UNIVERSITY
OF WOLLONGONG
IN DUBAI

TEAM ASSIGNMENT COVER SHEET

Name	Student ID	Person Submitting
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	
[redacted]	[redacted]	

Subject number and name	ENGG100 – Engineering Computing and Analysis
Subject Coordinator	Dr. Mohamad Nassereddine
Tutorial Coordinator	Mr. Mohamad Alrifai
Title of Assignment	ENGG100 Project Report
Date and Time Due	[redacted]
Total Number of Pages	[redacted]

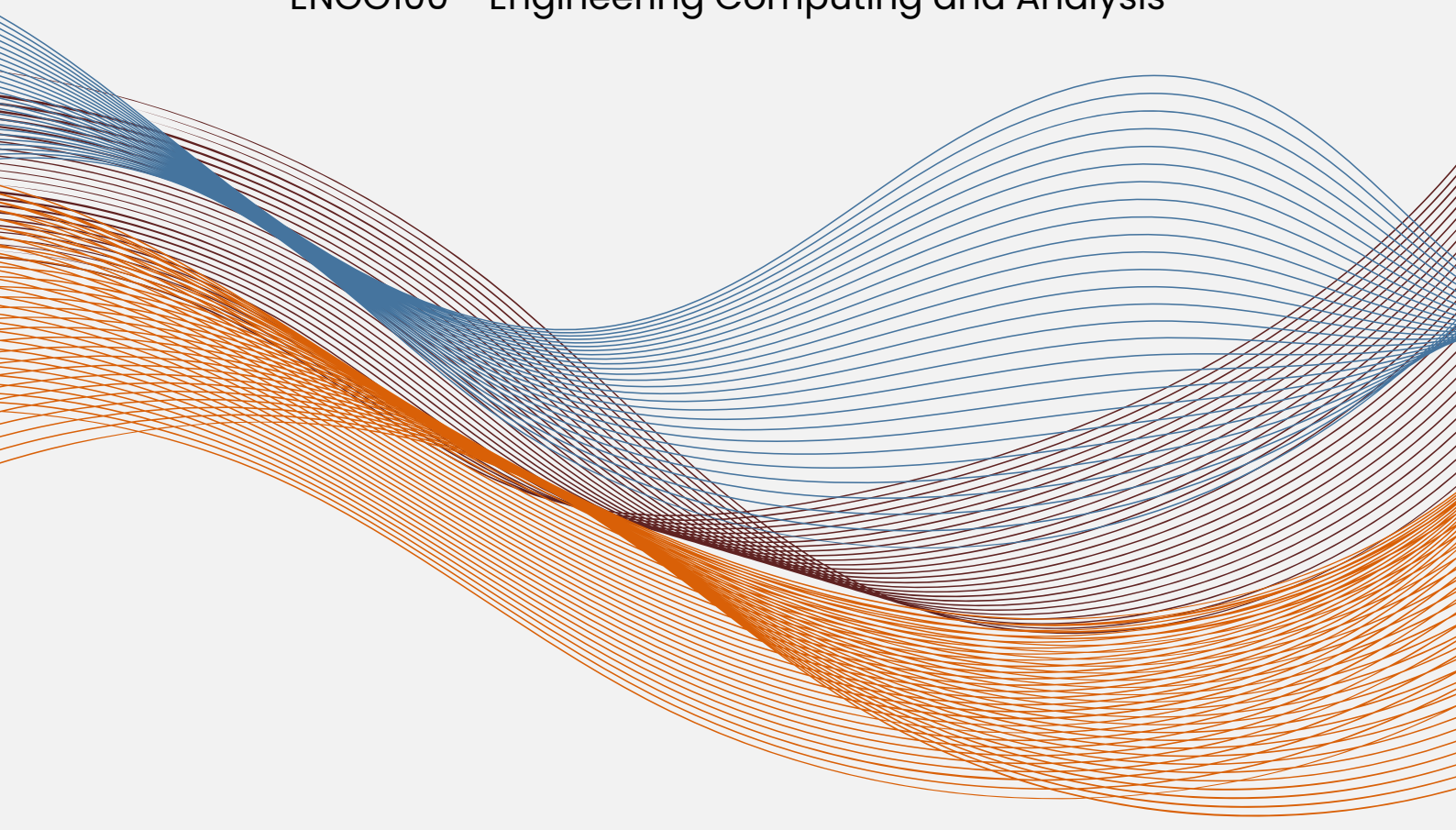
Student declaration and acknowledgement

By submitting this assignment online, the submitting student declares on behalf of the team that:

1. All team members have read the subject outline for this subject, and this assessment item meets the requirements of the subject detailed therein.
2. This assessment is entirely our work, except where we have included fully documented references to the work of others. The material in this assessment item has yet to be submitted for assessment.
3. Acknowledgement of source information is by the guidelines or referencing style specified in the subject outline.
4. All team members know the late submission policy and penalty.
5. The submitting student undertakes to communicate all feedback with the other team members.

Projectile Simulator

ENGG100 - Engineering Computing and Analysis



Acknowledgment

First and foremost, we express our deepest gratitude to Allah Almighty for His countless blessings, guidance, and strength throughout this project.

We would like to extend our sincere appreciation to our esteemed lecturer, Prof. Mohamad Nassereddine, whose invaluable guidance, expertise, and unwavering support were instrumental in the successful completion of this project.

We are also profoundly grateful to our instructor, Mr. Mohamad Mazen Alrifai, for his dedicated mentorship and tireless efforts in imparting knowledge and providing practical guidance throughout this project. His patience, enthusiasm, and commitment to excellence have been truly inspiring.

We would like to express our heartfelt gratitude to our parents and family members for their unconditional love, support, and constant encouragement, who have been the driving force behind our perseverance and determination.

Furthermore, we would like to acknowledge the contributions of our peers and colleagues, whose collaborative efforts, insightful discussions, and constructive feedback have significantly enriched and enhanced this project. Their continuous testing and constructive criticism of our project has helped us keep building our project and further improve it.

Finally, we extend our appreciation to all those who have directly or indirectly contributed to the success of this project, including the library staff, administrative personnel, and everyone else who has lent their support in various capacities.

Summary

The aim of this report is to document the findings and closely explain the functionality of the GUI. An exhaustive understanding of physics, dynamics and engineering has been used throughout the report.

This document is split into 4 distinct sections, all of which contribute to a comprehensive analysis of the projectile simulator.

Chapter 1 states the main problem and objectives as well as what is expected of it. It highlights the need for a user–friendly tool that helps the user experiment with different metrics the projectile can be about, such as the distance from the obstacle, and the height of the obstacle.

Chapter 2 delves into the theoretical design and explains the derivations of the dynamic's equations used as the foundations of code in the simulator in detail. This section covers the graphical representation and discusses various conditions that might affect the projectile during its time of flight.

Chapter 3 introduces, explains and extensively elaborates code properties and their functions, callbacks and outputs, that help allow the application to run as smoothly as possible. It also successfully provides an insight into the dependencies involved.

Lastly, Chapter 4 concludes the report and summarizes it with findings and achievements. The references include external links that have been used as brainstorming for the project. The appendix captures and visualizes the entire code as screenshots and all output examples from the GUI.

Table of Contents

Table of Figures

List of Equations

List of Tables

Chapter 1: Introduction and Background

1.1 Problem Definition

Knowledge and understanding of projectile motion in engineering dynamics contributes to utmost crucial importance in the study of forces and their applications that govern trajectorial characteristics of an object and its path of flight. A projectile, typically an object or particle thrown from near the Earth's surface, moves in a traditionally parabolic path under gravity's sole influence, assuming air resistance is negligible.

The horizontal and vertical components of motion and the SUVAT equations (specifically $v = u + at$ and $s = vt - 0.5at^2$) make use of the following variables to determine unknowns [1]:

- Initial Velocity (m/s)
- Angle of Launch (θ)
- Acceleration Due to Gravity (g)
- Range (m)
- Maximum Height (m)
- Time of Flight (s)

The applications of general dynamics in engineering play a vital role in the designs to withstand forces, analyze trajectories and predict models for the safety, reliability and optimization of theoretical concepts to validate simulation and testing. This allows engineers to create safer and more efficient solutions for systems.

Applications of dynamics and projectile motion problems can be depicted and solved through Matrix Laboratory (MATLAB) using a combination of mathematics, physics and code language to predict the trajectory as well as solving systems that accurately display the motion in real time simulation [2].

This project requires the incorporation of all these aspects to produce a graphical user interface (GUI) that asks the user for the distance from an obstacle, the height of the obstacle, and a constant acceleration of an object or particle launched to be input into a program that outputs velocity, time of flight, maximum height, and peak location, from the code written. Gravitational forces have been considered throughout the flight of the projectile.

The report aims to comprehensively analyze all the inputs and outputs, including the projectile's position at any given time and the maximum distance it can travel.

1.2 Objectives

There are numerous aspects that play into the final design and operations of this project. The objectives have been built around theoretical aspects learned in ENGG100 – Engineering Computing and Analysis and aim to elaborate and add detail to their practical applications.

1. **Fundamental understanding and basic sense of projectile motion, general dynamics, and its importance in engineering.**
    - A basic idea about key concepts such as kinematics, kinetics, and Newton's laws of motion are the foundations of this project.
    - Implementation and use of SUVAT equations to analyze and compute the behavior of the projectile.
    - Determine how different inputs of factors such as distance from obstacle and the height of obstacle that are put into the program result in contrasting sensitivities of outputs.

2. **Fundamental understanding and basic sense of MATLAB programming**
    - Ability to develop and include scripts in MATLAB and create functions to accurately simulate a conceptual projectile's trajectory.
    - Ensuring all code is well commented and easy to comprehend.
    - Ability to use code from scripts to help determine the projectile's position at any given time during flight.
    - Ability to plot velocity – time, acceleration – time, and displacement – time graphs from scripts.

3. **Implementation of dynamics equations into code.**
    - Ability of code to operate given specific minimum and maximum parameters.
    - Use of equations in relation to ENGG100 concepts and dynamics
    - Proper derivation of necessary equations to build code

4. **GUI – User Interaction**
    - Design a user–friendly interface where users can input variables.
    - Use loops to vary launch parameters [3].
    - The following values are the characteristics of the target position:
        - 3 meters above the ground.
        - 6 meters away from the obstacle.

5. **Parameter Optimization**
    - Develop an algorithm to find the optimal launch conditions to reach a specific target distance.
    - Ensure parameters are properly defined to prevent errors.
    - Make use of iterative calculations and loops to find the best solution.

6. **Graphical Representation**
    - Implementation of comprehensive graphical tools.
    - Creation of plots that allow comparison of different trajectories.
    - Addition of sliders and input fields to interactively adjust parameters.

1.3 Expected Outcomes

Upon completion of the project, the objectives are to be met, along with a proper understanding of general dynamics and the use of MATLAB to carry out tasks. To reiterate, it is expected to achieve the following in addition to the objectives:

- A comprehensive grasp of dynamics equations.
- Ability to tweak sensitivity of projectile's path subject to different starting conditions.
- Development of precise MATLAB scripts to produce accurate results.
- User-friendly and easy to understand code.
- Determination of projectile's position at any given time during its flight.
- Accurate calculations of the maximum horizontal distance the projectile can travel at different starting conditions.
- Finding how long the projectile stays in the air and how far up it goes before landing.
- Flexibility in changing launch conditions in GUI.
- Encourage users to think on how different constraints affect the projectile in real–world scenarios.
- Develop code to predict the best launch angle and velocity required for the projectile to reach a point.
- Use of loops to provide reliable results.
- Creation of graphs to depict the projectile's path.
- Create a practically convenient GUI.

- Compilation of this report to document the background, method, code, and results.
- Inclusion of all code in the report with comments to ensure it is simple to follow.

Chapter 2: Theoretical Design

2.1 Equations Used

The following are the main equations that have been used to track the motion of the projectile.

$$x = x_0 + v_0 \times \cos(\alpha) \times t \quad (1)$$

Equation 1 – Equation for horizontal projectile motion

$$y = y_0 + v_0 \times \sin(\alpha) \times t - \frac{1}{2} \times g \times t^2 \quad (2)$$

Equation 2 – Equation for vertical projectile motion

$$s = v_0 \times t - 0.5 \times g \times t^2 \quad (3)\ [4]$$

Equation 3 – Equation for displacement of an object experiencing free fall

$$v = v_0 - g \times t \quad (4)$$

Equation 4 – Equation for velocity of an object experiencing free fall

Table 1 shows the meanings of the symbols and variables used in Equations (1) through (4) above.

Table 1 – Variables of the Main Equations

Symbol / Variable	Definition
x	Distance (at given time)
x_0	Initial Distance
y	Height (at given time)
y_0	Initial Height
v_0	Initial Velocity
α	Angle
t	Time
g	Acceleration due to gravity
v	Velocity (at given time)
s	Displacement (at given time)

2.2 Dynamics Analysis

Dynamics analysis, in the context of projectile motion, involves examining the forces acting on a projectile, such as acceleration due to gravity and velocity, to predict the projectile's trajectory, velocity, and acceleration throughout its flight. With this in mind, two equations have been used to determine the initial velocity, initial angle, and the time required for the projectile to reach its destination.

2.2.1 Dynamics Analysis for Simulation

To determine the position of the projectile throughout its flight, Equation 1 and Equation 2 have been used in the simulation tab.

The meaning of the symbols and variables shown in the equations that are used are detailed in Table 2.

Table 2 – Variables of the equations used in this section

Symbol / Variable	Definition
x	Distance (at given time)
x_0	Initial Distance
x_f	Final Distance / Range
y	Height (at given time)
y_0	Initial Height
y_f	Final Height
v_0	Initial Velocity
α	Angle
t	Time
g	Acceleration due to gravity
D	Distance from obstacle
H	Height of obstacle
m	Safety margin

To ensure accurate results, the following assumptions had to be made.

1. The projectile starts from the origin (x_0 = 0 m, y_0 = 0 m).
2. The final position of the projectile is 6 m away from the obstacle (x_f = D + 6 m).
3. The projectile lands at an elevation of 3 m above ground (y_f = 3 m).
4. m is a safety margin to ensure that the projectile does not hit the obstacle. It also accounts for floating point inaccuracies in programming languages like MATLAB. Its value is set to 1e – 6, or 0.000001m, or 1 micrometer (μm). This safety margin is negligible and does not cause any significant change in the values of initial velocity, initial angle and the time of flight.

Equation 1 and Equation 2 were rearranged to obtain the initial velocity, initial angle and the time of flight.

$$x_f = v_0 \times \cos(\alpha) \times t \;\Rightarrow t = \frac{x_f}{v_0 \times \cos\alpha} = \frac{D+6}{v_0 \times \cos(\alpha)} \tag{5}$$

Equation 5 – Equation for total time of flight

$$y_f = 3 = v_0 \times \sin(\alpha) \times t - \frac{1}{2} \times g \times t^2 \tag{6}$$

Equation 6 – Equation for height at final position

Substituting Equation 5 in Equation 6, we obtain the formula for the height at the final position.

$$3 = v_0 \times \sin(\alpha) \times \frac{D+6}{v_0 \times \cos(\alpha)} - \frac{1}{2} \times g \times \left(\frac{D+6}{v_0 \times \cos(\alpha)}\right)^2 \quad \leftrightarrow$$

$$3 = \sin(\alpha) \times \frac{D+6}{\cos(\alpha)} - \frac{1}{2} \times g \times \left(\frac{D+6}{v_0 \times \cos(\alpha)}\right)^2 \quad \leftrightarrow$$

$$3 = \tan(\alpha) \times x_f - \frac{1}{2} \times g \times \left(\frac{x_f}{v_0 \times \cos(\alpha)}\right)^2 \quad (7)$$

Equation 7 – Substituting total time of flight in y_f

To rearrange the terms in Equation 7, we first start by multiplying throughout by $\cos^2 \alpha$ and then using the formula $\frac{1}{2} \times \sin(2\alpha) = \sin(\alpha) \times \cos(\alpha)$

$$3 \times \cos^2(\alpha) = \sin(\alpha) \times \cos(\alpha) \times x_f - \frac{1}{2} \times g \times \left(\frac{x_f}{v_0}\right)^2 \quad \leftrightarrow$$

$$3 \times \cos^2(\alpha) = \frac{1}{2} \times \sin(2\alpha) \times x_f - \frac{1}{2} \times g \times \left(\frac{x_f}{v_0}\right)^2 \quad (8)$$

Equation 8 – Using $sin(2\alpha)$ formula

We divide it by 0.5 to simplify it further.

$$6 \times \cos^2(\alpha) = x_f \times \sin(2\alpha) - g \times \left(\frac{x_f}{v_0}\right)^2 \quad (9)$$

Equation 9 – Dividing by 0.5

We then rearrange the equation to get v_0

$$g \times \left(\frac{x_f}{v_0}\right)^2 = x_f \times \sin(2\alpha) - 6 \times \cos^2(\alpha) \quad \leftrightarrow$$

$$g \times \frac{x_f^2}{v_0^2} = x_f \times \sin(2\alpha) - 6 \times \cos^2(\alpha) \quad (10)$$

Equation 10 – Rearranging to isolate v_0

$$v_0^2 = \frac{g \times x_f^2}{x_f \times \sin(2\alpha) - 6 \times \cos^2(\alpha)} \quad \leftrightarrow$$

$$v_0 = \sqrt{\frac{g \times x_f^2}{x_f \times \sin(2\alpha) - 6 \times \cos^2(\alpha)}} \quad (11)$$

Equation 11 – Final v_0 Equation

Given that velocity must be a positive value in this context, we discard the negative root obtained from Equation 11. The valid solution for velocity is therefore the positive root only.

At the coordinates of the obstacle (D, H), the equations are

$$D = v_0 \times \cos(\alpha) \times t_1 \Rightarrow t_1 = \frac{D}{v_0 \times \cos(\alpha)} \quad (12)$$

Equation 12 – Equation for time at distance of obstacle

$$H + m = v_0 \times \sin(\alpha) \times t_1 - \frac{1}{2} \times g \times {t_1}^2 \quad (13)$$

Equation 13 – Equation for height at height of obstacle

Substituting Equation 12 in Equation 13, we obtain

$$H + m = v_0 \times \sin(\alpha) \times \frac{D}{v_0 \times \cos(\alpha)} - \frac{1}{2} \times g \times \left(\frac{D}{v_0 \times \cos(\alpha)}\right)^2 \leftrightarrow$$

$$H + m = \sin(\alpha) \times \frac{D}{\cos(\alpha)} - \frac{1}{2} \times g \times \left(\frac{D}{v_0 \times \cos(\alpha)}\right)^2 \quad (14)$$

Equation 14 – Substituting time required to reach the obstacle

Substituting (11) in (14) to obtain obstacle height in terms of α only.

$$H + m = \sin(\alpha) \times \frac{D}{\cos(\alpha)} - \frac{1}{2} \times g \times \left(\frac{D}{\sqrt{\frac{g \times x_f^2}{x_f \times \sin(2\alpha) - 6 \times (1 - \sin^2(\alpha))} \times \cos(\alpha)}}\right)^2 \quad (15)$$

Equation 15 – Replacing initial velocity in equation for height

$$H + m = \sin(\alpha) \times \frac{D}{\cos(\alpha)} - \frac{1}{2} \times g \times \left(\frac{D^2 \times \left(x_f \times \sin(2\alpha) - 6 \times (1 - \sin^2(\alpha))\right)}{g \times x_f^2 \times \cos^2(\alpha)}\right) \leftrightarrow$$

$$H + m = D \times \frac{\sin(\alpha)}{\cos(\alpha)} - \frac{1}{2} \times \left(\frac{D^2 \times \left[x_f \times \sin(2\alpha) - 6 \times (1 - \sin^2(\alpha)) \right]}{x_f^2 \times \cos^2(\alpha)} \right) \quad (16)$$

Equation 16 – Final equation for height at obstacle in one variable (α)

Determining the minimum angle for obstacle clearance requires solving Equation 16. The solution to this equation yields the smallest angle necessary for the projectile to successfully pass over the obstacle and reach its intended target.

After identifying this critical angle, we can proceed to calculate the initial velocity using Equation 11. By inputting these values into Equation 5, we can compute the projectile's time of flight.

Trying to implement the complex Equation 16, Equation 11 and Equation 5 for initial velocity, angle of launch, and time of flight in MATLAB can be quite challenging. Not only is it a daunting task, but it might also lead to inaccurate results as there are multiple trigonometric functions in Equation 16. Moreover, creating a general-purpose code that works in all situations increases the complexity of the system. The calculations could fail in extreme cases, like when someone is standing too close to the obstacle or when the calculated launch angle is unreasonably steep.

To avoid these problems, a different approach can be taken. Instead of using the complicated equations above, a simpler system of Equation 7 and Equation 14 can be used. This method is not only more accurate but also faster than using a trial-and-error approach with multiple iterations. It gives reliable values for the initial velocity, launch angle, and time of flight.

2.2.2 Dynamics Analysis for Graphs

The graphs tab showcases three crucial visualizations: displacement – time, velocity – time, and acceleration – time graphs. These graphs were generated using two fundamental equations of motion, specifically Equation 3 and Equation 4.

Equation 3, which represents the displacement formula, was used solely for creating the displacement – time graph. This equation allows us to calculate the total displacement of a projectile at any given moment. It considers three key variables: time, initial velocity, and the acceleration due to gravity. By inputting various time values into this equation, we can plot the projectile's position at different instants, resulting in a detailed displacement – time curve.

Equation 4 was crucial to generate the velocity – time graph. This velocity equation enables us to determine the instantaneous velocity of the projectile at any point during its trajectory. Similar to Equation 3, it incorporates time, initial velocity, and gravitational acceleration. By systematically applying this equation across a range of time values, we can trace the projectile's velocity changes throughout its flight, resulting in an accurate velocity – time graph.

The acceleration – time graph is derived from the constant acceleration due to gravity. In a typical projectile motion scenario, acceleration remains constant, resulting in a straight horizontal line on the acceleration – time graph.

2.3 Impacts of Conditions on Dynamics Analysis

In order to ensure that the program runs correctly and produces accurate outputs, it is necessary to set appropriate constraints in place. If these limitations were not implemented, the program would be susceptible to various errors and unintended behavior, potentially leading to incorrect, inconsistent, or unreliable results.

There are two limitations put in place for successful flight of the projectile.

1. $90 > \alpha \geq \alpha_{min}$

   The range of angles permitted for successful calculation is constrained within a specific range to achieve practical and meaningful results.

   Two key minimum angles must be considered:

   - The angle from the origin to the obstacle's peak
   - The angle from the origin to the target destination

   The greater of these two becomes the minimum launch angle, ensuring both obstacle clearance and reaching the target. For example, if the obstacle exceeds the target's height, the minimum angle must clear the obstacle. Conversely, if the target is higher, the minimum angle should be the angle required to reach the target.

This critical minimum angle is calculated using the inverse tangent function $\tan^{-1}$(height / horizontal distance) where 'height' and 'horizontal distance' refer to the height and horizontal distance of either the obstacle or target, depending on which determines the minimum angle. Angles below this minimum result in collision with obstacle or undershooting the target position, rendering the trajectory impossible.

The maximum permissible angle is strictly less than 90°. Angles of 90° or greater result in vertical or backward trajectories, both of which fail to reach the intended target. A vertical launch under ideal conditions would return to its starting point, while angles exceeding 90° send the projectile in the opposite direction of the target.

2. $D \prec 0.01$

To ensure meaningful and applicable trajectory calculations, the horizontal distance must exceed 0.01 meters (1 centimeter). This minimum threshold is crucial because distances below it create an impractical scenario where the required minimum launch angle approaches 90 degrees. As previously established in the limitation above, a 90° launch angle is infeasible for reaching the target, rendering the calculations invalid and the trajectory unsolvable.

Chapter 3: MATLAB Code

This chapter covers the entire MATLAB Code of the application. The program consists of various tabs thus equipping the user with the tools to use for a given task. These tabs are:

1. Simulation Tab
2. Graphs Tab
3. History Tab

The following explanations contain only snippets of the code along with their various outputs, while the entire code will be presented in ***Appendix A***.

3.1 Dependencies

To operate the program, the software requires the MATLAB Optimization Toolbox. While this toolbox is not included in the standard MATLAB installation, users with a valid license can download it separately. [5]

MATLAB's *fsolve* function, part of the Optimization Toolbox, is an extremely powerful tool for solving systems of nonlinear equations. [6] In the context of this project, it is used to determine the initial angle and initial velocity needed for a projectile to clear the obstacle and reach the final position. The function works by taking initial guesses for these variables, which are set to very small values (10^{-6} radians for the angle and 10^{-6} m/s for the velocity) and iteratively refining them to satisfy the given equations. [7, 8]

The process begins with defining the system of equations in a separate function *eqnss*, which ensures the projectile meets the required conditions at different points in its trajectory. The equations used in the *eqnss* function are Equation 7 and Equation 14 from Section 2.2.1 above. After this, a function handle is created, pointing to the *eqnss* function with the initial guess as an input [9]. This allows *fsolve* to evaluate the system of equations for the given variables.

The solver employs sophisticated algorithms like trust – region – dogleg or Levenberg – Marquardt to handle the nonlinear equations. It evaluates the initial guesses, then estimates the Jacobian matrix to understand how changes in variables affect the equations' outcomes. The solver then computes steps to adjust the guesses, aiming to reduce residuals. Residuals are the differences between the left – hand side and right – hand side of equations when the current estimates of the variables are substituted into those equations.

This process of plugging in estimated values repeats until the solution converges or reaches the maximum allowed iterations. Throughout this process, user – defined options control various aspects of the optimization. For this project, the maximum number of function evaluations and maximum iterations of variables allowed during the optimization process are set to very high values, at 10^{50} evaluations and iterations each. This ensures that the solver has ample opportunities to find a solution.

After each update, *fsolve* checks if the solution has converged. Convergence criteria typically involve the size of the residuals and the changes in variables being smaller than specified tolerances. The default value of the tolerance, which is 10^{-6}, is used for this project.

Once a solution is found, the function returns the values for the launch angle and initial velocity that satisfy the system of equations, providing the necessary parameters for the projectile to meet the specified conditions.

3.2 Simulation Tab

The Simulation Tab is the main page of the application where the core of the program runs. It contains the Input, Output, Graph, Location Tracker and Real Time Simulator System (RTSS). *Figure 1* contains the screenshot of the Simulator Tab.

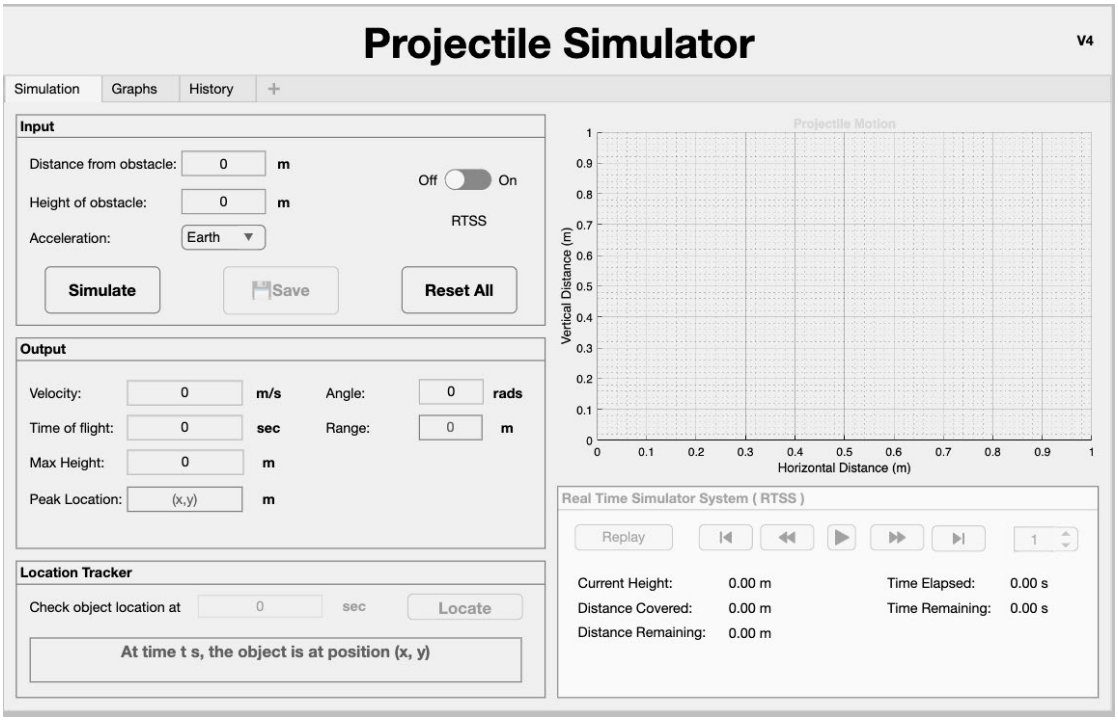


Figure 1 – Simulation Tab

In the Simulation tab, the user inputs the distance from the obstacle, the height of the obstacle, and the gravity of the system. By default, the program assumes the gravity of the Earth. Once the simulate button is pressed the values are shown in the Output panel which includes the velocity (m/s), angle (rads), time of flight (sec), maximum height (m), and the peak location of the projectile. The Location Tracker panel allows the user to input the time (s) to check the location of the projectile in the system, while the RTSS allows the user to view real – time data. This can be viewed by enabling it before the simulation.

3.2.1 Functions (Simulation Tab)

Various properties have been used in the program; these are the global properties utilized throughout the application. *Figure 2* contains the properties initialized at the start of the program.

```
properties (Access = private)
    X % Main - X Values
    Y % Main - Y Values
    T % Main Time
    TMAX % Maximum Time

    sAni = true % Should the program animate
    isAni = true % Is the program animating
    isPlay = true % Is the program playing the animation
    isUpdat = true % Is the program updating
    cTime = 0 % Start Time
    pSpeed = 1 % Playback speed
    timerr = timer('ExecutionMode','fixedRate', ...
        'Period', 0.1, ...
        'TimerFcn', @(~,~) updateRTSS(app));
    hisname = [] % Array of data for importing in graph (History Tab)
    playIcon = 'Play.png' % Play Icon
    pauseIcon = 'Pause.png' % Pause Icon
    new_tab = [] % Array of New-tabs appended in the History Tab
end
```

Figure 2 – Properties Initialized

The *Input* function takes the input values of the program such as the distance from the obstacle (D), height of the obstacle (H), and the acceleration due to gravity (g). These inputs are fed to the equation and used to determine the initial velocity (v), angle (alpha in radians) and time of flight (t). This function will be used in other parts of the application. It outputs the various inputs as an array. The following figure (*Figure 3)* contains the snippet of the function.

```
function [x, H, y_b, d, D, g] = inputs(app)
    x = app.Distance.Value; % Distance from obstacle
    H = app.Height.Value; % Height of obstacle
    a = app.acceleration.Value; % Acceleration ie. Gravity
    y_b = 3;    % Height of the basket in meters
    d = 6;      % Distance from obstacle to basket in meters
    D = x + d;  % Total horizontal distance from thrower to basket
    if a == "Earth"
        g = 9.81; % Earth Gravity
    elseif a == "Moon"
        g = 1.62; % Moon Gravity
    elseif a == "Mars"
        g = 3.71; % Mars Gravity
    end
end
```

Figure 3 – Inputs Function

The function *eqnss* is intended to set up a system of two nonlinear equations based on initial guesses for the variables involved. The function takes one input *vars*, which is a vector of initial guesses for the variables.

The function begins by calling *input function* to receive the required inputs of horizontal distance to obstacle (D), the height of the obstacle (H), the final vertical position (y_b), the final horizontal position (xf), and the acceleration due to gravity (g).

Next, the function extracts the initial guesses for alpha (angle) and v0 (velocity) from the *vars* vector. A small safety margin of 1μm is defined to avoid hitting the obstacle.

The function then defines two equations. The first equation (eq1) represents the vertical position at the final horizontal position y_b, which is based on Equation 7. The second equation (eq2) represents the height at the obstacle, considering the safety margin, and is based on Equation 14.

The function also checks if the height of the obstacle is higher than the target position. If it is, it defines an equation for the height of the obstacle. If not, it defines eq2 as zero, since the obstacle does not have to be considered in the calculations.

Finally, the function returns these two equations as a column vector F. This vector is then used to solve for the initial angle and initial velocity required to reach the target position while avoiding the obstacle in the *calculate* function below. Figure 4 contains a snippet of the function.

```
function F = eqnss(app, vars)
    [D, H, y_b, ~, xf, g] = inputs(app); % Get required inputs
    alpha = vars(1); % Angle initial guess
    v0 = vars(2); % Velocity initial guess
    safety_margin = 1e-6; % Margin to avoiid hitting obstacle

    % Equation for the vertical position at the final position
    eq1 = y_b - (xf * tan(alpha) - (1/2) * g * (xf / (v0 * cos(alpha)))^2);

    % If the building height is less than the basket height, the building constraint can be ignored
    if H < y_b
        eq2 = 0;
    else
        % Equation for the height at the obstacle
        eq2 = H + safety_margin - (D * tan(alpha) - (1/2) * g * (D / (v0 * cos(alpha)))^2);
    end

    % Return the system of equations
    F = [eq1; eq2];
end
```

Figure 4 – Eqnss Function

The *calculate* function (*Figure 5*) solves the equation vector F as detailed in Section 3.1 and calculates the metrics of the projectile such as maximum height, peak location and range. These will be used in the History tab for each of the simulations. It takes distance (D), gravity (g), as inputs and velocity (V), angle (alpha), time of flight (tofl), range (rangeh), maximum height (maxhis), distance at maximum height (x_max_height), time at maximum height (t_max_height) and peak value (peakvalue) as outputs. *Figure 5* shows the snippet of the function.

```
function [V, alpha, tofl, rangeh, t_max_height, max_height, x_max_height, peakvalue] = calculate(app, D, g)

    % Initial guesses for alpha and v0 1e-6 degrees and 1e-6 m/s
    initial_guess = [(1e-6) * pi / 180, 1e-6];
    ...
    t_max_height = V * sin(alpha) / g; % Time at max height
    max_height = V * sin(alpha) * t_max_height - 0.5 * g * t_max_height^2; % Max height
    x_max_height = V * cos(alpha) * t_max_height ; % Max Height X - Coordinates % X coordinate at max height
    peakvalue = ['(',num2str(round(x_max_height, 2)),',',num2str(round(max_height, 2)),')'];
end
```

Figure 5 – Calculate Function

The *plotgraph* function (*Figure 6*) plots a static graph without real time data and animation. It calculates and plots a graph for every x and y in the given time array. This can be used by disabling the RTSS switch.

```
function plotgraph(app) % Plots the static Graph
    [~, ~, ~, ~, D, g] = inputs(app);
    [V, alpha, tofl, ~, ~, ~, ~, ~] = calculate(app, D, g);
    t = 0:1e-5:tofl; % Time array
    x = V * cos(alpha) * t; % X axis - Horizontal Displacement
    y = V * sin(alpha) * t - 0.5 * g * t.^2; % Y Axis - Vertical Displacement
    plot(app.MainGraph,x,y)
end
```

Figure 6 – Plotgraph Function

The set of functions below (*Figure 7)* are for updating and maintaining a real – time simulation system (RTSS). The main function, *updateRTSS*, modifies the simulation's state continually in response to user interactions and the current time. It determines whether the current time (cTime) is within a valid range and if the animation (isAni) is

active. If so, it locates the most recent data point that matches the current time, adds it to the display, and shows the position at that moment along with the journey traveled so far on a graph. Plotting is done dynamically; the path is shown by a line, and the current position is represented by a red dot. In addition, if the animation is configured to update (isUpdat), the method manages time progression and guarantees that the display updates at a consistent rate. The simulation can be started from the beginning, restarted from where it left off, or paused using the *pauseRTSS*, *resumeRTSS*, and *startRTSS* functions respectively. The animation's timer is stopped using *pauseRTSS*, which also modifies the play/pause button symbol to show that the animation is paused.

resumeRTSS modifies the icon to indicate that the animation is playing when restarting the timer. *startRTSS* sets the animation state to active and initializes the current time to zero. Additionally, it also initiates a timer that calls *updateRTSS* on a regular basis. *updateRTSSv* function uses the most recent data point to update the display with the height, distance traveled, distance left, and time elapsed. *tweakRTSS* ensures that the animation speed can be adjusted as needed by recalculating the current time based on a specified speed factor. This enables customization of the animation playback speed.

```
function updateRTSS(app) ... % Updates the RTSS

function pauseRTSS(app) ... % Pauses the RTSS

function resumeRTSS(app) ... % Resumes the RTSS

function startRTSS(app) ... % Starts the RTSS

function updateRTSSv(app, x, y, t) ... % Updates the RTSS values

function tweakRTSS(app, pspeed) ... % Tweak the RTSS speed
```

Figure 7 – Real – Time Simulation System (RTSS) Functions

The *RTSSEnable* function (*Figure 8*) checks the value of the RTSS switch. If the value is set to "On", it enables the RTSS panel and its functions along with it.

```
function rtssenable(app) % This function enables or disables the RTSS
    rs = app.RTSSSwitch.Value;
    if strcmp(rs,'On') % Check the RTSS switch if it is ON and enables the RTSS panel
        app.RTSSPanel.Enable = "on";
        app.ReplayButton.Enable = "on";
        ...
        startRTSS(app);
        app.sAni = true;
    else % If off then disables the panel along with its buttons
        app.sAni = false;
        app.plotgraph;
        ...
        app.InputPlaybackSpeed.Enable= "off";
    end
end
```

Figure 8 – RTSSenable Function

3.2.2 Callbacks (Simulator Tab)

When the Simulate Button (*Figure 9)* is pressed, the *input* and *calculate* functions along with various conditions for the input are run while also displaying the velocity, angle, time of flight, peak location, maximum height, and range. It runs the *RTSSenable* function to check if the user has enabled it. It sets the value of the X, Y, TMAX, T, and enables the Location Tracker panel along with its functions, and the Save button to save the values for future use in the History Tab.

```
function simulateButtonPushed(app, event)
    [x, H, y_b, ~, D, g] = inputs(app);
    isValidDistance = true; % Initialize checking variable - Distance
    isValidHeight = true; % Initialize checking variable - Height

    % Error if distance from building is too less
    ...
            x = V * cos(alpha) * t; % X - Axis
            y = V * sin(alpha) * t - 0.5 * g * t.^2; % Y - Axis

            y(y < 0) = 0; % Condition to make sure Y is 0
            app.X = x; % Sets RTSS X Array
            app.Y = y; % Sets RTSS Y Array
            app.T = t; % Sets RTSS T Array
            app.TMAX = tofl; % Sets RTSS TMAX

            app.SaveButton.Enable = "on"; % Enables Save button
        end
    end
end
```

Figure 9 – Simulate Button Callback

The Reset All Button (*Figure 10*) resets the entire application and brings it back to its initial state.

```
% Button pushed function: resetall
function resetallButtonPushed(app, event)
    % Reset all values and set to default

    % RTSS Panel
    app.RTSSPanel.Enable = "off";
    ...
     app.MinDisplacement.Value = 0;
    app.MaxDisplacement.Value = 0;
end
```

Figure 10 – ResetAll Button Callback

The *locateButtonPushed* (*Figure 11*) function callback calculates and shows a projectile's position at a user specified time. The function collects the gravitational acceleration (g), launch angle (alpha), and initial velocity (V) when the button is pressed. The projectile's vertical (y_check) and horizontal (x_check) coordinates at the given time (t_check) are then determined. The function modifies the horizontal position to the maximum distance and sets the vertical position to zero, indicating that the projectile has landed, if t_check exceeds the projectile's total flight duration TMAX.

```
% Button pushed function: locate
function locateButtonPushed(app, event)
    [~, ~, ~, ~, D, g] = inputs(app);
    [V, alpha, tofl, ~, ~, ~, ~, ~] = calculate(app, D, g);
    t_check = app.timelocate.Value; % Time field
    ...
    app.locationdisp.Value= ['At time ' num2str(round(t_check, 2))
    ' s, the object is at position (' num2str(round(x_check, 2)) ', ' num2str(round(y_check, 2)) ')'];
     % Displays time, (x,y) coordinates in locationdisp
    end
end
```

Figure 11 – Locate Button Callback

The following button callbacks (*Figure 12*) all relate to the function of the Real Time Simulation System. *ReplayButton* replays the entire plot while *FastRewind* rewinds the graph. *SkipBack* starts the graph from the origin. *PlayPause* plays and pauses the graph when pressed. *SkipForward* skips to the final position of the projectile, whereas *FastForward* forwards the graph. The *InputPlayback* determines the speed of the graph.

```
function ReplayButtonPushed(app, event) ... % Replay button pushed

function FastRewindButtonPushed(app, event) ... % Fast rewind button pushed

function SkipbackPushed(app, event) ... % Skip back button pushed

function PlayPauseButtonValueChanged(app, event) ... %  Play/Pause button pushed

function SkipforwardPushed(app, event) ... % Skip forward button pushed

function FastForwardButtonPushed(app, event) ... % Fast forward button pushed

function InputPlaybackSpeedValueChanged(app, event) ... % Input playback speed value
```

Figure 12 – Real – time Simulation System (RTSS) Button Callbacks

Once enabled, the *SaveButton* (*Figure 13*) allows the user to save the values for future use to be viewed in the History Tab. It stores the value in a table allowing the user to choose which values to be saved.

```
% Button pushed function: SaveButton
function SaveButtonPushed(app, event)
    s = inputdlg("Enter the name of the Data","Save"); % Name of the Save
    if isempty(s)
        uialert(app.UIFigure, "Save Cancelled", "Failure", "Icon", "warning"); % Cancels the save
    else
        [~, H, ~, ~, D, g] = inputs(app);
        [V, alpha, tofl, ~, ~, ~, ~, ~] = calculate(app, D, g);
        data = [s, D, H, round(g, 2), round(V, 2), round(alpha, 2), round(tofl, 2)]; % Data to be saved
        sh = app.SimulationH.Data; % Data in History Tab
        app.SimulationH.Data = [sh;data]; % Saves the Data in database in History Tab
    end
end
```

Figure 13 – Save Button Callback

3.2.3 Outputs (Simulator Tab)

Figure 14 shows the basic working of the application including the various outputs and metrics. In this figure, the Real – Time Simulation System is not enabled.

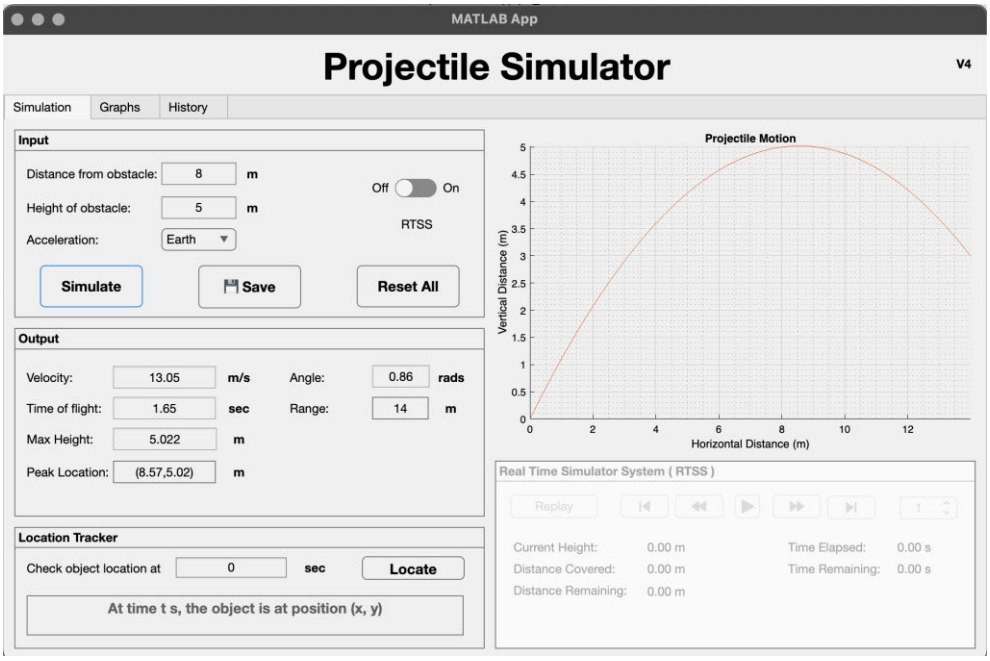


Figure 14 – Basic Output

Figure 15 showcases the application running when RTSS is enabled.

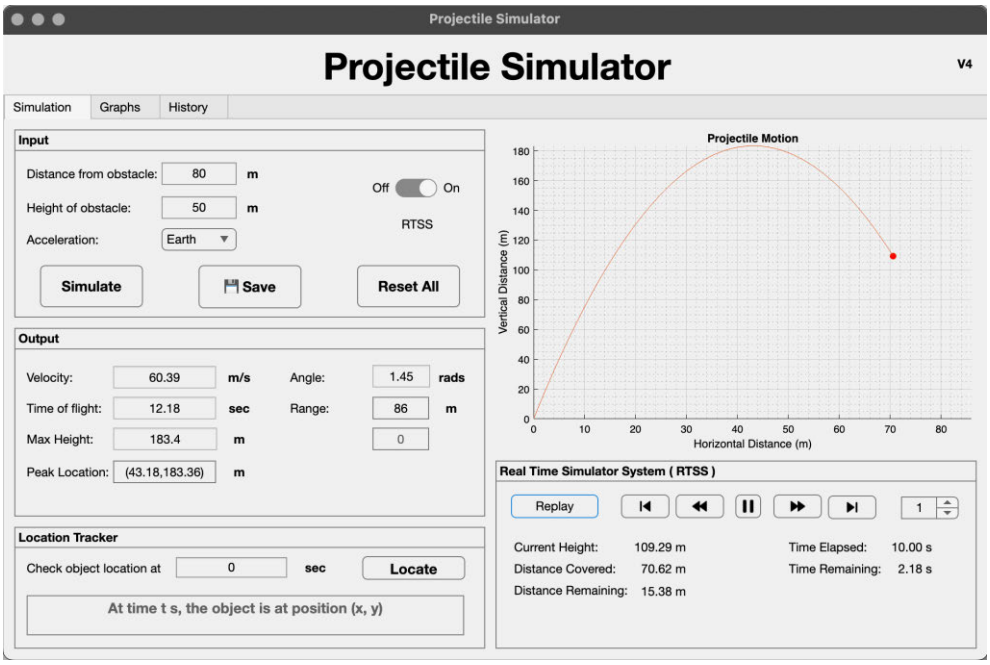


Figure 15 – RTSS Enabled

The *Figure 16* showcases the Location tracker function, while *Figure 17* shows the errors to the user inputs Distance (D) less than 0.01. *Figure 18* shows the graph when height is less than 3.

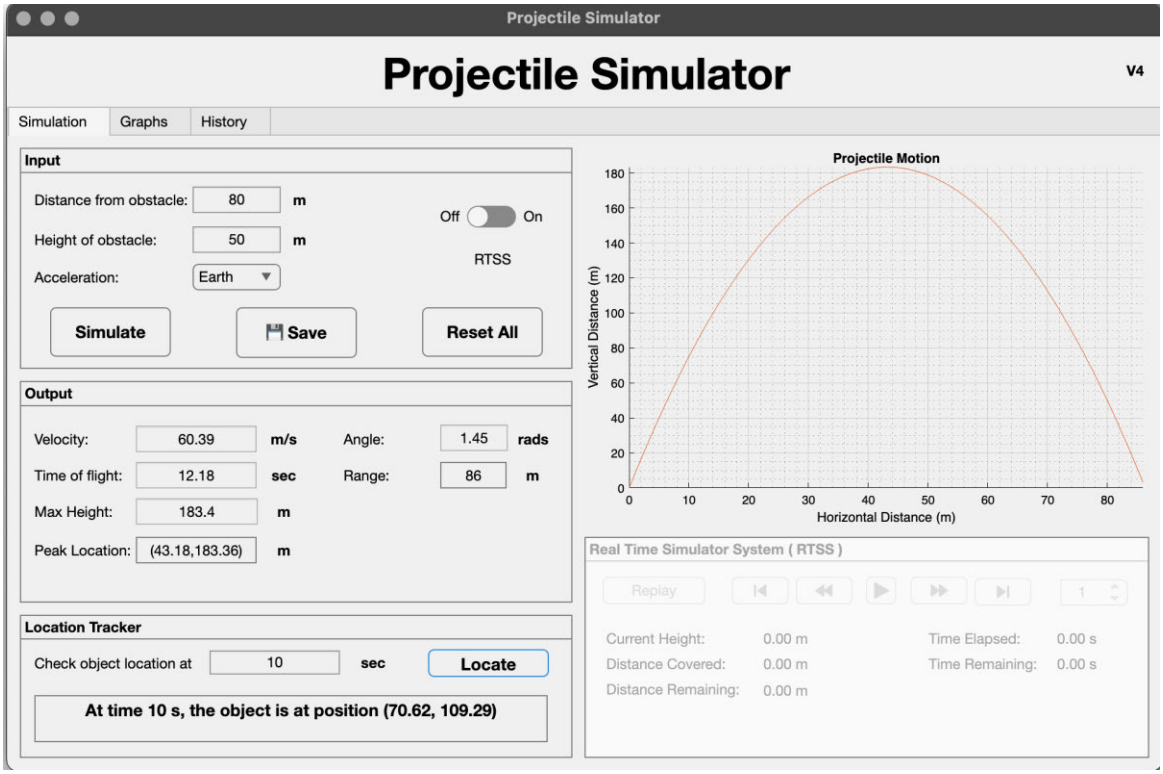


Figure 16 – Location Tracker

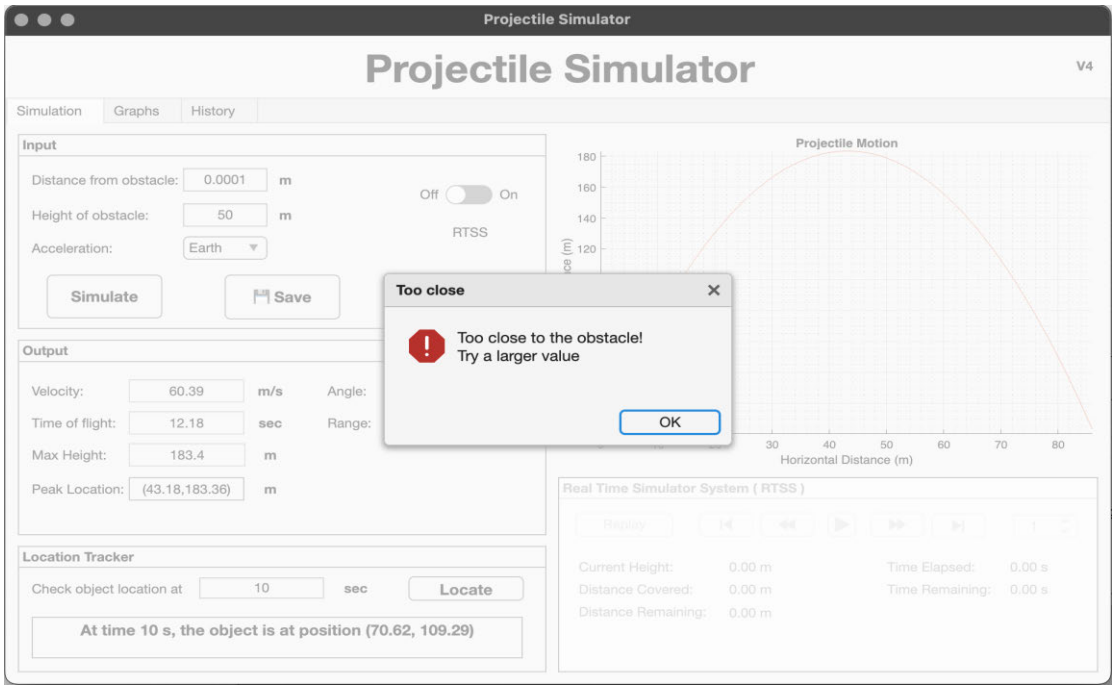


Figure 17 – Error Output

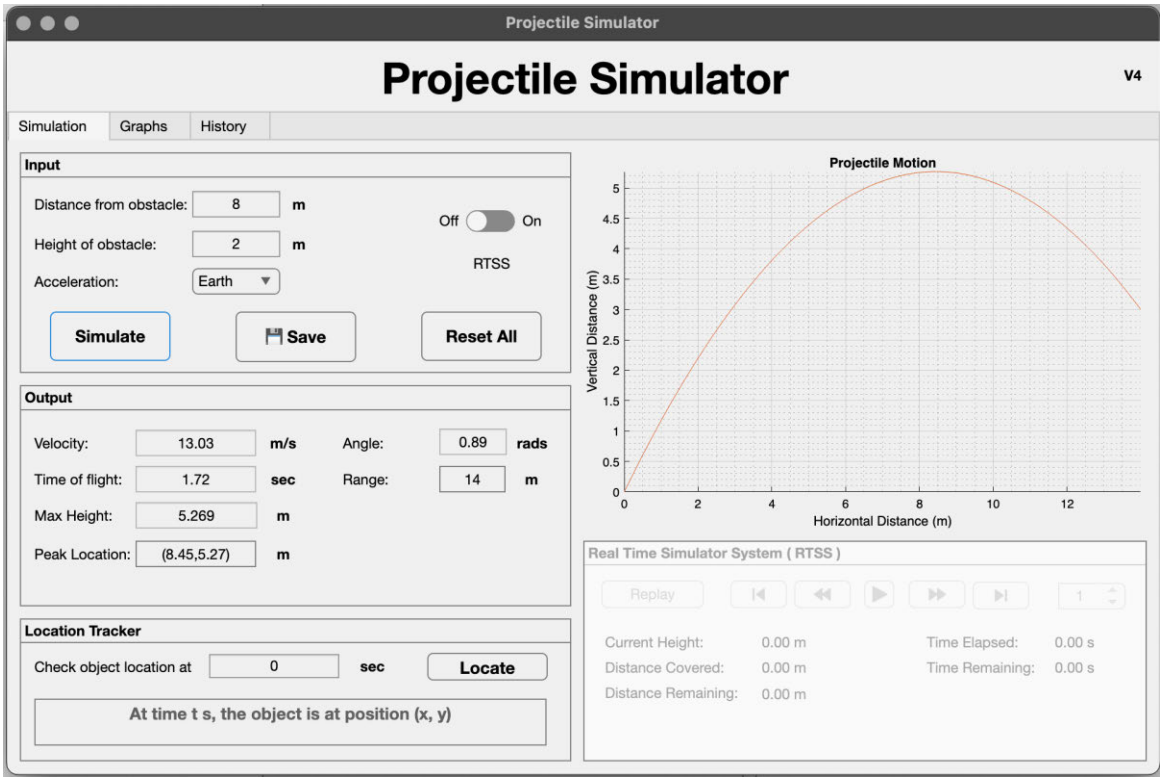


Figure 18 - When Height is less than 3

3.3 Graphs Tab

The Graphs Tab, illustrated in Figure 19, presents three graphical representations: the acceleration – time graph, the velocity – time graph, and the displacement – time graph. Underneath these graphs, various metrics are displayed, including the maximum and minimum values for displacement, velocity, and acceleration, as well as the average velocity, average acceleration, and jerk or jolt.

Upon clicking the "Simulate" button, the application generates the graphs immediately after calculating and populating the output values in the Simulate Tab.

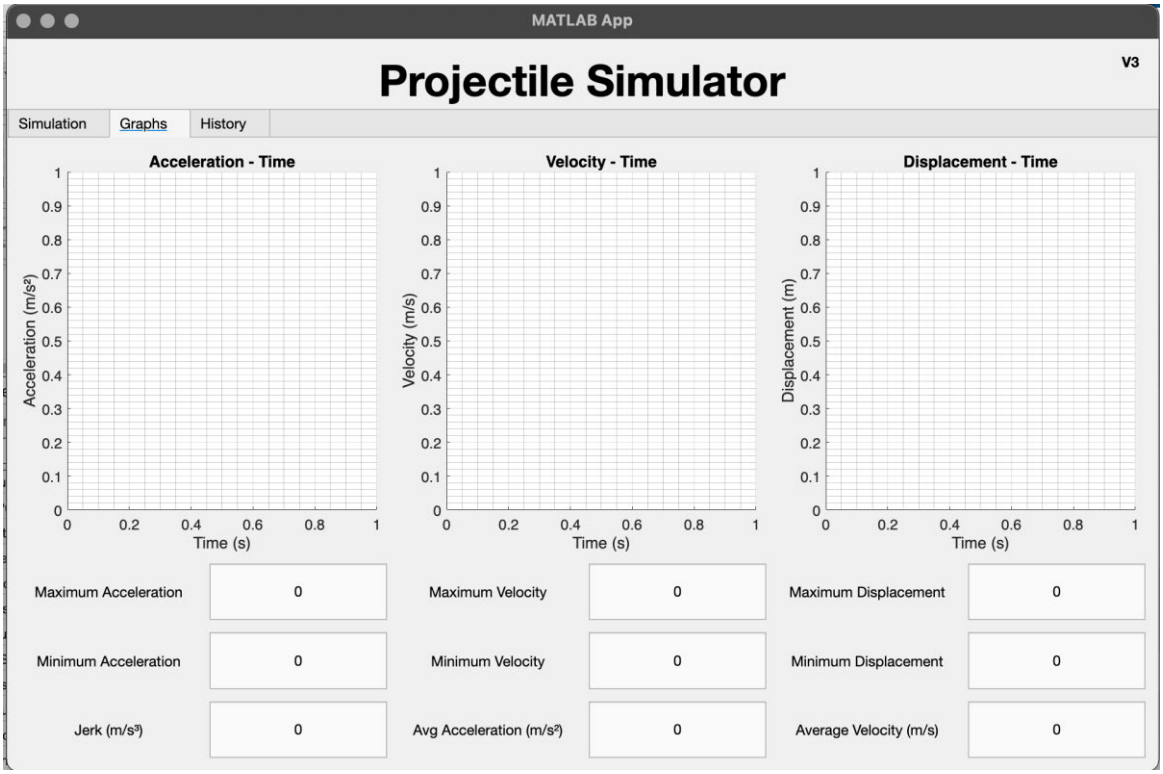


Figure 19 – Graphs Tab

3.3.1 Functions (Graphs Tab)

The *plotdisp* function obtains the distance to the obstacle (D) and the gravitational acceleration (g) from the *inputs* function. It then utilizes the *calculate* function to determine the initial velocity, initial angle, and time of flight. With these values, *plotdisp* generates an array of time values and computes the displacement using Equation 3.

Additionally, *plotdisp* calculates the maximum and minimum displacements, as well as the average velocity of the object's motion. Figure 20 shows the *plotdisp* function.

```
% This function plots displacement against time in graphs tab
function plotdisp(app)
    [~, ~, ~, ~, D, g] = inputs(app);
    [V, alpha, tofl, ~, ~, ~, ~, ~] = calculate(app, D, g);
    x = linspace(0, tofl, 1e+5); % Time array
    y = V * sin(alpha) * x - 0.5 * g * x.^2; % Displacement based on time
    plot(app.DisplacementTimeGraph, x, y)
    app.MaxDisplacement.Value = max(y); % Max Displacement
    app.AvgVelocity.Value = V * cos(alpha); % Average Velocity
end
```

Figure 20 – Displacement Time Graph Function

The *plotvel* function receives two parameters from the *inputs* function: the distance to the obstacle (D) and the gravitational acceleration (g). It invokes the *calculate* function to determine the initial velocity and the total time the projectile remains in flight. With these values, *plotvel* generates a linearly spaced vector of time and employs Equation 4 to compute the velocity at each time instant.

Furthermore, *plotvel* evaluates the highest and lowest velocities attained by the projectile during its trajectory. It also calculates the average acceleration experienced by the projectile over the course of its motion. The code of *plotvel* is shown in Figure 21.

```
% This fucntions plots velocity against time in the graphs tab
function plotvel(app)
    [~, ~, ~, ~, D, g] = inputs(app);
    [V, ~, tofl, ~, ~, ~, ~, ~] = calculate(app, D, g);
    x = linspace(0, tofl, 1e+5); % Time array
    y = linspace(-V, 0, 1e+5); % Velocity array
    plot(app.VelocityTimeGraph, x, y)
    app.MinVelocity.Value = -V; % Max velocity
    app.AvgAcceleration.Value = -g; % Average acceleration
end
```

Figure 21 – Velocity Time Graph Function

The *plotacc* function obtains the obstacle distance (D) and gravity due to acceleration (g) from *inputs*, calls *calculate* for the projectile's flight time, and creates a time array. Since gravity is constant, the projectile's acceleration remains unchanged throughout its motion, determined solely by the gravitational acceleration value from *inputs*. Jerk is not calculated since it is the change in acceleration over time, which is 0 in this case. Figure 22 displays the *plotacc* function.

```
% This function plots acceleration against time in the graphs tab
function plotacc(app)
    [~, ~, ~, ~, D, g] = inputs(app);
    [~, ~, tofl, ~, ~, ~, ~, ~] = calculate(app, D, g);
    x = linspace(0, tofl, 1e+5); % Time array
    y = linspace(-g, -g, 1e+5); % Acceleration array
    plot(app.AccelerationTimeGraph, x, y)
    app.MaxAcceleration.Value = -g; % Max Acceleration
    app.Jerk.Value = 0; % Jerk = 0 since dy/dt(constant)
end
```

Figure 22 – Acceleration – Time Graph

3.3.2 Callbacks (Graphs Tab)

The *plotdisp*, *plotvel* and *plotacc* functions are called when the Simulate button is pressed on the Simulate tab. These functions plot the acceleration – time graph, the velocity – time graph, and the displacement – time graph. Furthermore, they calculate and set the extrema for displacement, velocity, and acceleration, along with average velocity, average acceleration, and jerk.

The callback for the functions is shown in Figure 23.

```
function simulateButtonPushed(app, event)
    ...
    plotdisp(app); % Plot displacement time graph
    plotvel(app); % Plot velocity time graph
    plotacc(app); % Plot displacement time graph
    ...
end
```

Figure 23 – Velocity Time Graph Function

3.3.3 Output (Graphs Tab)

When the simulate button is pressed, the graphs tab is populated with 3 distinct graphs, which are the displacement – time, velocity – time, acceleration – time graphs. *Figure 24* shows a sample output.

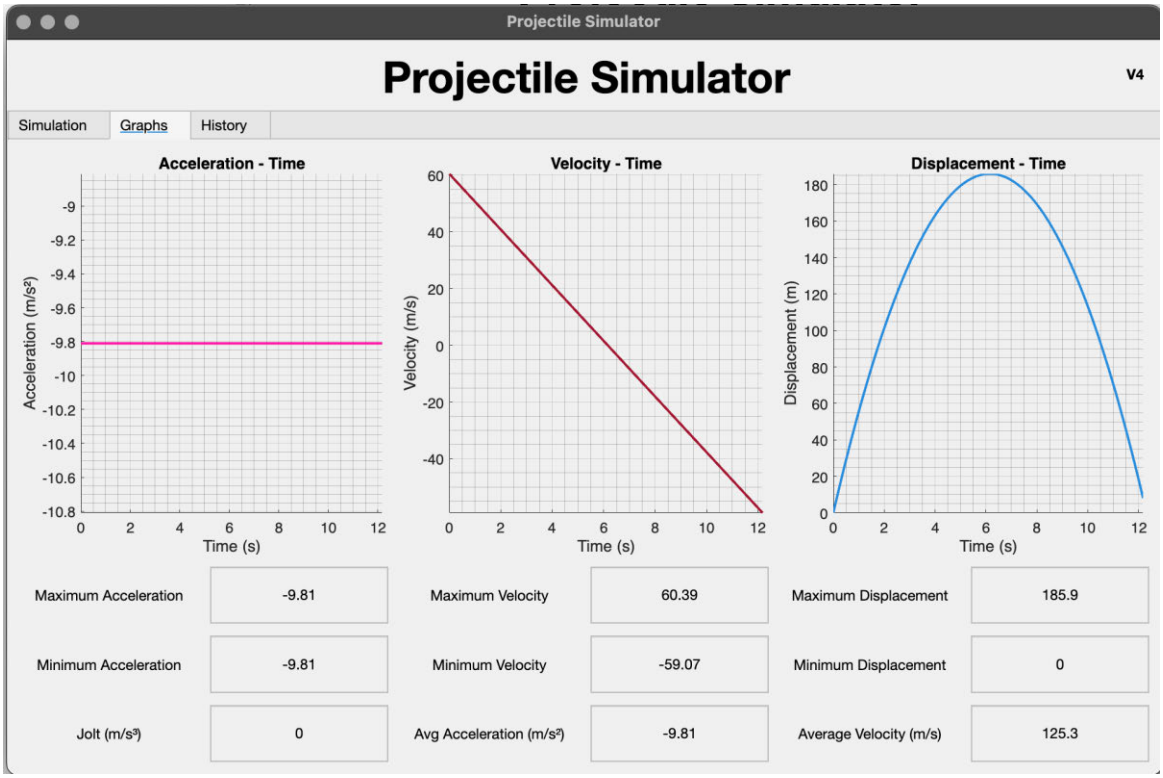


Figure 24 – Output of Graphs Tab

3.4 History Tab

The history tab is the third and final tab in the application. This tab stores information that the user chooses to save by clicking the Save button. It compiles all plots into a single graph that enables the user to compare inputs and facilitates comparison of all, or specific trial inputs. Figure 25 displays the history tab.

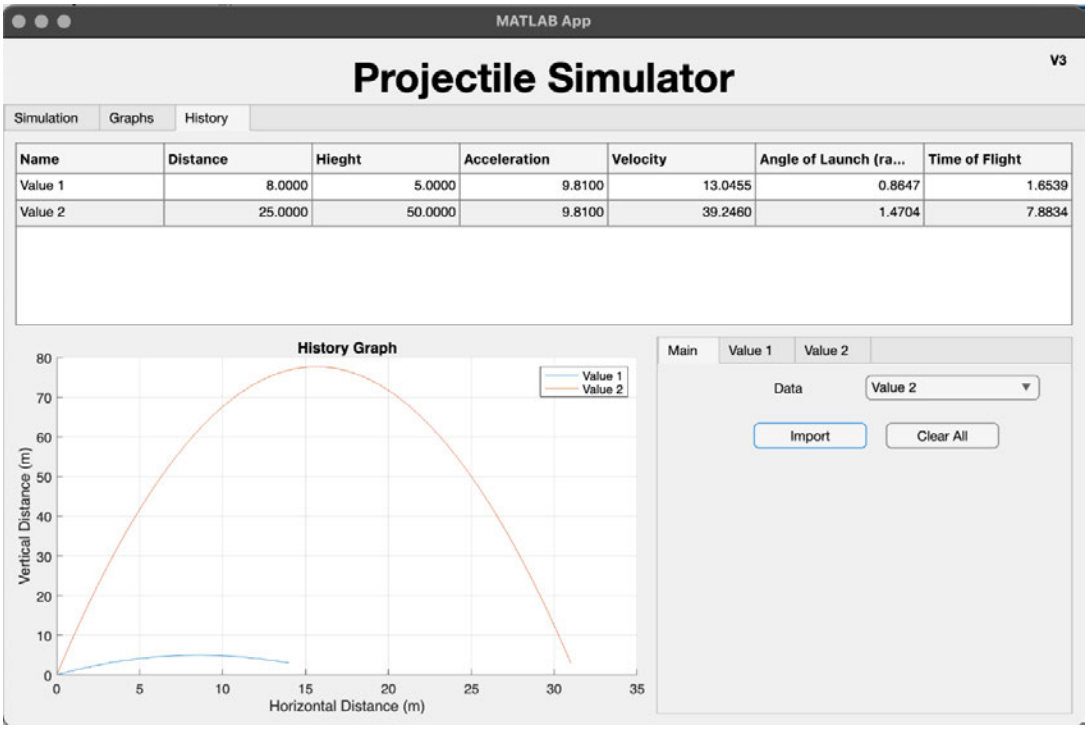


Figure 25 - History Tab

When the user saves a certain trial from the simulation tab, it appears on the history tab in the table. This table has the following columns: the name of the saved trial, distance the projectile traveled, maximum height of the projectile during flight, its acceleration, velocity, angle of launch, and time of flight. The graphing system under it permits the user to view the saved trials in a visual manner that allows comparison of different inputs. The panel next to it dedicates each tab (Value 1, Value 2, etc.) to a specific saved trial to show the parameters such as peak value, time of flight, range, etc.

3.4.2 Functions (History Tab)

The function *newTab* creates a new tab and populates it with interface components. It has five input arguments: name, tofh, rangeh, maxh, peakvalue. The tab is created with a specific title 'name' and is stored in app.new_tab. Labels and Edit Fields are created for displaying time of flight, maximum height, range, and peak location values.

```
% Function for creating a new tab with the imported data
function newtab(app,name,tofh,rangeh,maxh,peakvalue)

    tabname = uitab(app.TabGroup2,"Title",name);
    app.new_tab = [app.new_tab, tabname];
    ...
    peak_tab.Position = [93 247 100 22];
    peak_tab.Value = peakvalue;

end
```

Figure 26 – New Tab Function

importf first retrieves the data from the app's components. 'val' is a row vector extracted from 'db' based on the 'index' selected. The *calculate* function returns values assigned to tofl, rangeh, maxh, and peakvalue. This function imports and processes data based on user selection, to compute the metrics and update accordingly.

```
function importf(app)
    db = app.SimulationH.Data; % Import data from the table in History tab
    name = app.DataDropDown.Value; % Get the name of the data
    index = app.DataDropDown.ValueIndex; % Get the index of the drop down menu
    val = db(index,:); % Get the values of the row
    D = cell2mat(val(2)); % Get the value of Distance
    g = cell2mat(val(4)); % Get the value of gravity
    [~, ~, tofl, rangeh, ~, max_height, ~, peakvalue] = calculate(app, D, g); % Calculate the values
    newtab(app,name,tofl,rangeh,max_height,peakvalue)   % Create a new tab with the values
end
```

Figure 27 – importf Function

The *plothis* function is designed to plot based on the data selected from the dropdown menu. The 'index' gets the corresponding index from the dropdown menu in 'db' containing the data, whereas 'val' is the specific index. It extracts the parameters such as the acceleration, initial velocity, launch angle and time of flight needed to plot and computes the trajectory and updates the legend.

```
    function plothis(app)
        index = app.DataDropDown.ValueIndex; % Get the index of the data
        db = app.SimulationH.Data; % Get the data
        val = db(index,:); % Get the full array of Row
        name = cellstr(val(1)); % Get the name
        g= cell2mat(val(4)); % Get the gravity
        V= cell2mat(val(5)); % Get the velocity
        alpha = cell2mat(val(6));   % Get the angle
        tofl = cell2mat(val(7));  % Get the Time of flight
        t = linspace(0, tofl, 100); % Time array
        x = V * cos(alpha) * t; % X - Axis
        y = V * sin(alpha) * t - 0.5 * g * t.^2; % Y - Axis
        app.hisname = [app.hisname; name]; % Append the name to the array
        plot(app.Historygraph,x,y) % Plot the graph
        legend(app.Historygraph,app.hisname) % Add the legend
        hold(app.Historygraph,"on") % Hold the graph
    end
end
```

Figure 28 – Plothis Function

3.4.2 Callbacks (History Tab)

ImportButtonPushed is a function that is carried out when the user clicks on the Import button. It first calls the *import* function that imports the data and then invokes *plothis* which processes it.

```
% Button pushed function: ImportButton
function ImportButtonPushed(app, event)
    importf(app); % Import function
    plothis(app); % Plot history
end
```

Figure 29 – Import Button Pushed

In the *HistoryTabButtonDown* function, the 'db' database gets the Data stored in SimulationH, and its 'height', which refers to number of values, is stored in h. It then returns the number of rows in db. The loop iterates each row of 'db' and extracts the value in the column called 'name', then appends that 'name' to the items in the dropdown menu.

```
% Button down function: HistoryTab
function HistoryTabButtonDown(app, event)
    db = app.SimulationH.Data; % Get the data
    h = height(db);        % Get the height of the data
    app.DataDropDown.Items = {}; % Clear the drop down menu
    app.DataDropDown.ItemsData ={}; % Clear the drop down menu
    for i = 1:h % Loop through the data
        name = db(i,1); % Get the name
        ddrop = app.DataDropDown.Items; % Get the drop down menu
        app.DataDropDown.Items = [ddrop name]; % Append the name to the drop down menu
    end
end
```

Figure 30 – History Tab Button Down Function

The purpose of the *simulatorButtonDown* function is to clear the items from the dropdown menu by setting items and itemsData as empty arrays. It is triggered when the user interacts with the simulator tab. This ensures that the latest data is fetched every time the user visits the History page.

```
% Button down function: simulator
function simulatorButtonDown(app, event)
    % Clear the drop down menu
     app.DataDropDown.Items = {};
     app.DataDropDown.ItemsData ={};
end
```

Figure 31 – Simulator Button Down Function

ClearAllButtonPushed is set to be called when the Clear All button is pushed. It is designed to clear the plot on the History tab and removes all values in the dropdown menu. The function clears the hold on the history tab and iterates through each tab and deletes each one.

```
% Button pushed function: ClearAllButton
function ClearAllButtonPushed(app, event)
    hold(app.Historygraph,"off") % Hold the graph off
    app.hisname = []; % Clear the history name
    plot(app.Historygraph,0,0) % Plot the graph
    legend(app.Historygraph, 'off') % Turn off the legend
    tabs = app.new_tab; % Get the tabs
    for i  = tabs % Loop through the tabs
        delete(i) % Delete the tabs
    end
end
```

Figure 32 – Clear All Button Pushed Function

3.4.3 Output (History Tab)

When the user saves a trial from the Simulator tab, it is added to the table in the History tab. Say for example the name of the first trial is saved as ‘Value’ where the distance from the obstacle is 80 meters and the height of the obstacle is 50m. This saved value will show up as the first row in the table on the History tab.

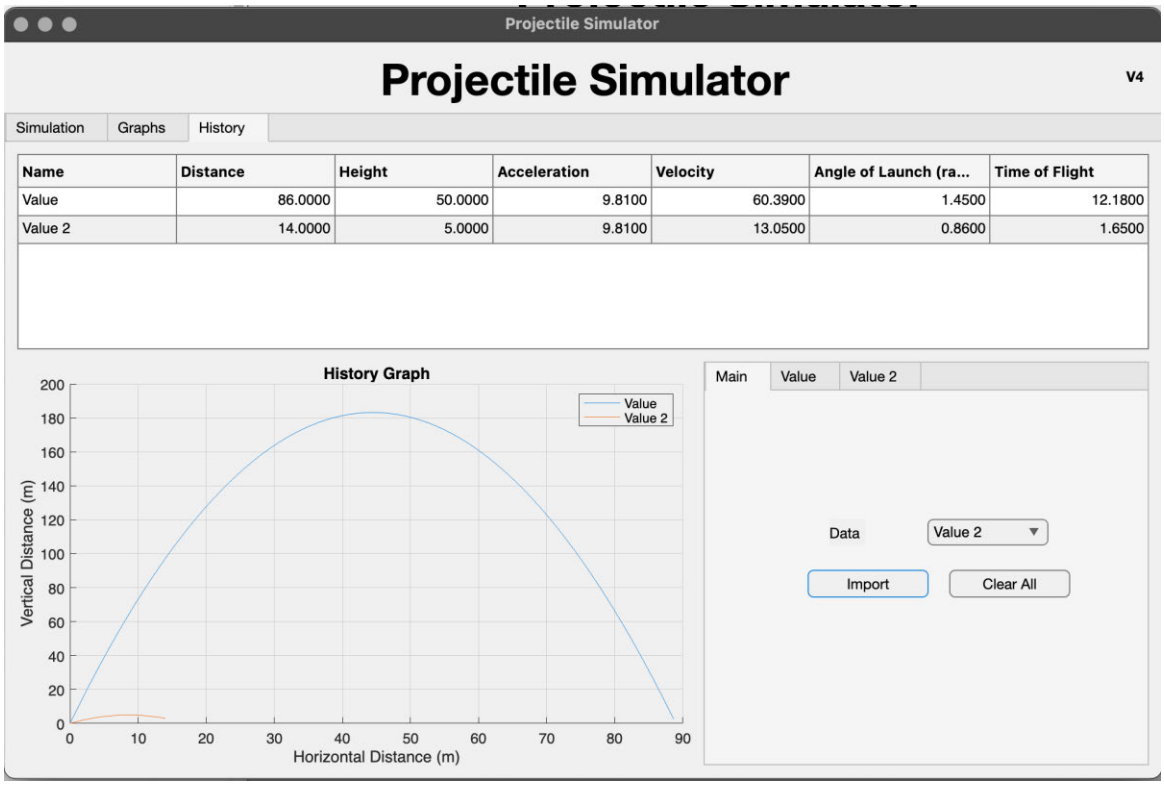


Figure 33 – History Output

Chapter 4: Conclusions

This project's purpose was to explore, analyze, and simulate the motion of an object to land at a target position behind the obstacle, where the target is 3 meters above the ground and 6 meters away from it. The program asked the user to input the distance (D) from the obstacle and the height (H) of the obstacle to find the velocity of the projectile and the angle with which it was launched to reach the target position in that system. Additionally, it calculated the time of flight and maximum height the projectile reached. The user is also able to determine what the peak location of the projectile was from this information. With MATLAB, this project design is a significant device in the interpretation of projectile dynamics, which is mandatory in engineering physics and has many applications such as ballistics and artillery.

By now, complete apprehension of the fundamental dynamic's equations and the development of MATLAB programs has been mastered. The importance of implementing position tracking based on various initial conditions has also been grasped. Furthermore, the creation of a user – friendly interface has been learned, and the optimization of launch angle and velocity parameters has been mastered, enabling the display of graphs within the graphical user interface (GUI) to be accomplished.

The approach to achieving all objectives was best when the derivations of equations were first done carefully and then programmed into MATLAB with the help of various online sources and guidance from tutors. Proficiency in iterative calculations like

loops contributed to the precision of results. Lastly and most importantly the creation of GUI enhanced user experience. Following this approach the project successfully combines theory and practice to produce a versatile instrument in the field.

References

[1] J. Adams, "MATLAB Projectile Motion Demo," YouTube, 01 August 2016. [Online]. Available: https://youtu.be/g9wwYyR7PgE. [Accessed 10 May 2024].

[2] M. Kuchle, "MATLAB – Projectile Motion Function & GUI," YouTube, 07 May 2018. [Online]. Available: https://youtu.be/i6Abe7VuABQ. [Accessed 10 May 2024].

[3] MathWorks, Inc., "Algebraic Loop Concepts," MathWorks, Inc., [Online]. Available: https://au.mathworks.com/help/simulink/ug/algebraic-loops.html?searchHighlight=loops&s_tid=srchtitle_support_results_1_loops%23d126e10921. [Accessed 12 May 2024].

[4] Khan Academy, "Plotting projectile displacement, acceleration, and," YouTube, 15 June 2011. [Online]. Available: https://youtu.be/T0zpF_j7Mvo. [Accessed 10 June 2024].

[5] MathWorks, Inc., "Optimization Toolbox," [Online]. Available: https://www.mathworks.com/help/optim/index.html. [Accessed 20 May 2024].

[6] MathWorks, Inc., "Solve systems of nonlinear equations – MATLAB fsolve," MathWorks, Inc., [Online]. Available: https://www.mathworks.com/help/optim/ug/fsolve.html. [Accessed 20 May 2024].

[7] MathWorks, Inc., "Optimization Options – MATLAB optimoptions," MathWorks, Inc., [Online]. Available: https://www.mathworks.com/help/optim/ug/optim.problemdef.optimizationproblem.optimoptions.html. [Accessed 20 May 2024].

[8] MathWorks, Inc., "Optimization options reference," MathWorks, Inc., [Online]. Available: https://www.mathworks.com/help/optim/ug/optimization-options-reference.html. [Accessed 20 May 2024].

[9] MathWorks, Inc., "Creating a function handle," MathWorks, Inc., [Online]. Available: https://www.mathworks.com/help/matlab/matlab_prog/creating-a-function-handle.html. [Accessed 20 May 2024].

Appendix

Appendix A: Complete Code

```
classdef Projectile_Simulator_Exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        ProjectileSimulatorUIFigure  matlab.ui.Figure
        Versionlabel                 matlab.ui.control.Label
        ProjectileSimulatorLabel     matlab.ui.control.Label
        TabGroup                     matlab.ui.container.TabGroup
        simulator                    matlab.ui.container.Tab
        simlayout                    matlab.ui.container.GridLayout
        InputPanel                   matlab.ui.container.Panel
        Distance                     matlab.ui.control.NumericEditField
        Height                       matlab.ui.control.NumericEditField
        acceleration                 matlab.ui.control.DropDown
        RTSSSwitch                   matlab.ui.control.Switch
        simulate                     matlab.ui.control.Button
        SaveButton                   matlab.ui.control.Button
        resetall                     matlab.ui.control.Button
        RTSSSwitchLabel              matlab.ui.control.Label
        mLabel_3                     matlab.ui.control.Label
        mLabel                       matlab.ui.control.Label
        RangeLabel                   matlab.ui.control.Label
        IntialHeightLabel            matlab.ui.control.Label
        AccelerationLabel            matlab.ui.control.Label
        OutputPanel                  matlab.ui.container.Panel
        mLabel_5                     matlab.ui.control.Label
        velocity                     matlab.ui.control.NumericEditField
        VelocityLabel                matlab.ui.control.Label
        timeofflight                 matlab.ui.control.NumericEditField
        TimeofflightLabel            matlab.ui.control.Label
        MH                           matlab.ui.control.NumericEditField
        MaxHieghtLabel               matlab.ui.control.Label
        peak                         matlab.ui.control.EditField
        PeakLocationEditFieldLabel   matlab.ui.control.Label
        angle                        matlab.ui.control.NumericEditField
        AngleofLaunchLabel           matlab.ui.control.Label
        Range                        matlab.ui.control.EditField
        RangeLabel_2                 matlab.ui.control.Label
        msLabel                      matlab.ui.control.Label
        secLabel_2                   matlab.ui.control.Label
        mLabel_2                     matlab.ui.control.Label
        radsLabel                    matlab.ui.control.Label
        mLabel_4                     matlab.ui.control.Label
        LocationTrackerPanel         matlab.ui.container.Panel
        timelocate                   matlab.ui.control.NumericEditField
        locate                       matlab.ui.control.Button
        secLabel                     matlab.ui.control.Label
        locationdisp                 matlab.ui.control.TextArea
        CheckobjectlocationatLabel   matlab.ui.control.Label
        RTSSPanel                    matlab.ui.container.Panel
        Skipforward                  matlab.ui.control.Button
        TimeElapsedLabel             matlab.ui.control.Label
        TimeElapsedValue             matlab.ui.control.Label
        TimeRemainingLabel           matlab.ui.control.Label
        TimeRemainingValue           matlab.ui.control.Label
        CurrentHeightLabel           matlab.ui.control.Label
        CurrentHeightValue           matlab.ui.control.Label
        DistanceCoveredLabel         matlab.ui.control.Label
```

```
    DistanceCoveredValue          matlab.ui.control.Label
    DistanceRemainingLabel        matlab.ui.control.Label
    DistanceRemainingValue        matlab.ui.control.Label
    ReplayButton                  matlab.ui.control.Button
    FastRewindButton              matlab.ui.control.Button
    Skipback                      matlab.ui.control.Button
    PlayPauseButton               matlab.ui.control.StateButton
    FastForwardButton             matlab.ui.control.Button
    InputPlaybackSpeed            matlab.ui.control.Spinner
    MainGraph                     matlab.ui.control.UIAxes
    GraphsTab                     matlab.ui.container.Tab
    GridLayout                    matlab.ui.container.GridLayout
    Jerk                          matlab.ui.control.NumericEditField
    MaxDisplacementLabel_10       matlab.ui.control.Label
    MinAcceleration               matlab.ui.control.NumericEditField
    MaxDisplacementLabel_9        matlab.ui.control.Label
    MaxAcceleration               matlab.ui.control.NumericEditField
    MaxDisplacementLabel_8        matlab.ui.control.Label
    AvgAcceleration               matlab.ui.control.NumericEditField
    MaxDisplacementLabel_7        matlab.ui.control.Label
    MinVelocity                   matlab.ui.control.NumericEditField
    MaxDisplacementLabel_5        matlab.ui.control.Label
    MaxVelocity                   matlab.ui.control.NumericEditField
    MaxDisplacementLabel_4        matlab.ui.control.Label
    AvgVelocity                   matlab.ui.control.NumericEditField
    MaxDisplacementLabel_3        matlab.ui.control.Label
    MinDisplacement               matlab.ui.control.NumericEditField
    MaxDisplacementLabel_2        matlab.ui.control.Label
    MaxDisplacement               matlab.ui.control.NumericEditField
    MaxDisplacementLabel          matlab.ui.control.Label
    DisplacementTimeGraph         matlab.ui.control.UIAxes
    VelocityTimeGraph             matlab.ui.control.UIAxes
    AccelerationTimeGraph         matlab.ui.control.UIAxes
    HistoryTab                    matlab.ui.container.Tab
    GridLayout2                   matlab.ui.container.GridLayout
    TabGroup2                     matlab.ui.container.TabGroup
    MainTab                       matlab.ui.container.Tab
    ClearAllButton                matlab.ui.control.Button
    DataDropDown                  matlab.ui.control.DropDown
    DataDropDownLabel             matlab.ui.control.Label
    ImportButton                  matlab.ui.control.Button
    SimulationH                   matlab.ui.control.Table
    Historygraph                  matlab.ui.control.UIAxes
end
properties (Access = private)
    X % Main - X Values
    Y % Main - Y Values
    T % Main Time
    TMAX % Maximum Time

    sAni = true % Should the program animate
    isAni = true % Is the program animating
    isPlay = true % Is the program playing the animation
    isUpdat = true % Is the program updating
    cTime = 0 % Start Time
    pSpeed = 1 % Playback speed
    timerr = timer('ExecutionMode','fixedRate', ...
        'Period', 0.1, ...
        'TimerFcn', @(~,~) updateRTSS(app));
    hisname = [] % Array of data for importing in graph (History Tab)
```

```
    playIcon = 'Play.png' % Play Icon
    pauseIcon = 'Pause.png' % Pause Icon
    new_tab = [] % TODO
end

methods (Access = public)
    % Simulation Tab

    % This is the input function of the program
    function [D, H, y_b, d, xf, g] = inputs(app)
        D = app.Distance.Value; % Distance from obstacle
        H = app.Height.Value; % Height of obstacle
        a = app.acceleration.Value; % Acceleration ie. Gravity
        y_b = 3;    % Height of the final position in meters
        d = 6;      % Distance from obstacle to final position in meters
        xf = D + d;  % Total horizontal distance
        if a == "Earth"
            g = 9.81; % Earth Gravity
        elseif a == "Moon"
            g = 1.62; % Moon Gravity
        elseif a == "Mars"
            g = 3.71; % Mars Gravity
        end
    end

    % Define the system of equations to solve
    function F = eqnss(app, vars)
        [D, H, y_b, ~, xf, g] = inputs(app); % Get required inputs
        alpha = vars(1); % Angle initial guess
        v0 = vars(2); % Velocity initial guess
        safety_margin = 1e-6; % Margin to avoiid hitting obstacle

        % Equation for the vertical position at the final position
        eq1 = y_b - (xf * tan(alpha) - (1/2) * g * (xf / (v0 * cos(alpha)))^2);

        % If the building height is less than the basket height, the building constraint can be ignored
        if H < y_b
            eq2 = 0;
        else
            % Equation for the height at the obstacle
            eq2 = H + safety_margin - (D * tan(alpha) - (1/2) * g * (D / (v0 * cos(alpha)))^2);
        end

        % Return the system of equations
        F = [eq1; eq2];
    end

    function [V, alpha, tofl, rangeh, t_max_height, max_height, x_max_height, peakvalue] = calculate(app, xf, g)

        % Initial guesses for alpha and v0 1e-6 degrees and 1e-6 m/s
        initial_guess = [1e-6 * pi / 180, 1e-6];

        % Generate function for fsolve
        fun = @(initial_guess)eqnss(app, initial_guess);

        % Solve the system of equations using fsolve
        % MaxFunEvals - How many times can the function be checked with values
        % MaxIterations - How many iterations of paramter can be made
        options = optimoptions('fsolve', 'MaxFunEvals', 1e+50, 'MaxIterations', 1e+50);
        solution = fsolve(fun, initial_guess, options); % fsolve function with initial guess and extra options

        % Extract the solutions
        alpha = abs(solution(1)); % Angle
        V = abs(solution(2)); % Velocity

        tofl = xf / (V * cos(alpha)); % Time of flight
        rangeh = xf; % Max X
        t_max_height = V * sin(alpha) / g; % Time at max height
        max_height = V * sin(alpha) * t_max_height - 0.5 * g * t_max_height^2; % Max height
        x_max_height = V * cos(alpha) * t_max_height ; % Max Height X - Coordinates % X coordinate at max height
        peakvalue = ['(',num2str(round(x_max_height, 2)),',',num2str(round(max_height, 2)),')'];
    end
```

```
function pauseRTSS(app) % This function pauses the animation
    app.isPlay = false; % Set the play to false
    app.isUpdat = false; % Set the update to false
    stop(app.timerr) % Stop the timer
    app.PlayPauseButton.Icon = app.playIcon;    % Change the icon to play
    app.timerr.running
end

function resumeRTSS(app) % This function resumes the animation
    app.isPlay = true; % Set the play to true
    start(app.timerr);  % Start the timer
    app.PlayPauseButton.Icon = app.pauseIcon; % Change the icon to pause
end

function startRTSS(app) % This function starts the RTSS
    app.cTime = 0; % Set the current time to 0
    app.isAni = true;  % Set the animation to true
    app.isPlay = true;  % Set the play to true
    app.PlayPauseButton.Icon = app.pauseIcon;   % Change the icon to pause
    app.timerr = timer('ExecutionMode','fixedRate', ...
        'Period', 0.1, ...
        'TimerFcn', @(~,~) updateRTSS(app)); % Set the timer function
    start(app.timerr);  % Start the timer
end

function updateRTSSv(app, x, y, t) % This function updates the values of the RTSS
    distanceRemaining = max(app.X) - x; % Distance remaining
    timeRemaining = app.TMAX - t; % Time remaining
    app.CurrentHeightValue.Text = sprintf('%.2f m', y); % Current Height
    app.DistanceCoveredValue.Text = sprintf('%.2f m', x);   % Distance Covered
    app.DistanceRemainingValue.Text = sprintf('%.2f m', distanceRemaining); % Distance Remaining
    app.TimeElapsedValue.Text = sprintf('%.2f s', t); % Time Elapsed
    app.TimeRemainingValue.Text = sprintf('%.2f s', timeRemaining); % Time Remaining
end
```

```
function pauseRTSS(app) % This function pauses the animation
    app.isPlay = false; % Set the play to false
    app.isUpdat = false; % Set the update to false
    stop(app.timerr) % Stop the timer
    app.PlayPauseButton.Icon = app.playIcon;    % Change the icon to play
    app.timerr.running
end

function resumeRTSS(app) % This function resumes the animation
    app.isPlay = true; % Set the play to true
    start(app.timerr);  % Start the timer
    app.PlayPauseButton.Icon = app.pauseIcon; % Change the icon to pause
end

function startRTSS(app) % This function starts the RTSS
    app.cTime = 0; % Set the current time to 0
    app.isAni = true;  % Set the animation to true
    app.isPlay = true;  % Set the play to true
    app.PlayPauseButton.Icon = app.pauseIcon;   % Change the icon to pause
    app.timerr = timer('ExecutionMode','fixedRate', ...
        'Period', 0.1, ...
        'TimerFcn', @(~,~) updateRTSS(app)); % Set the timer function
    start(app.timerr);  % Start the timer
end

function updateRTSSv(app, x, y, t) % This function updates the values of the RTSS
    distanceRemaining = max(app.X) - x; % Distance remaining
    timeRemaining = app.TMAX - t; % Time remaining
    app.CurrentHeightValue.Text = sprintf('%.2f m', y); % Current Height
    app.DistanceCoveredValue.Text = sprintf('%.2f m', x);   % Distance Covered
    app.DistanceRemainingValue.Text = sprintf('%.2f m', distanceRemaining); % Distance Remaining
    app.TimeElapsedValue.Text = sprintf('%.2f s', t); % Time Elapsed
    app.TimeRemainingValue.Text = sprintf('%.2f s', timeRemaining); % Time Remaining
end

function tweakRTSS(app, pspeed) % This function tweaks the speed of the animation
    if app.isAni
        pauseRTSS(app); % Pause the RTSS
        app.cTime = app.cTime + (app.pSpeed / pspeed); % Update the time
```

```
        if app.cTime < 0 % If time is less than 0
            app.cTime = 0; % Set time to 0
        end
        updateRTSS(app); % Update the RTSS
    end
end

function rtssenable(app) % This function enables or disables the RTSS
    rs = app.RTSSSwitch.Value;
    if strcmp(rs,'On') % Check the RTSS switch if it is ON and enables the RTSS panel
        app.RTSSPanel.Enable = "on";
        app.ReplayButton.Enable = "on";
        app.FastForwardButton.Enable = "on";
        app.FastRewindButton.Enable= "on";
        app.ReplayButton.Enable= "on";
        app.Skipback.Enable= "on";
        app.PlayPauseButton.Enable = "on";
        app.Skipforward.Enable= "on";
        app.InputPlaybackSpeed.Enable= "on";
        startRTSS(app);
        app.sAni = true;
    else % If off then disables the panel along with its buttons
        app.sAni = false;
        app.plotgraph;
        app.RTSSPanel.Enable = "off";
        app.ReplayButton.Enable = "off";
        app.FastForwardButton.Enable = "off";
        app.FastRewindButton.Enable= "off";
        app.ReplayButton.Enable= "off";
        app.Skipback.Enable= "off";
        app.PlayPauseButton.Enable = "off";
        app.Skipforward.Enable= "off";
        app.InputPlaybackSpeed.Enable= "off";
    end
end

% Graph Tab

% This function plots displacement against time in graphs tab
function plotdisp(app)
    [~, ~, ~, ~, xf, g] = inputs(app);
    [V, ~, tofl, ~, ~, ~, ~, ~] = calculate(app, xf, g);
    x = linspace(0, tofl, 1e+5); % Time array
        y = V * x - 0.5 * g * x.^2; % Displacement based on time
        plot(app.DisplacementTimeGraph, x, y, 'LineWidth', 2.0)
        app.MaxDisplacement.Value = max(y); % Max Displacement
        app.AvgVelocity.Value = mean(y); % Average Velocity
    end

    % This fucntions plots velocity against time in the graphs tab
    function plotvel(app)
        [~, ~, ~, ~, xf, g] = inputs(app);
        [V, ~, tofl, ~, ~, ~, ~, ~] = calculate(app, xf, g);
        x = linspace(0, tofl, 1e+5); % Time array
        y = V - g * x; % Velocity array
        plot(app.VelocityTimeGraph, x, y, 'LineWidth', 2.0)
        app.MinVelocity.Value = min(y); % Min velocity
        app.MaxVelocity.Value = max(y); % Max velocity
        app.AvgAcceleration.Value = -g; % Average acceleration
    end

    % This function plots acceleration against time in the graphs tab
    function plotacc(app)
        [~, ~, ~, ~, xf, g] = inputs(app);
        [~, ~, tofl, ~, ~, ~, ~, ~] = calculate(app, xf, g);
        x = linspace(0, tofl, 1e+5); % Time array
        y = linspace(-g, -g, 1e+5); % Acceleration array
        plot(app.AccelerationTimeGraph, x, y, 'LineWidth', 2.0)
        app.MaxAcceleration.Value = -g; % Max Acceleration
        app.MinAcceleration.Value = -g; % Min Acceleration
        app.Jerk.Value = 0; % Jerk = 0 since dy/dt(constant)
    end
```

```
% History Tab
function importf(app)
    db = app.SimulationH.Data; % Get the data from the table
    name = app.DataDropDown.Value; % Get the value from the dropdown
    index = app.DataDropDown.ValueIndex;    % Get the index of the value
    val = db(index,:); % Get the value from the table
    D = cell2mat(val(2)); % Get the distance
    g = cell2mat(val(4)); % Get the gravity
    [~, ~, tofl, rangeh, ~, max_height, ~, peakvalue] = calculate(app, D, g); % Calculate the values
    newtab(app,name,tofl,rangeh,max_height,peakvalue) % Create a new tab
end

function newtab(app,name,tofh,rangeh,maxh,peakvalue) % This function creates a new tab
    tabname = uitab(app.TabGroup2,"Title",name);
    app.new_tab = [app.new_tab, tabname];
    % Create TimeofflightLabel_2
    TimeofflightLabel_tab = uilabel(tabname);
    TimeofflightLabel_tab.Position = [8 278 78 22];
    TimeofflightLabel_tab.Text = 'Time of flight:';
    Timeofflight = uieditfield(tabname, 'text');
    Timeofflight.Editable = 'off';
    Timeofflight.HorizontalAlignment = 'center';
    Timeofflight.Value = num2str(tofh);
    Timeofflight.Position = [92 278 100 22];

    % Create MaxHeightLabel_2
    MaxHeightLabel_tab = uilabel(tabname);
    MaxHeightLabel_tab.Position = [8 215 70 22];
    MaxHeightLabel_tab.Text = 'Max Height:';

    % Create MH_2
    MH_tab = uieditfield(tabname, 'numeric');
    MH_tab.Editable = 'off';
    MH_tab.HorizontalAlignment = 'center';
    MH_tab.Position = [92 215 101 22];
    MH_tab.Value = maxh ;

    % Create RangeLabel_2
    RangeLabel_tab = uilabel(tabname);
    RangeLabel_tab.Position = [10 183 43 22];
    RangeLabel_tab.Text = 'Range:';

    % Create range_2
    range_tab = uieditfield(tabname, 'numeric');
    range_tab.Editable = 'off';
    range_tab.HorizontalAlignment = 'center';
    range_tab.Position = [92 183 100 22];
    range_tab.Value = rangeh;

    % Create PeakLocationEditFieldLabel_2
    PeakLocationEditFieldLabel_tab = uilabel(tabname);
    PeakLocationEditFieldLabel_tab.Position = [8 247 85 22];
    PeakLocationEditFieldLabel_tab.Text = 'Peak Location:';

    % Create peak_2
    peak_tab = uieditfield(tabname, 'text');
    peak_tab.Editable = 'off';
    peak_tab.HorizontalAlignment = 'center';
    peak_tab.Placeholder = '(x,y)';
    peak_tab.Position = [93 247 100 22];
    peak_tab.Value = peakvalue;
end

function plothis(app)
    index = app.DataDropDown.ValueIndex; % Get the index of the value
    db = app.SimulationH.Data; % Get the data from the table
    val = db(index,:); % Get the value from the table
    name = cellstr(val(1)); % Get the name
    g= cell2mat(val(4));   % Get the gravity
    V= cell2mat(val(5));  % Get the velocity
    alpha = cell2mat(val(6)); % Get the angle
    tofl = cell2mat(val(7));  % Get the time of flight
```

```
            t = linspace(0, tofl, 100); % Time array
            x = V * cos(alpha) * t; % X - Axis
            y = V * sin(alpha) * t - 0.5 * g * t.^2; % Y - Axis
            app.hisname = [app.hisname; name]; % Add the name to the history name
            plot(app.Historygraph,x,y) % Plot the graph
            legend(app.Historygraph,app.hisname)    % Add the legend
            hold(app.Historygraph,"on") % Hold the graph
        end
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: simulate
        function simulateButtonPushed(app, event)
            [D, ~, ~, ~, xf, g] = inputs(app);
            isValidDistance = true; % Initialize checking variable - Distance

            % Error if distance from obstacle is too less
            if D < 0.01
                uialert(app.ProjectileSimulatorUIFigure, ["Too close to the obstacle!", "Try a larger value"],
                "Too close", "Icon", "error"); % Error message
                isValidDistance = false;
            end

            % Check if both conditions are met
            if isValidDistance
                [V, alpha, tofl, ~, ~, max_height, ~, peakvalue] = calculate(app, xf, g);

                % Error if angle greater than 89.99
                if round(rad2deg(alpha), 2) > 89.99
                    uialert(app.ProjectileSimulatorUIFigure, "Angle is too large!", "Angle too large", "Icon", "error");
                else
                    app.velocity.Value = round(V, 2) ; % Displays Velocity
                    app.angle.Value = round(alpha, 2); % Displays Angle
                    app.timeofflight.Value = round(tofl, 2); % Displays Time of flight

                    rtssenable(app); % RTSS Enable

                    app.MH.Value = max_height; % Maximum Height
                    app.peak.Value = peakvalue; % Peak value (x,y)
                    app.Range.Value = num2str(xf); % Total range


                    plotdisp(app); % Plot displacement time graph
                    plotvel(app); % Plot velocity time graph
                    plotacc(app); % Plot displacement time graph

                    app.locate.Enable ="on"; % Enables the locate button
                    app.timelocate.Enable = "on"; % Enables the Time field
                    app.timelocate.Editable = "on"; % Makes time field editable
                    app.secLabel.Enable ="on";

                    t = 0:1e-5:tofl; % Time array
                    x = V * cos(alpha) * t; % X - Axis
                    y = V * sin(alpha) * t - 0.5 * g * t.^2; % Y - Axis

                    y(y < 0) = 0; % Condition to make sure Y is 0
                    app.X = x; % Sets RTSS X Array
                    app.Y = y; % Sets RTSS Y Array
                    app.T = t; % Sets RTSS T Array
                    app.TMAX = tofl; % Sets RTSS TMAX

                    app.SaveButton.Enable = "on"; % Enables Save button
                end
            end
        end
```

```
    % Button pushed function: resetall
    function resetallButtonPushed(app, event)
        % Reset all values and set to default

        % Input Panel
        app.Distance.Value = 0;
        app.Height.Value = 0;
        app.acceleration.Value = "Earth";
        app.RTSSSwitch.Value = "Off";

        % Output Panel
        app.velocity.Value = 0;
        app.timeofflight.Value = 0;
        app.MH.Value = 0;
        app.angle.Value = 0;
        app.Range.Value = "0";

        app.peak.Value = "(x,y)";

        % Main Graph
        plot(app.MainGraph,0,0)

        % Location Tracker
        app.timelocate.Value = 0;
        app.locationdisp.Value = "At time t s, the object is at position (x, y)";
        app.timelocate.Editable = "off";
        app.locate.Enable ="off";

        % RTSS Panel
        app.RTSSPanel.Enable = "off";
        app.ReplayButton.Enable = "off";
        app.FastForwardButton.Enable = "off";
        app.FastRewindButton.Enable= "off";
        app.ReplayButton.Enable= "off";
        app.Skipback.Enable= "off";
        app.PlayPauseButton.Enable = "off";
        app.Skipforward.Enable= "off";
        app.InputPlaybackSpeed.Enable= "off";

        % Graphs Tab
        % Graphs
        plot(app.DisplacementTimeGraph,0,0)
        plot(app.VelocityTimeGraph,0,0)
        plot(app.AccelerationTimeGraph,0,0)
    % Values under graph
    app.MaxAcceleration.Value = 0;
    app.MinAcceleration.Value = 0;
    app.AvgAcceleration.Value = 0;
    app.Jerk.Value = 0;
    app.MaxVelocity.Value = 0;
    app.MinVelocity.Value = 0;
    app.AvgVelocity.Value = 0;
    app.MinDisplacement.Value = 0;
    app.MaxDisplacement.Value = 0;
end

% Button pushed function: locate
function locateButtonPushed(app, event)
    [~, ~, ~, ~, xf, g] = inputs(app);
    [V, alpha, tofl, ~, ~, ~, ~, ~] = calculate(app, xf, g);
    t_check = app.timelocate.Value; % Time field
    if t_check >= tofl % If input time greater than total time
        x_check = V * cos(alpha) * tofl;
        app.locationdisp.Value=
        ['At time ' num2str(round(t_check, 2)) ' s, the object is at position (' num2str(round(x_check, 2)) ', 3)'];
         % Displays time, (x,y) coordinates in locationdisp if time input is greater or equal to time of flight
    else
        x_check = V * cos(alpha) * t_check; % X - Location
        y_check = V * sin(alpha) * t_check - 0.5 * g * t_check^2; % Y - Location
        app.locationdisp.Value=
        ['At time ' num2str(round(t_check, 2)) ' s, the object is at position (' num2str(round(x_check, 2)) ', ' num2str(round(y_check, 2)) ')'];
         % Displays time, (x,y) coordinates in locationdisp
    end
end
```

```
% Button pushed function: ReplayButton
function ReplayButtonPushed(app, event)
    if app.isAni
        pauseRTSS(app);
        app.cTime = 0;
        resumeRTSS(app);
    else
        startRTSS(app);
    end
end

% Button pushed function: FastRewindButton
function FastRewindButtonPushed(app, event)
    tweakRTSS(app, -1.5);
end

% Button pushed function: Skipback
function SkipbackPushed(app, event)
    if app.isAni
        pauseRTSS(app);
        app.cTime = 0;
        resumeRTSS(app);
    else
        startRTSS(app);
    end
end

% Value changed function: PlayPauseButton
function PlayPauseButtonValueChanged(app, event)
    if app.isAni
        if app.isPlay
            pauseRTSS(app);
            app.timerr.Running
        else
            resumeRTSS(app);
            app.timerr.Running
        end
    end
end
```

```
% Button pushed function: Skipforward
function SkipforwardPushed(app, event)
    if app.isAni
        pauseRTSS(app);
        app.cTime = app.TMAX;
        resumeRTSS(app);
    else
        startRTSS(app);
    end
end

% Button pushed function: FastForwardButton
function FastForwardButtonPushed(app, event)
    tweakRTSS(app, 1.5);
end

% Value changed function: InputPlaybackSpeed
function InputPlaybackSpeedValueChanged(app, event)
    app.pSpeed = app.InputPlaybackSpeed.Value;
end

% Button pushed function: SaveButton
function SaveButtonPushed(app, event)
    s = inputdlg("Enter the name of the Data","Save");
    if isempty(s)
        uialert(app.ProjectileSimulatorUIFigure, "Save Cancelled", "Failure", "Icon", "warning");
    else
        [~, H, ~, ~, xf, g] = inputs(app);
        [V, alpha, tofl, ~, ~, ~, ~, ~] = calculate(app, xf, g);
        data = [s, xf, H, round(g, 2), round(V, 2), round(alpha, 2), round(tofl, 2)];
        sh = app.SimulationH.Data;
        app.SimulationH.Data = [sh;data];
    end
end
```

```
% Button pushed function: ImportButton
function ImportButtonPushed(app, event)
    importf(app);
    plothis(app);
end

% Button down function: HistoryTab
function HistoryTabButtonDown(app, event)
    db = app.SimulationH.Data;
    h = height(db);
    app.DataDropDown.Items = {};
    app.DataDropDown.ItemsData ={};
    for i = 1:h
        name = db(i,1);
        ddrop = app.DataDropDown.Items;
        app.DataDropDown.Items = [ddrop name];
    end
end

% Button down function: simulator
function simulatorButtonDown(app, event)
    % Need to clear the drop down menu
    app.DataDropDown.Items = {};
    app.DataDropDown.ItemsData ={};
end

% Button down function: GraphsTab
function GraphsTabButtonDown(app, event)
    app.DataDropDown.Items = {};
    app.DataDropDown.ItemsData = {};
end

% Button pushed function: ClearAllButton
function ClearAllButtonPushed(app, event)
    hold(app.Historygraph,"off")
    app.hisname = [];
    plot(app.Historygraph,0,0)
    legend(app.Historygraph, 'off')
    tabs = app.new_tab;
    for i  = tabs
        delete(i)
    end
end
```

```
% Button pushed function: ImportButton
function ImportButtonPushed(app, event)
    importf(app); % Import the data
    plothis(app);   % Plot the data
end

% Button down function: HistoryTab
function HistoryTabButtonDown(app, event)
    db = app.SimulationH.Data; % Get the data from the table
    h = height(db); % Get the height of the data
    app.DataDropDown.Items = {}; % Clear the drop down menu
    app.DataDropDown.ItemsData ={};
    for i = 1:h % Loop through the data
        name = db(i,1); % Get the name
        ddrop = app.DataDropDown.Items; % Get the items
        app.DataDropDown.Items = [ddrop name];  % Add the name to the drop down menu
    end
end

% Button down function: simulator
function simulatorButtonDown(app, event)
    % Need to clear the drop down menu
    app.DataDropDown.Items = {};
    app.DataDropDown.ItemsData ={};
end

% Button down function: GraphsTab
function GraphsTabButtonDown(app, event)
    app.DataDropDown.Items = {};
    app.DataDropDown.ItemsData = {};
```

```
        end

        % Button pushed function: ClearAllButton
        function ClearAllButtonPushed(app, event)
            hold(app.Historygraph,"off") % Hold the graph off
            app.hisname = []; % Clear the history name
            plot(app.Historygraph,0,0) % Plot the graph
            legend(app.Historygraph, 'off')     % Turn off the legend
            tabs = app.new_tab; % Get the new tab
            for i  = tabs
                delete(i) % Delete the tab
            end
        end
```

```
    % Close request function: ProjectileSimulatorUIFigure
    function ProjectileSimulatorUIFigureCloseRequest(app, event)
        if isfield(app, 'timer') && isa(app.timerr, 'timer') && isvalid(app.timerr) % Check if the timer is valid
            if strcmp(app.timerr.Running, 'on') % Check if the timer is running
                stop(app.timerr); % Stop the timer
            end

            delete(app.timer); % Delete the timer
        end

        delete(app);    % Delete the app
    end
end

% Component initialization
methods (Access = private)

    % Create UIFigure and components
    function createComponents(app)

        % Get the file path for locating images
        pathToMLAPP = fileparts(mfilename('fullpath'));

        % Create ProjectileSimulatorUIFigure and hide until all components are created
        app.ProjectileSimulatorUIFigure = uifigure('Visible', 'off');
        app.ProjectileSimulatorUIFigure.Color = [0.9412 0.9412 0.9412];
        app.ProjectileSimulatorUIFigure.Position = [100 100 957 609];
        app.ProjectileSimulatorUIFigure.Name = 'Projectile Simulator';
        app.ProjectileSimulatorUIFigure.Icon = fullfile(pathToMLAPP, 'catapult-w.png');
        app.ProjectileSimulatorUIFigure.CloseRequestFcn = createCallbackFcn(app, @ProjectileSimulatorUIFigureCloseRequest, true);

        % Create TabGroup
        app.TabGroup = uitabgroup(app.ProjectileSimulatorUIFigure);
        app.TabGroup.Position = [1 1 957 550];

        % Create simulator
        app.simulator = uitab(app.TabGroup);
        app.simulator.AutoResizeChildren = 'off';
        app.simulator.Title = 'Simulation';
        app.simulator.BackgroundColor = [0.9412 0.9412 0.9412];
        app.simulator.ButtonDownFcn = createCallbackFcn(app, @simulatorButtonDown, true);
```

```
        % Create simlayout
        app.simlayout = uigridlayout(app.simulator);
        app.simlayout.ColumnWidth = {'1.5x', '1.5x', '1x', '1x', '1x'};
        app.simlayout.RowHeight = {'1x', '1x', '1x', '1x', '1x', '1x', '1x', '1x'};
        app.simlayout.BackgroundColor = [0.9412 0.9412 0.9412];

        % Create MainGraph
        app.MainGraph = uiaxes(app.simlayout);
        title(app.MainGraph, 'Projectile Motion')
        xlabel(app.MainGraph, 'Horizontal Distance (m)')
        ylabel(app.MainGraph, 'Vertical Distance (m)')
        app.MainGraph.Toolbar.Visible = 'off';
        app.MainGraph.AmbientLightColor = [0 0 0];
        app.MainGraph.XLimitMethod = 'tight';
        app.MainGraph.YLimitMethod = 'tight';
        app.MainGraph.ZLimitMethod = 'tight';
        app.MainGraph.XColor = [0 0 0];
        app.MainGraph.YColor = [0 0 0];
        app.MainGraph.BoxStyle = 'full';
        app.MainGraph.Color = 'none';
        app.MainGraph.XGrid = 'on';
        app.MainGraph.XMinorGrid = 'on';
        app.MainGraph.YGrid = 'on';
        app.MainGraph.YMinorGrid = 'on';
        app.MainGraph.ColorOrder = [0.850980392156863 0.325490196078431 0.0980392156862745
```

```
;0.850980392156863 0.325490196078431 0.0980392156862745
;0.752941176470588 0.36078431372549 0.984313725490196;0.286274509803922 0.858823529411765 0.250980392156863
;0.423529411764706 0.956862745098039 1;0.949019607843137 0.4 0.768627450980392];
app.MainGraph.FontSize = 10;
app.MainGraph.Layout.Row = [1 5];
app.MainGraph.Layout.Column = [3 5];
app.MainGraph.Interruptible = 'off';

% Create RTSSPanel
app.RTSSPanel = uipanel(app.simlayout);
app.RTSSPanel.AutoResizeChildren = 'off';
app.RTSSPanel.Enable = 'off';
app.RTSSPanel.Title = 'Real Time Simulator System ( RTSS )';
app.RTSSPanel.BackgroundColor = [0.9412 0.9412 0.9412];
app.RTSSPanel.Layout.Row = [6 8];
app.RTSSPanel.Layout.Column = [3 5];
app.RTSSPanel.FontWeight = 'bold';
```

```
% Create InputPlaybackSpeed
app.InputPlaybackSpeed = uispinner(app.RTSSPanel);
app.InputPlaybackSpeed.Step = 0.25;
app.InputPlaybackSpeed.Limits = [0.25 2];
app.InputPlaybackSpeed.ValueChangedFcn = createCallbackFcn(app, @InputPlaybackSpeedValueChanged, true);
app.InputPlaybackSpeed.HorizontalAlignment = 'center';
app.InputPlaybackSpeed.BackgroundColor = [0.9412 0.9412 0.9412];
app.InputPlaybackSpeed.Enable = 'off';
app.InputPlaybackSpeed.Tooltip = {'Set playback speed'};
app.InputPlaybackSpeed.Position = [394 126 58 22];
app.InputPlaybackSpeed.Value = 1;

% Create FastForwardButton
app.FastForwardButton = uibutton(app.RTSSPanel, 'push');
app.FastForwardButton.ButtonPushedFcn = createCallbackFcn(app, @FastForwardButtonPushed, true);
app.FastForwardButton.Icon = fullfile(pathToMLAPP, 'fast-forward.png');
app.FastForwardButton.BackgroundColor = [0.9412 0.9412 0.9412];
app.FastForwardButton.Enable = 'off';
app.FastForwardButton.Position = [270 127 47 23];
app.FastForwardButton.Text = '';

% Create PlayPauseButton
app.PlayPauseButton = uibutton(app.RTSSPanel, 'state');
app.PlayPauseButton.ValueChangedFcn = createCallbackFcn(app, @PlayPauseButtonValueChanged, true);
app.PlayPauseButton.Enable = 'off';
app.PlayPauseButton.Icon = fullfile(pathToMLAPP, 'Play.png');
app.PlayPauseButton.IconAlignment = 'center';
app.PlayPauseButton.Text = '';
app.PlayPauseButton.BackgroundColor = [0.9412 0.9412 0.9412];
app.PlayPauseButton.Position = [233 127 25 23];

% Create Skipback
app.Skipback = uibutton(app.RTSSPanel, 'push');
app.Skipback.ButtonPushedFcn = createCallbackFcn(app, @SkipbackPushed, true);
app.Skipback.Icon = fullfile(pathToMLAPP, 'Rewind.png');
app.Skipback.BackgroundColor = [0.9412 0.9412 0.9412];
app.Skipback.Enable = 'off';
app.Skipback.Position = [122 127 47 23];
app.Skipback.Text = '';
```

```
% Create ReplayButton
app.ReplayButton = uibutton(app.RTSSPanel, 'push');
app.ReplayButton.ButtonPushedFcn = createCallbackFcn(app, @ReplayButtonPushed, true);
app.ReplayButton.BackgroundColor = [0.9412 0.9412 0.9412];
app.ReplayButton.Enable = 'off';
app.ReplayButton.Position = [15 127 85 23];
app.ReplayButton.Text = 'Replay';

% Create DistanceRemainingValue
app.DistanceRemainingValue = uilabel(app.RTSSPanel);
app.DistanceRemainingValue.HorizontalAlignment = 'right';
app.DistanceRemainingValue.Position = [132 45 53 22];
app.DistanceRemainingValue.Text = '0.00 m';

% Create DistanceRemainingLabel
app.DistanceRemainingLabel = uilabel(app.RTSSPanel);
app.DistanceRemainingLabel.Position = [18 46 116 22];
app.DistanceRemainingLabel.Text = 'Distance Remaining:';
```

```
% Create DistanceCoveredValue
app.DistanceCoveredValue = uilabel(app.RTSSPanel);
app.DistanceCoveredValue.HorizontalAlignment = 'right';
app.DistanceCoveredValue.Position = [122 67 63 22];
app.DistanceCoveredValue.Text = '0.00 m';

% Create DistanceCoveredLabel
app.DistanceCoveredLabel = uilabel(app.RTSSPanel);
app.DistanceCoveredLabel.Position = [18 67 104 22];
app.DistanceCoveredLabel.Text = 'Distance Covered:';

% Create CurrentHeightValue
app.CurrentHeightValue = uilabel(app.RTSSPanel);
app.CurrentHeightValue.HorizontalAlignment = 'right';
app.CurrentHeightValue.Position = [122 88 63 22];
app.CurrentHeightValue.Text = '0.00 m';

% Create CurrentHeightLabel
app.CurrentHeightLabel = uilabel(app.RTSSPanel);
app.CurrentHeightLabel.Position = [18 88 86 22];
app.CurrentHeightLabel.Text = 'Current Height:';

% Create TimeRemainingValue
app.TimeRemainingValue = uilabel(app.RTSSPanel);
app.TimeRemainingValue.HorizontalAlignment = 'right';
app.TimeRemainingValue.Position = [380 67 44 22];
app.TimeRemainingValue.Text = '0.00 s';

% Create TimeRemainingLabel
app.TimeRemainingLabel = uilabel(app.RTSSPanel);
app.TimeRemainingLabel.Position = [285 67 95 22];
app.TimeRemainingLabel.Text = 'Time Remaining:';

% Create TimeElapsedValue
app.TimeElapsedValue = uilabel(app.RTSSPanel);
app.TimeElapsedValue.HorizontalAlignment = 'right';
app.TimeElapsedValue.Position = [380 88 44 22];
app.TimeElapsedValue.Text = '0.00 s';

% Create TimeElapsedLabel
app.TimeElapsedLabel = uilabel(app.RTSSPanel);
app.TimeElapsedLabel.Position = [285 88 85 22];
app.TimeElapsedLabel.Text = 'Time Elapsed: ';

% Create Skipforward
app.Skipforward = uibutton(app.RTSSPanel, 'push');
app.Skipforward.ButtonPushedFcn = createCallbackFcn(app, @SkipforwardPushed, true);
app.Skipforward.Icon = fullfile(pathToMLAPP, 'Forward.png');
app.Skipforward.IconAlignment = 'center';
app.Skipforward.BackgroundColor = [0.9412 0.9412 0.9412];
app.Skipforward.Enable = 'off';
app.Skipforward.Position = [323 126 47 23];
app.Skipforward.Text = '';

% Create LocationTrackerPanel
app.LocationTrackerPanel = uipanel(app.simlayout);
app.LocationTrackerPanel.AutoResizeChildren = 'off';
app.LocationTrackerPanel.Title = 'Location Tracker';
app.LocationTrackerPanel.BackgroundColor = [0.9412 0.9412 0.9412];
app.LocationTrackerPanel.Layout.Row = [7 8];
app.LocationTrackerPanel.Layout.Column = [1 2];
app.LocationTrackerPanel.FontWeight = 'bold';

% Create CheckobjectlocationatLabel
app.CheckobjectlocationatLabel = uilabel(app.LocationTrackerPanel);
app.CheckobjectlocationatLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.CheckobjectlocationatLabel.Position = [12 68 186 21];
app.CheckobjectlocationatLabel.Text = 'Check object location at';
```

```
% Create locationdisp
app.locationdisp = uitextarea(app.LocationTrackerPanel);
app.locationdisp.Editable = 'off';
app.locationdisp.HorizontalAlignment = 'center';
app.locationdisp.WordWrap = 'off';
app.locationdisp.FontSize = 14;
app.locationdisp.FontWeight = 'bold';
app.locationdisp.BackgroundColor = [0.9412 0.9412 0.9412];
app.locationdisp.Placeholder = 'At time t s, the object is at position (x, y)';
app.locationdisp.Position = [12 13 425 39];

% Create secLabel
app.secLabel = uilabel(app.LocationTrackerPanel);
app.secLabel.HorizontalAlignment = 'center';
app.secLabel.FontWeight = 'bold';
app.secLabel.Enable = 'off';
app.secLabel.Position = [280 68 25 22];
app.secLabel.Text = 'sec';

% Create locate
app.locate = uibutton(app.LocationTrackerPanel, 'push');
app.locate.ButtonPushedFcn = createCallbackFcn(app, @locateButtonPushed, true);
app.locate.BackgroundColor = [0.9412 0.9412 0.9412];
app.locate.FontSize = 14;
app.locate.FontWeight = 'bold';
app.locate.Enable = 'off';
app.locate.Position = [338 67 100 23];
app.locate.Text = 'Locate';

% Create timelocate
app.timelocate = uieditfield(app.LocationTrackerPanel, 'numeric');
app.timelocate.Editable = 'off';
app.timelocate.HorizontalAlignment = 'center';
app.timelocate.BackgroundColor = [0.9412 0.9412 0.9412];
app.timelocate.Enable = 'off';
app.timelocate.Position = [157 69 108 20];
```

```
% Create OutputPanel
app.OutputPanel = uipanel(app.simlayout);
app.OutputPanel.AutoResizeChildren = 'off';
app.OutputPanel.Title = 'Output';
app.OutputPanel.BackgroundColor = [0.9412 0.9412 0.9412];
app.OutputPanel.Layout.Row = [4 6];
app.OutputPanel.Layout.Column = [1 2];
app.OutputPanel.FontWeight = 'bold';

% Create mLabel_4
app.mLabel_4 = uilabel(app.OutputPanel);
app.mLabel_4.HorizontalAlignment = 'center';
app.mLabel_4.FontWeight = 'bold';
app.mLabel_4.Position = [412 94 25 22];
app.mLabel_4.Text = 'm';

% Create radsLabel
app.radsLabel = uilabel(app.OutputPanel);
app.radsLabel.HorizontalAlignment = 'center';
app.radsLabel.FontWeight = 'bold';
app.radsLabel.Position = [410 124 30 22];
app.radsLabel.Text = 'rads';

% Create mLabel_2
app.mLabel_2 = uilabel(app.OutputPanel);
app.mLabel_2.HorizontalAlignment = 'center';
app.mLabel_2.FontWeight = 'bold';
app.mLabel_2.Position = [206 64 25 22];
app.mLabel_2.Text = 'm';

% Create secLabel_2
app.secLabel_2 = uilabel(app.OutputPanel);
app.secLabel_2.HorizontalAlignment = 'center';
app.secLabel_2.FontWeight = 'bold';
app.secLabel_2.Position = [206 94 25 22];
app.secLabel_2.Text = 'sec';
```

```
% Create msLabel
app.msLabel = uilabel(app.OutputPanel);
app.msLabel.HorizontalAlignment = 'center';
app.msLabel.FontWeight = 'bold';
app.msLabel.Position = [205 124 27 22];
app.msLabel.Text = 'm/s';

% Create RangeLabel_2
app.RangeLabel_2 = uilabel(app.OutputPanel);
app.RangeLabel_2.BackgroundColor = [0.9412 0.9412 0.9412];
app.RangeLabel_2.Position = [268 94 43 22];
app.RangeLabel_2.Text = 'Range:';

% Create Range
app.Range = uieditfield(app.OutputPanel, 'text');
app.Range.Editable = 'off';
app.Range.HorizontalAlignment = 'center';
app.Range.BackgroundColor = [0.9412 0.9412 0.9412];
app.Range.Placeholder = '0';
app.Range.Position = [348 94 55 22];

% Create AngleofLaunchLabel
app.AngleofLaunchLabel = uilabel(app.OutputPanel);
app.AngleofLaunchLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.AngleofLaunchLabel.Position = [268 124 88 22];
app.AngleofLaunchLabel.Text = 'Angle:';

% Create angle
app.angle = uieditfield(app.OutputPanel, 'numeric');
app.angle.Editable = 'off';
app.angle.HorizontalAlignment = 'center';
app.angle.BackgroundColor = [0.9412 0.9412 0.9412];
app.angle.Position = [348 125 56 22];

% Create PeakLocationEditFieldLabel
app.PeakLocationEditFieldLabel = uilabel(app.OutputPanel);
app.PeakLocationEditFieldLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.PeakLocationEditFieldLabel.Position = [12 32 85 22];
app.PeakLocationEditFieldLabel.Text = 'Peak Location:';

% Create peak
app.peak = uieditfield(app.OutputPanel, 'text');
app.peak.Editable = 'off';
app.peak.HorizontalAlignment = 'center';
app.peak.BackgroundColor = [0.9412 0.9412 0.9412];
app.peak.Placeholder = '(x,y)';
app.peak.Position = [96 32 100 22];

% Create MaxHieghtLabel
app.MaxHieghtLabel = uilabel(app.OutputPanel);
app.MaxHieghtLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxHieghtLabel.Position = [12 64 70 22];
app.MaxHieghtLabel.Text = 'Max Height:';

% Create MH
app.MH = uieditfield(app.OutputPanel, 'numeric');
app.MH.Editable = 'off';
app.MH.HorizontalAlignment = 'center';
app.MH.BackgroundColor = [0.9412 0.9412 0.9412];
app.MH.Position = [96 64 101 22];

% Create TimeofflightLabel
app.TimeofflightLabel = uilabel(app.OutputPanel);
app.TimeofflightLabel.Position = [12 94 78 22];
app.TimeofflightLabel.Text = 'Time of flight:';

% Create timeofflight
app.timeofflight = uieditfield(app.OutputPanel, 'numeric');
app.timeofflight.Editable = 'off';
app.timeofflight.HorizontalAlignment = 'center';
app.timeofflight.BackgroundColor = [0.9412 0.9412 0.9412];
app.timeofflight.Position = [96 94 100 22];
```

```
% Create VelocityLabel
app.VelocityLabel = uilabel(app.OutputPanel);
app.VelocityLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.VelocityLabel.Position = [12 124 53 22];
app.VelocityLabel.Text = 'Velocity: ';

% Create velocity
app.velocity = uieditfield(app.OutputPanel, 'numeric');
app.velocity.Editable = 'off';
app.velocity.HorizontalAlignment = 'center';
app.velocity.BackgroundColor = [0.9412 0.9412 0.9412];
app.velocity.Position = [96 124 100 22];

% Create mLabel_5
app.mLabel_5 = uilabel(app.OutputPanel);
app.mLabel_5.HorizontalAlignment = 'center';
app.mLabel_5.FontWeight = 'bold';
app.mLabel_5.Position = [206 32 25 22];
app.mLabel_5.Text = 'm';

% Create InputPanel
app.InputPanel = uipanel(app.simlayout);
app.InputPanel.AutoResizeChildren = 'off';
app.InputPanel.Title = 'Input  ';
app.InputPanel.BackgroundColor = [0.9412 0.9412 0.9412];
app.InputPanel.Layout.Row = [1 3];
app.InputPanel.Layout.Column = [1 2];
app.InputPanel.FontWeight = 'bold';

% Create AccelerationLabel
app.AccelerationLabel = uilabel(app.InputPanel);
app.AccelerationLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.AccelerationLabel.Position = [12 65 75 22];
app.AccelerationLabel.Text = 'Acceleration:';

% Create IntialHeightLabel
app.IntialHeightLabel = uilabel(app.InputPanel);
app.IntialHeightLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.IntialHeightLabel.Position = [12 96 108 22];
app.IntialHeightLabel.Text = 'Height of obstacle: ';
```

```
% Create RangeLabel
app.RangeLabel = uilabel(app.InputPanel);
app.RangeLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.RangeLabel.Position = [12 129 134 22];
app.RangeLabel.Text = 'Distance from obstacle: ';

% Create mLabel
app.mLabel = uilabel(app.InputPanel);
app.mLabel.HorizontalAlignment = 'center';
app.mLabel.FontWeight = 'bold';
app.mLabel.Position = [219 96 25 22];
app.mLabel.Text = 'm';

% Create mLabel_3
app.mLabel_3 = uilabel(app.InputPanel);
app.mLabel_3.HorizontalAlignment = 'center';
app.mLabel_3.FontWeight = 'bold';
app.mLabel_3.Position = [219 129 25 22];
app.mLabel_3.Text = 'm';

% Create RTSSSwitchLabel
app.RTSSSwitchLabel = uilabel(app.InputPanel);
app.RTSSSwitchLabel.HorizontalAlignment = 'center';
app.RTSSSwitchLabel.Tooltip = {'Real Time Simulator System'};
app.RTSSSwitchLabel.Position = [374 80 35 22];
app.RTSSSwitchLabel.Text = 'RTSS';

% Create resetall
app.resetall = uibutton(app.InputPanel, 'push');
app.resetall.ButtonPushedFcn = createCallbackFcn(app, @resetallButtonPushed, true);
app.resetall.BackgroundColor = [0.9412 0.9412 0.9412];
app.resetall.FontSize = 14;
```

```
app.resetall.FontWeight = 'bold';
app.resetall.Position = [333 10 100 41];
app.resetall.Text = 'Reset All';

% Create SaveButton
app.SaveButton = uibutton(app.InputPanel, 'push');
app.SaveButton.ButtonPushedFcn = createCallbackFcn(app, @SaveButtonPushed, true);
app.SaveButton.WordWrap = 'on';
app.SaveButton.BackgroundColor = [0.9412 0.9412 0.9412];
app.SaveButton.FontSize = 14;
app.SaveButton.FontWeight = 'bold';
app.SaveButton.Enable = 'off';
app.SaveButton.Position = [179 9 100 42];
app.SaveButton.Text = '💾 Save';

% Create simulate
app.simulate = uibutton(app.InputPanel, 'push');
app.simulate.ButtonPushedFcn = createCallbackFcn(app, @simulateButtonPushed, true);
app.simulate.BackgroundColor = [0.9412 0.9412 0.9412];
app.simulate.FontSize = 14;
app.simulate.FontWeight = 'bold';
app.simulate.Position = [25 10 100 41];
app.simulate.Text = 'Simulate';

% Create RTSSSwitch
app.RTSSSwitch = uiswitch(app.InputPanel, 'slider');
app.RTSSSwitch.Position = [370 117 41 18];

% Create acceleration
app.acceleration = uidropdown(app.InputPanel);
app.acceleration.Items = {'Earth', 'Moon', 'Mars'};
app.acceleration.BackgroundColor = [0.9412 0.9412 0.9412];
app.acceleration.Position = [143 65 73 22];
app.acceleration.Value = 'Earth';

% Create Height
app.Height = uieditfield(app.InputPanel, 'numeric');
app.Height.HorizontalAlignment = 'center';
app.Height.BackgroundColor = [0.9412 0.9412 0.9412];
app.Height.Position = [143 96 73 22];
```

```
% Create Distance
app.Distance = uieditfield(app.InputPanel, 'numeric');
app.Distance.HorizontalAlignment = 'center';
app.Distance.BackgroundColor = [0.9412 0.9412 0.9412];
app.Distance.Position = [143 129 73 22];

% Create GraphsTab
app.GraphsTab = uitab(app.TabGroup);
app.GraphsTab.AutoResizeChildren = 'off';
app.GraphsTab.Title = 'Graphs';
app.GraphsTab.BackgroundColor = [0.9412 0.9412 0.9412];
app.GraphsTab.ButtonDownFcn = createCallbackFcn(app, @GraphsTabButtonDown, true);

% Create GridLayout
app.GridLayout = uigridlayout(app.GraphsTab);
app.GridLayout.ColumnWidth = {'1x', '1x', '1x', '1x', '1x', '1x'};
app.GridLayout.RowHeight = {'1x', '1x', '1x', '1x', '1x', '1x', '1x', '1x', '1x'};
app.GridLayout.BackgroundColor = [0.9412 0.9412 0.9412];

% Create AccelerationTimeGraph
app.AccelerationTimeGraph = uiaxes(app.GridLayout);
title(app.AccelerationTimeGraph, 'Acceleration - Time')
xlabel(app.AccelerationTimeGraph, 'Time (s)')
ylabel(app.AccelerationTimeGraph, 'Acceleration (m/s²)')
zlabel(app.AccelerationTimeGraph, 'Z')
app.AccelerationTimeGraph.XLimitMethod = 'tight';
app.AccelerationTimeGraph.YLimitMethod = 'tight';
app.AccelerationTimeGraph.ZLimitMethod = 'tight';
app.AccelerationTimeGraph.MinorGridLineStyle = '-';
app.AccelerationTimeGraph.Color = 'none';
app.AccelerationTimeGraph.XGrid = 'on';
app.AccelerationTimeGraph.XMinorGrid = 'on';
app.AccelerationTimeGraph.YGrid = 'on';
app.AccelerationTimeGraph.YMinorGrid = 'on';
```

```
app.AccelerationTimeGraph.ColorOrder = [1 0.0705882352941176 0.650980392156863;
0.96078431372549 0.466666666666667 0.16078431372549;1 0.909803921568627 0.392156862745098;
0.752941176470588 0.36078431372549 0.984313725490196;0.286274509803922 0.858823529411765 0.250980392156863;
0.423529411764706 0.956862745098039 1;1 0.0705882352941176 0.650980392156863];
app.AccelerationTimeGraph.Layout.Row = [1 6];
app.AccelerationTimeGraph.Layout.Column = [1 2];
colormap(app.AccelerationTimeGraph, 'copper')

% Create VelocityTimeGraph
app.VelocityTimeGraph = uiaxes(app.GridLayout);
title(app.VelocityTimeGraph, 'Velocity - Time')
xlabel(app.VelocityTimeGraph, 'Time (s)')
ylabel(app.VelocityTimeGraph, 'Velocity (m/s)')
zlabel(app.VelocityTimeGraph, 'Z')
app.VelocityTimeGraph.Toolbar.Visible = 'off';
app.VelocityTimeGraph.XLimitMethod = 'tight';
app.VelocityTimeGraph.YLimitMethod = 'tight';
app.VelocityTimeGraph.ZLimitMethod = 'tight';
app.VelocityTimeGraph.MinorGridLineStyle = '-';
app.VelocityTimeGraph.Color = 'none';
app.VelocityTimeGraph.XGrid = 'on';
app.VelocityTimeGraph.XMinorGrid = 'on';
app.VelocityTimeGraph.YGrid = 'on';
app.VelocityTimeGraph.YMinorGrid = 'on';
app.VelocityTimeGraph.ColorOrder = [0.63921568627451 0.0784313725490196 0.180392156862745
;0.96078431372549 0.466666666666667 0.16078431372549;1 0.909803921568627 0.388235294117647
;0.749019607843137 0.36078431372549 0.980392156862745;0.290196078431373 0.858823529411765 0.250980392156863
;0.423529411764706 0.956862745098039 1;0.949019607843137 0.403921568627451 0.772549019607843
;0.63921568627451 0.0784313725490196 0.180392156862745];
app.VelocityTimeGraph.SortMethod = 'depth';
app.VelocityTimeGraph.Layout.Row = [1 6];
app.VelocityTimeGraph.Layout.Column = [3 4];

% Create DisplacementTimeGraph
app.DisplacementTimeGraph = uiaxes(app.GridLayout);
title(app.DisplacementTimeGraph, 'Displacement - Time')
xlabel(app.DisplacementTimeGraph, 'Time (s)')
ylabel(app.DisplacementTimeGraph, 'Displacement (m)')
zlabel(app.DisplacementTimeGraph, 'Z')
app.DisplacementTimeGraph.Toolbar.Visible = 'off';
app.DisplacementTimeGraph.XLimitMethod = 'tight';
app.DisplacementTimeGraph.YLimitMethod = 'tight';
app.DisplacementTimeGraph.ZLimitMethod = 'tight';
app.DisplacementTimeGraph.ZLimitMethod = 'tight';
app.DisplacementTimeGraph.MinorGridLineStyle = '-';
app.DisplacementTimeGraph.Color = 'none';
app.DisplacementTimeGraph.XGrid = 'on';
app.DisplacementTimeGraph.XMinorGrid = 'on';
app.DisplacementTimeGraph.YGrid = 'on';
app.DisplacementTimeGraph.YMinorGrid = 'on';
app.DisplacementTimeGraph.ColorOrder = [0.149 0.549 0.866;0.96 0.466 0.16;1 0.909 0.392
;0.752 0.36 0.984;0.286 0.858 0.25;0.423 0.956 1;0.949 0.403 0.772];
app.DisplacementTimeGraph.Layout.Row = [1 6];
app.DisplacementTimeGraph.Layout.Column = [5 6];

% Create MaxDisplacementLabel
app.MaxDisplacementLabel = uilabel(app.GridLayout);
app.MaxDisplacementLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel.HorizontalAlignment = 'center';
app.MaxDisplacementLabel.Layout.Row = 7;
app.MaxDisplacementLabel.Layout.Column = 5;
app.MaxDisplacementLabel.Text = 'Maximum Displacement';

% Create MaxDisplacement
app.MaxDisplacement = uieditfield(app.GridLayout, 'numeric');
app.MaxDisplacement.Editable = 'off';
app.MaxDisplacement.HorizontalAlignment = 'center';
app.MaxDisplacement.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacement.Layout.Row = 7;
app.MaxDisplacement.Layout.Column = 6;
```

```
% Create MaxDisplacementLabel_2
app.MaxDisplacementLabel_2 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_2.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_2.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_2.Layout.Row = 8;
app.MaxDisplacementLabel_2.Layout.Column = 5;
app.MaxDisplacementLabel_2.Text = 'Minimum Displacement';

% Create MinDisplacement
app.MinDisplacement = uieditfield(app.GridLayout, 'numeric');
app.MinDisplacement.Editable = 'off';
app.MinDisplacement.HorizontalAlignment = 'center';
app.MinDisplacement.BackgroundColor = [0.9412 0.9412 0.9412];
app.MinDisplacement.Layout.Row = 8;
app.MinDisplacement.Layout.Column = 6;
```

```
% Create MaxDisplacementLabel_3
app.MaxDisplacementLabel_3 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_3.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_3.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_3.Layout.Row = 9;
app.MaxDisplacementLabel_3.Layout.Column = 5;
app.MaxDisplacementLabel_3.Text = 'Average Velocity (m/s)';

% Create AvgVelocity
app.AvgVelocity = uieditfield(app.GridLayout, 'numeric');
app.AvgVelocity.Editable = 'off';
app.AvgVelocity.HorizontalAlignment = 'center';
app.AvgVelocity.BackgroundColor = [0.9412 0.9412 0.9412];
app.AvgVelocity.Layout.Row = 9;
app.AvgVelocity.Layout.Column = 6;

% Create MaxDisplacementLabel_4
app.MaxDisplacementLabel_4 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_4.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_4.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_4.Layout.Row = 7;
app.MaxDisplacementLabel_4.Layout.Column = 3;
app.MaxDisplacementLabel_4.Text = 'Maximum Velocity';

% Create MaxVelocity
app.MaxVelocity = uieditfield(app.GridLayout, 'numeric');
app.MaxVelocity.Editable = 'off';
app.MaxVelocity.HorizontalAlignment = 'center';
app.MaxVelocity.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxVelocity.Layout.Row = 7;
app.MaxVelocity.Layout.Column = 4;

% Create MaxDisplacementLabel_5
app.MaxDisplacementLabel_5 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_5.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_5.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_5.Layout.Row = 8;
app.MaxDisplacementLabel_5.Layout.Column = 3;
app.MaxDisplacementLabel_5.Text = 'Minimum Velocity';

% Create MinVelocity
app.MinVelocity = uieditfield(app.GridLayout, 'numeric');
app.MinVelocity.Editable = 'off';
```

```
app.MinVelocity.HorizontalAlignment = 'center';
app.MinVelocity.BackgroundColor = [0.9412 0.9412 0.9412];
app.MinVelocity.Layout.Row = 8;
app.MinVelocity.Layout.Column = 4;

% Create MaxDisplacementLabel_7
app.MaxDisplacementLabel_7 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_7.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_7.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_7.Layout.Row = 9;
app.MaxDisplacementLabel_7.Layout.Column = 3;
app.MaxDisplacementLabel_7.Text = 'Avg Acceleration (m/s²)';

% Create AvgAcceleration
app.AvgAcceleration = uieditfield(app.GridLayout, 'numeric');
app.AvgAcceleration.Editable = 'off';
app.AvgAcceleration.HorizontalAlignment = 'center';
app.AvgAcceleration.BackgroundColor = [0.9412 0.9412 0.9412];
app.AvgAcceleration.Layout.Row = 9;
app.AvgAcceleration.Layout.Column = 4;

% Create MaxDisplacementLabel_8
app.MaxDisplacementLabel_8 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_8.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_8.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_8.Layout.Row = 7;
app.MaxDisplacementLabel_8.Layout.Column = 1;
app.MaxDisplacementLabel_8.Text = 'Maximum Acceleration';

% Create MaxAcceleration
app.MaxAcceleration = uieditfield(app.GridLayout, 'numeric');
app.MaxAcceleration.Editable = 'off';
app.MaxAcceleration.HorizontalAlignment = 'center';
app.MaxAcceleration.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxAcceleration.Layout.Row = 7;
app.MaxAcceleration.Layout.Column = 2;

% Create MaxDisplacementLabel_9
app.MaxDisplacementLabel_9 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_9.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_9.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_9.Layout.Row = 8;
app.MaxDisplacementLabel_9.Layout.Column = 1;
app.MaxDisplacementLabel_9.Text = 'Minimum Acceleration';
```

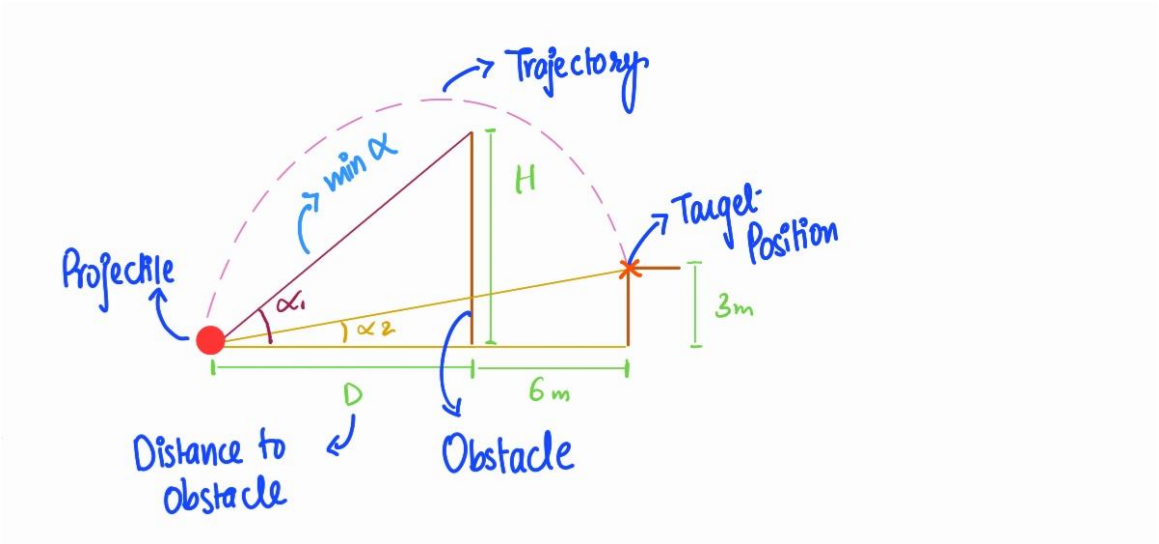
```
% Create MaxDisplacementLabel_9
app.MaxDisplacementLabel_9 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_9.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_9.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_9.Layout.Row = 8;
app.MaxDisplacementLabel_9.Layout.Column = 1;
app.MaxDisplacementLabel_9.Text = 'Minimum Acceleration';

% Create MinAcceleration
app.MinAcceleration = uieditfield(app.GridLayout, 'numeric');
app.MinAcceleration.Editable = 'off';
app.MinAcceleration.HorizontalAlignment = 'center';
app.MinAcceleration.BackgroundColor = [0.9412 0.9412 0.9412];
app.MinAcceleration.Layout.Row = 8;
app.MinAcceleration.Layout.Column = 2;

% Create MaxDisplacementLabel_10
app.MaxDisplacementLabel_10 = uilabel(app.GridLayout);
app.MaxDisplacementLabel_10.BackgroundColor = [0.9412 0.9412 0.9412];
app.MaxDisplacementLabel_10.HorizontalAlignment = 'center';
app.MaxDisplacementLabel_10.Layout.Row = 9;
app.MaxDisplacementLabel_10.Layout.Column = 1;
app.MaxDisplacementLabel_10.Text = 'Jerk (m/s³)';

% Create Jerk
app.Jerk = uieditfield(app.GridLayout, 'numeric');
app.Jerk.Editable = 'off';
app.Jerk.HorizontalAlignment = 'center';
app.Jerk.BackgroundColor = [0.9412 0.9412 0.9412];
app.Jerk.Layout.Row = 9;
app.Jerk.Layout.Column = 2;

% Create HistoryTab
app.HistoryTab = uitab(app.TabGroup);
app.HistoryTab.AutoResizeChildren = 'off';
app.HistoryTab.Title = 'History';
app.HistoryTab.BackgroundColor = [0.9412 0.9412 0.9412];
app.HistoryTab.ButtonDownFcn = createCallbackFcn(app, @HistoryTabButtonDown, true);

% Create GridLayout2
app.GridLayout2 = uigridlayout(app.HistoryTab);
app.GridLayout2.ColumnWidth = {'1x', '1x', '1x', '1x', '1x'};
app.GridLayout2.RowHeight = {'1x', '1x', '1x', '1x', '1x', '1x', '1x', '1x', '1x'};
app.GridLayout2.BackgroundColor = [0.9412 0.9412 0.9412];
```

```
% Create Historygraph
app.Historygraph = uiaxes(app.GridLayout2);
title(app.Historygraph, 'History Graph')
xlabel(app.Historygraph, 'Horizontal Distance (m)')
ylabel(app.Historygraph, 'Vertical Distance (m)')
zlabel(app.Historygraph, 'Z')
app.Historygraph.Color = 'none';
app.Historygraph.XGrid = 'on';
app.Historygraph.YGrid = 'on';
app.Historygraph.ColorOrder = [0.149 0.549 0.866;0.96 0.466 0.16;1 0.909 0.392
;0.752 0.36 0.984;0.286 0.858 0.25;0.423 0.956 1;0.949 0.403 0.772];
app.Historygraph.Layout.Row = [4 9];
app.Historygraph.Layout.Column = [1 3];

% Create SimulationH
app.SimulationH = uitable(app.GridLayout2);
app.SimulationH.BackgroundColor = [1 1 1;0.9412 0.9412 0.9412];
app.SimulationH.ColumnName = {'Name'; 'Distance'; 'Height'; 'Acceleration';
'Velocity'; 'Angle of Launch (rads)'; 'Time of Flight'};
app.SimulationH.RowName = {};
app.SimulationH.Layout.Row = [1 3];
app.SimulationH.Layout.Column = [1 5];

% Create TabGroup2
app.TabGroup2 = uitabgroup(app.GridLayout2);
app.TabGroup2.AutoResizeChildren = 'off';
app.TabGroup2.Layout.Row = [4 9];
app.TabGroup2.Layout.Column = [4 5];

% Create MainTab
app.MainTab = uitab(app.TabGroup2);
app.MainTab.AutoResizeChildren = 'off';
app.MainTab.Title = 'Main';
app.MainTab.BackgroundColor = 'none';

% Create ImportButton
app.ImportButton = uibutton(app.MainTab, 'push');
app.ImportButton.ButtonPushedFcn = createCallbackFcn(app, @ImportButtonPushed, true);
app.ImportButton.BackgroundColor = [0.9412 0.9412 0.9412];
app.ImportButton.Position = [86 138 100 23];
app.ImportButton.Text = 'Import';
```

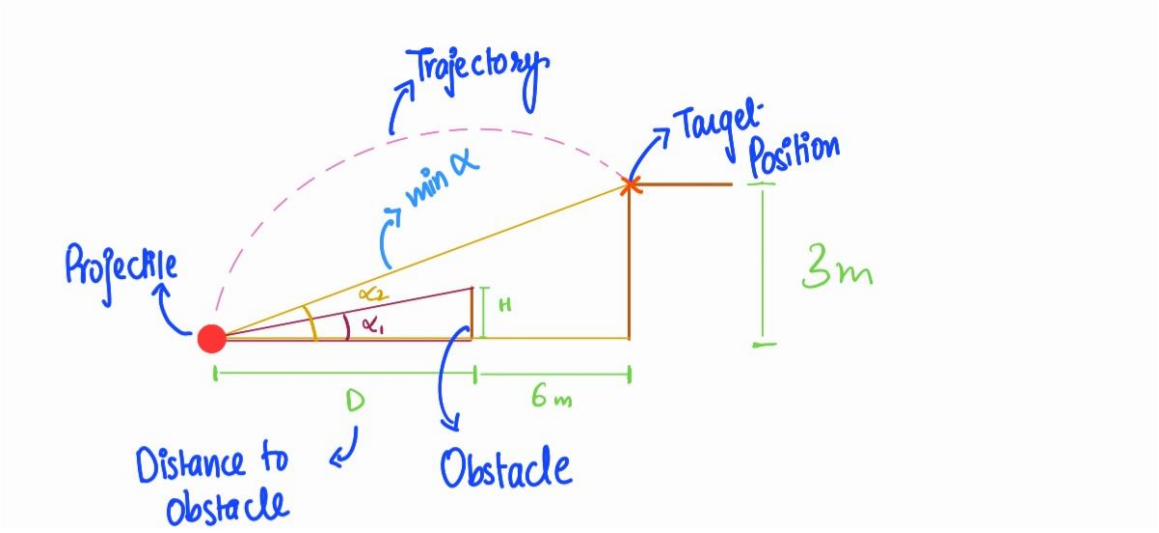
```
% Create DataDropDownLabel
app.DataDropDownLabel = uilabel(app.MainTab);
app.DataDropDownLabel.BackgroundColor = [0.9412 0.9412 0.9412];
app.DataDropDownLabel.Position = [104 181 30 22];
app.DataDropDownLabel.Text = 'Data';

% Create DataDropDown
app.DataDropDown = uidropdown(app.MainTab);
app.DataDropDown.Items = {''};
app.DataDropDown.BackgroundColor = [0.9412 0.9412 0.9412];
app.DataDropDown.Position = [185 181 100 22];
app.DataDropDown.Value = '';

% Create ClearAllButton
app.ClearAllButton = uibutton(app.MainTab, 'push');
app.ClearAllButton.ButtonPushedFcn = createCallbackFcn(app, @ClearAllButtonPushed, true);
app.ClearAllButton.BackgroundColor = [0.9412 0.9412 0.9412];
app.ClearAllButton.Position = [203 138 100 23];
app.ClearAllButton.Text = 'Clear All';

% Create ProjectileSimulatorLabel
app.ProjectileSimulatorLabel = uilabel(app.ProjectileSimulatorUIFigure);
app.ProjectileSimulatorLabel.HorizontalAlignment = 'center';
app.ProjectileSimulatorLabel.FontSize = 36;
app.ProjectileSimulatorLabel.FontWeight = 'bold';
app.ProjectileSimulatorLabel.Position = [307 557 348 48];
app.ProjectileSimulatorLabel.Text = 'Projectile Simulator';
```

```
            % Create Versionlabel
            app.Versionlabel = uilabel(app.ProjectileSimulatorUIFigure);
            app.Versionlabel.HorizontalAlignment = 'center';
            app.Versionlabel.FontWeight = 'bold';
            app.Versionlabel.Position = [923 570 25 22];
            app.Versionlabel.Text = 'V4';

            % Show the figure after all components are created
            app.ProjectileSimulatorUIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = Projectile_Simulator_Exported

            runningApp = getRunningApp(app);

            % Check for running singleton app
            if isempty(runningApp)

                % Create UIFigure and components
                createComponents(app)

                % Register the app with App Designer
                registerApp(app, app.ProjectileSimulatorUIFigure)
            else

                % Focus the running singleton app
                figure(runningApp.ProjectileSimulatorUIFigure)

                app = runningApp;
            end

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

           % Delete UIFigure when app is deleted
           delete(app.ProjectileSimulatorUIFigure)
        end
    end
end
```

Appendix B: Sketches



Trajectory when height of obstacle is greater than target position



Trajectory when height of target position is greater than obstacle

Appendix C: Flowchart