` EEE4119F Milestone 4

Caide Lander - LNDCAI001

Contents

Introduction

For this project, I have been tasked with designing a controller for a planetary lander that can safely touchdown on a planet with unknown gravity. The project is divided into three milestones, each with its own set of requirements and deliverables.

Milestone 1

The first milestone involves modelling the lander in MATLAB and Simulink. This includes determining the equations of motion (EOM), the planet's gravitational acceleration (g value), the mass moment of inertia (Izz) and calculating the lander's centre of mass offset (dL).

Milestone 2

The second milestone focuses on the control of the lander in two different scenarios. The controller must meet specific requirements related to the lander's final height, velocity, and angle. Additional marks were awarded for minimizing fuel usage and landing near the desired position.

Milestone 3

The third milestone introduces a third, more challenging landing scenario with a consistent g value for all students to introduce competition – bonus marks were awarded for speed of landing. The controller must still meet the same requirements as in Milestone 2, but with slightly harder criteria. This milestone, along with the final report, is meant to demonstrate a deep understanding of the modelling and control concepts.

Commented [PN1]: The recommended layout had some stuff like goals and problem parameters in the introduction section, I thought you might want to know.

Modelling

When modeling the system was defined as shown in the figure below:

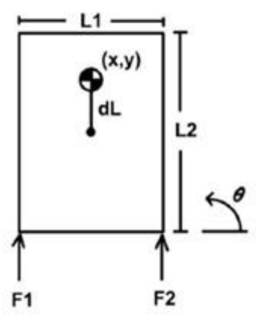


Figure 1 - An illustration of the lander's dimensions and the position of x and y [1]

When writing this report it was determined that the best method to describe modelling of the system would be a table that explains key parameters. The table is as follows:

Parameter	Code or Value	Reasoning/ Explanation
Frames	Inertial and body	These are the only two frames that can be used in this case – single body and rotation.
Rocket Positions	[x; y; 0]	The rocket was simulated in 2D and thus only has x and y coordinates. The sensors read the positions at the COM, with respect to ground.
Force Positions (body)	[± L1/2; - L2/2 - dL; 0]	Based on x and y being at (0, 0). Left thruster @ - L1/2 and Right thruster @ L1/2.
Force Vectors	[0; F1; 0]	The thrusters only act in the y direction.
Partial Force Derivative	diff(rF$_i$, q)	The partial derivative was calculated for each F, on x and y.
Q Vector	$\sum_{k=0}^{1} transpose(F_i\_0) * partial\_q$	Each F$_i$ is transposed and multiplied my the partial to determine the Q_x, Q_y and the Q_{th}.
Q matrix	-sin(th)*(F1 + F2) cos(th)*(F1 + F2) -(L1*(F1 - F2))/2	This is a simplified version of Q_x, Q_y and Q_{th} all put into a singular Q matrix.
Mass Matrix	[m, 0, 0] [0, m, 0] [0, 0, Izz]	Mass matrix (M) is derived from the system's kinetic energy with respect to the generalized coordinates (dq).
Coriolis Matrix	0 0 0	The Coriolis matrix (C) is calculated from the derivative of the mass matrix (M) with respect to the generalized coordinates (q) and their rates of change (dq).
Gravity Matrix	[0, g*m, 0]	The gravity matrix (G) is derived from the potential energy of the system with respect to the generalized coordinates (q).
Manipulator Equation	M*ddq + C + transpose(G) - Q == 0	The manipulator equation describes the dynamic equilibrium of the system's generalized coordinates (q) and their accelerations (ddq). It combines four terms: the mass matrix (M) multiplied by the second derivative of q (ddq), the Coriolis matrix (C), the transpose of the gravity matrix (G), and the generalized forces (Q). The equation asserts that in a state of equilibrium, the sum of these terms equals zero, indicating no net forces or torques acting on the system. This equation is fundamental for analysing the dynamics and control of robotic systems, such as the rocket in this context.

Table 1 - Explaining key parameters

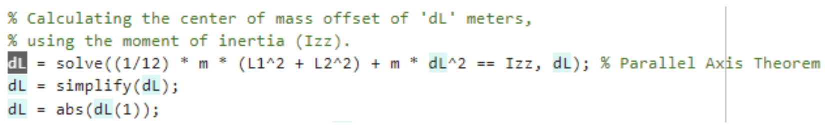
After considering the defined system, the main ~~objetive~~objective was to determine the EOM, the g value, the Izz value and dL. Below was the method used to determine g and Izz:

Commented [PN2]: spelling error

1. **Determining *g***: By setting the thrusters' outputs to 0, the lander falls freely under the influence of gravity. The constant acceleration observed during free fall is assumed to be equal to the planet's gravitational acceleration g.

2. **Determining *Izz***: Applying a constant torque to the lander causes it to spin with a constant angular acceleration. By analysing the behaviour of the lander under this torque and subbing the observed values into the manipulator equation, the moment of inertia *Izz* can be calculated.

The entire code for modelling can be found in Figure 8 and Figure 9, dL could be solved using the following code snippet:



```
% Calculating the center of mass offset of 'dL' meters,
% using the moment of inertia (Izz).
dL = solve((1/12) * m * (L1^2 + L2^2) + m * dL^2 == Izz, dL); % Parallel Axis Theorem
dL = simplify(dL);
dL = abs(dL(1));
```

Figure 2 - Code snippet showing how dL was calculated.

Determining the magnitude of dL sufficed since only its absolute value was needed. In this instance, COM fell below the specified origin. Below are the found values within milestone 1:

$$g = 11.1606\,\mathrm{m\cdot s^{-2}} \qquad Izz = 62500\,kg\cdot\mathrm{m}^2 \qquad dL = 2.7\mathrm{m}$$

Commented [PN3]: I think you skipped an equation here

Commented [CL4R3]: What eqn do you rate should be here?

Commented [PN5R3]: The statement above could be removed, because I see you talked about dL and then doubled down on it and just ended with that "Below....." statement

Commented [CL6R3]: I was just explaining the use of the ABS and then on a separate note talking about the found values

Control Scheme

In this section, various important controller details will be covered.

Controller requirements

The controller must be designed to handle three distinct landing scenarios, each with varying initial conditions for the lander's height, angle, and angular velocity. The specific requirements for each scenario are as follows:

Mode 1: y_0 = 1000±5 m, θ_0 = 0 rad, ω_0 = 0 rad/s
Mode 2: y_0 = 1000±50 m, θ_0= ±0.5 rad, ω_0 = 0 rad/s
Mode 3: y_0 = 1300±200 m, θ_0 = π±0.5 rad, ω_0 = ±0.1 rad/s

Regardless of the scenario, the controller must ensure the lander:

- Touches down within the landing platform range: 0 < y < 0.5 m

- Achieves a vertical velocity $|dy|$ < 2 m/s

- Lands vertically with an angle $|\theta|$ < 10 degrees
  Additionally, the controller should aim to minimize the time taken to land, the horizontal landing position x, and the fuel consumption denoted by $\int(F1 + F2)\,dt$.

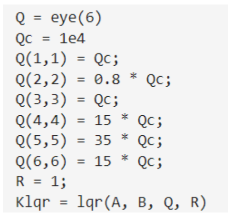
Controller design

Once the controller requirements are well defined the controller may be designed, bellow is an explanation of the code used to generate specific K values for the controller.

1. **Equations of Motion**: The code starts by defining the equations of motion (EOM) for the lander system, including the gravitational acceleration (g), the moment of inertia (Izz),

and the center of mass offset (dL). These equations are represented symbolically using the syms function. The EOM's were simplified using small angle approximation.

2. **State-Space Representation**: The state vector X is defined as [x; y; th (θ); dx; dy; dth (dθ)], and the state-space equations are derived by substituting the equations of motion into the state derivatives dX , defined by: [dx, dy, dth (dθ), EOM1, EOM2, EOM3] .
3. **Equilibrium Conditions**: The code finds the equilibrium conditions by setting the state derivatives dX equal to zero and solving for the control inputs F1 and F2.
4. **Linearization**: The Jacobian matrices A and B are computed by taking the partial derivatives of the state derivatives dX with respect to the state X and the control inputs U, respectively. These matrices represent the linearized state-space model around the equilibrium point.
5. **LQR Controller Design**: The code defines the state and control weighting matrices Q and R, respectively, and uses the lqr function to compute the optimal state feedback gain matrix Klqr. This gain matrix is used to design the linear quadratic regulator (LQR) controller. Below in figure 3, the Q and R values that were used for milestone 2 are shown:



```
Q = eye(6)
Qc = 1e4
Q(1,1) = Qc;
Q(2,2) = 0.8 * Qc;
Q(3,3) = Qc;
Q(4,4) = 15 * Qc;
Q(5,5) = 35 * Qc;
Q(6,6) = 15 * Qc;
R = 1;
Klqr = lqr(A, B, Q, R)
```

Figure 3 - Q and R values used for milestone 2

6. **Closed-Loop System**: The closed-loop system matrix Acl is computed by subtracting the product of B and Klqr from the state matrix A. The closed-loop system is then converted to state-space form and its step response is plotted.

The preferred control approach is optimal control using LQR optimization, which determines the optimal gain K to minimize a linear system. Q and R matrices decide how much emphasis to put on penalizing state variables and control actions, respectively. By picking the right values, the LQR optimization algorithm can determine the correct K control values based on these penalty weights. This method is way more effective than the alternative method of pole placement for controlling state spaces. With LQR tweaking a weight slightly can change control over the selected state or control variable. Whereas with pole placement a slight change may end up impacting all poles at once. That's why LQR optimization wins out over the trickier pole placement method, plus there's the option to boost performance with augmented LQR optimization. However, in preparation for calculating the KLQR values, the Simulink simulation control model was first completed. This controller design involved implementing a Simulink function that takes in the state variables and the controller gain matrix KLQR (after testing, these values where hard-coded inside the actual controller in order to just hand in a .slx file), and outputs the two thruster forces. To address the algebraic loop error, unit delays were incorporated into the feedback loop before the controller. The Simulink model and the controller function code are provided in the appendix in Figure 11 and Figure 12.

Optimization

The optimization process in this code is focused on tuning the weighting matrices Q and R to achieve the desired closed-loop performance. Specifically:

1. **State Weighting Matrix Q**: The diagonal elements of Q are adjusted to prioritize the different state variables. For example, the position states x and y are weighted more heavily than the angular states th (θ) and dth (dθ) – as can be seen in Figure 3.
2. **Control Weighting Matrix R**: The scalar value R is set to 1, which means the control effort is not heavily penalized in the cost function, originally there was an attempt to heavily penalize the control effort in order to reduce fuel consumption. However, it was found that the controller performed more unreliably and failed for specific seeds. Therefore, reliability was prioritized as ensuring the safety of potential passengers and of the rocket itself takes priority over reducing fuel consumption.

The optimization process aimed to strike a balance between minimizing state errors and control effort to meet the specified landing criteria. An iterative approach was employed, starting with random Q values and adjusting the weighting matrices (Q and R) for the Linear Quadratic Regulator (LQR) controller. As stated above after some failed attempts, the value for R was kept at 1.

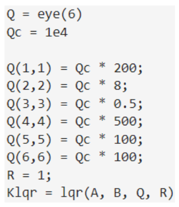
During the iterative refinement, careful adjustments were made to each matrix element to evaluate its impact on controller performance (If Q was over-penalized it was reduced whereas if it was under-penalized it was increased). Over-penalization of key state variables like vertical position (y) and its derivative (dy) could lead to undesirable outcomes, such as rapid descents or erratic movements, affecting stability. Under-penalization of y and dy could result in challenges like poor altitude control, unstable vertical motion, trajectory tracking issues, and reduced system stability. These variables were crucial as errors in their optimization could lead to the rocket crashing, emphasizing their significance in the control process. All state variables were adjusted in a similar way to y and dy – these two variables were just used as an example due to their significance. It is important to note that the optimal derivative penalization is higher than that of the respective state variable.

While y and dy were critical for safe landing, suboptimal values for other state variables might affect landing accuracy or duration (for e.g. if overshooting occurred it would cause instability beyond 90 degrees for theta, and oscillation around the landing platform for both x and theta). The iterative approach ensured a balanced set of weights for all state variables, enabling effective control while avoiding instability or erratic behaviour. This meticulous process aimed to optimize the controller for reliable and precise landings, prioritizing safety and mission success. When testing optimal values for the controller, it was tested in batches of 100 random seeds, whenever a failure was encountered it was taken into consideration and the Q values were adjusted. State variables aiming for small and low deviation necessitate larger weightings.

In Milestone 3, like Milestone 2, the controller code remained consistent, with a change in the gravitational value from Milestone 1. The new g value was determined using the code from Milestone 1, adapting to altered dynamics. Bonus marks in Milestone 3 were based on landing speed, rewarding the fastest student with full marks and scaling others' scores accordingly. While speed was a factor in controller design, safety and absolute reliability took precedence over competition. The controller prioritized reliability within the simulated conditions, ensuring a balance between speed and dependability.

Below are the found values within milestone 3 as well as the Q values used:

$$g = 9.7284 \text{ m} \cdot \text{s}^{-2} \qquad Izz = 62500\ kg \cdot \text{m}^2 \qquad dL = 2.7\text{m}$$



```
Q = eye(6)
Qc = 1e4

Q(1,1) = Qc * 200;
Q(2,2) = Qc * 8;
Q(3,3) = Qc * 0.5;
Q(4,4) = Qc * 500;
Q(5,5) = Qc * 100;
Q(6,6) = Qc * 100;
R = 1;
Klqr = lqr(A, B, Q, R)
```

Figure 4 - Q and R values used for milestone 3

Results and Discussion

When thoroughly testing the final controller, 10 000 line plots were executed (simulations of randomly generated seeds in scenario 3 - the most difficult scenario - and plotting the important state variables; It is important to note that the controller worked for all three scenarios), resulting in 20 failed seeds, indicating a 0.2% failure rate, which is considered reasonable. Across all failed seed variations, the landing speed was near -2m/s (the threshold for a crash), with an approximate average speed of -2.094984m/s. A table showing the failed seeds can be found at Table 2. Various other line plots were generated in testing, which can be found from Figure 13 onwards within the appendix. The 10 000 line plots are included below:

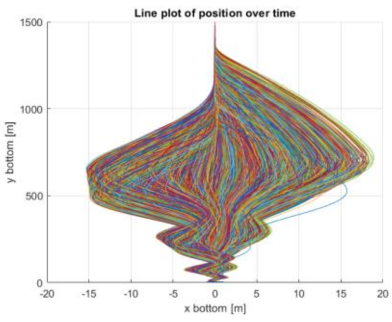


Figure 5 - Position over time for 10 000 line plots

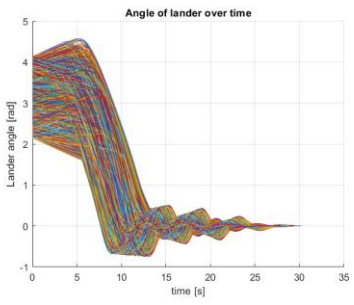


Figure 6 - Angle of lander over time for 10 000 line plots

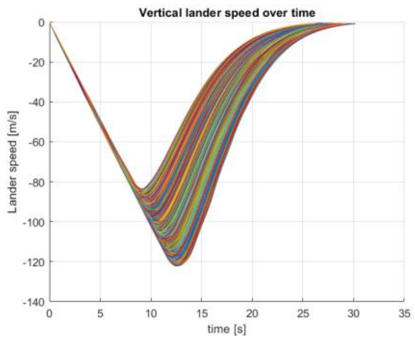


Figure 7 - Vertical lander speed over time for 10 000 line plots

The controller's robustness and reliability are evident in the plots, showcasing its ability to handle uncertainties and ensure safe landings consistently. Each simulation meets all requirements, as seen in the following figures: Figure 5 displays successful landings at the platform coordinates $(x, y) \approx (0, 0)$; Figure 6 illustrates vertical landings with controlled angles near $\theta \approx 0$; and Figure 7 demonstrates controlled vertical speeds with safe landings below 2 m/s.

The rocket largely landed within approximately ±30 seconds, with the majority landing in under 30 seconds and a few outliers exceeding this duration, as depicted in Figure 7.

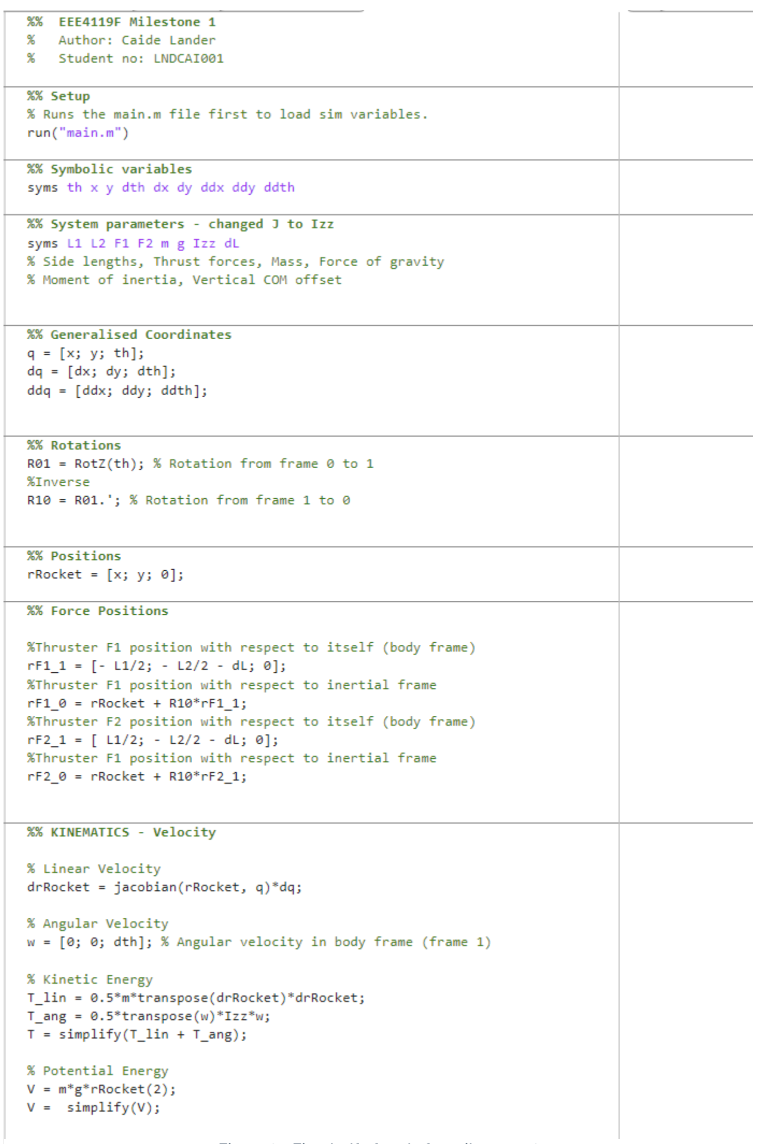
Conclusion

In conclusion, the final designed controller successfully met all the requirements for landing the planetary lander on the unknown planet. The controller was robust and reliable, ensuring safe and precise landings across all three modes. The iterative approach to optimizing the weighting matrices Q and R allowed for a balanced set of weights that prioritized safety and reliability over the competition (speed). The controller's performance was tested extensively, with 10,000 line plots executed to simulate various seed variations in mode 3, the most challenging scenario. The results showed a success rate of 99.8%, indicating a high level of reliability. The controller's ability to adapt to changing dynamics and handle unexpected scenarios demonstrates its effectiveness in real-world applications. Although given the ability to generate large datasets – by adjusting the line plots code – it may be possible to train a machine learning model to create even more optimal KLQR values than the ones used. Using a model may be the next step in designing the most optimal controller possible. Overall, the designed controller successfully achieved the project's objectives, showcasing a deep understanding of modelling and control concepts.

References

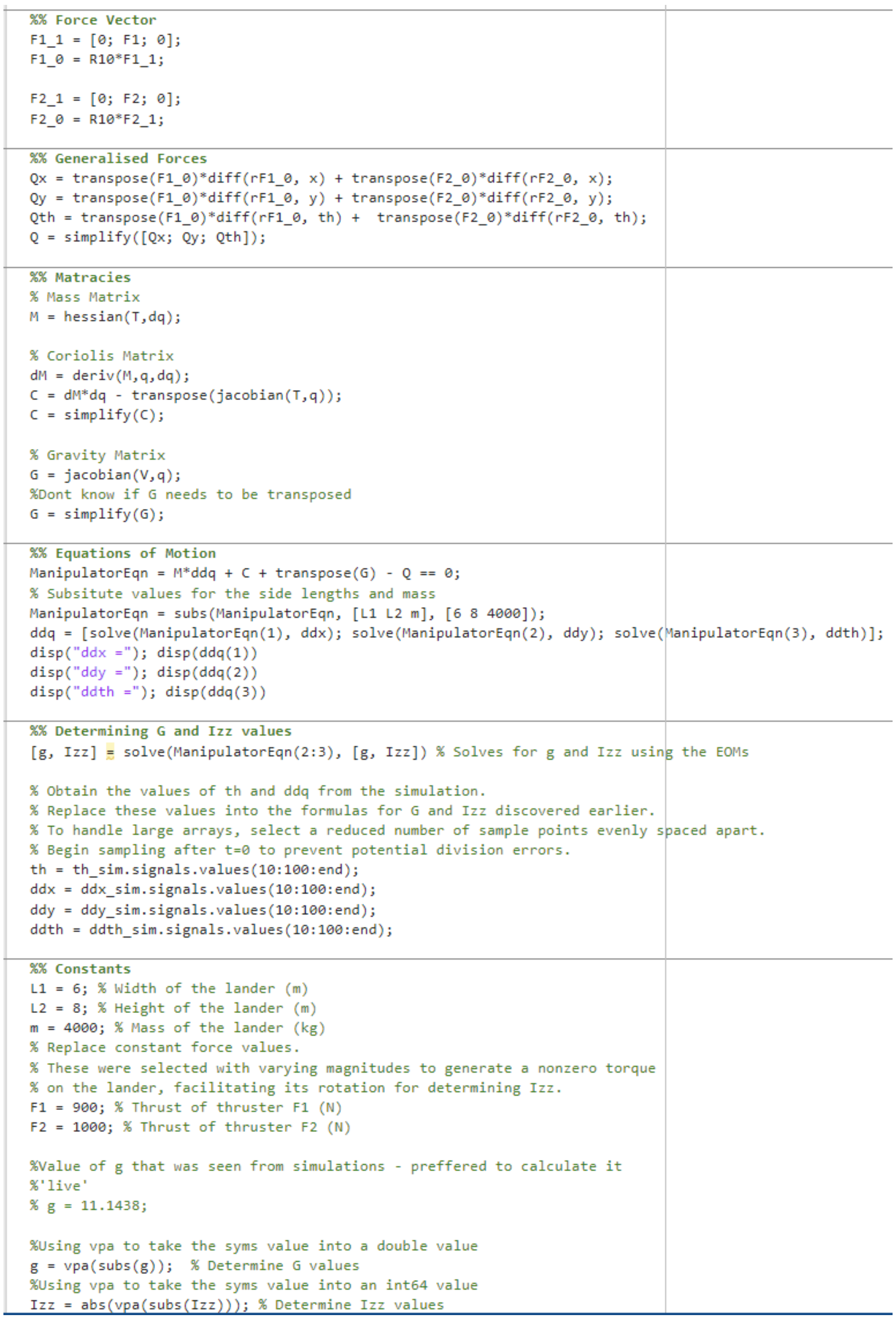
[1] EEE4119F, “Project brief - landing on an unknown planet,” 2024

Appendix



```
%%  EEE4119F Milestone 1
%   Author: Caide Lander
%   Student no: LNDCAI001

%% Setup
% Runs the main.m file first to load sim variables.
run("main.m")

%% Symbolic variables
syms th x y dth dx dy ddx ddy ddth

%% System parameters - changed J to Izz
syms L1 L2 F1 F2 m g Izz dL
% Side lengths, Thrust forces, Mass, Force of gravity
% Moment of inertia, Vertical COM offset

%% Generalised Coordinates
q = [x; y; th];
dq = [dx; dy; dth];
ddq = [ddx; ddy; ddth];

%% Rotations
R01 = RotZ(th); % Rotation from frame 0 to 1
%Inverse
R10 = R01.'; % Rotation from frame 1 to 0

%% Positions
rRocket = [x; y; 0];

%% Force Positions

%Thruster F1 position with respect to itself (body frame)
rF1_1 = [- L1/2; - L2/2 - dL; 0];
%Thruster F1 position with respect to inertial frame
rF1_0 = rRocket + R10*rF1_1;
%Thruster F2 position with respect to itself (body frame)
rF2_1 = [ L1/2; - L2/2 - dL; 0];
%Thruster F1 position with respect to inertial frame
rF2_0 = rRocket + R10*rF2_1;

%% KINEMATICS - Velocity

% Linear Velocity
drRocket = jacobian(rRocket, q)*dq;

% Angular Velocity
w = [0; 0; dth]; % Angular velocity in body frame (frame 1)

% Kinetic Energy
T_lin = 0.5*m*transpose(drRocket)*drRocket;
T_ang = 0.5*transpose(w)*Izz*w;
T = simplify(T_lin + T_ang);

% Potential Energy
V = m*g*rRocket(2);
V =  simplify(V);
```

Figure 8 - First half of code for milestone 1



```
%% Force Vector
F1_1 = [0; F1; 0];
F1_0 = R10*F1_1;

F2_1 = [0; F2; 0];
F2_0 = R10*F2_1;
```

```
%% Generalised Forces
Qx = transpose(F1_0)*diff(rF1_0, x) + transpose(F2_0)*diff(rF2_0, x);
Qy = transpose(F1_0)*diff(rF1_0, y) + transpose(F2_0)*diff(rF2_0, y);
Qth = transpose(F1_0)*diff(rF1_0, th) +  transpose(F2_0)*diff(rF2_0, th);
Q = simplify([Qx; Qy; Qth]);
```

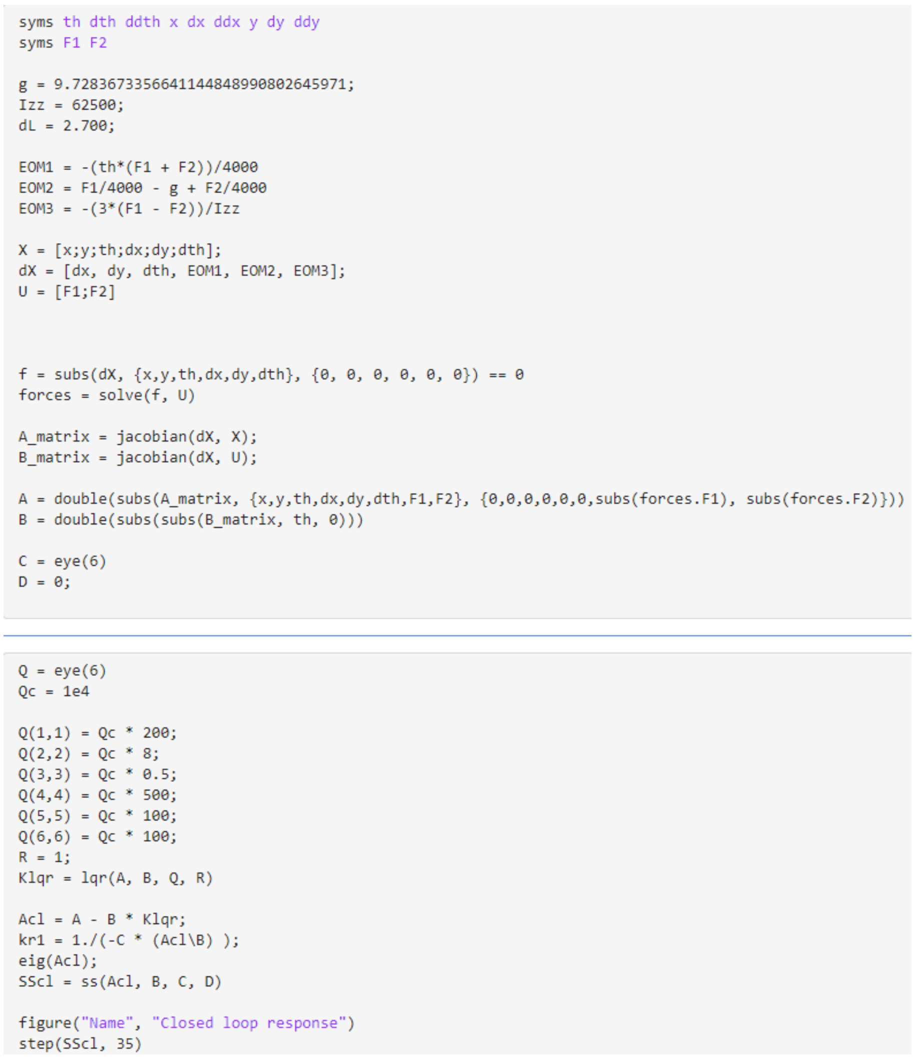
```
%% Matracies
% Mass Matrix
M = hessian(T,dq);

% Coriolis Matrix
dM = deriv(M,q,dq);
C = dM*dq - transpose(jacobian(T,q));
C = simplify(C);

% Gravity Matrix
G = jacobian(V,q);
%Dont know if G needs to be transposed
G = simplify(G);
```

```
%% Equations of Motion
ManipulatorEqn = M*ddq + C + transpose(G) - Q == 0;
% Subsitute values for the side lengths and mass
ManipulatorEqn = subs(ManipulatorEqn, [L1 L2 m], [6 8 4000]);
ddq = [solve(ManipulatorEqn(1), ddx); solve(ManipulatorEqn(2), ddy); solve(ManipulatorEqn(3), ddth)];
disp("ddx ="); disp(ddq(1))
disp("ddy ="); disp(ddq(2))
disp("ddth ="); disp(ddq(3))
```

```
%% Determining G and Izz values
[g, Izz] = solve(ManipulatorEqn(2:3), [g, Izz]) % Solves for g and Izz using the EOMs

% Obtain the values of th and ddq from the simulation.
% Replace these values into the formulas for G and Izz discovered earlier.
% To handle large arrays, select a reduced number of sample points evenly spaced apart.
% Begin sampling after t=0 to prevent potential division errors.
th = th_sim.signals.values(10:100:end);
ddx = ddx_sim.signals.values(10:100:end);
ddy = ddy_sim.signals.values(10:100:end);
ddth = ddth_sim.signals.values(10:100:end);
```

```
%% Constants
L1 = 6; % Width of the lander (m)
L2 = 8; % Height of the lander (m)
m = 4000; % Mass of the lander (kg)
% Replace constant force values.
% These were selected with varying magnitudes to generate a nonzero torque
% on the lander, facilitating its rotation for determining Izz.
F1 = 900; % Thrust of thruster F1 (N)
F2 = 1000; % Thrust of thruster F2 (N)

%Value of g that was seen from simulations - preffered to calculate it
%'live'
% g = 11.1438;

%Using vpa to take the syms value into a double value
g = vpa(subs(g));  % Determine G values
%Using vpa to take the syms value into an int64 value
Izz = abs(vpa(subs(Izz))); % Determine Izz values
```

Figure 9 - Second half of code for milestone 1



```
syms th dth ddth x dx ddx y dy ddy
syms F1 F2

g = 9.7283673356641144848990802645971;
Izz = 62500;
dL = 2.700;

EOM1 = -(th*(F1 + F2))/4000
EOM2 = F1/4000 - g + F2/4000
EOM3 = -(3*(F1 - F2))/Izz

X = [x;y;th;dx;dy;dth];
dX = [dx, dy, dth, EOM1, EOM2, EOM3];
U = [F1;F2]



f = subs(dX, {x,y,th,dx,dy,dth}, {0, 0, 0, 0, 0, 0}) == 0
forces = solve(f, U)

A_matrix = jacobian(dX, X);
B_matrix = jacobian(dX, U);

A = double(subs(A_matrix, {x,y,th,dx,dy,dth,F1,F2}, {0,0,0,0,0,0,subs(forces.F1), subs(forces.F2)}))
B = double(subs(subs(B_matrix, th, 0)))

C = eye(6)
D = 0;
```

```
Q = eye(6)
Qc = 1e4

Q(1,1) = Qc * 200;
Q(2,2) = Qc * 8;
Q(3,3) = Qc * 0.5;
Q(4,4) = Qc * 500;
Q(5,5) = Qc * 100;
Q(6,6) = Qc * 100;
R = 1;
Klqr = lqr(A, B, Q, R)

Acl = A - B * Klqr;
kr1 = 1./(-C * (Acl\B) );
eig(Acl);
SScl = ss(Acl, B, C, D)

figure("Name", "Closed loop response")
step(SScl, 35)
```

Figure 10 - Code used to determine KLQR for controller

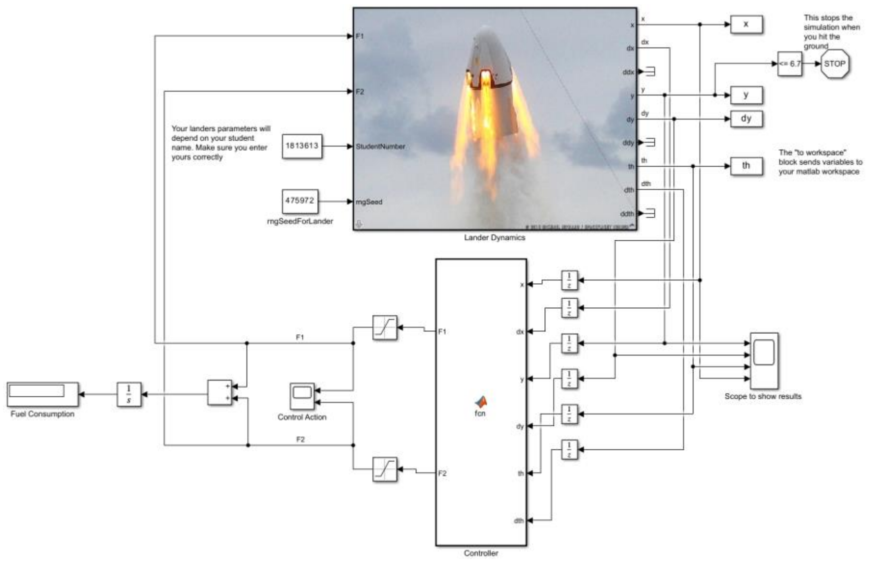
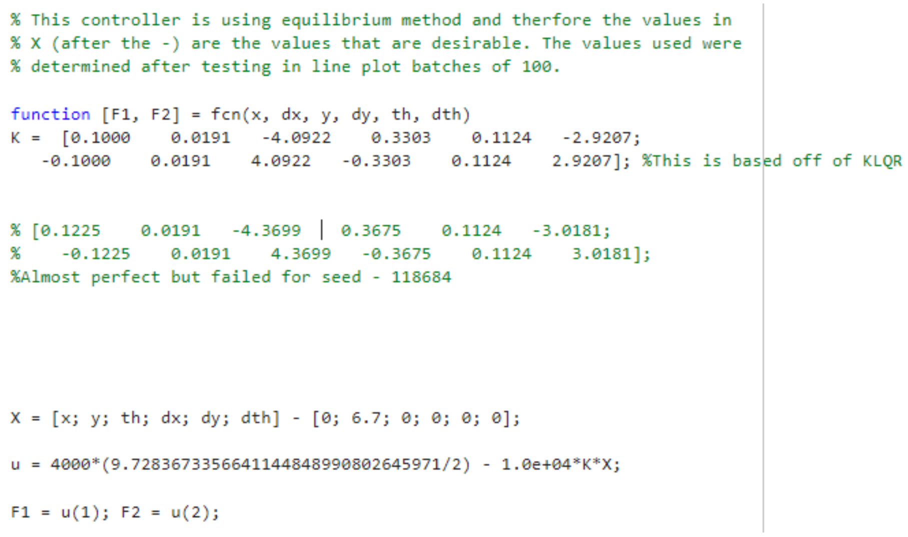

F1
F2
Your landers parameters will depend on your student name. Make sure you enter yours correctly
1813613
StudentNumber
475972
rngSeed
rngSeedForLander
Lander Dynamics
x
dx
ddx
y
dy
ddy
th
dth
ddth
This stops the simulation when you hit the ground
<= 6.7
STOP
The "to workspace" block sends variables to your matlab workspace
Scope to show results
Fuel Consumption
Control Action
fcn
Controller

Figure 11 - State space control simulink model



```
% This controller is using equilibrium method and therfore the values in
% X (after the -) are the values that are desirable. The values used were
% determined after testing in line plot batches of 100.

function [F1, F2] = fcn(x, dx, y, dy, th, dth)
K =  [0.1000    0.0191   -4.0922    0.3303    0.1124   -2.9207;
   -0.1000    0.0191    4.0922   -0.3303    0.1124    2.9207]; %This is based off of KLQR


% [0.1225    0.0191   -4.3699 | 0.3675    0.1124   -3.0181;
%    -0.1225    0.0191    4.3699   -0.3675    0.1124    3.0181];
%Almost perfect but failed for seed - 118684




X = [x; y; th; dx; dy; dth] - [0; 6.7; 0; 0; 0; 0];

u = 4000*(9.72836733566411448489908026459‌71/2) - 1.0e+04*K*X;

F1 = u(1); F2 = u(2);
```

Figure 12 - Code used within the controller block

Seed	Landing speed (m/s)
59250	-2.021852
453704	-2.024578
743655	-2.035632
625364	-2.154876
43835	-2.231833
844571	-2.092866
888011	-2.069261
609881	-2.097451
724748	-2.036040
800460	-2.146630
33158	-2.016423
523113	-2.117916
476890	-2.117070
858483	-2.047475
385794	-2.120683
76455	-2.162360
111999	-2.068996
355235	-2.165996
21736	-2.070018
989218	-2.159615

Table 2 - Failed seeds and their respective landing speeds

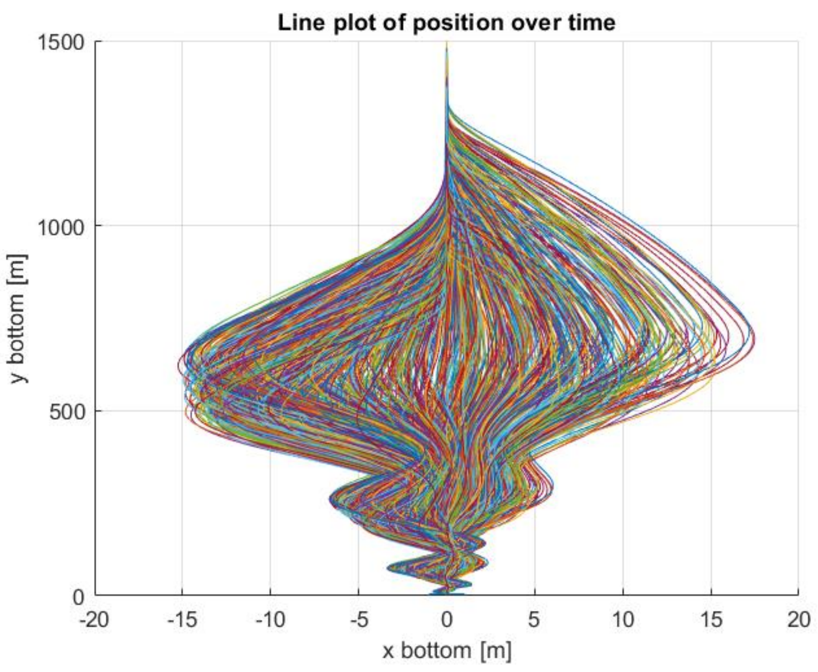


Figure 13 - Line plot of position over time for 5 000 line plots

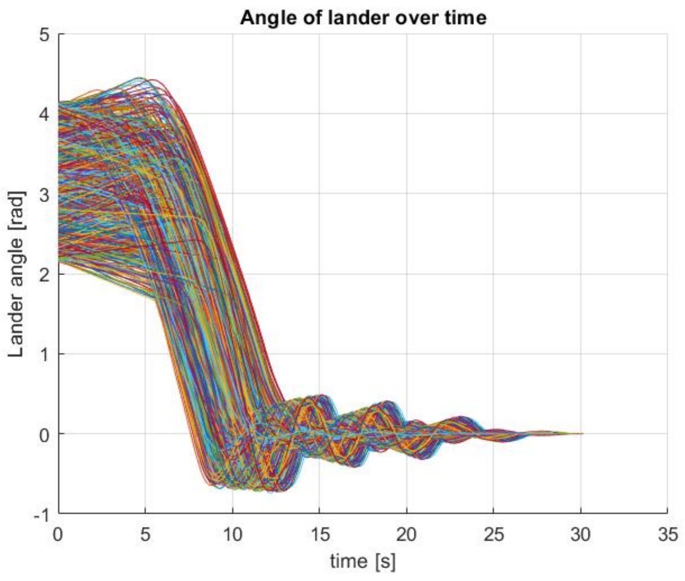


Figure 14 - Angle of lander over time for 5 000 line plots

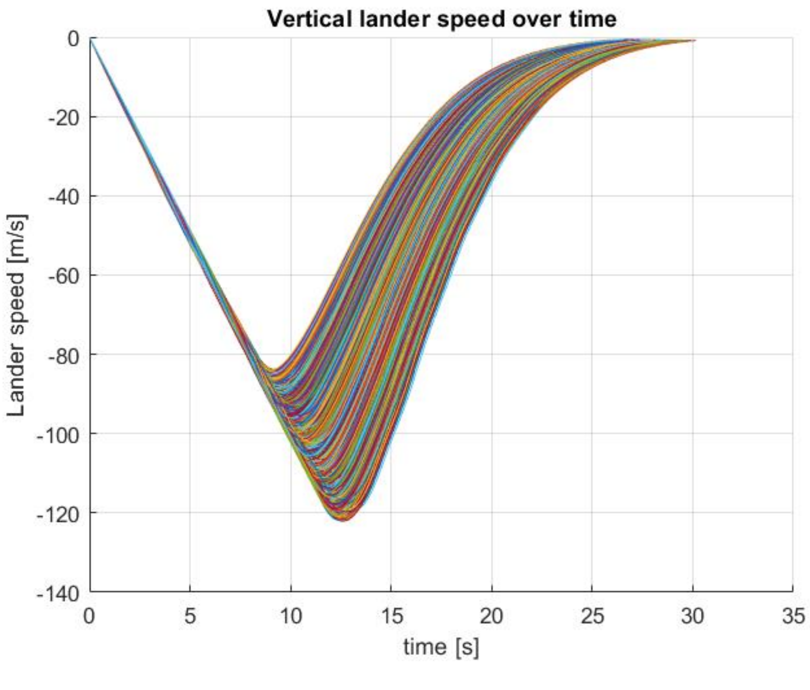


Figure 15 - Vertical lander speed over time for 5 000 line plots

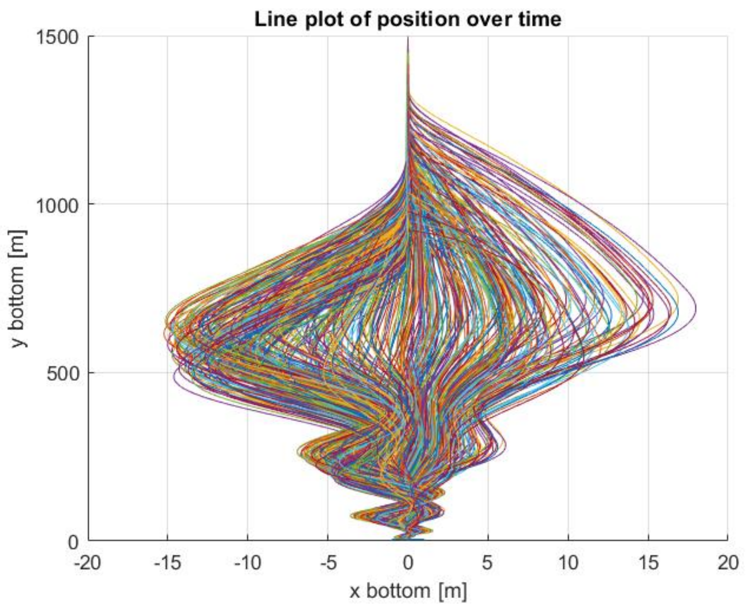


Figure 16 - Line plot of position over time for 500 line plots

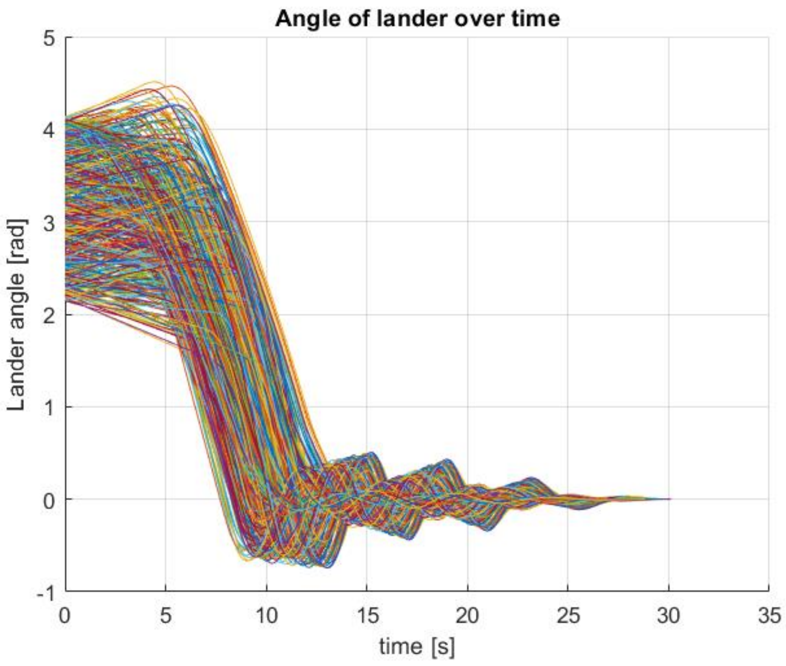


Figure 17 - Angle of lander over time for 500 line plots

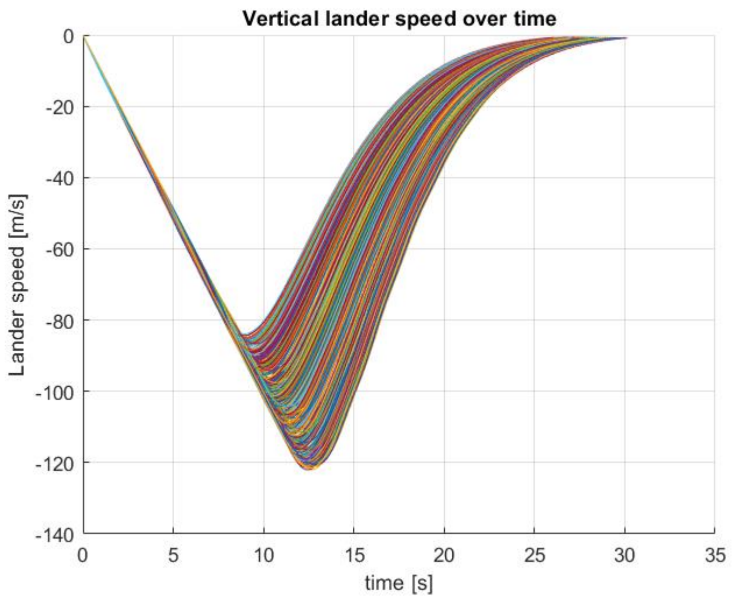


Figure 18 - Vertical lander speed over time for 500 line plots

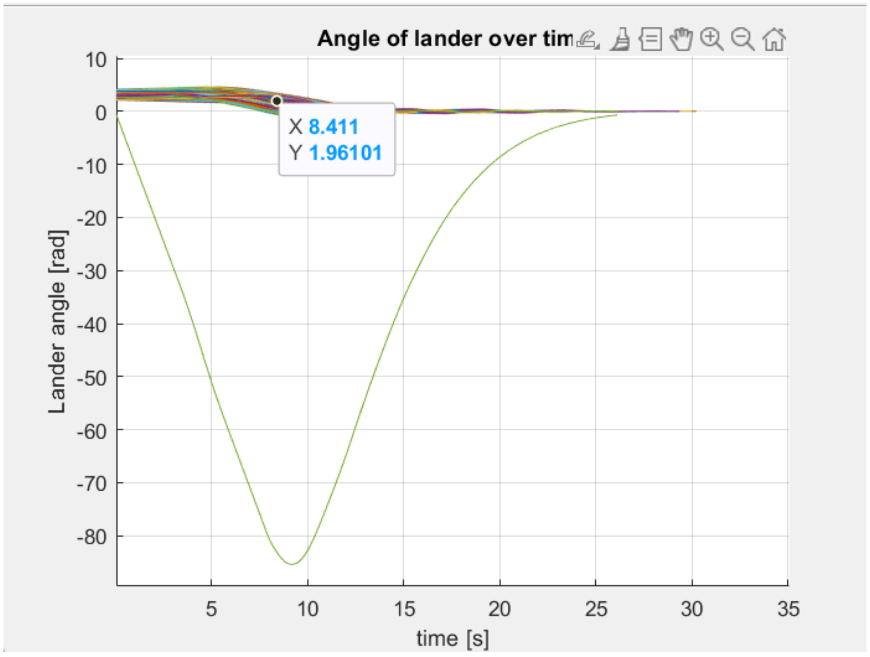


Figure 19 - Strange seed that occurred when testing 10 000 seeds the second time.

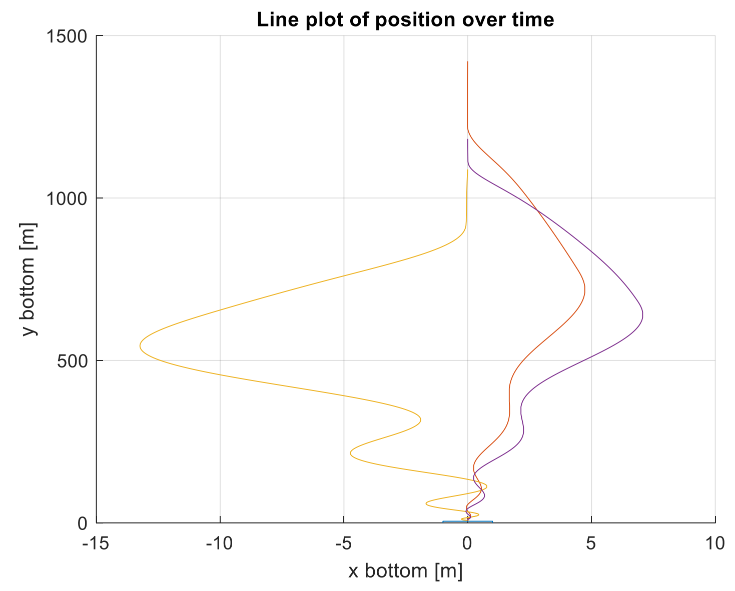


Figure 20 - Line plot of position over time for 3 line plots

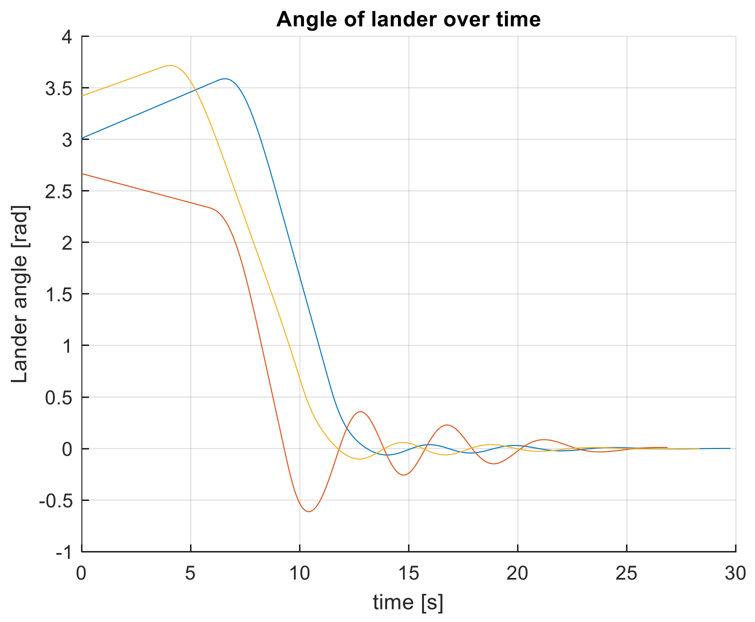


Figure 21 - Angle of lander over time for 500 line plots

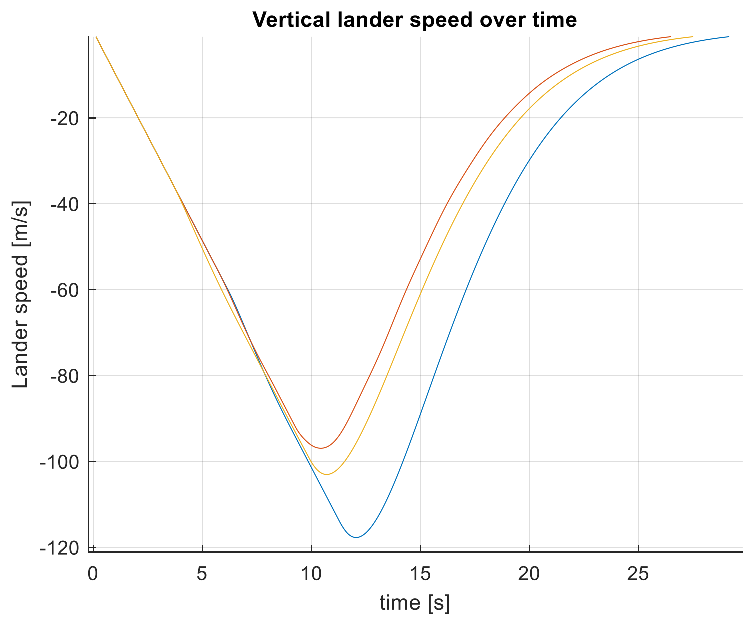


Figure 22 - Vertical lander speed over time for 3 line plots