2020

EEE4119F - Project

MECHATRONICS 2

JAMIE ARONSON – ARNJAM004

Table of Contents

Introduction

For this project, students were tasked with saving a city by utilizing a rocket to intercept an asteroid. In order to do so, both the rocket and asteroid dynamics need to be modelled. Once this is done, an appropriate method of control can be used to make sure that the rocket correctly intercepts the asteroid based on known dynamics.

There were two scenarios to be undertaken, as well as an additional optional scenario. For the first case, the rocket is able to fly straight up and meet the asteroid on its trajectory, however in the second case, this is not possible and the rocket must use some form on control to track the trajectory of the asteroid and turn accordingly. For the third and optional scenario, the rocket needs to impact the asteroid at an angle due to inherent weaknesses in the structure of the asteroid, thus ensuring that the asteroid is destroyed.

Modelling the rocket dynamics

The rocket that we have been given can be approximated by a two-dimensional rectangle with dimensions $L = 15m$ and $W = 5m$. The centre of mass of the rocket was given as $(x; y) = (0; 3.5)$. The rocket is actuated by an input force F which acts at the base of the rocket. The force can be angled by

$\pi/2 > alpha > -\pi/2.$

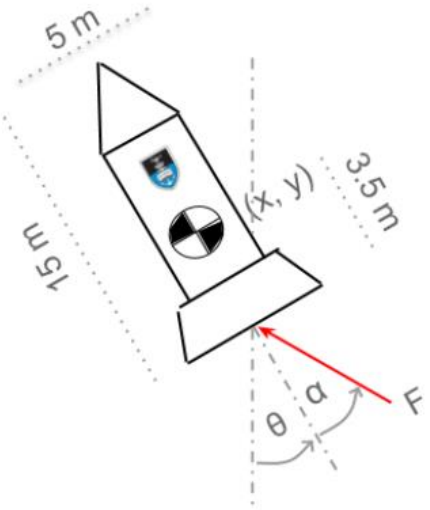


Figure 1.1: Rocket model

A generalized coordinate system was used for this system. In order to fully describe the dynamics of the system, the generalized coordinates were chosen as $q = (x, y, theta)$, $dq = (dx, dy, dtheta)$ and $ddq = (ddx, ddy, ddtheta)$.

The technique used in order to find the equations of motion of the rocket involved the use of the operator matrices. This equation states that $M + C + G = B * u$ where M is the mass matrix, C is the velocity-product matrix, G is the gravity term and $B * u$ is describing the input to the system.

In order to calculate the mass matrix, the kinetic energy of the rocket needed to be calculated.

```
%kinetic energy of rocket
Tt = (1/2)*mass*transpose(v_body)*v_body;
Tr = (1/2)*inertia*dth^2;
Ttot = simplify(Tt + Tr);
```

Figure 1.2 Kinetic energy calculation using rotational and tangential kinetic energy

The rocket is both moving tangentially direction and rotating, so it has both rotational and tangential kinetic energies. The inertia of the rocket is given as:

$$Inertia = \frac{1}{12} * mass(W^2 + D^2) + mass * d^2$$

Where W is the width of the rectangle, D is the height and d is the distance from the geometric centre of mass to the actual centre of mass.

In order to find the mass matrix, the hessian of $Ttot$ is taken with respect to dq.

The $C\ and\ G$ matrices are found as follows where dM is the derivative of mass matrix M.

```
%C Matrix
C = dM * dq - jacobian(Ttot, q)';
C = simplify(C);
%G Matrix
G = transpose(simplify(jacobian(Vtot,q)));
```

Figure 1.3: Operator Matrices calculations

In order to find the B matrix, the input force to the system needs to be generalized. It can be said that the input vector is $[F; alpha]$ however, this can be simplified for now into two pseudo forces $[F1; F2]$ being a force in the X and Y directions. These forces, and the point relative to the centre of mass that they operate, are both rotated into the inertial frame. The generalized forces Q, and thus B matrices are then found as follows:

```
%Generalized Force
R_th = [ cos(th) -sin(th)
    sin(th)  cos(th)];
r1 = [x; y] + R_th * [ 0; -3.5];
Q = simplify((R_th*[F1;F2])'*jacobian(r1,q))';
B = jacobian(Q, [F1;F2])
```

Figure 1.4: Generalized forces

Based on the above methodology, the dynamics of the rocket were found to be:

```
acceleration_eqns =
```

$$\begin{pmatrix} \frac{F_1 \cos(\text{th}) - F_2 \sin(\text{th})}{\text{mass}} \\ \frac{100\,F_2 \cos(\text{th}) - 981\,\text{mass} + 100\,F_1 \sin(\text{th})}{100\,\text{mass}} \\ \frac{7\,F_1}{2\,\text{inertia}} \end{pmatrix}$$

Figure 1.5: Acceleration Equations in x, y and theta directions

Modelling the asteroid dynamics

In modelling the asteroid, it was important to consider the drag model. This came in the form of:

$$F_{drag} \propto c * \sqrt{dx_{ast}^2 + dy_{ast}^2} \ldots (1)$$

$$ddx_{ast} = \frac{F_{drag_x}}{m} + noise_x \ldots (2)$$

$$ddy_{ast} = \frac{F_{drag_y}}{m} - g + noise_y \ldots (3)$$

$$ddtheta_{ast} = noise_{theta} \ldots (4)$$

The asteroid dynamics given to us, when run, produced dx_{ast}, dy_{ast} and $dtheta_{ast}$. By differentiating these signals and applying a low-pass filter in Simulink, a relatively accurate signal corresponding to the accelerations in the various directions was achieved by attenuating the high frequency noise.

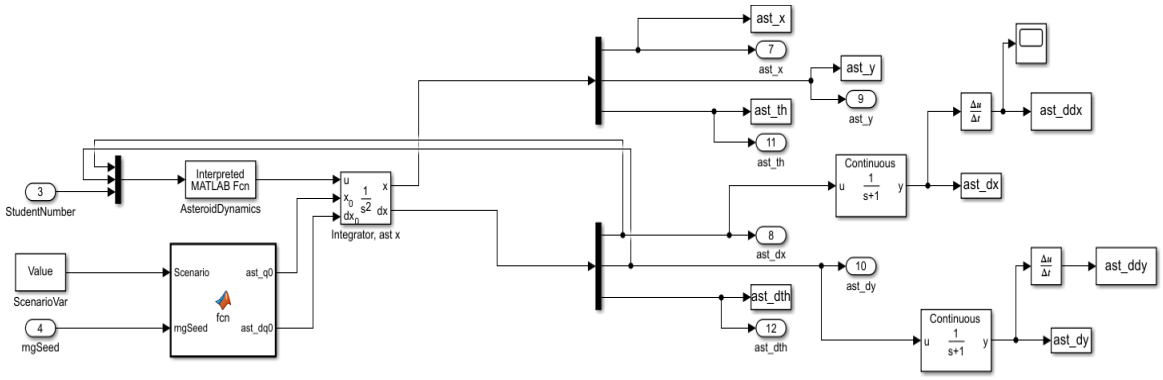


Figure 2.1: Simulink Diagram showing filtering and differentiation of dx and dy to find ddx and ddy

With these values of acceleration, ddx_{ast} and ddy_{ast}, and some rearranging of equations 2 and 3, a column vector for F_{drag_x} and F_{drag_y} was found. Simply taking $\sqrt{F_{drag_x}^2 + F_{drag_y}^2}$ gives a column of values for F_{drag}. Now rearranging equation 1, using values in the workspace, we can find a column vector for c. Taking the mean of these values gave the result $c = 118.54$

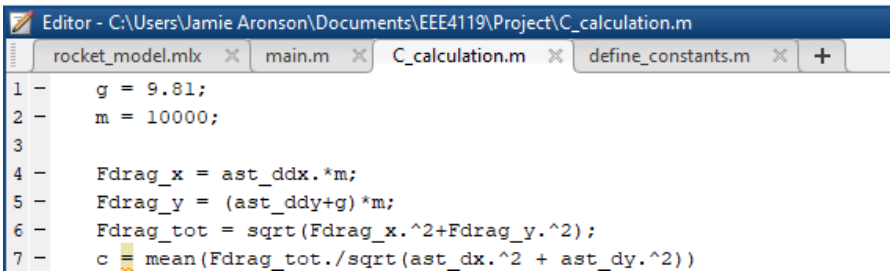


Figure 2.2: Calculated of C based on F_drag, dx_ast and dy_ast

With c, and using equations 2,3 and 4, the dynamics of the asteroid can be successfully modelled.

Controller design

The type of control method used to operate the rocket was proportional navigation which is a guidance law used in most homing air target missiles. The law states that two objects are on a collision course when their direct line of sight does not change as the range closes.

Proportional navigation can be achieved using an acceleration normal to the instantaneous velocity difference

$$\vec{a} = N * \overrightarrow{V_r} \times \vec{\Omega}$$

Where $\vec{\Omega}$ is the rotation vector of the line of sight.

$$\vec{\Omega} = {\vec{R} \times \overrightarrow{V_r}}/{\vec{R} \cdot \vec{R}}$$

Where $\overrightarrow{V_r} = \overrightarrow{V_t} - \overrightarrow{V_m}$ is the relative difference in velocity between the target and the 'missile'.

And $\vec{R} = \overrightarrow{R_t} - \overrightarrow{R_m}$ is the range from the 'missile' to target, or the difference in position.

This method gives us a correct value to track, however full-state feedback method will be utilized in order to achieve this. For this method to work, it requires that all n states are measurable.

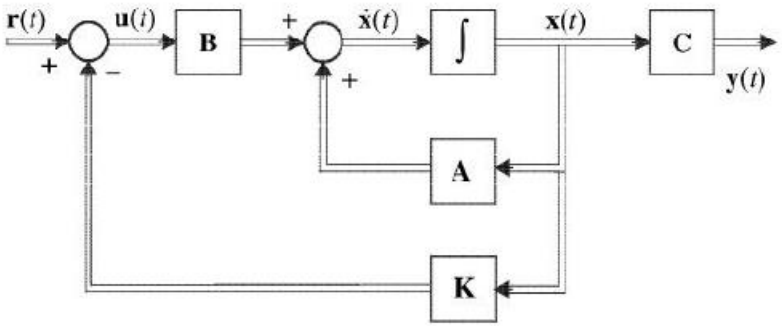


Figure 3.1: Full state-feedback block diagram

Next, a suitable K matrix needs to be found. This is where the process of linearizing the system comes in. In order to find such a matrix, a method known as LQR is utilized. This method provides optimally controlled feedback gains to enable the closed loop stable and high-performance design of systems.

It is also worth noting that the output of the controller is $[F1; F2]$, two forces, one in the x-direction and one in the y-direction, however the inputs to the system dynamics are $[F; alpha]$. In order to convert back to this, we take $F = \sqrt{F_1^2 + F_2^2}$ and $alpha = atan2(F1, F2)$ and feed these values back into the dynamics block.

Controller Implementation

Full-state feedback was successfully implemented in the Simulink mode in conjunction with a proportional navigation controller.

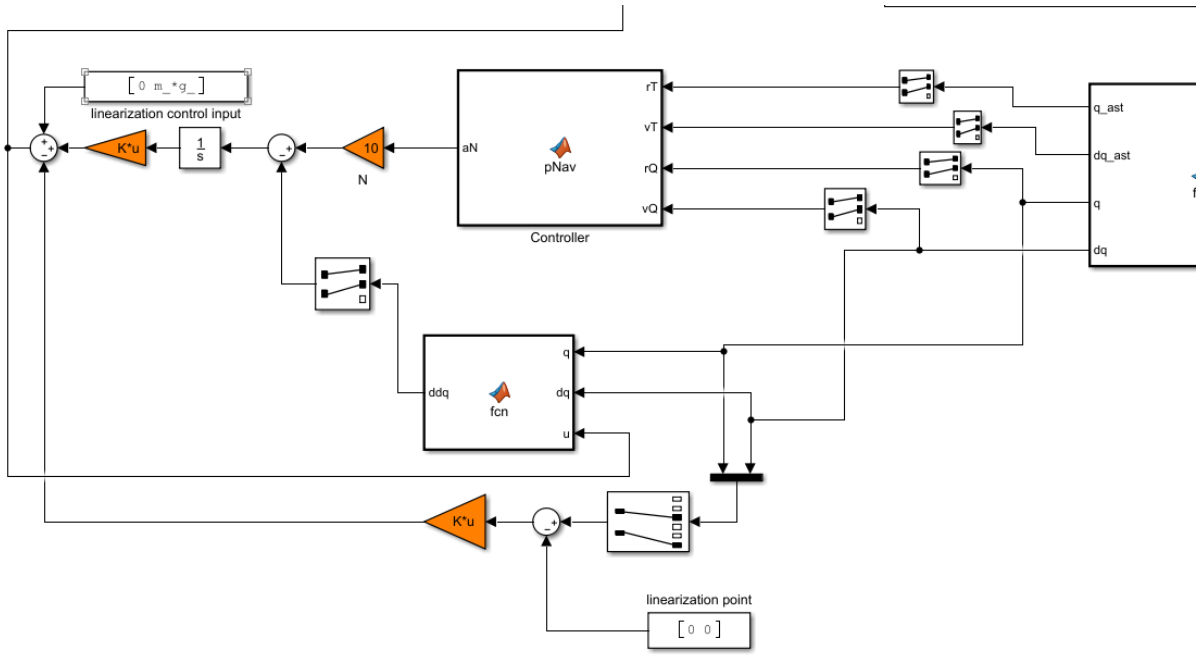


Figure 3.2: Simulink model showing full-state feedback implementation

```
function aN = pNav(rT, vT, rQ, vQ)
% this produces an acceleration which the controller should track
R = rT - rQ;
Rtemp = [R; 0];
Vr = vT - vQ;
Vtemp = [Vr; 0];
omega = cross(Rtemp, Vtemp) / dot(Rtemp, Rtemp);
temp = cross(Vtemp, omega);
aN = temp(1:2);
```

Figure 3.3: Code snippet showing proportional navigation implementation

Linearizing the system

In this undertaking, the rocket is the body we are trying to control. From its dynamics, we can see that the rocket has a non-linear model. We could employ some or other non-linear control method, however we have decided that a simpler task would be to linearize the system.

In order to linearize the system, an appropriate operating point needs to be chosen. This is $x = dx = y = dy = th = dth = 0$ and a thrust that is purely in the y-direction in order to balance the weight of the rocket so that forces are balanced, which is the equilibrium condition.

Firstly, the acceleration equations are represented in a state-space equation form. There are three variables of interest for our state space model, which corresponds to a total of 6 states. These are the first and second derivatives of $x, y\ and\ theta$:

```
dyn = [acceleration_eqns; dq]
```

```
dyn =
```

$$\begin{pmatrix} \frac{F_1\cos(\mathrm{th}) - F_2\sin(\mathrm{th})}{\mathrm{mass}} \\ \frac{100\,F_2\cos(\mathrm{th}) - 981\,\mathrm{mass} + 100\,F_1\sin(\mathrm{th})}{100\,\mathrm{mass}} \\ \frac{7\,F_1}{2\,\mathrm{inertia}} \\ \mathrm{dx} \\ \mathrm{dy} \\ \mathrm{dth} \end{pmatrix}$$

Figure 4.1: State-space model

Based on these states, we can now find the $A, B, C\ and\ D$ matrices of the state space model. It is important to keep in mind that we are controlling $ddx\ and\ ddy$, so these are the outputs.

Find A, B, C, D for state space equations:

```
A = jacobian(dyn, [dth, th]);
A = subs(A,[dth, th, u.'],[0,0,0,m_*g_])

B = jacobian(dyn, [u.']);
B = subs(B,[dth, th, u.'],[0,0,0,m_*g_]);

% this time weed to linearise the output equation
C = jacobian(Y, [dth, th]);
C = subs(C,[dth, th, u.'],[0,0,0,m_*g_]);

% and linearise feedforward
D = jacobian(Y, [u.']);
D = subs(D,[dth, th, u.'],[0,0,0,m_*g_]);
```

As can be seen here, we are solving for our $A, B, C\ and\ D$ matrices. Y is the output equation which is characterized by $ddx\ and\ ddy\ (acceleration\_eqns(1{:}2))$. Each of these matrices is then linearized around the chosen points and hovering condition.

Figure 4.2: Matrix Calculations

Next, the Jacobian is augmented with integral states for acceleration. The result are workable matrices that enable us to find a suitable feedback gain matrix, K.

```
newA = [0, -g_, 0, 0;...
        0,  0, 0, 0];
tempA = horzcat(A, zeros(2,2));

% New A Matrix
A_aug = vertcat(tempA, newA);

% New B Matrix
B_aug = vertcat(B, D);

% New C Matrix
newC = [0, 0, 0, 0;...
        0, 0, 0, 0];
tempC = horzcat(C,zeros(2,2));
C_aug = vertcat(tempC, newC);

% New D Matrix - STAYS THE SAME
D_aug = vertcat(D, zeros(2,2));

% get numerical versions (right now they're symbolic)
A = double(A_aug);
B = double(B_aug);
C = double(C_aug);
D = double(D_aug);
```

```
sys_ss =

  A =
         dth      th    ddxE    ddyE
  dth      0       0       0       0
  th       1       0       0       0
  ddxE     0   -9.81       0       0
  ddyE     0       0       0       0

  B =
              u1          u2
  dth   9.502e-05           0
  th            0           0
  ddxE      0.001           0
  ddyE          0       0.001

  C =
         dth      th    ddxE    ddyE
  ddx      0   -9.81       0       0
  ddy      0       0       0       0
  ddxE     0       0       0       0
  ddyE     0       0       0       0

  D =
           u1      u2
  ddx   0.001       0
  ddy       0   0.001
  ddxE      0       0
  ddyE      0       0
```

Figure 4.2: Augmented State-Space Matrix Calculation

The LQR (Linear Quadratic Regulator) algorithm is then implemented which determines the optimal gain K to minimise a linear system using

$$J(u) = \int_0^\infty (x^T Q x + u^t R u) dt$$

Where Q is the penalisation on state and R is the penalisation on control action. Both are diagonal matrices.

Using MATLAB's built in LQR function and choosing state penalization values, an optimal K matrix was computed. These values were chosen based on trial and error by observing the simulated outcome.

```
% LQR - regulator/stabilising states
Q = eye(4);
Q(1,1) = 20;
Q(2,2) = 20;
Q(3,3) = 10;
%for Q(4,4) if scenario 1 -> = 1000, if scenario 2, change to 10000
Q(4,4) = 1000;
R = eye(2);
% calculate LQR gain
K = lqr(A, B, Q, R)
```

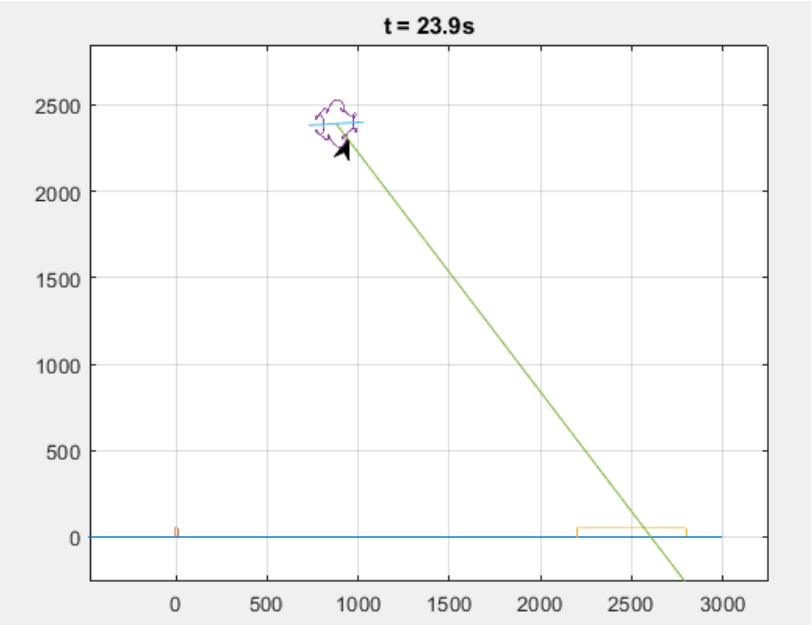
```
K = 2×4
10^3 ×
    3.0625    0.4359   -0.0032    0.0000
    0.0000    0.0000   -0.0000    0.0316
```

Figure4.2: Calculated K gain matrix

Results

Scenario 1

It is required that the rocket detonate within 150m of the asteroid. The parameter was set on the Simulink block and then main.m was run. If the scenario is successful, a logic 1 is printed, otherwise a logic 0 is printed.



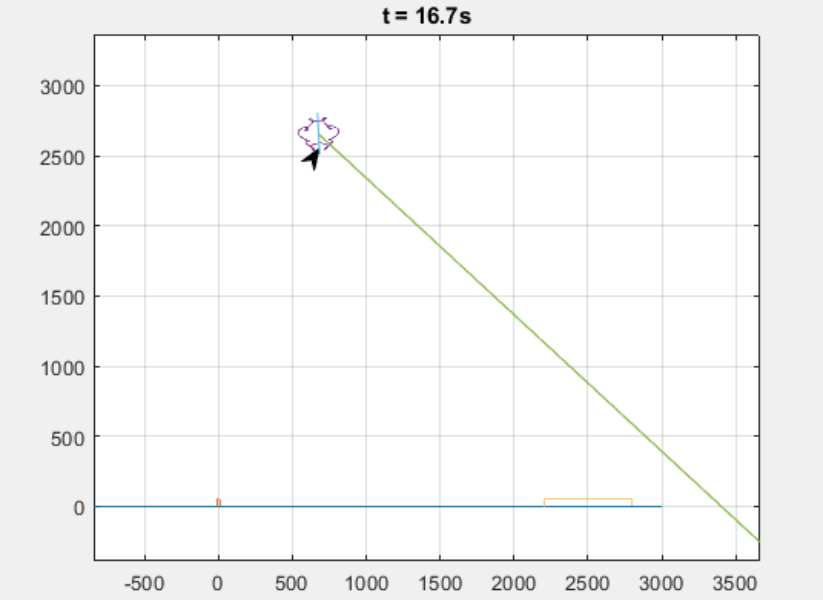

```
Mission complete: well done! Make sure to test with multiple seeds

ans =

  logical

   1
```

Figure 5.1: Simulation of scenario 1 showing a successful attempt

The process was run several times for several random seeds with a success rate of 97%. Despite the condition being allowed to simply meet the asteroid on its trajectory straight above it, the same kind of controller was used as scenario 2 which resulted in the rocket turning.

Scenario 2

In order to achieve a successful scenario 2, the penalization on the states must be changed. $Q(4,4)$ was changed from $1000\ to\ 10000$. The Simulink block parameter was set to 2, a new K matrix was found and then main.m was run.




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

ans =

  logical

   1
```

Figure 5.1: Simulation of scenario 2

Again, the process was repeated several times with several seeds. The success rate for this scenario was 94%. The effect of increasing Q (4,4) from 1000 to 10000 resulted in a faster response which was necessary to intercept the asteroid in scenario 2 due to it starting at a lower y-value with a faster velocity.

Conclusion

In conclusion to this project, it can be said that the requirements for scenarios 1 and 2 were partially satisfied, since a different K matrix, and hence a slightly different controller, was implemented in order to satisfy requirements. However, within this compromise, both requirements were fully satisfied.

Scenario 3 was not implemented due to time constraints and complexity however, conceptual thought was given to the possible implementations of the scenario.

Bibliography

Alexis, D. K. (n.d.). *LQR Control*. Retrieved from Dr. Kostas Alexis: http://www.kostasalexis.com/lqr-control.html

N.A. (n.d.). *Full State Feedbackfor State Space Approach*. Retrieved from mercer.edu: http://faculty.mercer.edu/jenkins_he/documents/FullStateFeedbackegr386.pdf

Proportional navigation. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Proportional_navigation