EEE4119F
Milestone 4 - Report

Prepared by:
Luke Baatjes,

Prepared for:
EEE4119F
Department of Electrical Engineering
University of Cape Town

May 22, 2023

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

L. D. Baatjes

May 22, 2023

Luke Baatjes

Date

Contents

List of Figures

Chapter 1

Introduction

The risk of asteroid strikes poses a grave threat to our planet's security and well-being. This project aims to develop a rocket controller to prevent an asteroid from colliding with a city on Earth. The objective is to create an efficient control system that can successfully guide the rocket to intercept the asteroid and avert the potentially catastrophic consequences of such an impact. This project focuses on simulation-based design using MATLAB and Simulink, providing a virtual environment to model and test the rocket's behavior and evaluate different control systems. Through these software tools, we seek to comprehend the intricate dynamics of the rocket-asteroid system, design appropriate control algorithms, and assess their effectiveness in preventing the impending catastrophe.

1.1 Problem Statement

Asteroid impacts pose a severe threat to life and infrastructure, necessitating reliable control systems for interception. This project aims to develop a rocket controller to divert an asteroid approaching a city on Earth. Using simulation-based modeling and control techniques, we aim to create an effective control system that intercepts the asteroid and reduces collision risks. The key challenge is to develop a precise control algorithm that accurately maneuvers the rocket and predicts the dynamics of the rocket-asteroid combination. Meeting requirements for asteroid interception and human safety is crucial. Addressing these challenges will advance aerospace engineering and inspire innovative control systems for mitigating the risk of asteroid impacts.

1.2 Limitations

Recognizing project constraints is crucial. Simulations and control algorithms used in a simulated environment may not fully capture real-world complexities and uncertainties. Model quality and underlying assumptions significantly influence result accuracy.

- The project assumes simplified mathematical models can capture rocket motion and asteroid behavior. Real-world dynamics may involve additional complexities, such as fuel consumption, varying mass, and advanced aerodynamic effects.
- This project concentrates on a certain set of criteria and situations due to time constraints, therefore it might not cover all potential variations that might occur during an asteroid interception mission.

Chapter 2

Modelling

Accurately describing the rocket's dynamics is vital for designing a successful control system. We outline our modeling strategy to analyze the rocket's accelerations and dynamics, deriving the manipulator equation in MATLAB. This equation captures the dynamic interaction between the rocket's parameters, control inputs, and external effects. In asteroid modeling, we calculate the drag force coefficient, C, using simulation data, a crucial input for subsequent control design. The simulation results provide valuable insights into the aerodynamic effects it experiences. The Matlab script to develop the dynamics or the rocket and asteroid can be found in appendix A

2.1 Rocket Modelling

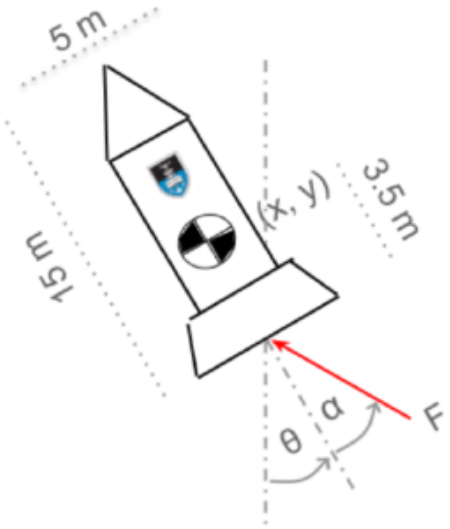


Figure 2.1: Diagram of Rocket Parameters

The rocket's parameters can be found in the milestone 1 brief [1] and will not be reiterated here because of page number limitations.

The rocket's Inertia was calculated using the parrallel axis theorem:

$$I = \frac{1}{12} * m * (h^2 + w^2) + md^2 = 36833 kgm^2 \tag{2.1}$$

2.1.1 Rocket Dynamics

In this section, MATLAB was used to create a mathematical model for the rocket. Both the dynamics of the rocket's motion in the translational and rotational degrees of freedom are captured by the model. The development of this model is intended to examine and comprehend the behavior of the rocket under various circumstances and to develop control schemes for its trajectory.

To begin, the rocket's position, velocity, and acceleration in the x and y directions, and the angle about the z-axis were defined as symbolic variables. These symbolic variables are then used to define the generalised coordinates of the rocket as follows:

$$q = [x, y, th]^T \tag{2.2}$$

The next step was to calculate the rocket's position and velocities in the inertial frame, but before doing so we defined the 3x3 rotation matrix about the z-axis to rotate the positions into the inertial or world frame. After using the rotation matrix, we end up with the following position vectors for the rocket, and the rocket's thruster force:

$$R_{Rocket} = [x, y, 0]^T \tag{2.3}$$

$$R_{Force} = \begin{vmatrix} x + d * sin(\theta) \\ y - d * cos(\theta) \\ 0 \end{vmatrix} \tag{2.4}$$

The translational and rotational velocities of the rocket are:

$$v_{translational} = [\dot{x}, \dot{y}, 0]^T \tag{2.5}$$

$$v_{rotational} = \omega = \dot{\theta} \tag{2.6}$$

Next, to analyze the rocket's energy, we need to compute the kinetic and potential energy of the rocket. The total kinetic energy, Ttotal, is determined by summing translational kinetic energies, T1 and T2, and its final value is:

$$T_{total} = \frac{2531167393109333 * \dot{\theta}^2}{137438953472} + \frac{m * \dot{x}^2}{2} + \frac{m * \dot{y}^2}{2} \tag{2.7}$$

And the potential energy:

$$V_{total} = m * g * y \tag{2.8}$$

Where m is the mass as stated in [1].

To establish the dynamics of the rocket, the mass matrix, M, and additionally, the mass matrix derivative, $\dot{M}$, gravity matrix, G, and coriolis matrix, C must be defined as follows:

$$M_{i,j} = \frac{\partial T_{total}}{\partial \dot{q}_i \partial \dot{q}_j} = \begin{vmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & 36833 \end{vmatrix} = \begin{vmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 36833 \end{vmatrix} \tag{2.9}$$

$$\dot{M} = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{vmatrix} \tag{2.10}$$

$$G = \frac{\partial V}{\partial q} = \begin{vmatrix} 0 \\ m * g \\ 0 \end{vmatrix} = \begin{vmatrix} 0 \\ 9810 \\ 0 \end{vmatrix} \tag{2.11}$$

$$C = \dot{M}\dot{q} - \frac{\partial T}{\partial q} = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} \tag{2.12}$$

Moving on, the thruster force vector in the inertial frame and generalised forces are calculated, and then resolved into the inertial frame. The partial derivatives of the force positions are determined, and the generalised force components are obtained by multiplying the force components with corresponding partial derivatives:

$$f_j = \begin{vmatrix} -F * sin(\alpha + \theta) \\ F * cos(\alpha + \theta) \\ 0 \end{vmatrix} \tag{2.13}$$

The generalised force equation used to compute generalised force vector, Q, is as follows:

$$Q_i = \sum_{j=1}^{m} f_j * \frac{\partial r_j}{\partial q_i} \tag{2.14}$$

Which leads to:

$$Q = \begin{vmatrix} -F * sin(\alpha + \theta) \\ F * cos(\alpha + \theta) \\ -F * d * sin(\alpha) \end{vmatrix} \tag{2.15}$$

The mass matrix (M), coriolis matrix (C), gravity matrix (G), and generalized forces are used to define the manipulator equation:

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = Q \tag{2.16}$$

The accelerations ($\ddot{q}$) and the forces and torques acting on the rocket are related in this equation, which reflects the rocket's dynamics. Finally, the rocket's dynamics are:

$$\begin{vmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{vmatrix} = \begin{vmatrix} \frac{-(F*sin(\alpha+\theta))}{m} \\ \frac{-(m*g-F*cos(\alpha+\theta))}{m} \\ \frac{-((6.87e11))*F*d*sin(\alpha))}{2.53e16} \end{vmatrix} \tag{2.17}$$

2.2 Asteroid Modelling

Figure 2.2: Asteroid hurtling towards Earth

As mentioned earlier for the rocket's parameters, the asteroid's parameters can be found in the milestone 1 brief [1] and will not be reiterated here because of page number limitations.

Of importance however, are the asteroid's dynamics:

- $$\ddot{x} = \frac{F_{dragx}}{m} + noise_x \tag{2.18}$$
- $$\ddot{y} = \frac{F_{dragy}}{m} - g + noise_y \tag{2.19}$$
- $$\ddot{\theta} = noise_\theta \tag{2.20}$$

Simulations and data analysis estimate the proportionality constant, c, considering process noise and the asteroid's tumbling velocity. The predicted value of c is expected in the range of approximately 100, but it can vary. The process for determining c is as follows:

1. **Simulation**
   To acquire velocity and time data, the script runs a simulation with the 'AsteroidImpact' model.

This simulation represents the asteroid's motion.

2. **Determining Acceleration from Velocities and Filtering Acceleration Data**
   By using the time derivative of the input velocities acquired from the simulation, the accelerations in the x and y directions are determined and displayed in Figure 2.3:

```
ast_ddx = diff(ast_dx)./diff(simulation_time);
ast_ddy = diff(ast_dy)./diff(simulation_time);

```

Listing 2.1: Determining Accelerations

Filtering techniques reduce noise in the acceleration data. Initially, a standard filter is ineffective, but a Gaussian-weighted moving average filter with a 20-bit window size improves noise reduction. Filtering is applied to both x and y directions.

```
ddx_ast_filter = smoothdata(ast_ddx);
ddx_ast_filter1 = smoothdata(ast_ddx,"gaussian",20);
ddy_ast_filter1 = smoothdata(ast_ddy,"gaussian",20);

```

Listing 2.2: Filtering Accelerations

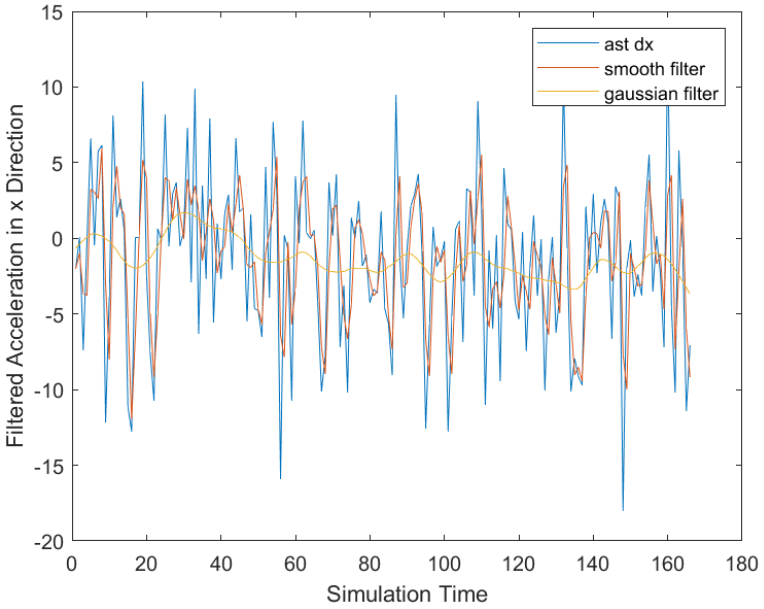


Figure 2.3: Initial and Filtered Accelerations

3. **Determine Drag Forces**
   The filtered x-acceleration is divided by the asteroid's mass to calculate drag force in that direction. The y-direction drag force is obtained by multiplying the mass-scaled force with the filtered y-acceleration and gravitational acceleration (g = 9.81 m/s²). The overall drag force

magnitude is determined using Pythagoras' theorem for the combined x and y forces. The magnitude of the overall velocity is then determined using the Pythagorean theorem based on the filtered velocity data.

```
% Determine drag force in the X direction
Fdrag_x = 10000*ddx_ast_filter1;

% Determine drag force in the Y direction
g_array = 9.81*ones(length(ddy_ast_filter1));
Fdrag_y = 10000*(ddy_ast_filter1 + g_array);

% Determine the magnitude of the total drag force by using pythagoras
% theorem
Fdrag = sqrt(Fdrag_x.^2 + Fdrag_y.^2);

% Determine the magnitude of the total velocity using pythagoras theorem
ast_dx_filter = smoothdata(ast_dx, "gaussian", 20);
ast_dy_filter = smoothdata(ast_dy, "gaussian", 20);
dq = sqrt(ast_dx_filter.^2 + ast_dy_filter.^2);

```

Listing 2.3: Obtaining Drag Force and Velocity

4. **Final Drag Coefficient**

The drag coefficient array is obtained by dividing the magnitude of the overall drag force by the magnitude of the velocity. This reveals the variation in the drag coefficient as the asteroid tumbles. Equation 2.21 represents the relationship between the drag force and velocity through the drag coefficient.

$$F_{drag} \propto c\sqrt{\dot{x}^2 + \dot{y}^2} \tag{2.21}$$

```
% Determine array of drag coefficients
c_array = Fdrag./dq(1:end-1);

% Final drag coefficient value
c = mean(c_array);

```

Listing 2.4: Final Drag Coefficient

The final value for the drag coefficient is:

$$c = 106.9934$$

Chapter 3

Control Scheme

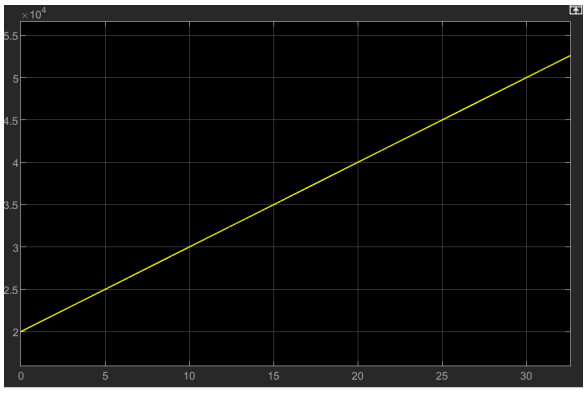
In order to control the rocket's trajectory as it tracked the asteroid, a simple lead lag compensator was designed to control the rocket's angle θ (Figure 2.1), by manipulating angle, α, but first, we need to use $\ddot{\theta}$ from Equation 2.2 to determine a relationship between alpha and theta. We compute the Laplace Transform of Equation 2.2 using the following transform:

$$\mathcal{L}\{\ddot{\theta}\} = s^2\theta - s\theta(0) - \dot{\theta}(0) \tag{3.1}$$

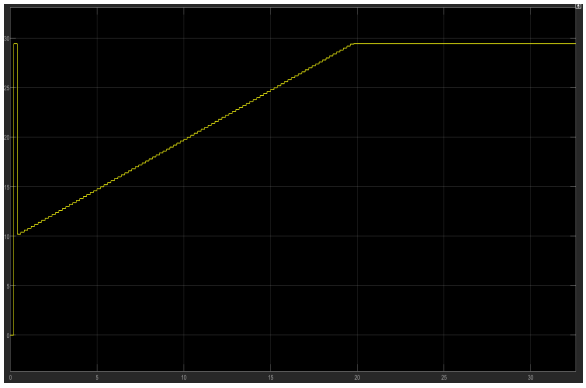
And we end up with a Plant model as follows representing the relationship between α and θ:

$$G = \frac{-3.5F}{I_z s^2} \tag{3.2}$$

Where F is the force of the rocket's thrusters which will be held constant while we control θ, and I_z is the inertia calculated in Equation 2.1. To proceed with the development of the lead-lag controller we need to determine a value for F. This is because F saturates after a certain value in the simulation. This will be done by plotting the force in response to a ramp input until it saturates and obtaining this value using scopes. The force plot can be seen in Figure 3.1a below, while the acceleration in the y-direction can be seen in Figure 3.1b:



(a) Input Force ramped to rocket



(b) Rocket Acceleration in the y-direction

Figure 3.1: Ramp Input Force and Rocket Y Acceleration

To get the saturation force value we observe the instant when the acceleration becomes constant and

simply obtain the force at the same instant in time. This value was found to be:

$$F \approx 40000N$$

Next we may choose a value for force that is slightly below the saturation value. For this project we will proceed with:

$$F = 39245N$$

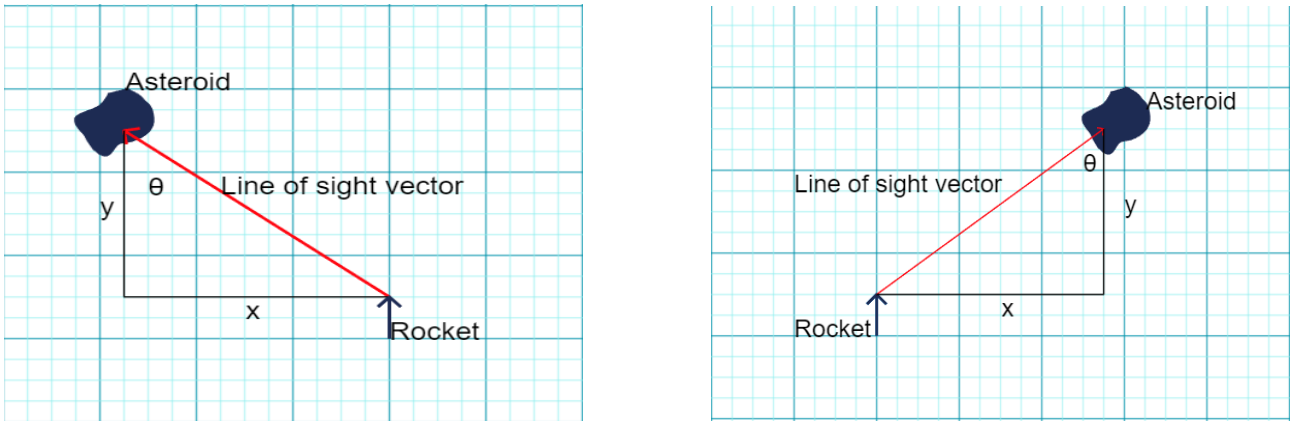
Now we proceed to use Equation 3.2 and Matlab's sistool function to design a lead-lag controller, and we end up with the following transfer function for the controller:

$$C = \frac{1.102s + 0.404}{s + 3.005} \tag{3.3}$$

Now, we are able to control the rockets angle θ while keeping force constant, however, this is not enough to track the missile. We still need to tell the rocket what angle to track as the asteroid hurtles toward the city. In this project we will consider three cases for the rocket's tracking angle. This 'auxiliary' control scheme was implemented as a Matlab function block in the Simulink file which can be seen in appendix D along with its code listing in appendix B.

3.1 Case 1

The asteroid is **above** the rocket as shown in Figure 3.2 below:



(a) Asteroid is above and to the left of Rocket

(b) Asteroid is above and to the right of Rocket

Figure 3.2: Scenario where **asteroid** is above **rocket**

In this case the angle that the rocket needs to track is:

$$\theta = \arctan \frac{x}{y} \tag{3.4}$$

3.2 Case 2

Please note, this scenario was accounted for because along with telling the rocket which angle to track, it was necessary to slightly predict the asteroid's trajectory. This will be discussed in more detail in section 4. The asteroid is **below** the rocket as shown in Figure 3.3 below:

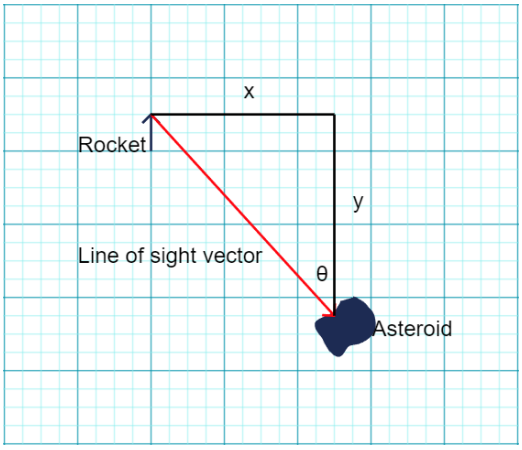


Figure 3.3: Scenario where **asteroid** is below **rocket**

In this case the angle that the rocket needs to track is:

$$\theta = -\arctan\frac{x}{y} \tag{3.5}$$

The control scheme developed works across all three scenarios of the asteroid's trajectory, with the exception of scenario 3 where it was required of the rocket to intercept either the front or back face of the asteroid. In this regard, it under-performed. However, this control scheme proved to be highly effective at controlling the rocket's angle θ, while the force was held constant at 39245N, and ultimately, intercepting the asteroid within the requirements of the scope of scenario 2 and 3. We will see its effectiveness in section 4.

Chapter 4

Results and Discussion

Earlier, we mentioned that it was necessary to predict the trajectory of the asteroid in order to enhance the functionality of the 'auxiliary' control scheme. The estimation was done by using the asteroid's dynamics given in Equations 2.18 and 2.19, as well as the drag coefficient, c, obtained earlier. The predicted trajectory compared to the actual trajectory after simulation can be seen in Figure 4.1 below where the blue curve is the actual trajectory:

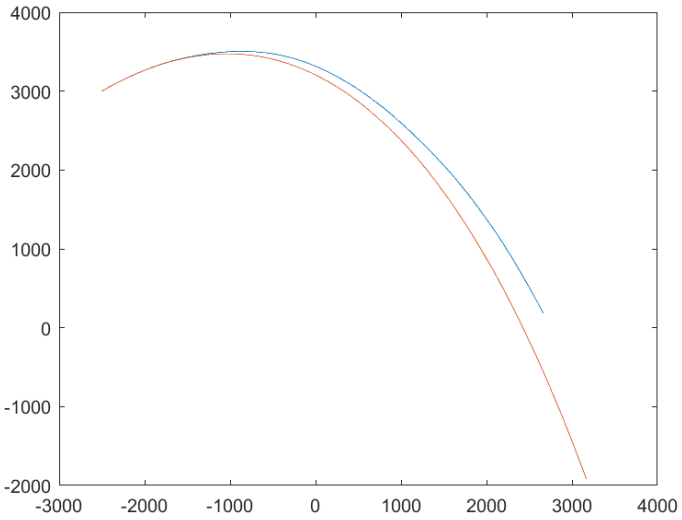
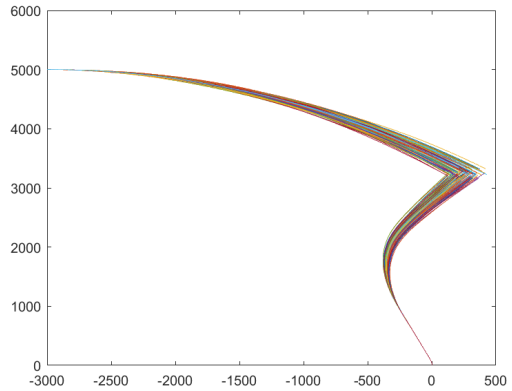


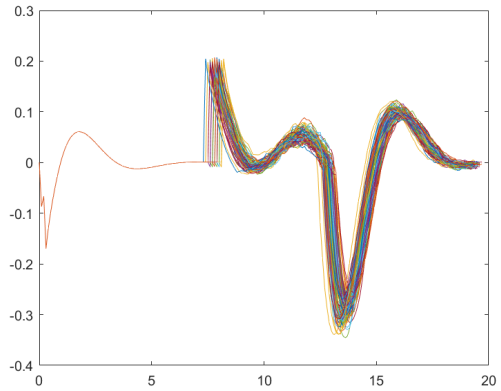
Figure 4.1: Predicted versus Actual Trajectory of Asteroid

4.1 Scenario 1 Testing after 100 simulations

The effectiveness of the control scheme was evaluated by running the simulation 100 times, each time with a randomly generated seed value in the range of 100-1000. After each simulation, a plot of the asteroid's and rocket's trajectory was plotted on the same curve. The Matlab script used to test the control scheme 100 times can be seen in appendix C. Its results can be seen in Figure 4.2a below:



(a) Scenario 1 Hit Rate after 100 randomly generated seed simulations



(b) Rocket angle α versus simulation time for each simulation

```
Command Window
  ---------------------------------------------
  Mission complete: well done! Make sure to test with multiple seeds
  Seed number 99
  Seed value: 307
  x distance between city and rocket-asteroid intercept: 1829.4053m
  Distance between rocket and asteroid: 73.7852m
  ---------------------------------------------
  Mission complete: well done! Make sure to test with multiple seeds
  Seed number 100
  Seed value: 619
  x distance between city and rocket-asteroid intercept: 1570.521m
  Distance between rocket and asteroid: 81.2798m
  ---------------------------------------------
  Accuracy: 98
fx >>
```

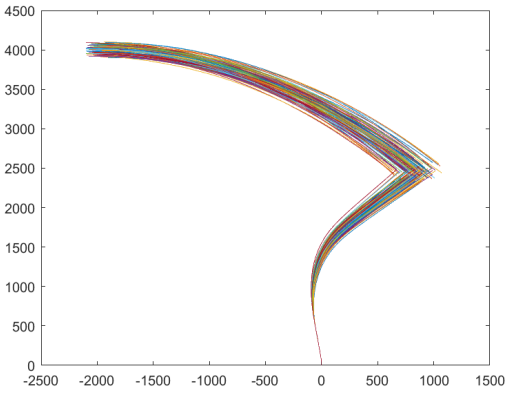
(c) Accuracy of Scenario 1 after 100 simulations

Figure 4.2: Scenario 1 Results after 100 simulations, each with a random seed value

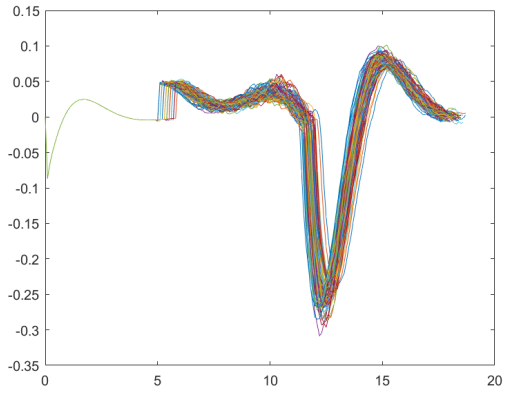
As you can see in Figure 4.2c, the control scheme has a 98% hit success rate for scenario 1. Along with this we can see from Figure 4.2b, that angle α never violates its constraints, $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$.

4.2 Scenario 2 Testing after 100 simulations

The same method was used to test the effectiveness of the control scheme for scenario 2. Once again, after each simulation, a plot of the asteroid's and rocket's trajectory was plotted on the same curve. Its results can be seen in Figure 4.3a below:



(a) Scenario 2 Hit Rate after 100 randomly generated seed simulations



(b) Rocket angle α versus simulation time for each simulation

```
Command Window
  ------------------------------------------------
  Mission complete: well done! Make sure to test with multiple seeds
  Seed number 99
  Seed value: 133
  x distance between city and rocket-asteroid intercept: 1142.4314m
  Distance between rocket and asteroid: 84.8376m
  ------------------------------------------------
  Mission complete: well done! Make sure to test with multiple seeds
  Seed number 100
  Seed value: 536
  x distance between city and rocket-asteroid intercept: 1200.5997m
  Distance between rocket and asteroid: 76.2368m
  ------------------------------------------------
  Accuracy: 98
fx >>
```

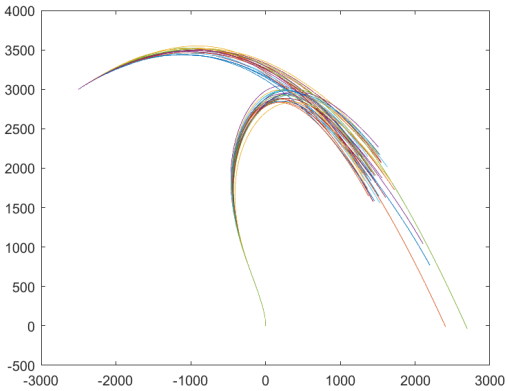
(c) Accuracy of Scenario 2 after 100 simulations

Figure 4.3: Scenario 2 Results after 100 simulations, each with a random seed value

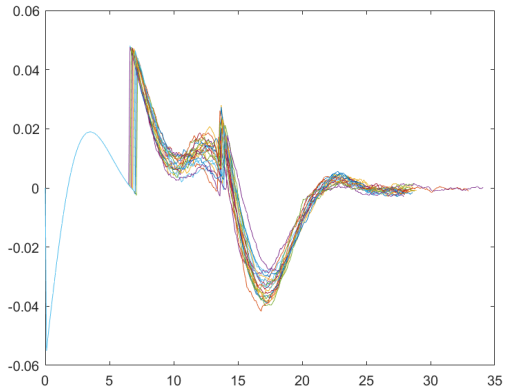
As you can see in Figure 4.3c, the control scheme has a 98% hit success rate for scenario 2. Along with this we can see from Figure 4.3b, that angle α never violates its constraints, $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$.

4.3 Scenario 3 Testing after 20 simulations

Once again, the same method was used to test the effectiveness of the control scheme for scenario 3. However, in this case, only 20 randomly generated seeds were tested using appendix C. This is because any more simulations would be unnecessary because the control scheme does not work well for scenario 3. Once again, after each simulation, a plot of the asteroid's and rocket's trajectory was plotted on the same curve. Its results can be seen in Figure 4.4a below:



(a) Scenario 3 Hit Rate after 20 randomly generated seed simulations



(b) Rocket angle α versus simulation time for each simulation

```
Command Window

  Mission complete: well done! Make sure to test with multiple seeds
  Seed number 19
  Seed value: 813
  x distance between city and rocket-asteroid intercept: 275.2716m
  Distance between rocket and asteroid: 86.9269m
  ------------------------------------------------
  Mission complete: well done! Make sure to test with multiple seeds
  Seed number 20
  Seed value: 964
  x distance between city and rocket-asteroid intercept: 437.6038m
  Distance between rocket and asteroid: 98.5933m
  ------------------------------------------------
  Accuracy: 40
fx >>
```

(c) Accuracy of Scenario 3 after 20 simulations

Figure 4.4: Scenario 3 Results after 20 simulations, each with a random seed value

As you can see in Figure 4.4c, the control scheme has a 40% hit success rate for scenario 3. Along with this we can see from Figure 4.4b, that once again, the angle α never violates its constraints, $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$.

Chapter 5

Conclusions

In conclusion, through the comprehensive modelling and analysis carried out in this report, we have successfully determined the drag coefficient (c) of the asteroid using simulation data and filtering techniques, and have successfuly modelled the rocket's dynamics in order to control its trajectory while attempting to intercept the asteroid. The developed control scheme has demonstrated its effectiveness in addressing the requirements and achieving desired outcomes in scenario 1 and 2 with a hit success rate of 98% each. However, in scenario 3, with an approximate 40% success rate, it became apparent that further design improvements and testing are necessary to meet the requirements of milestone 3 accurately. The results obtained provide valuable insights into the dynamics of the asteroid and lay the foundation for future iterations and enhancements to the control scheme, enabling more precise control and mitigation of potential risks in real-world scenarios.

Bibliography

[1] "Eee4119f_project_milestone1_brief.pdf," 2023. [Online]. Available: https://amathuba.uct.ac.za/content/enforced/14446-EEE4119F_2023/Project/EEE4119F_Project_MileStone1_Brief.pdf

Appendix A

Code listing for Modelling the Rocket and Asteroid Dynamics

```
%% Author: _Luke Baatjes_
% _EEE4119F Mechatronics II Project Milestone 1_
% Date: _05/03/2023_
%
%
%
%% *ROCKET MODEL*
%% _Symbolic Variables and Generalized Coordinates_

syms x dx ddx y dy ddy th dth ddth 'real'

% ROCKET PARAMETERS
syms alph F m d g Iz

% The rocket is allowed to translate in the x and y directions and allowed
% to rotate about an angle theta in the z axis of rotation.
% GENERALISED COORDINATES
q = [x; y; th];
dq = [dx; dy; dth];
ddq = [ddx; ddy; ddth];

% Inertia
Iz = (1/12 * 1000 * (15^2 + 5^2)) + (1000 * 4^2);

%% _Rotation Into, Positions, and Velocities in Inertial Frame_

% ROTATIONS
% There is only 1 other frame relative to the Inertial frame hence the need
% for only one rotation about the Z axis
R01 = Rotz(th);                 % Rotation from Inertial Frame to Body Frame
```

```
R10 = transpose(R01);           % Rotation from Body Frame to Inertial Frame

% POSITIONS
rR_0 = [x;y;0];                 % Rocket's center of mass Position in Inertial Frame
rF_B = [0;-d;0];                % Position of Force F located at position -d in body Frame
rF_0 = rR_0 + R10*rF_B;         % Position of Force F in Inertial Frame (Sum of position of
    ↪ centre of mass of rocket and position of force rotated to the Inertial frame)

% VELOCITIES
drR_0 = jacobian(rR_0,q)*dq;    % Translational velocity of rocket
wR_0 = dth;                     % Angular velocity of the rocket

%% _Energy of Rocket_

% KINETIC ENERGY
T1 = 0.5*m*transpose(drR_0)*drR_0;     % Translational Kinetic Energy
T2 = 0.5*transpose(wR_0)*Iz*wR_0;      % Angular Kinetic Energy. In this case the Mass
    ↪ Moment of Inertia is calculated using the parallel axis theorem (1/12*m*(L1^2 + L2
    ↪ ^2) + md^2).
Ttot = simplify(T1 + T2);

% POTENTIAL ENERGY
Vtot = simplify(m*g*rR_0(2));
%% _Define Mass Matrix_

% MASS MATRIX
M = hessian(Ttot,dq);
%% _Define Mass Matrix Deriv_

dM = sym(zeros(length(M),length(M)));
for i=1:length(M)
    for j=1:length(M)
        dM(i,j) = jacobian(M(i,j),q)*dq;
    end
end
dM = simplify(dM);
%% _Define Gravity Matrix_

% GRAVITY MATRIX
G = jacobian(Vtot,q);
G = simplify(G);
%% _Define Coriolis Matrix_
```

```
70
71 % CORIOLIS MATRIX
72 C = dM*dq - transpose(jacobian(Ttot,q));
73 C = simplify(C);
74 %% _Force Position and Generalised Forces_
75
76 % FORCES
77 % Magnitude of force in Inertial Frame
78 Fvec_B = [-F*sin(alph);F*cos(alph);0];              % Force in Frame 1, using trigonometry
       ↪ to resolve force into components in the body frame
79 Fvec_0 = simplify(R10*Fvec_B);                      % Force in Inertial frame
80
81 % GENERALIZED FORCES
82 % Partial derivatives of position where force acts
83 partX = jacobian(rF_0,x);
84 partY = jacobian(rF_0,y);
85 partTh = jacobian(rF_0,th);
86
87 % Generalised force components
88 Qx = simplify(Fvec_0(1)*partX(1) + Fvec_0(2)*partX(2) + Fvec_0(3)*partX(3));
89 Qy = simplify(Fvec_0(1)*partY(1) + Fvec_0(2)*partY(2) + Fvec_0(3)*partY(3));
90 Qth = simplify(Fvec_0(1)*partTh(1) + Fvec_0(2)*partTh(2) + Fvec_0(3)*partTh(3));
91
92 Q = [Qx; Qy; Qth];
93 %% _Define Manipulator Equation_
94
95 % MANIPULATOR EQUATION
96 ManipulatorEqn = M*ddq + C + G.' == Q;
97
98 % Solve for accelerations
99 ddx = simplify(solve(ManipulatorEqn(1), ddx));
100 ddy = simplify(solve(ManipulatorEqn(2), ddy));
101 ddth = simplify(solve(ManipulatorEqn(3), ddth));
102 %% _Rocket Accelerations in the X, Y, and Theta Directions_
103
104 disp("Rocket Accelerations:")
105 acceleration = [ddx;ddy;ddth]
106 %% *ASTEROID MODEL*
107 %
108 %% _Determining the Asteroids Drag Coefficient Value_
109
110 % System accepts velocities in the x, y, and theta directions and is
```

```
% expected to output the accelerations of the Asteroid in simulation
% Run Simulation to populate workspace
sim('AsteroidImpact');

% Determine Accelerations in the x and y directions by taking the time
% derivitive of the input velocities
ast_ddx = diff(ast_dx)./diff(simulation_time);
ast_ddy = diff(ast_dy)./diff(simulation_time);

plot(ast_ddx, "DisplayName","ast dx")
xlabel("Simulation Time")
ylabel("Filtered Acceleration in x Direction")
hold on

% Using the acceleration of the asteroid in the x direction as an example,
% it is clear that using a regular filter results in a poor reduction of
% noise (Orange curve)
ddx_ast_filter = smoothdata(ast_ddx);

% Filter the moise accelerations by using a Guassian-Weighted Moving
% Average Filter with a window size of 20
% It is clear that a gaussian filter produces a much better result (Yellow curve)
ddx_ast_filter1 = smoothdata(ast_ddx,"gaussian",20);
ddy_ast_filter1 = smoothdata(ast_ddy,"gaussian",20);

% plot(ddx_ast_filter, "DisplayName","smooth filter")
% plot(ddx_ast_filter1, "DisplayName","gaussian filter")
% legend
% hold off

% Determine drag force in the X direction
Fdrag_x = 10000*ddx_ast_filter1;

% Determine drag force in the Y direction
g_array = 9.81*ones(length(ddy_ast_filter1));
Fdrag_y = 10000*(ddy_ast_filter1 + g_array);

% Determine the magnitude of the total drag force by using pythagoras
% theorem
Fdrag = sqrt(Fdrag_x.^2 + Fdrag_y.^2);

% Determine the magnitude of the total velocity using pythagoras theorem
```

```
ast_dx_filter = smoothdata(ast_dx, "gaussian", 20);
ast_dy_filter = smoothdata(ast_dy, "gaussian", 20);
dq = sqrt(ast_dx_filter.^2 + ast_dy_filter.^2);

% Determine array of drag coefficients
c_array = Fdrag./dq(1:end-1);

% Final drag coefficient value
c = mean(c_array);
disp("Drag force coefficient c:")
disp(c(1))
%% _Rotation Function_

function A = Rotz(th)
    A = [cos(th)   sin(th) 0;...
         -sin(th)  cos(th) 0;...
         0        0        1];
end

```

Appendix B

'Auxiliary' control scheme code listing

```
function [th, detonate, rx, ry, Force] = pro_nav(x, y, ast_x, ast_dx, ast_y, ast_dy, time,
    ↪ cond)

%% _Determine Scenario From Initial Conditions_
%
if (cond == -3000)
    scenario = 1;
else
    scenario = 2;
end

%% _Determine Trajectory_

if scenario == 2
    % _Trajectory Estimation_
    %
    % Using equation of motion to predict position of asteroid after a few increments
    % r = r_0 + V_0*t + 0.5*a*t^2

    Force = 29275;
    Ts = 0.1;
    dt = 8*Ts;
    rx = ast_x + ast_dx*(time+dt);
    ry = ast_y + ast_dy*(time+dt) + 0.5*(-9.81)*((time+dt)^2);


    % _PROPORTIONAL NAVIGATION_
    % Track using theta
    dist = (x - rx);

    xrange = x - rx;
    yrange = ry - y;
```

```
32
33      % Determine tracking angle based on where (above or below) asteroid is
34      if ry > y % Asteroid above rocket
35
36          % Only start tracking after asteroid is close to position directly
37          % above rocket
38          if dist > 100
39              th = pi/39.84;
40          else
41              th = atan(xrange/yrange);
42          end
43
44      else % ASTEROID BELOW ROCKET
45          th = -atan(xrange/yrange);
46      end
47
48  else
49      Force = 31825;
50      Ts = 0.1;
51      dt = 10*Ts;
52      rx = ast_x + ast_dx*(time+dt);
53      ry = ast_y + ast_dy*(time+dt) + 0.5*(-9.81)*((time+dt)^2);
54
55      dist = (x - rx);
56      xrange = x - rx;
57      yrange = ry - y;
58
59      % Asteroid above rocket
60      if ry > y
61
62          % Only start tracking after asteroid is close to position directly
63          % above rocket
64          if dist > 0
65              th = pi/17;
66          else
67              th = atan(xrange/yrange);
68          end
69
70      else % ASTEROID BELOW ROCKET
71          th = -atan(xrange/yrange);
72      end
73
```

```
end

%% _SET DETONATION_

Xrange = abs(x - ast_x);
Yrange = abs(y - ast_y);
Dist = sqrt((Xrange^2) + (Yrange^2));

if Dist < 150
    detonate = 1;
else
    detonate = 0;
end

%% _OUTPUTS_

out = [th, detonate, rx, ry, Force];

```

Appendix C

Seed Test used to test control scheme 100 times

```
%% _SEED TEST_
%% ***_PLEASE NOTE THAT YOU HAVE TO OPEN THE AsteroidImpact.slx FILE BEFORE RUNNING THIS
    ↪ SCRIPT_***
close all;
clc;
%
% This script will run the simulation a specified number of times (number
% of times run is set by 'runs' variable below), each time with a randomly
% generated seed value in the range of 100 to 500. This is to show the
% robustness of the controller.
% In the command window you will be able to see both the x distance between
% the rocket-asteroid intercept and city and the distance
% between the rocket and asteroid when the rocket detonates

Scenario = 1; % This doesnt really matter for scenario 1 and 2

HIT = 0;
runs = 100;
seed = randi([100 1000], 1, runs);

for i = 1:runs
    set_param(['AsteroidImpact', '/rngSeed'], 'Value', 'seed(i)');
    sim('AsteroidImpact.slx');  % tell simulink to simulate the model

    if (mission_complete(x, y, ast_x, ast_y, ast_th, Scenario)) == true

        % Plot trajectories and interception
        plot(simulation_time, alpha.signals.values)
        hold on
%         plot(x,y)
```

```
30
31          HIT = HIT + 1;
32          xrange = 2000 - ast_x(end);
33          xrange1 = abs(x(end) - ast_x(end));
34          yrange1 = abs(y(end) - ast_y(end));
35          dist = xrange;
36          run_instance = ['Seed number ', num2str(i)];
37          disp(run_instance);
38          seed_value = ['Seed value: ', num2str(seed(i))];
39          disp(seed_value)
40          dist_from_city = ['x distance between city and rocket-asteroid intercept: ', num2str
    ↪ (dist), 'm'];
41          disp(dist_from_city)
42          dist1 = sqrt(xrange1^2 + yrange1^2);
43          dist_betw_rocket_and_ast = ['Distance between rocket and asteroid: ', num2str(dist1)
    ↪ , 'm'];
44          disp(dist_betw_rocket_and_ast)
45          disp('-----------------------------------------')
46      else
47          mission_complete(x, y, ast_x, ast_y, ast_th, Scenario)
48          HIT = HIT;
49      end
50
51  end
52
53  accuracy = (HIT/runs)*100;
54  acc = ['Accuracy: ', num2str(accuracy)];
55  disp(acc)
56
```

Appendix D

Simulink Diagram

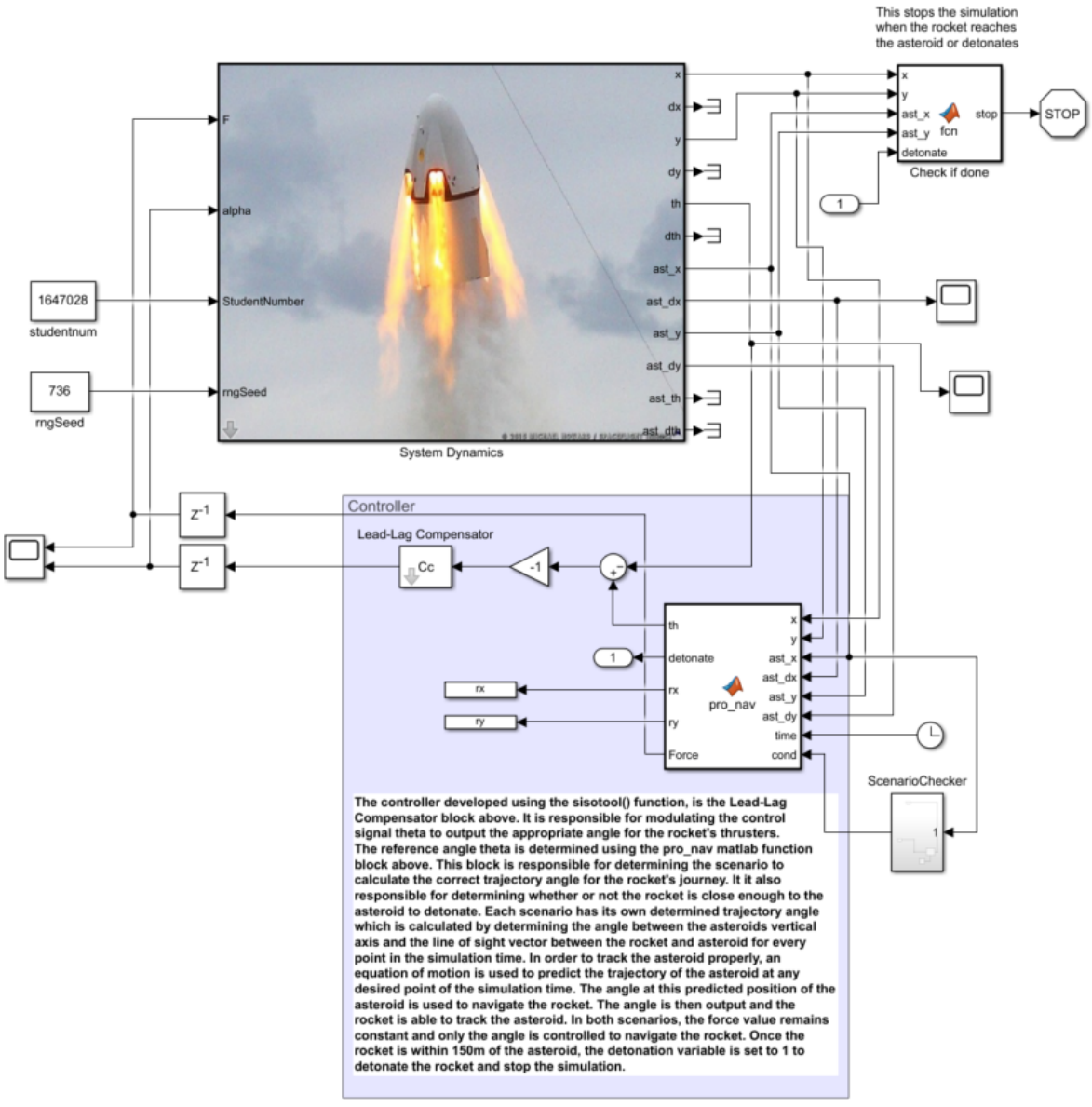


Figure D.1: Illustration of Simulink file