Student Number: SLYTAR001

U N I V E R S I T Y   O F   C A P E   T O W N

Department of Electrical Engineering

EEE4119F – Nonlinear Control of Mechanical Systems
Asteroid Project Report

Tareeq Saley

13-06-2020

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretending 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 of as their own work or part thereof.

Name: Tareeq Saley
Student Number: SLYTAR001
Date: 13/06/2020

Executive Summary

This report concerns an investigation into the position and acceleration control of a rocket control simulation system. The report is developed through the modelling, enchancement and simulation of a plant system and goes on to identify a controller to be able to control that system efficiently. The reports controller implementation section is based on a demonstration of the physical solution that is to be conducted by the TA.

Contents

1. Introduction

This report focuses on a project that relates to designing a controller to intercept an asteroid hurtling towards a city on Earth. A simulation was developed using Matlab and Simulink to achieve this.

2. Specifications

The requirements set out for the successful interception of the asteroid is listed below:

- The output range of the thrust force, F, is $0 < F < Fmax$
- The output range of the thrust angle, α, is $-\frac{\pi}{2} < \alpha < \frac{\pi}{2}$
- The rocket must detonate within 150 meters of the center of mass of the asteroid, before the center of mass of the asteroid falls to within 200 meters of the city, measured as height above ground

Additionally;

- The rocket should strike the asteroid in the range $-30 \leq \psi \leq 30$ or $150 \leq \psi \leq 210$, where ψ is the angle between the rocket and the asteroid
- The distance between the asteroid and city should be maximized at the time of detonation
- The rocket should detonate in between the asteroid and the city, so that any remaining shrapnell is pushed away from the city

The following specifications has been developed based on the above requirements and the respective scenarios:

- A stable, settled response
  It is possible to have the rocket intercept the asteroid along its trajectory and somewhere in between its anticipated response. However, emphasis was placed especially for position control to have the rocket intercept the asteroid after it has settled. This allows for a more robust controller and leads to easier and more refined controller tuning.
- Overshoot < 20%
  The reason for the relatively high allowance of overshoot, especially for position control, is so that the rocket 'hangs' in the air over a range of values rather than settling at a specific value. The noise in the asteroid system can't allow for completely predicatable trajectories, therefore causing it to possibly hit the city at a value offset by ±20%.
- Settling time < time it takes for rocket to pass the same point
  For the case where the rocket should settle at a specific positition and 'wait' for the rocket to the same position so that it can intercept it, the time it takes to reach that position should be less than the time it takes for the asteroid to pass the same position. This specification is optional and might not always be achieved, it would however allow for a more robust controller to be designed.
- Minimal Steady State Error
  Due to the fact to the asteroid and rocket being non-linear and to a certain extent unpredicatable. In order for the rocket to therefore strike the asteroid successfully whilst catering for the the instability and noise that comes with it, accuracy is imperitave. Steady state error should therefore be as small as possible to cater for this.

3. Model Identification

The ‘C’ Parameter

In order for the value of ‘c’ to be calculated, the given equations of motion for the asteroid need to be solved as ODEs so that the known data values from the simulation can be substituted into these equations in order to solve for c.

To ensure that the additive process noise contribution of the asteroid’s equations of motion does not have a significant effect on the accuracy of the estimated c value, this noise initially has to be filtered out and the data smoothened. The equation which relates to the motion in the x-direction was used for calculation purposes. The graph of the filtered data and unfiltered data as well as the code used for the filter can be found in Appendix A and Appendix B respectively.

Considering the filtered asteroid equation of motion in the y-direction and replacing the drag force with its equivalent representation in terms of the drag co-effiecient ‘c’, we have,

$$\ddot{x} = \frac{Fdrag_x}{m}$$

$$\therefore \ddot{x} = \frac{cv_x}{m}$$

Solving the above ODE to find a representation for y’ we have,

$$\frac{d\dot{x}}{dt} = \frac{c\dot{x}}{m}$$

This can then be rearranged as ,

$$\frac{md\dot{x}}{c\dot{x}} = dt$$

Finally, by integrating using the u-substitution technique and solving further we get,

$$\dot{x} = \dot{x}_o e^{\frac{-ct}{m}}$$

After running a single simulation, the simulation data in the Matlab workpace could be used to find $\dot{x}$ and t values for any point in the simulation. With the mass of the asteroid $m = 10000kg$ and the initial x-velocity of the asteroid $\dot{x}_o = 182\ m/s$, the value of c could be calculated for every simulation step. The cumulative total of all of these c values could then be divided by the number of steps to get an average value of c. For accuracy purposes and for smoother data, the step size was set at 0.1s. The corresponding code related to solving for c can be found in the appendix.

$$c = 97.71$$

(Note that a smaller step size could not be supported by my laptop. My laptop would freeze every time I tried to reduce the step size futher, unfortunately this led to slightly different c values for every simulation that was run).

Equations of Motion for the Rocket

The rocket’s motion has 3 degrees of freedom, it can move in both the x and y directions and can also rotate about its own axis. Therefore, the chosen generalized co-ordinates and position vector using the center of mass as the origin of the body axis are given below,

$$q = [x, y, \theta]^T$$

$$r = [x, y]^T$$

The total kinetic (translational and rotational) and potential energies of the rocket can then be given by,

$$T_{total} = \frac{1}{2} m\dot{r}^T\dot{r} + \frac{1}{2} I_{cm}\dot{\theta}^2$$
$$V_g = m[0, g]^T r$$

The following manipulator equation is then developed using the kinetic and potential energies with $\ddot{q}$ being the vector representing the desired equations of motion

$$M\ddot{q} + C + G = Q$$

With ,

$$M_{i,j} = \frac{\partial T}{\partial \dot{q}_i \partial \dot{q}_j}$$
$$C = \dot{M}\dot{q} - \frac{\partial T}{\partial q}$$
$$G = \frac{\partial V}{\partial q}$$

The generalized force vector, Q, for the thrust force which acts on the rocket now needs to be calculated. Since the rockets motion is 2-dimensional across the x and y planes, the following rotation matrix is used

$$R_1^0 = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

The thrust force can be broken down into its respective x and y components in order to decouple to the inputs when in state space modeling. The force acts at a distance of 3.5m below the center of mass. The position vector of the force with respect to inertial is then,

$$r_{Fx}^0 = \begin{bmatrix} x \\ y \end{bmatrix} + R_1^0 \begin{bmatrix} 0 \\ -3.5 \end{bmatrix}$$
$$r_{Fy}^0 = \begin{bmatrix} x \\ y \end{bmatrix} + R_1^0 \begin{bmatrix} 0 \\ -3.5 \end{bmatrix}$$

And the input forces with respect to inertia are,

$$F_x^0 = R_1^0 \begin{bmatrix} -F_1 \\ 0 \end{bmatrix}$$
$$F_y^0 = R_1^0 \begin{bmatrix} 0 \\ F_2 \end{bmatrix}$$

The generalized force vector is then given by

$$Q = \begin{bmatrix} Qx \\ Qy \\ Q\theta \end{bmatrix} = \begin{bmatrix} -F_1cos\theta - F_2sin\theta \\ F_2cos\theta - F_1sin\theta \\ -3.5F_1 \end{bmatrix}$$

By substituting back into the manipulator equation and solving for $\ddot{q}$, the equations of motion are:

$$\ddot{q} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{-F_1cos\theta - F_2sin\theta}{m_{rocket}} \\ \frac{F_2cos\theta - F_1sin\theta - m_{rocket}g}{m_{rocket}} \\ \frac{-3.5F_1}{I_{cm}} \end{bmatrix}$$

With $m_{rocket} = 1000kg$ and I_{cm} calculated below.

The Moment of Inertia of the Rocket

The rocket can be approximated as a box. The moment of inertia, using the parallel axis theorem is for a box is given by:

$$I_{cm} = \frac{1}{12}mh^2 + md^2$$

Where h is the height of the rocket and d is the distance between the center of mass of the rocket and its geometric center. With $m = 1000kg$, $h = 15m$ and $d = 7.5 - 3.5 = 4m$ we find that

$$I_{cm} = 34750\ kg.m^2$$

The Maximum Applied Force of the Rocket

To find the maximum force, the acceleration value at which the rocket saturates is required. The following graph details the rockets acceleration after the model was simulated.

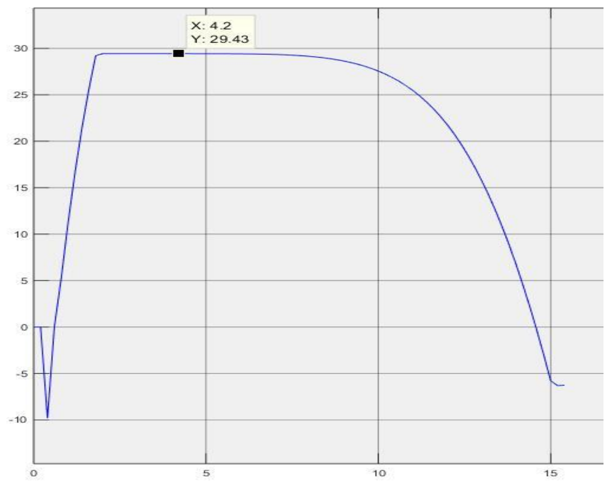


Figure 1: Graph depicitng saturation of acceleration in the y direction

From the above figure which displays the acceleration in the y direction of the rocket its clear to see that the maximum achievable acceleration is $29.43\frac{m}{s^2}$. The maximum thrust force which corresponds to this value can be found using Newtons 2nd Law where,

$$ma = F_{sat} - mg$$

Therefore, with $m = 1000kg$ and $g = 9.81\frac{m}{s^2}$ we have,

$$F_{sat} = 39.24kN$$

Linearization

Since the rockets motion is non-linear, in order for a controller to be designed for the rocket, the rockets model first has to be linearized. This is in order for a state feedback controll system to be designed. The reason why a state feedback controller has been chosen is because it is a lot less computationally expensive for MIMO systems compared to a PID controller and allows access to all states. The new state vector for the linearized system is then,

$$\dot{\boldsymbol{x}} = [\ddot{x}\ \ddot{y}\ \ddot{\theta}\ \dot{x}\ \dot{y}\ \dot{\theta}]^T$$

By using Taylor linearization around the hover condition where $F_1 = x = dx = y = dy = \theta = d\theta = 0$ and $F_2 = mg$, the state space represenatation of the linearized system can be found,

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -g \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}_{(g=9.81)} , \quad B = \begin{bmatrix} \frac{-1}{m} & 0 \\ 0 & \frac{1}{m} \\ -\frac{7}{2I_{cm}} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}_{(m=1000, I_{cm}=34750)}$$

Model Validation

In order to validate the calculated equations of motion and determine whether they make sense, a check was done to see whether or not the above derived, linearized state space model was indeed controllable. If the system was not controllable, it would mean that there was a flaw in deriving the equations of motion. For a controllable system, the rank of the matrix should be greater than the order of the system. Since this is a 3rd order system, the rank of the matrix should be greater than 3. Also, we require that each state be controllable since the applied force can act in both the x and y directions. The following two lines were used in Matlab to find the controllability matrix and then find its rank.

```
Co = ctrb(sys_ss.A,sys_ss.B)
r =  rank(Co)
```

The rank of the matrix was found to be 6. This is what was expected since all the states can be controlled.

Another form of validiation involved doing research into the equations of motion for other 2D planar quadrotor models and comparing them to the calculated equations of motion for our rocket. The following video of a 2D Quadrotor model suggests equations of motion that are identical to ours, therefore assisting us in validating our rocket state space model. The link to the video is https://youtu.be/iS5JFuopQsA.

4. Controller Design

With the model of the rocket being linearized, it was then possible to apply linear control techniques and tune the designed controllers to allow for the rocket to intercept the asteroid successfully. Both a position and an acceleration controller were designed, and an analysis between the two controllers and their respective characteristics was performed. As suggested before, a state feedback controller system was designed and the linear quadratic algorithm was used to determine the gains, K, for the system. Furthermore, integral action was introduced into both the acceleration and position controllers to reduce steady state error. The block diagram of a general state feedback controller with integral action is given below.

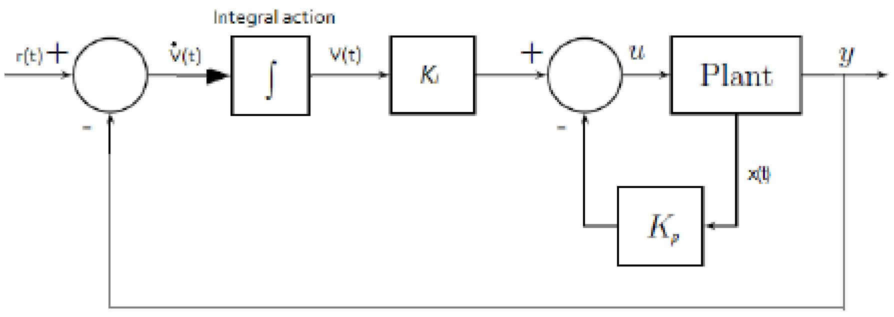


Figure 2: Block diagram of a general state-feedback controller

The LQR algorithm which was used for calculating the gains has the ability to control all the states of the system via a single controller. It minimizes the following cost function by penalizing the states, Q, or the control action, R, depending on the desired controller results. The cost function is given by:

$$\mathrm{J(u)} = \int_0^\infty (x^T \mathrm{Q}\mathrm{x} + u^T \mathrm{R}\mathrm{u})\,\mathrm{dt}$$

The controllers for both position and acceleration were designed such that these gains did not need to be recalculated based on the respective scenario. Note that scenario 3 was attempted for both controllers but the the rocket was not striking the asteroid at the correct angle. This scenario was, therefore, not considered in analysis.

Position Controller

The position controller was designed such that the rocket was able to intercept the asteroid every single time the asteroid was anticipated to land in the demarcated region in the animation (which represents the city). Furthermore, one of the specifications for scenario 2 specifically required that the rocket settle at a specific position along its trajectory and 'wait' in anticipation for the asteroid to pass this position. In doing this, controller robustness would be enhanced and controller tuning would become easier. The Simulink block diagram for this controller as well as its code can be found in Appendix D and Appendix C respectively.

The state and control action penalty matrices in using the LQR algorithm are given below. The state matrix, Q, consists of the penalties for the states $\dot{x},\ \dot{y},\ \dot{\vartheta},\ x, y, \theta, x_{ss}, y_{ss}$ respectively while the control action penalty matrix, R, consists of penalties for the inputs F_1, F_2.

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 200000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 180 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2900 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 430000 \end{pmatrix}_{\dot{x}, \dot{y}, \dot{\vartheta}, x, y, \theta, x_{ss}, y_{ss}}$$

$$R = \begin{bmatrix} 8050 & 0 \\ 0 & 8050 \end{bmatrix}_{F_1, F_2}$$

The step response of the controller for the integrated x, y and θ states are given below.

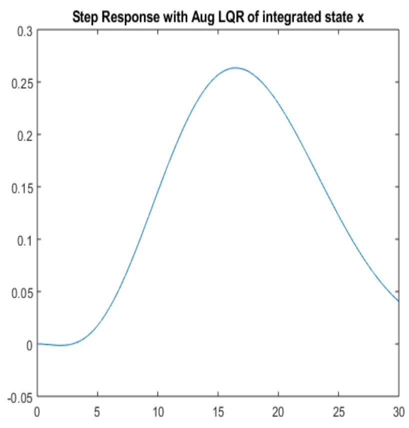


Figure 3: Step response of x

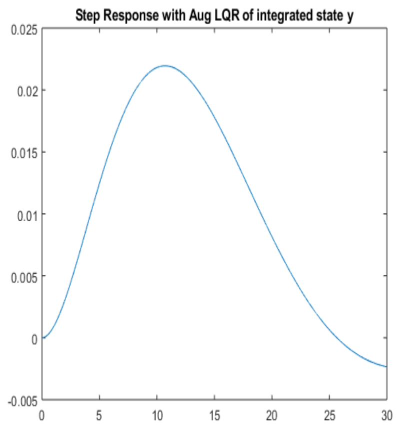


Figure 4: Step response of y

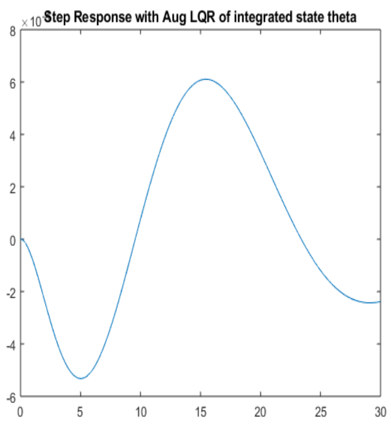


Figure 5: Step response of theta

As can be seen from the above graphs, the response times for each state is quite slow. Initially, much emphasis was placed on a faster response (through choosing a higher y-linearizing point), however, what became evident is the tradeoff between response time and stability. The reason for this is because the rocket was linearized vertically and therefore horizontal motion in a quick space of time causes the the rocket to spin on its axis uncontrollably regardless of how much the controller is tuned.

Without a phase controller impelemented to control the angle θ, a faster response had to be sacrificed such that the controller satisfies the requirement for scenario 1 without it settling at the respective y-position prior to doing so. However, for scenario 2, the rocket does indeed settle in the y direction before the asteroid passes it. It was designed specifically to not settle in the x-direction at a specific x-position before the asteroid passes it because the asteroid is unstable and does not pass a specific point in the x and y domain every single time along its trajectory. Hence, the rocket was designed such that it intercepts the asteroid just about as it is approaching its settling point in the x-direction. This increased the frequency of successful interception.

Additionally, to increase the chances of a successful interception, the rocket was designed to be at a specific angle as it approaches it settling point to cover a wider range of x-values that the asteroid might pass through. Emphasis was also placed on detonating the rocket in between the asteroid and the city, so that any remaining shrapnell is pushed away from the city. The following graphs depict the x and y trajectories for scenario 1 and scenario 2 respectively. For scenario 1 the reference point is chosen as (x=0m, y=3530m) and for scenario 2 the reference point is chosen as (x=1750, y =1600m).

Scenario 1

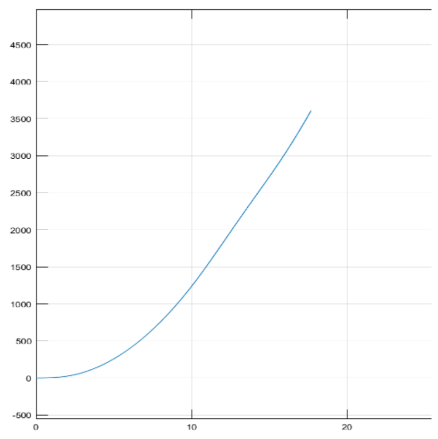


Figure 6: y trajectory - scenario 1

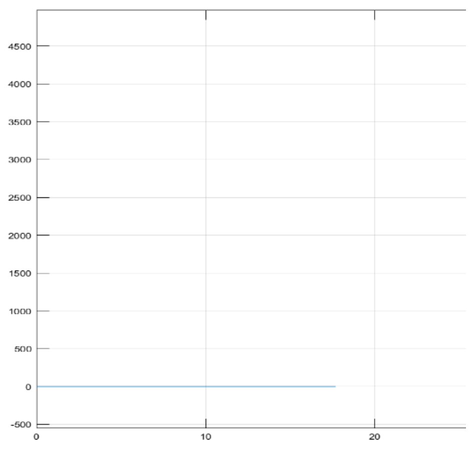


Figure 7: x trajectory - scenario 1

Scenario 2

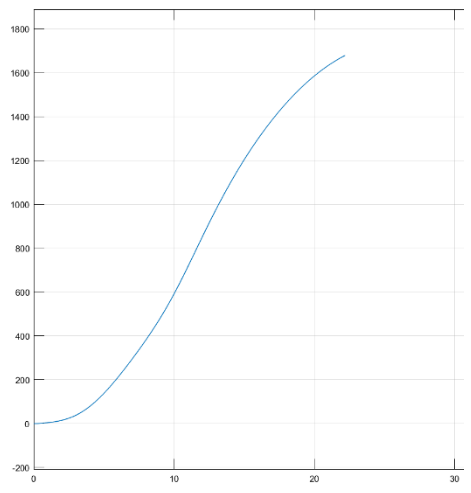


Figure 8: y trajectory - scenario 2

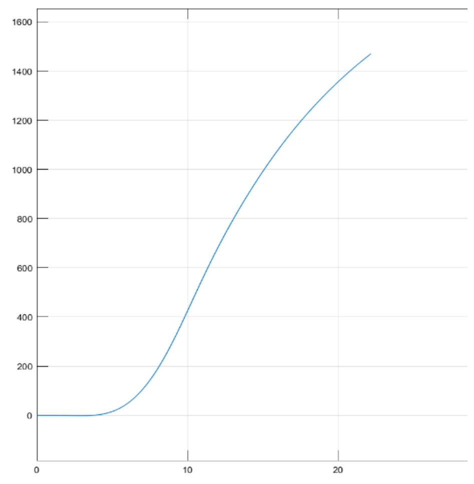


Figure 9: x trajectory - scenario 2

From the above graph its easy to see that the response for scenario 1 is unstable. This is because, for position control, it is a lot harder to control the rocket at large values of y. This is because of the slow response of the controller and could be remedied by the use of anti-windup techniques for the integrator. As expected, for scenario 2, the rocket intercepts the asteroid as it's about to settle in both the x and y direction. The overshoot in the y direction is about 8%. The reason why x has a large steady state error was because this was not severely penalized in the lqr algorithm in order to maintain controller stability and avoid the rocket from spinning on its axis.

Acceleration Controller

Unlike the position controller, the acceleration controller that was designed did not require that the rocket settle at a specific position. Rather, what was required was that, through the proportional navigation technique, the acceleration of the rocket in the x and & y direction is adjusted according to the difference in its position and velocity relative to that of the asteroid. For the acceleration controller, only the states related to the roll dynamics of the rocket θ and $\dot{\theta}$ are considered during linearization. The Simulink block diagram for this controller as well as its code can be found in Appendix F and Appendix E respectively.

The state and control action penalty matrices in using the LQR algorithm are given below. The state matrix, Q, consists of the penalties for the states $\dot{\vartheta}, \theta, x_{ss}, y_{ss}$ respectively while the control action penalty matrix, R, consists of penalties for the inputs F_1, F_2.

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 13 & 0 \\ 0 & 0 & 0 & 170000 \end{pmatrix}_{\dot{\vartheta},\theta,x_{ss},y_{ss}}$$

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{F_1,F_2}$$

The step response of the controller for the $\dot{\theta}$ state is given below.

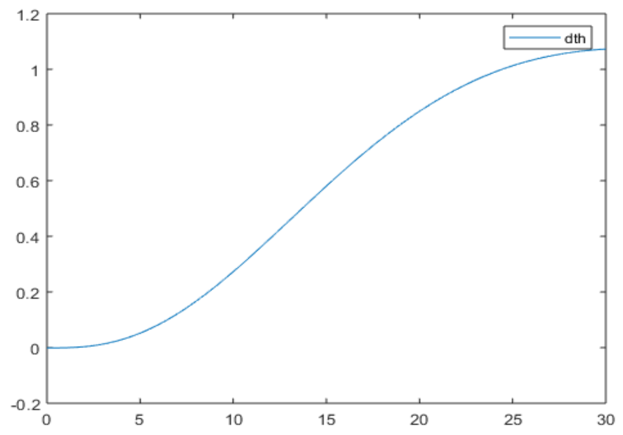


Figure 10: Step response of dth for acceleration control

For acceleration control using proportional navigation, not much emphasis needs to be placed on the settling time since the controller does not need to reach a specific $\dot{\theta}$ value within a set amount of time. The acceleration only needs to be adjusted relative to that of the reference point. The slow response in the above graph does not negatively affect the system. What can be seen is that the controller settles with no overshoot and 10% steady state error. The steady state error on the $\dot{\theta}$ state however does not need to be compensated for and is low enough to not have a significant impact on controller stability given the selected penalties for the LQR algorithm. The respective responses for the acceleration in the x and y direction for scenario 1 and scenario 2 are given below

Scenario 1

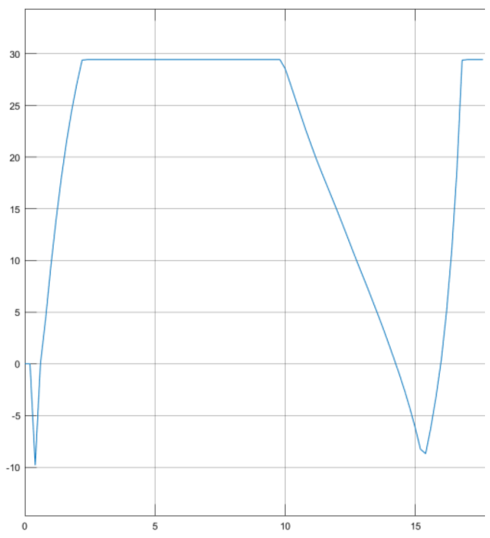


Figure 11: ddy response for scenario 1

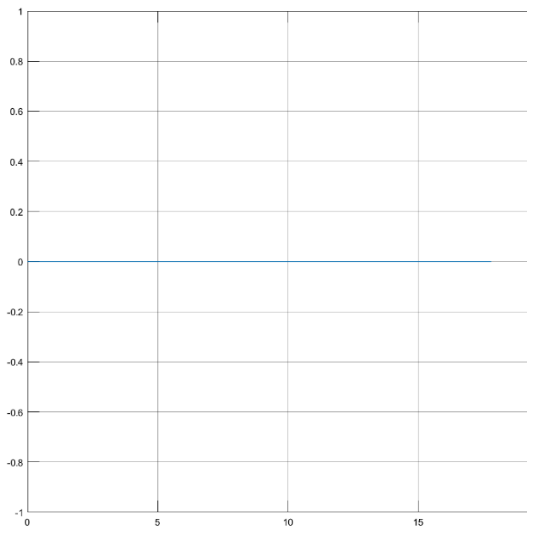


Figure 12: ddx response for scenario 1

Scenario 2

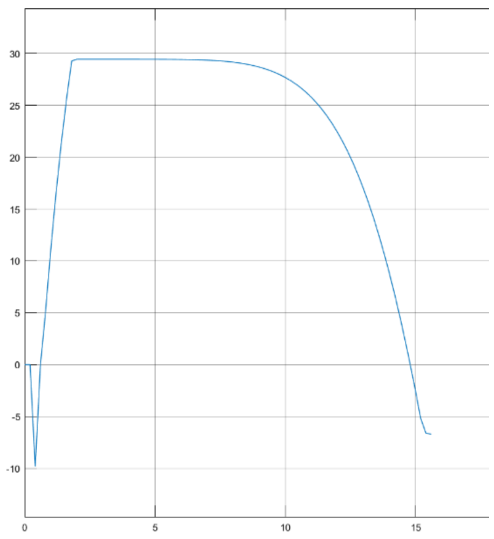


Figure 13: ddy response for scenario 2

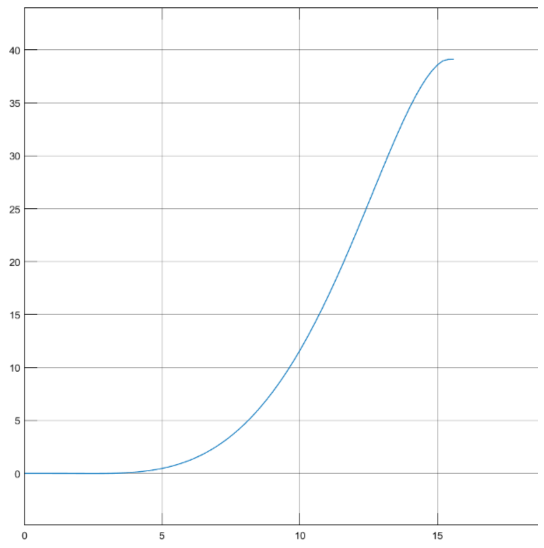


Figure 14: ddx response for scenario 2

From the above graphs, its easy to notice that the acceleration in both the x and y direction can increase or decrease with ease relative to the target and still maintain great controller stability whilst doing so. There is almost no or overshoot in the system and the controller works as intended.

Controller Tuning

The following tables illustrate how penalizing the respective the LQR algorithm variables affected the response of the system for each controller. Some states are not listed in the tables because they did not need to be penalized in controller design.

Position Control

$\boldsymbol{Q(x)}$	By increasing this value, the controller spun less on its axis. However, increasing it too much or using too small of a value caused the rocket to lose distance in the x-direction
$\boldsymbol{Q(y)}$	Increasing this value increased the height the rocket climbed up to in the y-direction

$\boldsymbol{Q(\theta)}$	There was no noticeable change when in controller performance when this value was altered.
$\boldsymbol{Q(x_{ss})}$	Increasing this value increased the response time of the controller, this came at the expense of losing distance in the c direction if this value was too large or too small. It had to be significantly smaller than $\boldsymbol{Q(y_{ss})}$ and even smaller than $\boldsymbol{R(F_1\& F_2)}$ to stop the rocket from spinning uncontrollably.
$\boldsymbol{Q(y_{ss})}$	Increasing this value increased the the height that the rocket climbed up to. The speed of the system was not affected since this value was already far greater than $\boldsymbol{Q(x_{ss})}$
$\boldsymbol{R(F_1\& F_2)}$	Increasing this value increased control action allowing for less spinning in the rocket at the expense of response time.

Acceleration Control

$\boldsymbol{Q(x_{ss})}$	Increasing this penalty allowed for the rocket to travel a greater distance in the x-direction, but also led to rocket spinning and becoming unstable. It had to be set to a value a lot lower than that for y steady state penalty in order to minimize the effect of spinning.
$\boldsymbol{Q(y_{ss})}$	Increasing this penalty allowed for the rocket to travel a greater distance in the y-direction. However, if the ratio between this and $\boldsymbol{Q(x_{ss})}$ was too small, the rocket would settle at a really small value in the y-direction but continue moving in the x-direction. This value had to be 13076 times greater than $\boldsymbol{Q(x_{ss})}$ for it to achieve 7 times as much distance in its respective direction. Also, in order to achieve satisfactory results for scenario 1 this value needed to be at least 30000 times greater than that of $\boldsymbol{R(F_1\& F_2)}$.
$\boldsymbol{R(F_1\& F_2)}$	Increasing this value increased control action allowing for less spinning in the rocket at the expense of response time.

Results & Analysis

For each controller, an analysis was done to determine the frequency of successful interceptions. For each scenario, the simulation was run twenty times. A successful interception involves the rocket intercepting the asteroid only if the asteroid was destined to land in the demarcated city region ($x = 2000 \pm 500m$). The cost or fuel usage can be calculated as follows :

$$\int F_{Tot} dt$$

Where F_{Tot} is the sum of the square of the forces in the x and y directions respectively.

Position Control

	y-Overshoot	*Interceptions*	*Misses*	*% Accuracy*	*Cost (N)*	*Time Taken(s)*	*Cost per second(kN/s)*
Scenario 1	N/A	18	2	90	5.377×10^5	17.4	30.90
Scenario 2	$\pm 8\%$	15	5	75	3.74×10^5	22	17.00

Acceleration Control

	y-Overshoot	*Interceptions*	*Misses*	*% Accuracy*	*Cost (N)*	*Time Taken(s)*	*Cost per second(kN/s)*
Scenario 1	None	20	0	100	5.082×10^5	18.2	27.92
Scenario 2	None	19	1	95	5.718×10^5	15	38.12

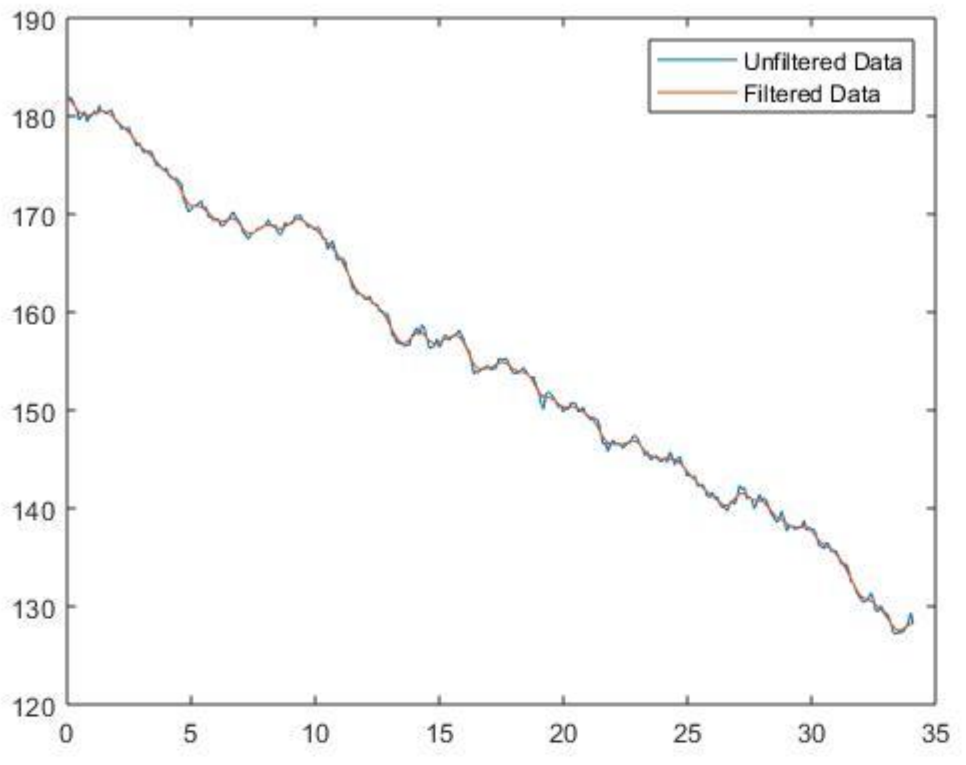
(controlling the angle at which it strikes is difficult)

5. Conclusion

Both the position and acceleration controllers worked as expected. However, because of the significant impact noise had on the asteroids trajectory, the accuracy of both controllers was affected. However, the acceleration controller out-performed the position controller in this aspect for both scenarios. This is because the acceleration controller uses proportional navigation which automatically adjusts the controller across small ranges of values without much controller action required. Since the position controller has more states to control, it makes it more difficult to tune and hence more difficult to successfully intercept a moving target. The acceleration controller was also able to control the rocket at larger values of y compared to the position controller, thus allowing for a greater distance between the point of detonation and the city. The acceleration controller also has better controller performance characteristics in that it displays no overshoot or steady state error. It could be argued, however, that on average the position controller costs less in terms of the force required at the input. However, the cost factor was not a requiremement that was needed to be considered in completing the project successfully. Hence, the controller elected for simulation purposes was the acceleration controller.

6. Appendix

Appendix A – Graph representing the filtered and unfiltered data that was used for calculating the value of c



Appendix B - Code used for calculating c parameter

```
syms c
w =5;
i = 0;
o = (1/w)*ones(1,5);
cval= 0;
banana = filtfilt(o,1,ast_dx);
```

```
plot(simulation_time,ast_dx)
hold on
plot(simulation_time,banana)
legend ('Unfiltered Data', 'Filtered Data')

apple = banana == 182*exp(-c*simulation_time/10000)

for i = 1:342
c_mat  = solve(apple(i), c);
cval = c_mat+cval;
end

cval =cval/342;
cval = round(cval*100)/100
```

Appendix C – Code used for calculating gains of position controller

```
clear;
syms x y th dx dy dth ddx ddy ddth F1 F2 F alph 'real'
syms mass width g inertia 'real'
q = [x; y; th];
dq = [dx; dy; dth];
ddq = [ddx; ddy; ddth];
% position and velocity
pBody = [x;
         y];
vBody = jacobian(pBody, q) * dq;
% potential energy
Ttot = 0.5*mass*transpose(vBody)*vBody ...
    + 0.5*inertia*(dth)^2;
Ttot = simplify(Ttot)
% kinetic energy
Vtot = mass*[0 g]*pBody
M = simplify(hessian(Ttot, dq));
dM = sym(zeros(size(M)));
for i = 1:size(dM, 1)
    for j = 1:size(dM, 2)
        dM(i,j) = jacobian(M(i,j), q) * dq;
    end

end
dM = simplify(dM);
C = dM*dq - transpose(jacobian(Ttot, q));
C = simplify(C);
G = simplify(jacobian(Vtot, q)');
R = [ cos(th) -sin(th)
      sin(th)  cos(th)];

% location of the input forces
r1 = [x; y] + R * [0; -3.5];
r2 = [x; y] + R * [0; -3.5];
% input force with respect to inertia
f1 = R * [-F1; 0];
f2 = R * [0; F2];
% partial derivatives
```

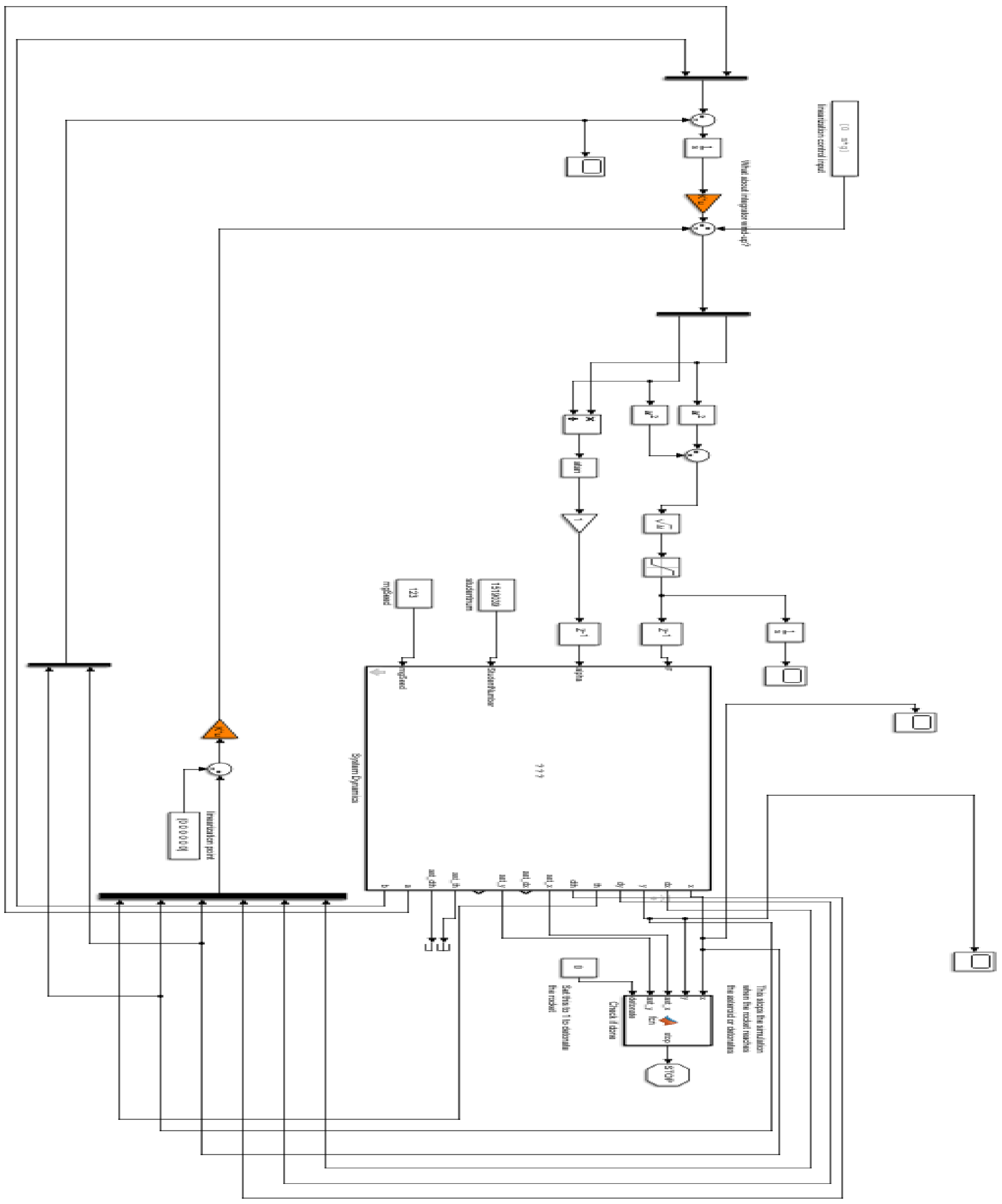
```
Q1 = simplify(f1' * jacobian(r1, q))'
Q2 = simplify(f2' * jacobian(r2, q))'
% calculate B matrix
B = jacobian(Q1 + Q2, [F1; F2]);
u = [F1;F2];
B*u
% the backslash operator \ solves the equation Ax = b
% more efficienctly than x = inv(A) * b
% the backslash operator \ solves the equation Ax = b
% more efficienctly than x = inv(A) * b
acceleration_eqns = simplify(M \ (-C - G + B*u))
dyn = [acceleration_eqns; dq];
% Jacobian of dynamics, with the operating point subbed in
A = jacobian(dyn, [dq.', q.']);
A = subs(A, [dq.', q.', u.'], [0,0,0,0,0,0,0,mass*g])
B = jacobian(dyn, [u.']);
B = subs(B, [dq.',q.',u.'],[0,0,0,0,0,0,0,mass*g])
```

Substitue in specific parameters

```
% Params needed for simulation
m_ = 1000;
w_ = 200;
I_ =34750;
g_ = 9.81;
% Subsitute to get numerical A and B
A = double(subs(A, [mass,width,inertia,g], [m_,w_,I_,g_]));
B = double(subs(B, [mass,width,inertia,g], [m_,w_,I_,g_]));
C = eye(6);
D = [];
states = {'dx' 'dy' 'dth' 'x' 'y' 'th'};
inputs = {'u1' 'u2'};
outputs = {'dx' 'dy' 'dth' 'x' 'y' 'th'};
sys_ss = ss(A, B, C, D, 'statename', states,...
                        'inputname', inputs,...
                        'outputname', outputs)

Co = ctrb(sys_ss.A,sys_ss.B);
r =  rank(Co);
Q = 1*eye(6);
Q(4,4) =25000;
Q(5,5) = 25000;
Q(6,6) = 25000;
R = 1*eye(2);
% calculate LQR gain
K = lqr(A, B, Q, R)
Ac = [(A-B*K)];
Bc = [B];
Cc = [C];
Dc = [D];
sys_cl = ss(Ac, Bc, Cc, Dc, 'statename', states,...
                            'inputname', inputs,...
                            'outputname', outputs);
t = 0:0.001:30;
```

```
r = 2*ones(length(t), 2);
[y,t,x] = lsim(sys_cl, r, t);
% plot the position states
plot(t, y(:, 4:6));
legend(["x"; "y"; "th"]);
title('Step Response with LQR Control')
% New integrator states for error on positions x and y
% New integrator states for error on positions x and y
newA = [0, 0, 0, 1, 0, 0, 0, 0;...
        0, 0, 0, 0, 1, 0, 0, 0];
tempA = horzcat(A, zeros(6,2));
% New A Matrix (aug = augmented)
A_aug = vertcat(tempA, newA);
% New B Matrix
B_aug = vertcat(B, zeros(2,2));
% New C matrix
C_aug = eye(8);
Q = eye(8);
Q(4,4) =200000;
Q(5,5) = 10000;
Q(6,6) = 180;
Q(7,7) = 2900;
Q(8,8) = 430000;
R = 8050*eye(2);
% weighting system - Weighting integral states more
% for SS tracking
% calculate new LQR gain
K_aug = lqr(A_aug, B_aug, Q, R)
Ac = [(A_aug - B_aug*K_aug)];
Bc = [B_aug];
Cc = [C_aug];
Dc = [D];
states = {'dx' 'dy' 'dth' 'x' 'y' 'th' 'xInt' 'yInt'};
inputs = {'u1' 'u2'};
outputs = {'dx' 'dy' 'dth' 'x' 'y' 'th' 'xInt' 'yInt'};
sys_cl = ss(Ac, Bc, Cc, Dc, 'statename', states,...
                            'inputname', inputs,...
                            'outputname', outputs);
t = 0:0.001:30;
r = 2*ones(length(t), 2);
[y,t,x] = lsim(sys_cl, r, t);
plot(t, y(:,5));
%legend(states)
title('Step Response with Aug LQR of integrated states')
% these constants must be in your workspace for the
% Simulink model
m = m_;
g = g_;
I = I_;
L = w_;
% you might want to copy-paste the final commands
% into the Command Window to avoid having to re-run
% all the above code every time
% OTHERWISE just commenting out the plots above
% should speed things up enough
% run the regular LQR
% sim('quad_LQR_sim.slx');
% run augmented LQR
% sim('quad_LQRwithIntegrators_sim.slx');
% plot!
```

Appendix D – Simulink diagram used for position controller implementation

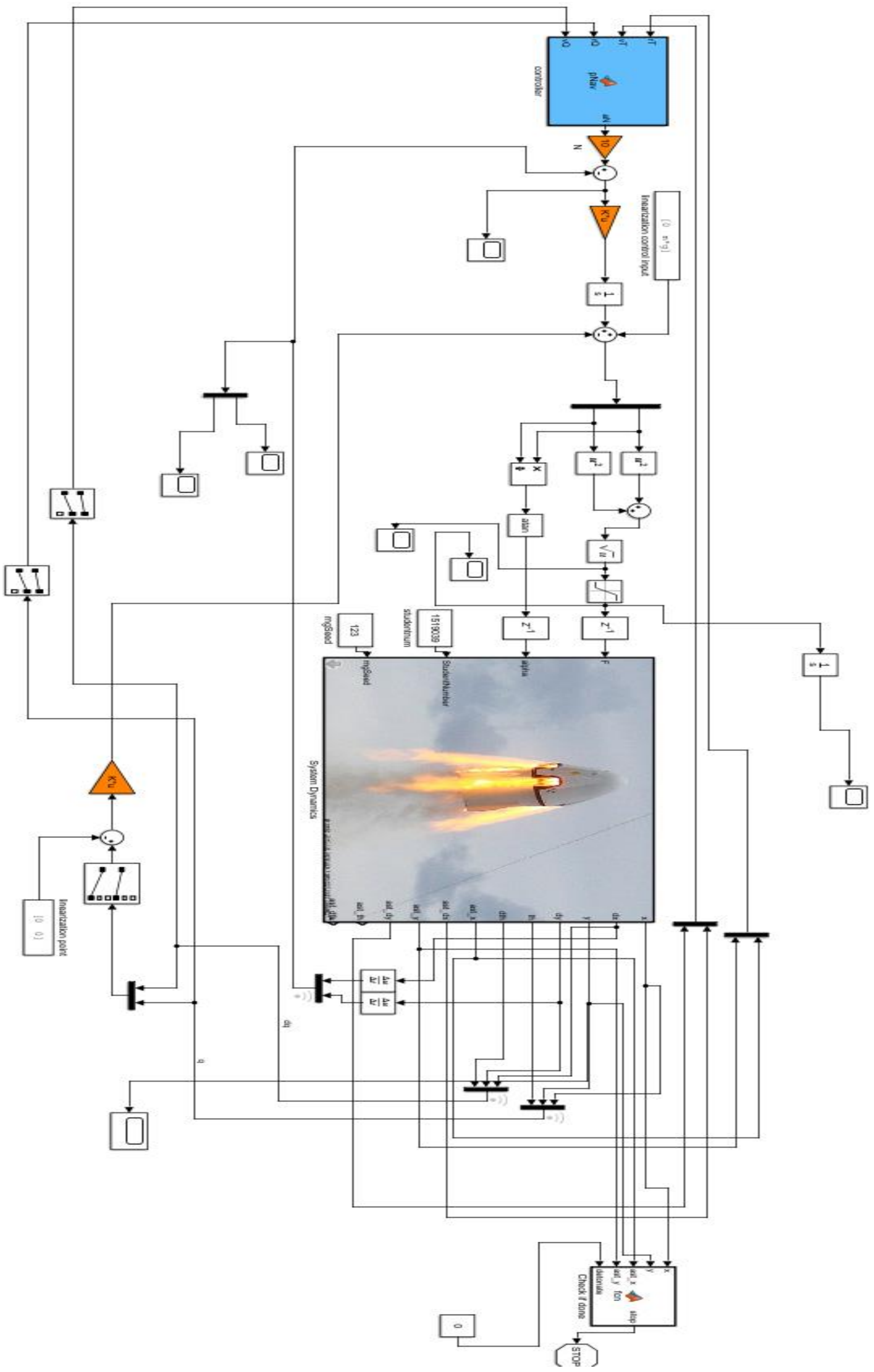


Appendix E - Code used to calculate gains for position controller

```
clear;
syms x y th dx dy dth ddx ddy ddth F1 F2 F alph 'real'
syms mass width g inertia 'real'
q = [x; y; th];
dq = [dx; dy; dth];
ddq = [ddx; ddy; ddth];
% position and velocity
pBody = [x;
         y];
vBody = jacobian(pBody, q) * dq;
% potential energy
Ttot = 0.5*mass*transpose(vBody)*vBody ...
    + 0.5*inertia*(dth)^2;
Ttot = simplify(Ttot)
% kinetic energy
Vtot = mass*[0 g]*pBody
M = simplify(hessian(Ttot, dq));
dM = sym(zeros(size(M)));
for i = 1:size(dM, 1)
    for j = 1:size(dM, 2)
        dM(i,j) = jacobian(M(i,j), q) * dq;
    end

end
dM = simplify(dM);
C = dM*dq - transpose(jacobian(Ttot, q));
C = simplify(C);
G = simplify(jacobian(Vtot, q)');
R = [ cos(th) -sin(th)
      sin(th)  cos(th)];

% location of the input forces
r1 = [x; y] + R * [0; -3.5];
r2 = [x; y] + R * [0; -3.5];
% input force with respect to inertia
f1 = R * [-F1; 0];
f2 = R * [0; F2];
% partial derivatives
Q1 = simplify(f1' * jacobian(r1, q))';
Q2 = simplify(f2' * jacobian(r2, q))';
% calculate B matrix
B = jacobian(Q1 + Q2, [F1; F2]);
u = [F1;F2];
B*u
% the backslash operator \ solves the equation Ax = b
% more efficienctly than x = inv(A) * b
% the backslash operator \ solves the equation Ax = b
% more efficienctly than x = inv(A) * b
acceleration_eqns = simplify(M \ (-C - G + B*u))
```

```
u = [F1; F2];
dyn = [acceleration_eqns(3); dth];
dyn = subs(dyn, [mass,inertia,g], [m_,I_,g_])
% OUTPUTS are acceleration: ddx, ddy
Y = subs(acceleration_eqns(1:2), [mass,inertia,g], [m_,I_,g_])
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])
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)
% linearized system
states = {'dth' 'th' 'ddxE' 'ddyE'};
inputs = {'u1' 'u2'};
outputs = {'ddx' 'ddy' 'ddxE' 'ddyE'};
sys_ss = ss(A, B, C, D, 'statename', states,...
                        'inputname', inputs,...
                        'outputname', outputs)
% LQR - regulator/stabilising states
Q = 1*eye(4);
Q(3,3) = 13;
Q(4,4) = 185000;
R = eye(2);
% calculate LQR gain
K = lqr(A, B, Q, R)
% closed loop
Ac = [(A-B*K)];
Br = [0 0; 0 0; -1 0; 0 -1];
Cc = [C];
Dc = [D];
sys_cl = ss(Ac, Br, Cc, Dc, 'statename', states,...
                            'inputname', inputs,...
                            'outputname', outputs);
t = 0:0.001:30;
step(sys_cl, t)
legend
title('Step Response with LQR Acceleration Control')
t = 0:0.001:30;
r = ones(length(t), 2);
[y,t,x] = lsim(sys_cl, r, t);
plot(t, y(:,1:1));

legend(states)
```

Appendix F – Simulink implementation of acceleration controller


controller
pNav
N
linearization control input
System Dynamics
studentnum
rngSeed
linearization point
Check if done
STOP