University of Cape Town

EEE4119F

Mechatronics

Milestone 4: Report

Kamryn Norton
NRTKAM001

May 17, 2023

Contents

1 Introduction

The purpose of this project was to model the system of an asteroid hurtling towards earth, and a rocket which is commissioned to stop it - and using these models, to come up with a controller for the rocket which enables it to destroy the asteroid before the asteroid hits a city, within certain parameters and conditions, which are different for the three scenarios provided.

1.1 Problem Parameters

The problem parameters are different for the asteroid and rocket and are interpreted as the constraints on the system, as well as the given values. These can be seen in Table 1:

Measurement	Asteroid	Rocket
Mass (kg)	10 000	1 000
Drag (N)	$\approx$ 100 (Milestone 1)	None
Dimensions	Negligible	5m x 15m (COM at 3.5m height)
Thrust angle range	No thrust	$\frac{-\pi}{2} \leq \alpha \leq \frac{\pi}{2}$
Initial Position (m)	Unknown: (x, y)	(0,0)
Starting angle (rad)	$\theta_{ast} = 0$	$\theta_r = 0$
Acceleration (gravity) ($m.s^{-2}$)	-9.81	-9.81

Table 1: Table of initial conditions of the system

1.2 Goals

The goals for the project were to complete each milestone and create a controller to achieve the the goals of each scenario. The scenarios, taken from the Milestone Briefs [1] [2], include the goals as follows:

1. For scenario 1: "The rocket must detonate within 150 meters of the centre of mass of the asteroid, before [it] falls to within 200 meters of the city, measured as height above ground" [1].

2. For scenario 2: The conditions for scenario 1 apply, however "In this scenario, the asteroid begins in a state such that flying straight up is not an option" [1] and so the rocket needs to be flying at an angle all the time.

3. For scenario 3: The conditions for Scenario 1 still apply here, however "The rocket must strike/detonate on/near the asteroid in a 60-degree arc in front the asteroid or behind the asteroid to inflict maximum damage and destroy the asteroid" [2].

2 Modelling

2.1 Rocket Modelling

To model the rocket, it's manipulator equation is the end result of the model we are looking for. To get this manipulator equation, the following procedure was followed:

1. Instantiate the rocket's parameters, rotations, position and velocity vectors. The diagram below was used to aid the thinking process, showing the dimensions, centre of mass, as well as the thought behind the rotation matrices needed (Rotation around Z through θ), and the thrust angle (α) considerations in conjunction with theta (θ).

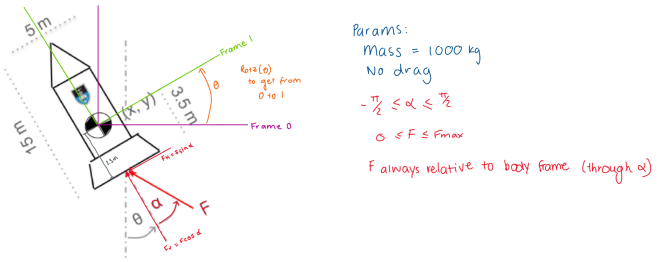

Figure 2.1.1: Diagram used in modelling thought process

2. Calculate the Rocket's Kinetic and Potential energies
3. Calculate the Rocket's Mass matrix, Coriolis Matrix, Gravity Matrix and Generalised forces.
4. The mass moment of inertia was calculated separately, and confirmed by the calculations in for the manipulator equation.
5. The manipulator equation was presented.

Manipulator Equation vector

$$\begin{pmatrix} \text{IzR ddth} = F\,d\,\sin(\text{alph}) \\ \text{ddx mR} = -F\,\sin(\text{alph} - \text{th}) \\ \text{mR (ddy} + g) = F\,\cos(\text{alph} - \text{th}) \end{pmatrix}$$

var =

$$\begin{pmatrix} 3.6833\text{e+}04\ \text{ddth} = 3.5000\,F\,\sin(\text{alph}) \\ 1000\ \text{ddx} = -F\,\sin(\text{alph} - \text{th}) \\ 1000\ \text{ddy} + 9810 = F\,\cos(\text{alph} - \text{th}) \end{pmatrix}$$

To confirm the above is correct: Iz = 36833.333333
Therefore the euqations match (symbolic = with floats)

$I_{zR} = 36833.333$ as calculated using L_1, L_2, d and m_r using perpendicular axis theorem. $I_{zR} = (\frac{1}{12}) * m_r * (L_1^2 + L_2^2) + mR * (\frac{L_2}{2} - d)^2 = 36833.333$

Figure 2.1.2: Manipulator Equation Output

The code and workings for this section can be found in Appendix 1, subsection 5.2.

2.2 Asteroid Modelling

To properly model the asteroid, the final value needed is it's drag coefficient. The following equations were given to aid the process:

$$F_{\text{drag}} \propto \sqrt{\dot{x}^2 + \dot{y}^2} \tag{1}$$

$$\ddot{x} = \frac{F_{\text{drag}_x}}{m} + \text{noise}_x \tag{2}$$

$$\ddot{y} = \frac{F_{\text{drag}_y}}{m} - g + \text{noise}_y \tag{3}$$

$$\underset{\text{ast}}{\ddot{\theta}} = \text{noise}_\theta \tag{4}$$

The drag coefficient, c, was then calculated for both the x and y components:

$$c \propto \frac{F_{\text{drag}_x}}{\dot{x}} = \frac{\ddot{x}m}{\dot{x}} = \frac{F_{\text{drag}_y}}{\dot{y}} = \frac{(\ddot{y}+g)\,m}{\dot{y}} \tag{5}$$

The components were then averaged to find the final drag coefficient.

```
len = length(ast_ddx);
c = zeros(len-1,1);
cy = zeros(len-1,1);
sum = 0;
sumy = 0;
for i = 2:1:len %start at 2 to eliminate the first ast_dy element =0
    c(i) = ast_ddx(i)*m/ast_dx(i); %x component
    sum=sum+c(i);
    cy(i) = (ast_ddy(i)+g)*m/ast_dy(i);
    sumy= sumy+cy(i);
end
cx = -sum/(len-1) %average of the drag x component
cy = -sumy/(len-1) %average of the drag y component
cfinal = (cx+cy)/2
fprintf("Final/Resultant drag coefficient, c = %f", cfinal)
```

The necessary variables were initialised prior to their use in the for loop.

Figure 2.2.1: Code to Find the Drag Coefficient

3 Control Scheme

3.1 Scenario 1

This scenario was the simplest and most trivial. To control the rocket, the approach was to model the asteroid's trajectory, and find out when (the point in time, t) it's horizontal position (x) would be the same as the rocket's - meaning when the asteroid was directly above the rocket. Then the only control needed to be applied to the rocket was thrust control, to get the rocket to fly straight up to the vertical (y) position of the asteroid in the given time, t. An online resource [3] was very helpful in giving some equations for projectile motion which incorporate drag, and were adaptable for the purposes of this project. The steps taken for the control of the rocket in this scenario were:

1. Find the asteroid's terminal velocity: $v_t = \frac{m_a \times g}{c}$ where c is the drag coefficient of the asteroid calculated in the asteroid modelling . process.

2. Calculate the initial horizontal distance between the asteroid and rocket: $x = x_{r0} - x_{ast0}$

3. Calculate the time it will take (t) for the asteroid to cover the distance between the rocket and asteroid. Because it is dealing with projectile motion, the acceleration is $g = 9.81ms^-2$ downwards due to gravity. The velocity of the asteroid is it's terminal velocity, and it's initial velocity is accounted for too: $t = \frac{-v_t}{g} \times \log\left(1 - \frac{xg}{\dot{x_{ast0}}) \times v_t}\right)$

4. Find the vertical target distance (y_{target}) of the asteroid above ground/sea level (the neutral axis) at the time t calculated above: $y_{target} = \frac{vt}{g} \times (\dot{y_{ast0}} + v_t) \times (1 - e^{(-gt)/v_t}) - t \times v_t + y_{ast0}$

5. The acceleration of the rocket ($\dot{y}_{rocket}$) needed to reach y_{target} in t seconds is calculated as: $\dot{y}_{rocket} = (y_{target} - \dot{y_{r0}}t) \times \frac{2}{t^2}$

6. Using Newton's Second Law of Motion, $F = ma$, the Force needed as thrust to get the rocket to the specified position in t seconds is calculated, where $a = a_{rocket} + g$ to take acceleration due to gravity into account, and $m = m_{rocket}$.

7. Alpha (the thrust angle) is set to 0 radians because the rocket is travelling directly upwards.

```
%Scenario 1
if scenario ==1
    % Use eqns from website
    % https://farside.ph.utexas.edu/teaching/336k/Newton/node29.html
    % find Vt of asteroid
    vt = ma*g/c;

    %calculate initial x distance between objects
    x = xinit_r-xinit_ast;

    % rearrange eqn 189 to find the time to location
    t = (-vt/g)*log(1-(x*g)/(dxinit_ast*vt));

    %y posn at time
    y_target = (vt/g)*(dyinit_ast+vt)*(1-exp((-g*t)/vt))-vt*t+yinit_ast;

    %finding accel of rocket in y to reach y position in given time
    accel_y = (y_target-dy_init_r*t)*2/(t^2);

    %f=ma
    F =mr*(accel_y+g); %account for gravity effects
    alpha = 0; %because straight up
end
```

Figure 3.1.1: Code for Scenario 1 Controller

The full code can be found in Appendix 1.

3.2 Scenario 2

For this Scenario, the rocket could not travel directly upwards, and the thrust angle needed to be controlled by observing the angle ($\theta_{correct}$) between the objects, and attempting to correct that angle to zero at each time iteration/instance of the simulation. The control algorithm for this scenario follows the steps:

1. A relatively small initial position to aim for: $x = 5$ is defined randomly, to give the controller a starting point (for angle adjustments to be made appropriately).
2. The steps as in Scenario 1 are followed until the end of step 5, finding the acceleration needed for the rocket to reach the target position.
3. The initial angle between the rocket and asteroid is calculated as $\theta_0 = \arctan(\frac{y_{target}}{x_{ast0}-x_{r0}})$
4. The angle of correction, which the controller needs to compensate for, is the difference between the current angle (θ) and the initial angle (θ_0): $\theta_{correct} = \theta - \theta_0$
5. The compensation angle is then scaled by a factor so that it represents the control angle of thrust, α: as $\alpha = K \times \theta_{correct}$, where K was found as $K = 0.001$ through trial and error. The pure correction angle was too large, and $K = 0.0001$ was too small, both causing the rocket to fly erratically and in loops.
6. The thrust is calculated as in Scenario 1, step 6.
7. The detonation condition is continually checked by the controller in an if statement: the rocket is set to detonate if it reaches within 150m of the asteroid, where this momentary distance is $d = \sqrt{(x_{ast} - x_r)^2 + (y_{ast} - y_r)^2}$.

The full code can be found in Appendix 1.

3.3 Scenario 3

For this scenario, there was an added complexity of the rocket having to hit the asteroid within a certain angle arc on the asteroid as mentioned in the Goals. Then, the controller needs to control the thrust and thrust angle to get the rocket to the asteroid in due time. The steps taken are listed below.

1. Firstly, the trajectory of the asteroid needed to be solved. The equations for this were given in class, and the y_{target} for this scenario was solved for using the symbolic maths toolbox in MATLAB.

```
    syms dy(t) dyinit_ast yinit_ast c ma g dt;
    ddy = diff(dy,t);
    ode = ddy == -c*dy/ma -g;
    cond = dy(0) == dyinit_ast;
    dy_traj = dsolve(ode, cond);
    y_pos = sum(dy_traj)*dt + yinit_ast
```

```
%%this returned:
%y_pos = yinit_ast - (dt*(g*ma - exp(-(c*t)/ma)*(c*dyinit_ast + g*ma
    )))/c
```

y_{pos} is then used in the controller.

2. Using angular kinematics, the angular acceleration of the asteroid is: $\ddot{\theta}_{ast} = \frac{\dot{\theta}_{ast}^2 - \dot{\theta}_{ast0}^2}{2*(\theta_{ast} - \theta_{ast0})}$

3. An array for time, which is linearly spaced between 0 and 30 (seconds) by the time step of the simulation $dt = 0.01$ (seconds), and theta were created, as we can use rotational kinematics to find the pitch(angle) of the asteroid at all times of the simulation. The Newtonian Rotational Kinematics dictate that $\theta = \theta_0 + w_0 * t + 1/2at^2$ Thus, the angle at a point in time (index i in the "time" and "theta" arrays) is: $\theta(i) = \theta_{ast0} + \dot{\theta}_{ast0} * time(i) + 0.5 * \ddot{\theta}_{ast} * (time(i))^2$

4. The time at which the asteroid enters the correct position and rotational range is when $\frac{\pi}{4} < \theta < \frac{\pi}{3}$, so when the asteroid angle enters this range, the index of the arrays is stored, and further multiplied by $dt = 0.01$ to get the time in seconds.

5. The target vertical distance to reach, y_{pos}, is now calculated as in the above block of code.

6. From here, the same procedure for finding vertical acceleration of the rocket, the correction angle, thrust angle and thrust force (as well as checking detonation conditions) were followed as done in the

3.4 Controller Implementation

The controller was implemented by adding it in the "AsteroidImpact.slx" Simulink model given as part of the project. The full controller code is given in Appendix 1. The following image represents the placement of the controller within the Simulink Model:

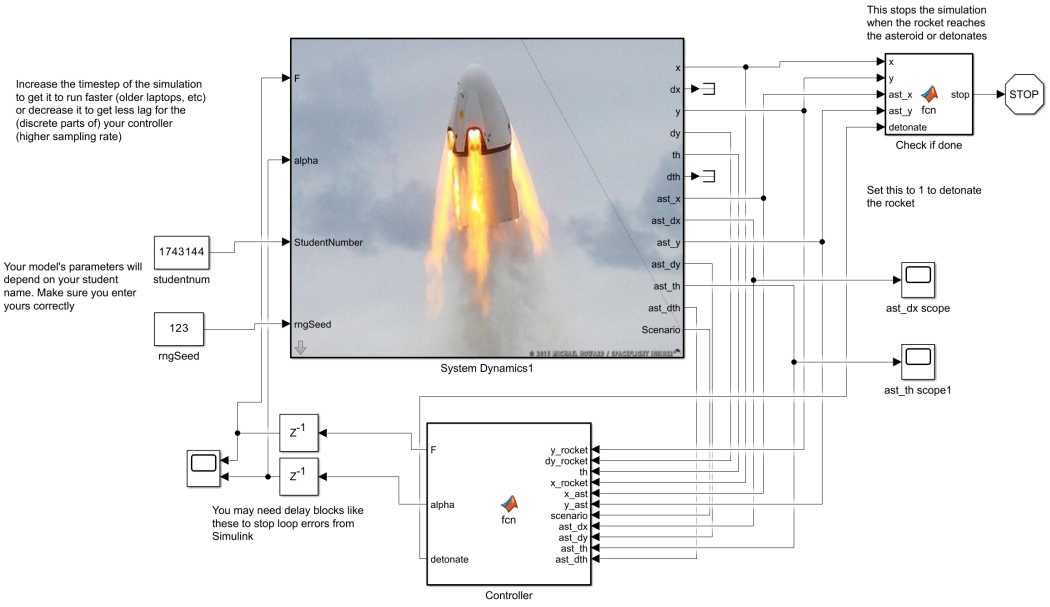

Figure 3.4.1: Simulink Setup of the Controlled System Model

4 Results and Discussion

4.1 Modelling

The modelling process was successful for the rocket, however the drag coefficient found was slightly incorrect the first time. The initial value found was $c \approx 146$, and since the feedback was given and marked, the values were revised and the average c value was $c = 87.25$. Although this was not ever confirmed again, because the guideline in the milestone descriptions were that $c \approx 100$, $c = 100$ was the value used in the simulations and in the controller. The results for the revised calculations are shown in Figure 4.1.1.

```
cx = -sum/(len-1) %average of the drag x component
cx = 97.3043
cy = sumy/(len-1) %average of the drag y component
cy = 77.1862
cfinal = (cx+cy)/2
cfinal = 87.2453
fprintf("Final/Resultant drag coefficient, c = %f", cfinal)
Final/Resultant drag coefficient, c = 87.245256
```

Figure 4.1.1: Result of the Drag Coefficient Calculations

4.2 Controller Performance

The controller designed for each Scenario performed well within the given Goals and Parameters. It is illustrated how the rocket and asteroid behaved, and that the end result was always the rocket successfully hitting the asteroid within the given scenario objectives for each scenario. When the files were run to simulate the situation and evaluate performance, if the rocket was unsuccessful in hitting the asteroid in any way, an appropriate error message would be displayed. These error messages were defined in the MATLAB code file given as part of the project, called "mission_complete.m". If the rocket was successful the following message was displayed in the MATLAB Terminal:

```
>> define_constants
>> main
Mission complete: well done! Make sure to test with multiple seeds
```

Figure 4.2.1: Success Message after running files

4.2.1 Scenario 1

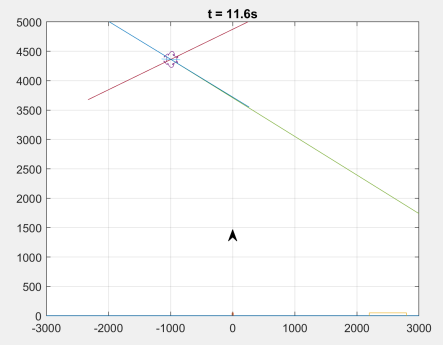

Figure 4.2.2: During the flight of the rocket

During it's flight, the rocket is staying upright, not turning and is flying straight upwards as directed.

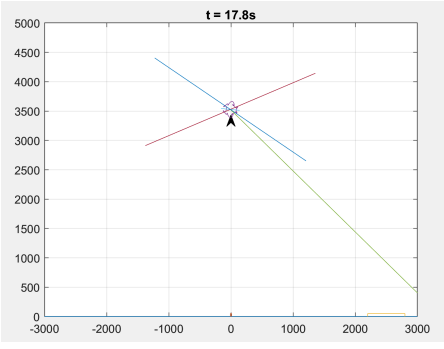

Figure 4.2.3: Strike!

The rocket hits the asteroid directly above the rocket's starting point, and was successful in it's mission!

4.2.2 Scenario 2

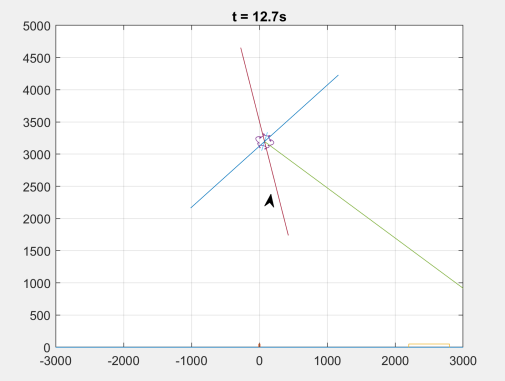

Figure 4.2.4: During the flight of the rocket

We can see that mid-flight, the rocket is tracking an angle as it has flown to the right.

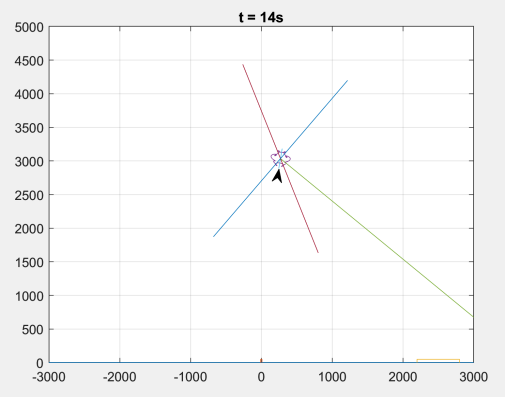

Figure 4.2.5: Strike!

The rocket has found the asteroid and hit it, while changing the thrust angle, α.

4.2.3 Scenario 3

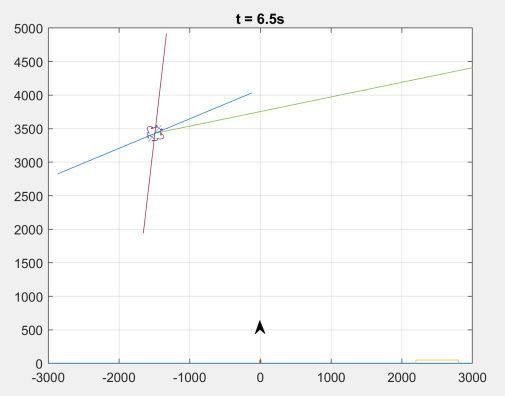

Figure 4.2.6: During the flight of the rocket

Here, it is visible that the rocket is starting to depart from the centre line, towards the left of the screen, as the asteroid is rotating on it's axis (blue line).

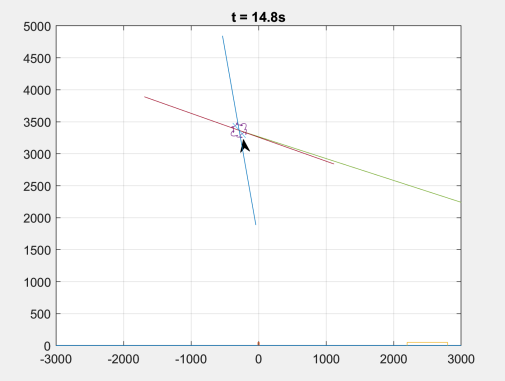

Figure 4.2.7: Strike!

The area between the red line and the asteroid's axis (blue) is indicative of the asteroid's arc in which it needs to be hit, and this is where the rocket finds itself!

It can be concluded after examining the figures above in line with the scenario objectives, that the goals were all met, and the rocket's controller design was successful.

References

[1] E. Mechatronics, "Milestone 2 brief." https://amathuba.uct.ac.za/content/enforced/14446-EEE4119F_2023/Project/EEE4119F_Project_MileStone2_Brief.pdf.

[2] E. Mechatronics, "Milestone 3 brief." https://amathuba.uct.ac.za/content/enforced/14446-EEE4119F_2023/Project/EEE4119F_Project_MileStone3_Brief.pdf.

[3] R. Fitzpatrick, "Projectile motion with air resistance." https://farside.ph.utexas.edu/teaching/336k/Newton/node29.html, Mar. 2011.

5 Appendix 1

5.1 MATLAB Code for the Controller

```
function [F, alpha, detonate] = fcn(y_rocket, dy_rocket, th,x_rocket,
   x_ast, y_ast, scenario, ast_dx, ast_dy, ast_th, ast_dth)
    persistent xinit_r xinit_ast yinit_ast dy_init_r dxinit_ast
       dyinit_ast ast_th_0 ast_dth_0

    c= 100;%approximate value given in milestone 1
    %Note: my value was c=146 and also works. Using c=100 to be safest.
    %Feel free to swap for c=146 if more appropriate
    mr = 1000;
    ma = 10000;
    g = 9.81;

    %initialise output variables
    F = 0;
    alpha = 0;
    detonate = 0;

    %storing initial values as persistent variables for calculations
    if isempty (xinit_r)
        xinit_r =x_rocket;
        %yinit = y_rocket;
        xinit_ast = x_ast;
        yinit_ast = y_ast;
        dyinit_ast = ast_dy;
        dxinit_ast = ast_dx;
        dy_init_r = dy_rocket;
        ast_th_0 = ast_th;
        ast_dth_0 = ast_dth;
    end

    %Scenario 1
    if scenario ==1
        % Use eqns from website
        % https://farside.ph.utexas.edu/teaching/336k/Newton/node29.html
        % find Vt of asteroid
        vt = ma*g/c;

        %calculate initial x distance between objects
        x = xinit_r-xinit_ast;

        % rearrange eqn 189 to find the time to location
        t = (-vt/g)*log(1-(x*g)/(dxinit_ast*vt));

        %y posn at time
        y_target = (vt/g)*(dyinit_ast+vt)*(1-exp((-g*t)/vt))-vt*t+
           yinit_ast;

        %finding accel of rocket in y to reach y position in given time
```

```
46          accel_y = (y_target-dy_init_r*t)*2/(t^2);
47
48          %f=ma
49          F =mr*(accel_y+g); %account for gravity effects
50          alpha = 0; %because straight up
51      end
52
53      if scenario ==2
54          %need an x to aim for initially (chosen randomly, but needs to
                be
55          % reltaively small for angle adjustments to be made
                appropriately):
56          x = 5;
57
58          %same process as scenario 1 until accel
59          % find Vt of asteroid
60          vt = ma*g/c;
61
62          %calculate initial x distance between objects
63          x = xinit_r-xinit_ast;
64
65          % rearrange eqn 189 to find the time to location
66          t = (-vt/g)*log(1-(x*g)/(xinit_ast*vt));
67
68          %y posn at time
69          y_target = (vt/g)*(dyinit_ast+vt)*(1-exp((-g*t)/vt))-vt*t+
                yinit_ast;
70
71          %finding accel of rocket in y to reach y position in given time
72          accel_y = (y_target-dy_init_r*t)*2/(t^2);
73
74          %calculate intial angle between objects:
75          th_init = atan(y_target/(xinit_ast-xinit_r));
76          correct_th = th-th_init;
77
78          %direction to apply
79          alpha = 0.001*correct_th; %alpha is a ratio of the correction
                angle
80          %gain factor of 0.001 was found using trial and error. 1 was too
                large
81          %and 0.0001 was too small - both caused erraticism
82
83          %Newtonian dynamics and account for gravity
84          F = mr*(accel_y+g);
85
86          %need distance between asteroid and rocket using euclidian geom
                to
87          %check if detonation needed
88          dist = sqrt((x_ast-x_rocket)^2 + (y_ast-y_rocket)^2);
89          if dist <= 150
90              detonate = 1;
91          end
```

```
92      end
93
94
95
96      if scenario ==3
97          %solve trajectory:done in define_constants.m
98          %th_desired = pi/3; %60 degrees
99      %     dy = ast_dy;
100     %     ode = ddy == -c*dy/ma -g;
101     %     cond = dy(0) == dyinit_ast;
102     %     dy_traj = dsolve(ode, cond);
103     %     y_pos = sum(dy_traj)*dt + yinit_ast;
104
105
106         %angular motion kinematics
107         %find angular acceleration of asteroid
108         ast_ddth = (ast_dth^2-ast_dth_0^2)/(2*(ast_th-ast_th_0));
109
110         %time array and theta array
111         TARR = 0:0.01:30;
112         thARR = zeros(size(TARR));
113
114
115         %find time to get to th= between 45 and 60degreees
116         ind =0;
117         for i=1:3000 %size of TARR
118             thARR(i) = ast_th_0 + ast_dth_0*TARR(i) + 0.5*ast_ddth*(TARR
                    (i))^2; %th = th0+w0*t+1/2at^2
119             if thARR(i)<pi/3 %has to be less than 60 degrees
120                 if thARR(i)>(pi/4) %has to be greater than 45 degrees
121                     ind = i;
122                     th = thARR(i);
123                     break;
124                 end
125             end
126         end
127         t = ind*0.01;
128         %finding y position at the time
129         dt = 0.01;
130         y_pos = yinit_ast - (dt*(g*ma - exp(-(c*t)/ma)*(c*dyinit_ast +g*
                ma)))/c;
131
132
133         %use the the remaining equations from scenario 2
134         %finding accel of rocket in y to reach y position in given time
135         accel_y = (y_pos-dy_init_r*t)*2/(t^2);
136
137         %calculate intial angle between objects:
138         %th_init = atan(y_pos/(xinit_ast-xinit_r));
139         correct_th = th-ast_th_0; %use the theta found
140
141         %direction to apply
```

```
        alpha = -0.001*correct_th; %alpha is a ratio of the correction
            angle
        %gain factor of 0.001 was found using trial and error. 1 was too
            large
        %and 0.0001 was too small - both caused erraticism

        %Newtonian dynamics and account for gravity
        F = mr*(accel_y+g);

        %need distance between asteroid and rocket using euclidian geom
            to
        %check if detonation needed
        dist = sqrt((x_ast-x_rocket)^2 + (y_ast-y_rocket)^2);
        if dist <= 150
            detonate = 1;
      end

    end
```

5.2 Full Modelling Code

```
    %variables
syms th dth ddth alph x dx ddx y dy ddy F
%generalised coordinates
q = [th; x; y];
dq = [dth; dx; dy];
ddq = [ddth; ddx; ddy];
%constants
syms g;

%force components (in frame 1 = body) for thrust
Fv = F*cos(alph);
Fh = -F*sin(alph);

%rocket parameters
syms mR L1 L2 d %m= 1000kg;L1 = 5m; L2 = 15m; %dimensions %d from COM =
    3.5m
%x0 = 0; y0 = 0; d =3.5; L1 = 5; L2 = 15; mR = 1000;
syms IxR IyR IzR %inertias not needed yet
I_R = diag([IxR IyR IzR]); %inertia tensor

%Rotations
R01 = RotZ(th);
R10 = transpose(R01);

%Positions
r_1 = [x;y;0];
r_0 = r_1; %rotation correct?

Fvector_1 = [Fh;Fv;0];
Fvector_0 = R10*Fvector_1;
```

```
forcepos = [0;-d;0]; %force position in body frame
forceP = R10*forcepos+ r_0; %force position in inertial frame

%Velocities
vR = jacobian(r_0, q)*dq;
wR = [0;0;dth];

%Kinetic Energy
Tr = 0.5*mR*transpose(vR)*vR + 0.5*transpose(wR)*I_R*wR;
TtotalR = simplify(Tr);
Potential Energy
Vr = mR*g*r_0(2);
Vtot = simplify(Vr);
Mass Matrix & derivative
M = hessian(TtotalR, dq);
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
G = jacobian(Vtot,q);
G = simplify(G);

%Define Coriolis Matrix
C = dM*dq - transpose(jacobian(TtotalR,q));
C = simplify(C);

%Generalised Forces
%partial derivatives
partTh = diff(forceP, th);
partX = diff(forceP, x);
partY = diff(forceP, y);

%Gen force components dot products
Qth = Fvector_0(1)*partTh(1) + Fvector_0(2)*partTh(2) + Fvector_0(3)*
   partTh(3);
Qx = Fvector_0(1)*partX(1) + Fvector_0(2)*partX(2) + Fvector_0(3)*partX
   (3);
Qy = Fvector_0(1)*partY(1) + Fvector_0(2)*partY(2) + Fvector_0(3)*partY
   (3);
Q = [Qth; Qx; Qy];
Q = simplify(Q);

%Manipulator Equation FINAL ANSWER
%IzR = 36833.333 as calculated using L1, L2, d and MR using
   perpendicular
%axis theorem
%IzR = (1/12) * mR * (L1^2 + L2^2) + mR*(L2/2-d)^2;
```

```
% calculated to be = 36833.333333333333

ManipEqn = M*ddq + C + transpose(G) == Q; %Manipulator Equation
ManipEqn = simplify(ManipEqn);

disp('Manipulator Equation vector')
disp(ManipEqn) %display the manipulator equation
sympref('FloatingPointOutput',true); %so we can see the floats rather
    than fractions
var = subs(ManipEqn,[IzR,d,g,mR],[36833.33, 3.5, 9.81, 1000])
%disp("We know the values of L1, L2, d and mR and they are constants")
%to ensure it is the same: outputting the calculated value of IzR:
Iz = (1/12)*1000*(15^2+5^2)+1000*(15/2-3.5)^2;
fprintf("To confirm the above is correct: Iz = %f", Iz)
disp("Therefore the euqations match (symbolic = with floats)");

%ASTEROID: CALCULATION OF C VALUE

m = 10000;
g = 9.81;

ast_ddx = diff(ast_dx)/0.001;
ast_ddy = diff(ast_dy)/0.001;

len = length(ast_ddx);
c = zeros(len-1,1);
cy = zeros(len-1,1);
sum = 0;
sumy = 0;
for i = 2:1:len %start at 2 to eliminate the first ast_dy element =0
    c(i) = ast_ddx(i)*m/ast_dx(i); %x component
    sum=sum+c(i);
    cy(i) = (ast_ddy(i)+g)*m/ast_dy(i);
    sumy= sumy+cy(i);
end
cx = -sum/(len-1) %average of the drag x component
cy = sumy/(len-1) %average of the drag y component
cfinal = (cx+cy)/2
fprintf("Final/Resultant drag coefficient, c = %f", cfinal)

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