK01	WK02	WK03	WK04	WK05	WK06	WK07	WK08	WK09	WK10	WK11	WK12	WK13	WK14	WK15	WK16	WK17	WK18	WK19	WK20	WK21	WK22	WK23	WK24	WK25	WK26	WK27	WK28	WK29	WK30	WK31	WK32
1	**A Term**	**WK01**	**WK08**																																	
01.01	Objective Setting	WK01	WK02		■	■																														
01.02	Research/ Literature Review	WK03	WK08				■	■	■	■	■	■																								
01.03	Develop Trajectory Equations	WK05	WK16						■	■	■	■	■	■	■	■	■	■	■	■																
01.04	Create Base Code	WK05	WK11						■	■	■	■	■	■	■																					
2	**B Term**	**WK09**	**WK16**																																	
02.01	Order Parts/ Components	WK12	WK14													■	■	■																		
02.02	Create Experimental Set Up	WK10	WK15											■	■	■	■	■	■																	
02.03	Revise Trajectory Calculations	WK15	WK23																■	■	■	■	■	■	■	■	■									
02.04	Revise Code	WK15	WK28																■	■	■	■	■	■	■	■	■	■	■	■	■	■				
3	**C Term**	**WK17**	**WK23**																																	
03.01	Build Library of Kicks	WK19	WK25																				■	■	■	■	■	■	■							
03.02	Combine Imaging and Trajectory	WK19	WK26																				■	■	■	■	■	■	■	■						
03.03	Integrate Image Acquisition	WK23	WK27																								■	■	■	■	■					
03.04	Conduct Statistical Analysis	WK23	WK27																								■	■	■	■	■					
4	**D Term**	WK25	WK32																																	
04.01	Submit Poster	WK26	WK29																											■	■	■	■			
04.02	Submit Report	WK25	WK31																										■	■	■	■	■	■	■	

Appendix C: Decision Matrices

Software	Arduino	Python	Matlab
Time to learn (4)	8	2	7
Ease of use (3)	9	5	9
Compatibility (5)	8	3	8
CPU (2)	7	4	10
Final score:	113	46	115

Hardware	Camera (Singular Side)	UltraSonic	Laser Gate	Stereo Cameras	Force Sensors
Ease of use (3)	8	9	4	5	2
Cost (3)	8	9	3	4	8
Compatibility (5)	9	3	3	7	3
Capability (3)	10	2	5	10	2
Processing speed (4)	7	2	8	10	7
Final Score:	151	83	83	132	79

Appendix D: Final MATLAB Code

```
    % Produced by Aidan Lynn ME'23, Joseph Durocher ME'23, Michael Sposato ME'23,
% Noah Litzinger ME'23, and Noah Skinner ME'23 for the completion of their
% undergraduate requirements at Worcester Polytechnic Institute

% MQP 2022-2023: Smart Kicking Net
% Advisor: Alireza Ebadi
%=====================================================================
```

Clear Workspace, Command Window, and Figures

```
    clear
clc
close all
```

Constants and predefined variables to be used throughout the code

```
    % Video File Upload
 vid_file_path = 'C:\Users\bunks\OneDrive - Worcester Polytechnic Institute (wpi.edu)\Desktop\MQP\22_23_Kicking_Cage\01 Presentation day';
video_file = 'IMG_9489 1.mov';
Vr = VideoReader(strcat([vid_file_path,'\',video_file]));
FrameSkip = 1; %frame skip (should stay at 1)
ToatalFrames = round(Vr.NumFrames/FrameSkip)-1; %amount of frames

% Centroid, Ball Spin, and Ball Angle Matrices
Centroid = zeros(ToatalFrames,2);
AngleinFlight = zeros(ToatalFrames,2);
ballAngle = zeros(ToatalFrames,1);
frame_time = zeros(ToatalFrames,1);
CameraDistanceFromBall = 10;
MtoP=0.2875/(1303*((CameraDistanceFromBall)^-1.03));
```

```
        DragCo = xlsread('SimDrag.xlsx');
LiftCo = xlsread('SimLift.xlsm');
SurfaceArea = xlsread('SurfaceArea.xlsx');

% Constants for Trajectory Calculations
fps = 240;
framerate = 1/fps;
dt = 0.0001; %Change in time in seconds
t_tot = 10; %Total time of flight in seconds
t = [0:dt:t_tot]'; %Array of time in seconds
m = 0.4; %Average mass of a football in kg
rho = 1.2; %Density of air in kg/m^3
g = 9.81; %Force of gravity in m/s
d = 0.285; %Length of long axis of football in meters
l = 0.17; %Length of short axis of football in meters

% Drag with Respect to Changing Surface Area
angle = zeros(numel(t), 1);
mydrag = zeros(numel(t), 1);
mylift = zeros(numel(t), 1);
SA = zeros(numel(t),1);

%Goal Post Measurements
GPH = 3 * 1.09361; %Goal post height meters
KD = [0,80];%Target kick distance yards
```

Frame by Frame Analysis of Interpreted Video File

```
        for i = 1:ToatalFrames
        n_frame = 1+(i-1)*FrameSkip;


RGB = read(Vr,n_frame);
```

```
% Convert RGB image to chosen color space
I = rgb2hsv(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.149;
channel1Max = 0.172;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.429;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.672;
channel3Max = 1.000;

% Create mask based on chosen histogram thresholds
% Creates binary matrix for masked values
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
        (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
        (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
BW =(sliderBW);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

% Whole ball color value identification for centroid calculation
[rows, columns] = find(BW);
```

```
topRow = min(rows);
bottomRow = max(rows);
leftColumn = min(columns);
rightColumn = max(columns);

% Capturing the angles using the recorded edges of the football
anglex = rightColumn-leftColumn;
angley = bottomRow-topRow;
ballAngle(i,1) = (atan2d(angley,anglex));
frame_time(i,1) = (n_frame*framerate)+((n_frame-1)*framerate);
AngleinFlight(i,1) = frame_time(i,1);
AngleinFlight(i,2) = ballAngle(i,1);

% Plotting theoretical centroid and writing Centroid matrix
[j,k]=size(BW);
ypos=(topRow+bottomRow)/2;
ypos1=j-ypos;
xpos=(leftColumn+rightColumn)/2;
Centroid(i,1) = xpos;
Centroid(i,2) = ypos1;

end
```

Plotting and calculating Spin Rate of ball

```
        dc = diff (Centroid,1);
RowsToDelete = [0;any(dc >= 0 & dc <7,2)];
Centroid = Centroid (~RowsToDelete, :);

H=size(Centroid,1);
AngleinFlight = AngleinFlight(end-H+1:end, :);
```

Median Filter for Flight Path

```
        Q=3;
PostFiltX = medfilt1(Centroid(1:H,1),Q);
PostFiltY = medfilt1(Centroid(1:H,2),Q);
PostFilt = [PostFiltX, PostFiltY];
```

Calculating Initial Velocity and Angle of Ball

```
        change = diff(PostFilt);
V=((change)/framerate)*MtoP;
Angle=atan2d(V(:,2),V(:,1));

rows_with_0 = any(Angle == 0, 2);
Angle(rows_with_0, :) = [];
LaunchAngle = mean(Angle);

v_squared = V.^2;
V2 = [v_squared(:,1) + v_squared(:,2)];

rows_to_delete = V2(:,1) < 5;
V2(rows_to_delete, :) = [];

DeltaAngle = abs(diff(AngleinFlight(:,2)));
DeltaAngle0 = DeltaAngle(:, 1) ~= 0;
DeltaAngle_Nonzero = DeltaAngle(DeltaAngle0, :);
AngularVelo = mean(DeltaAngle_Nonzero)/(60/fps); %Spin rate of football in air
SR = ((AngularVelo)/(2*pi));

Theta = (pi/180)*LaunchAngle; %Launch Angle in Radians

V0 = mean(sqrt(V2)); %Initial velocity in m/s
```

```
Vx0 = V0*cos(Theta); %X component of initial velocity in m/s
Vy0= V0*sin(Theta); %Y component of initial velocity in m/s

Vx = zeros(size(t));
Vx(1) = Vx0;
Vy = zeros(size(t));
Vy(1) = Vy0;

Angle0 = 90; %Initial angle of football
```

Trajectory with changing Drag

```
    Vx = zeros(size(t));
Vx(1) = Vx0;
Vy = zeros(size(t));
Vy(1) = Vy0;
angle = zeros(numel(t), 1);
mydrag = zeros(numel(t), 1);
mylift = zeros(numel(t), 1);
SA = zeros(numel(t), 1);

for ii = 2:numel(t)
    angle(ii) = mod((SR*dt*ii+Angle0), 360);
    mydrag(ii) = interp1(DragCo(:,1), DragCo(:,3), angle(ii));
    mylift(ii) = interp1(LiftCo(:,1), LiftCo(:,3), angle(ii));
    SA(ii) = interp1(SurfaceArea(:,1), SurfaceArea(:,3), angle(ii));

    Vx(ii)                    =                    Vx(ii-1)                    +
dt*(-0.5/m*rho*SA(ii))*(mylift(ii)*Vy(ii-1)+mydrag(ii)*Vx(ii-1))*sqrt(Vx(ii-1)^2+Vy(ii-1)^2);
    Vy(ii)                    =                    Vy(ii-1)                    +
dt*((0.5/m*rho*SA(ii))*(mylift(ii)*Vx(ii-1)-mydrag(ii)*Vy(ii-1))*sqrt(Vx(ii-1)^2+Vy(ii-1)^2)-g
);
```

```
end



X = cumtrapz(t, Vx);
Y = cumtrapz(t, Vy);

% Trajectory Plot
%close all;
figure Name Trajectory
plot(X,Y,'b');
yline(3.3);
xlim ([1 100]);
ylim([0 30]);
hold on
```

Message After Kick

```
        [xi,yi] = intersections(X,Y,KD, [GPH GPH]); % Download the intercepts function from
https://www.mathworks.com/matlabcentral/fileexchange/11837-fast-and-robust-curve-intersectio
ns

 % Save 1 intersection only (the one on the way down, not the one on the way up)
 plot(X, Y, 'b-', [10 40], [GPH GPH], 'r-', xi(2), yi(2), 'g*'); % Intersection point between height
of field goal and trajectory arc of kick

 distance = round(xi(2)*10)/10; % X distance of intersection point
 msg_str = sprintf('Kick is good from %1.0f yards away!', distance); % Creates the message
based on the result of the intersection point
 msg = msgbox(msg_str, 'Kick Distance'); % Creates the message in a new text box
 set(msg,'position',[375 125 400 100]);
 set(findall(msg,'Type','text'), 'FontSize',20);
```