3801 Lab 4 – Quadrotor Dynamics and Control

November 29th, 2023

Matthew Bradford
Lauren Christenson
Reece Fountain
Jared Steffen

Lab Task 1: Nonlinear Equations of Motion

1.1

See the Appendix for PlotAircraftSim function.

1.2

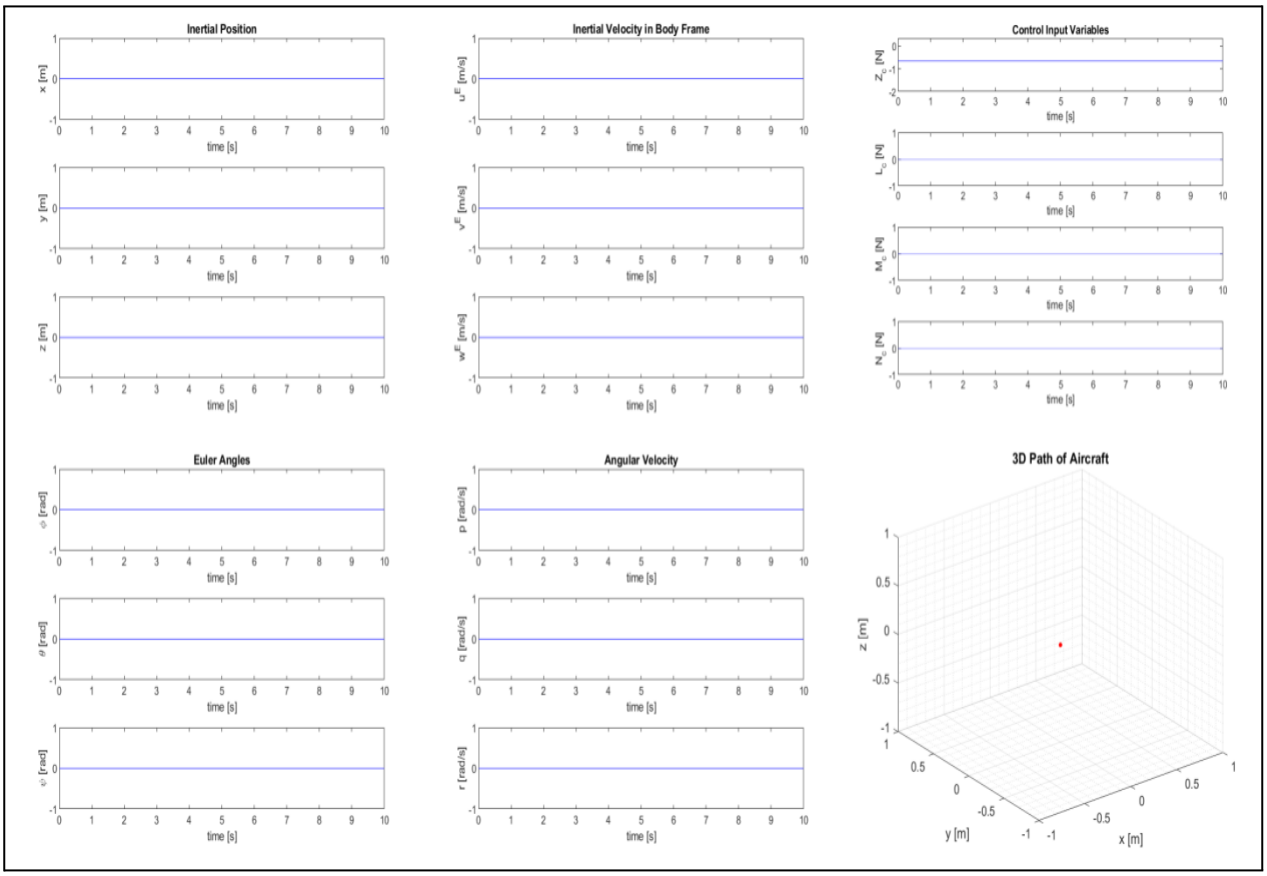

Figure 1: Problem 1.2 Steady Hover without Aerodynamic Forces/Moments

In order to maintain steady hover, the thrust for each motor would be equal to ¼ the weight of the quadcopter. The plots show that there is no translation or rotation being performed, therefore the quadrotor is in steady hover.

1.3

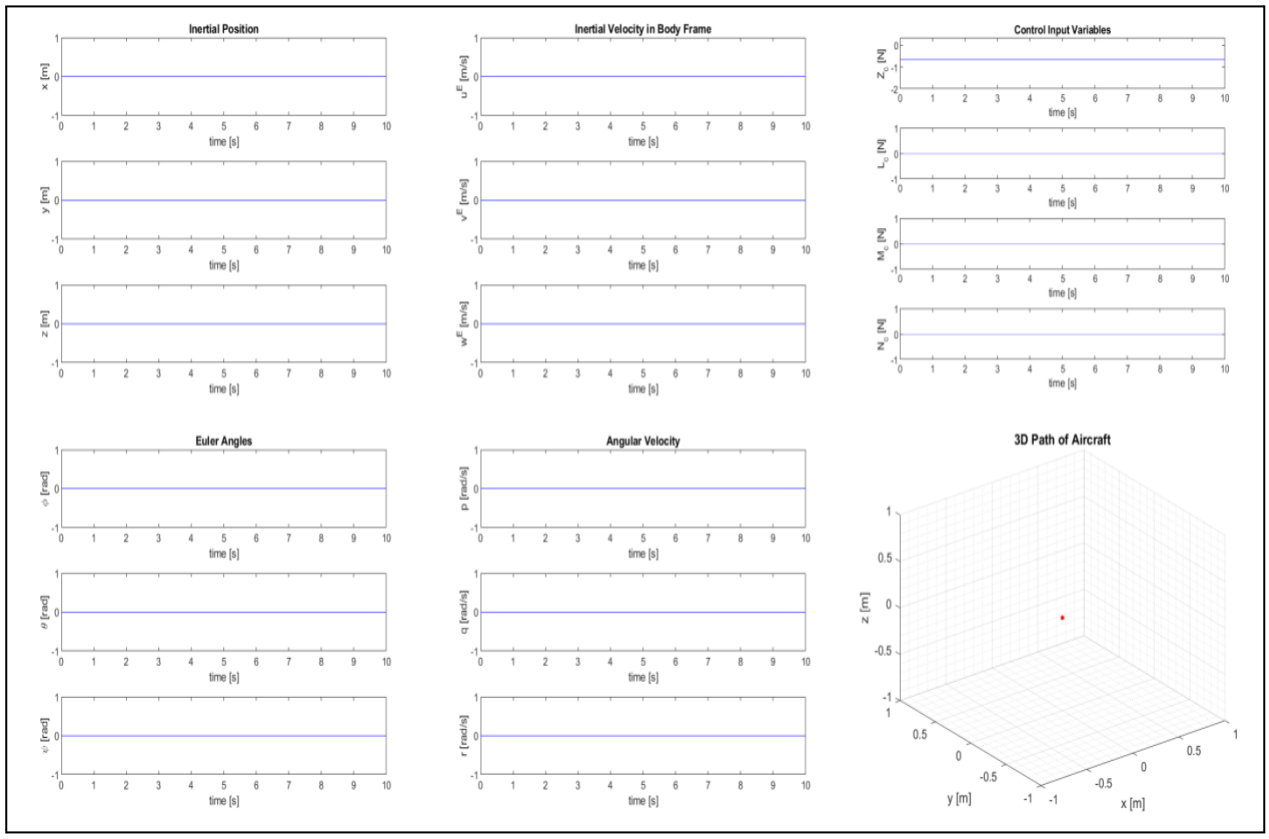

Figure 2: Problem 1.3 Steady Hover with Aerodynamic Forces/Moments

Plots before and after introducing these changes showed no difference. This is because the quadrotor is not moving in its stead hovering state, and so no aerodynamic forces or moments are created, and its velocities, and therefore position, all remain at zero.

1.4

From the derivations below, it can be seen that roll angle needs to be an angle such that the net force in the z direction is zero and the thrust in the y direction needs to be equal to the drag at 5 m/s. To do this, the y and z components were broken up into a system of equations which was then solved to find trim state and thrust.

Additionally for a yaw angle equal to 90°, the above statement is also true, with the exception that the x and z components were used. However, the pitch angle would be negative by convention.

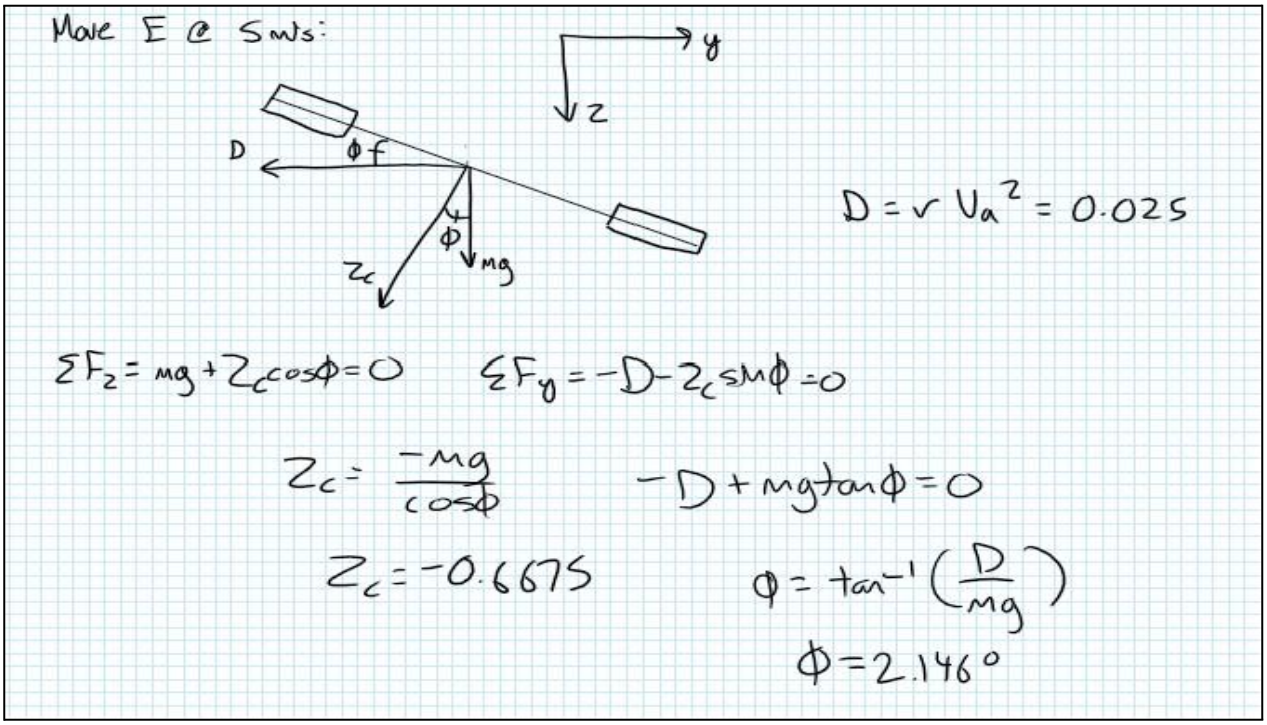

Figure 3a: Derivation of Roll Angle for Moving 5 m/s East with 0° Yaw

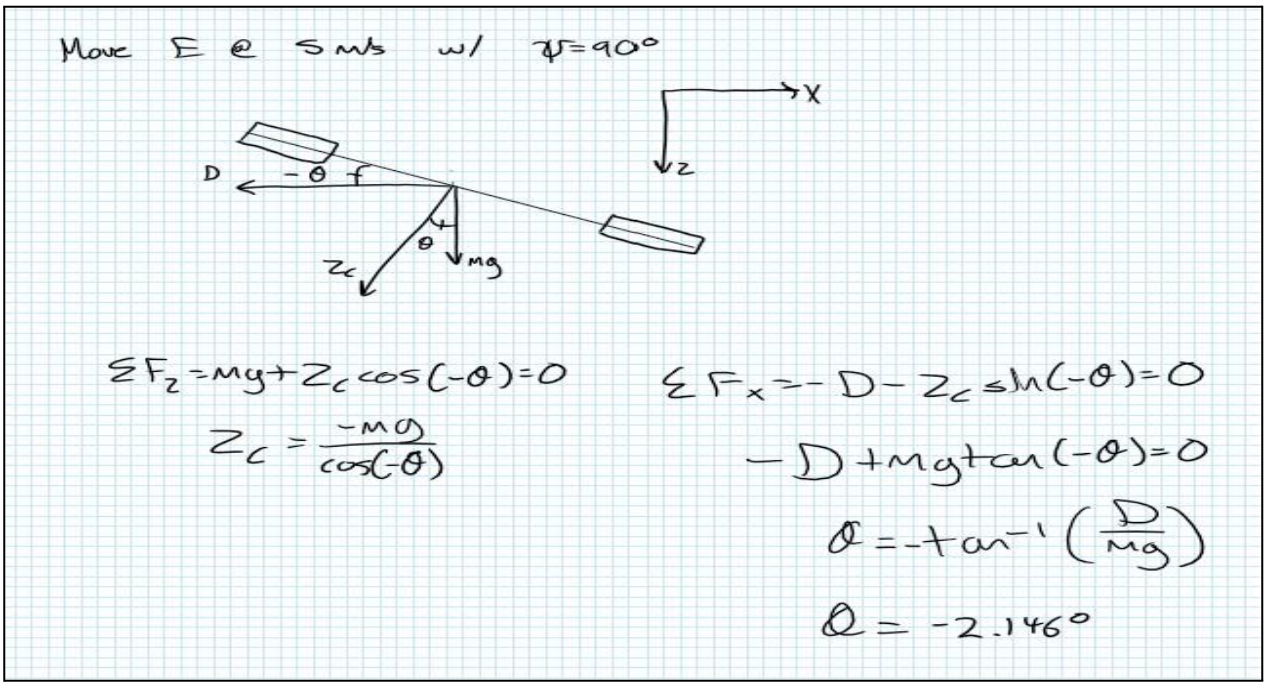

Figure 4a: Derivation for Pitch Angle for Moving 5 m/s East with 90° Yaw

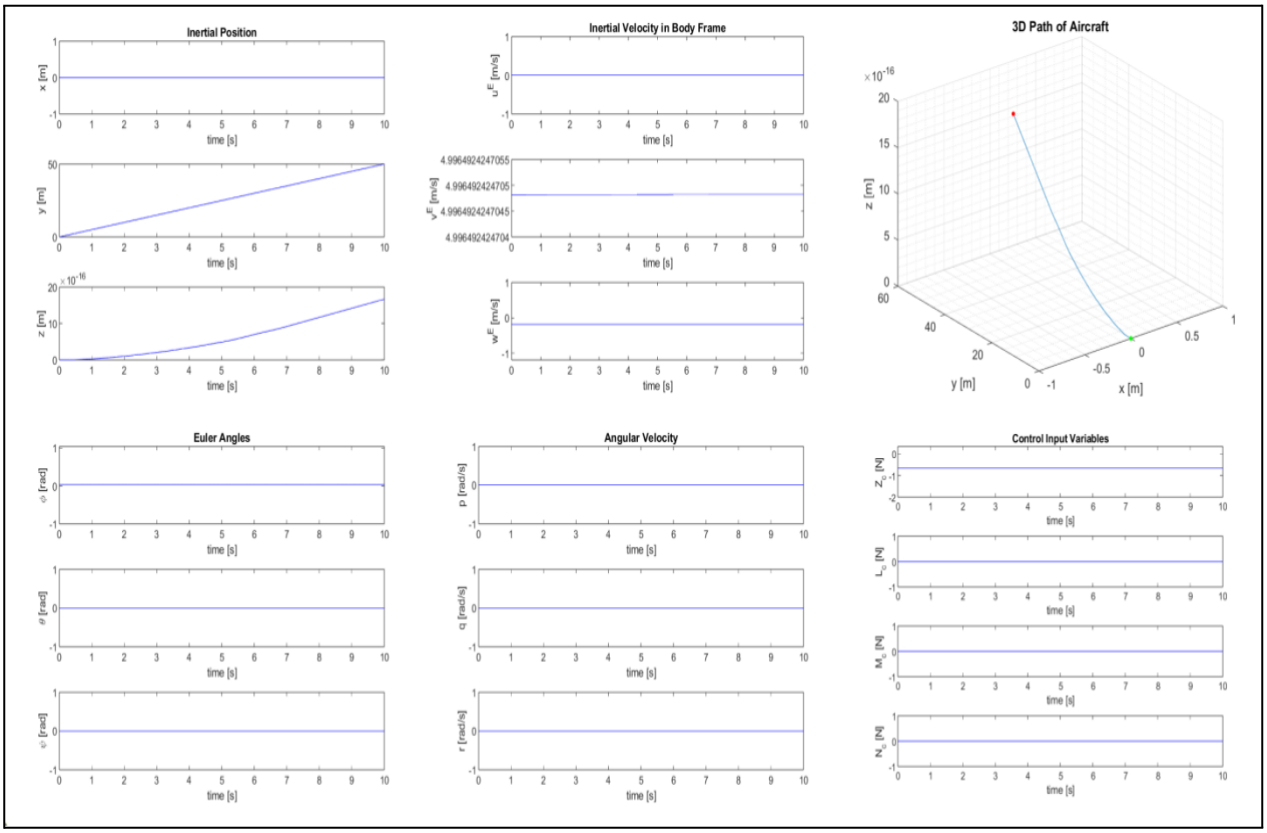

Figure 3b: Problem 1.4 Moving East at 5 m/s with 0° Yaw

The trim state of moving East at 5 m/s can best be verified by the inertial position and inertial velocity in body frame graphs in figure 3b. We see that after 10 seconds, the y inertial position is 50 meters and that the inertial velocity in the body y axis is maintained at ~5 m/s for the whole duration.

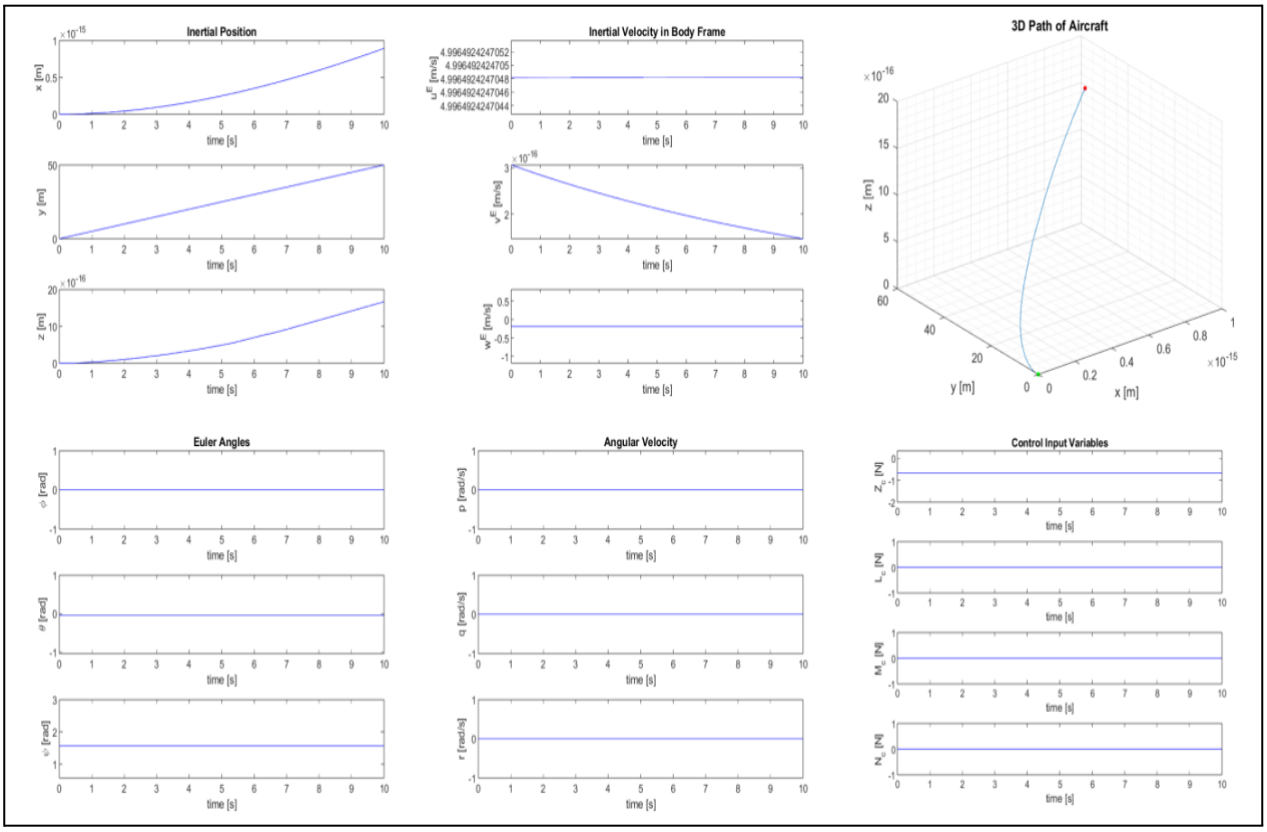

Figure 4b: Problem 1.4 Moving East at 5 m/s with 90° Yaw

The trim state of moving East at 5 m/s can best be verified by the inertial position and inertial velocity in body frame graphs in figure 3b. We see that after 10 seconds, the x inertial position is 50 meters and that the inertial velocity in the body y axis is maintained at ~5 m/s for the whole duration. Note that for both of the above sets of graphs, some axis are on the magnitude of $\sim 10^{-15}$, so they are considered to be zero.

1.5

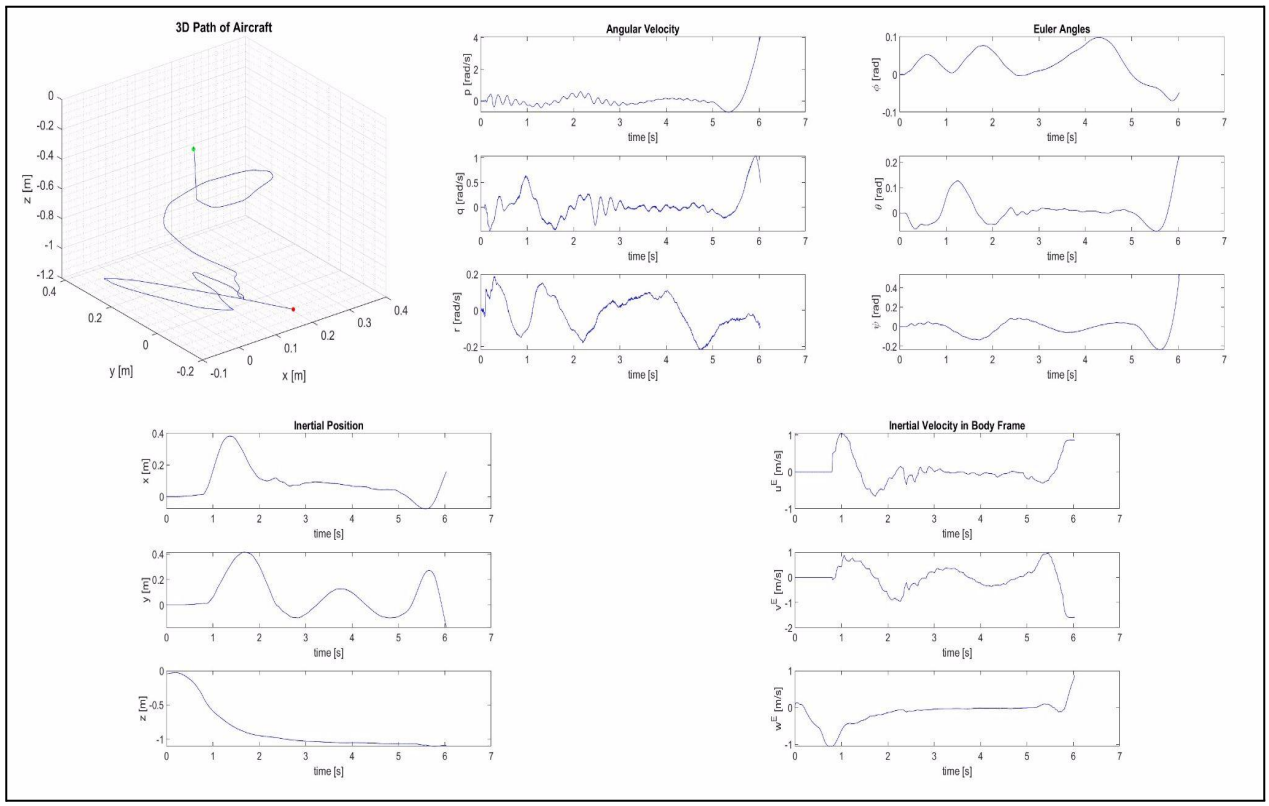

Figure 5: Problem 1.5 Given Data without Control Implementation

Steady hovering flight is not stable for the quadcopter once control is removed. When a small disturbance was introduced during the hardware demonstration it quickly grew in magnitude and caused a crash. In the plots below you can see that angular velocity rapidly oscillates when controls are removed, causing changes in the Euler angles that, alongside the translational motion, lead to the crash. This is also shown in the model we developed, in which any sort of small disturbance along multiple axes causes a degeneration of the stability and a crash. Obviously our model does not show minor air currents in the room that would provide this instability, but when we manually add disturbance this reaction is shown.

Lab Task 2: Linear Equations and Simple Control

2.1

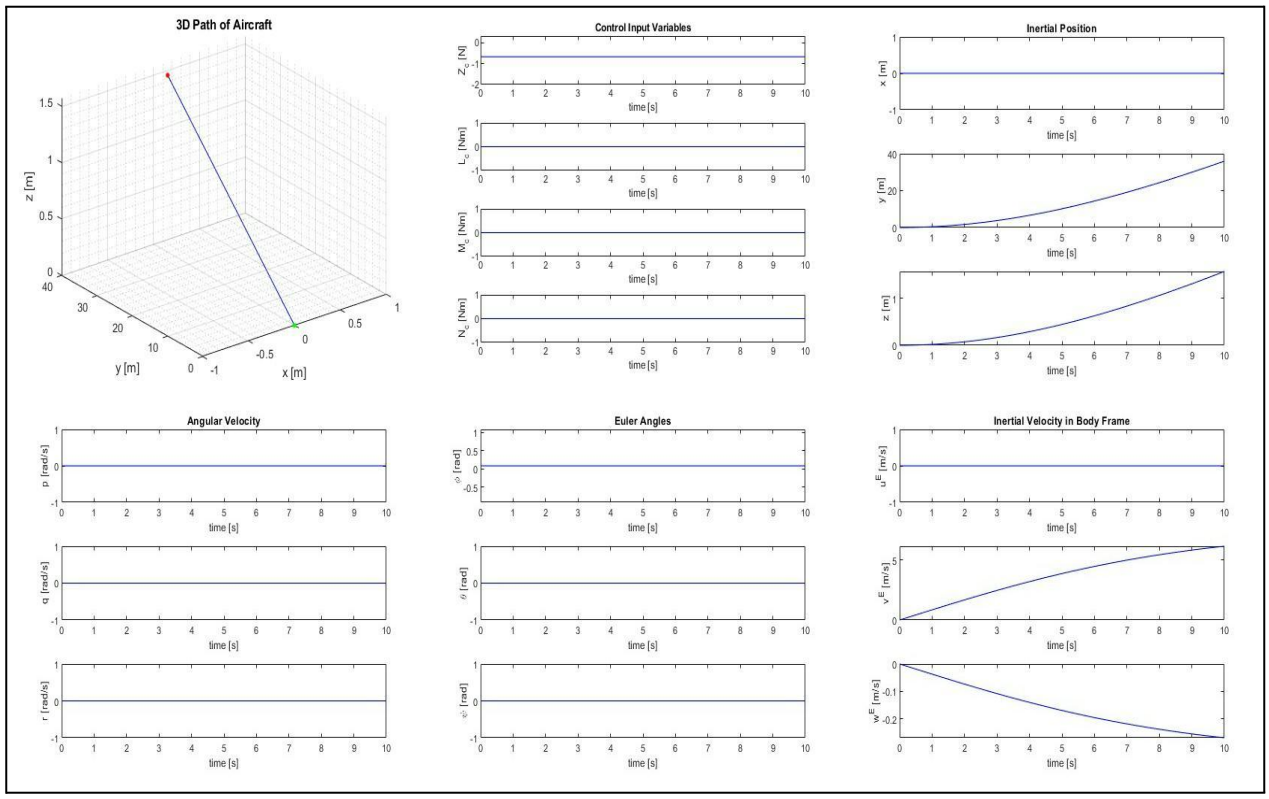

Figure 6: Problem 2.1 Deviation of +5° Roll

From our nonlinear equations of motion, we can see that a deviation of roll will cause a change in the y velocity which will then cause a change in the y position. This makes physical sense as the drone isn't level and therefore has a y component of lift.

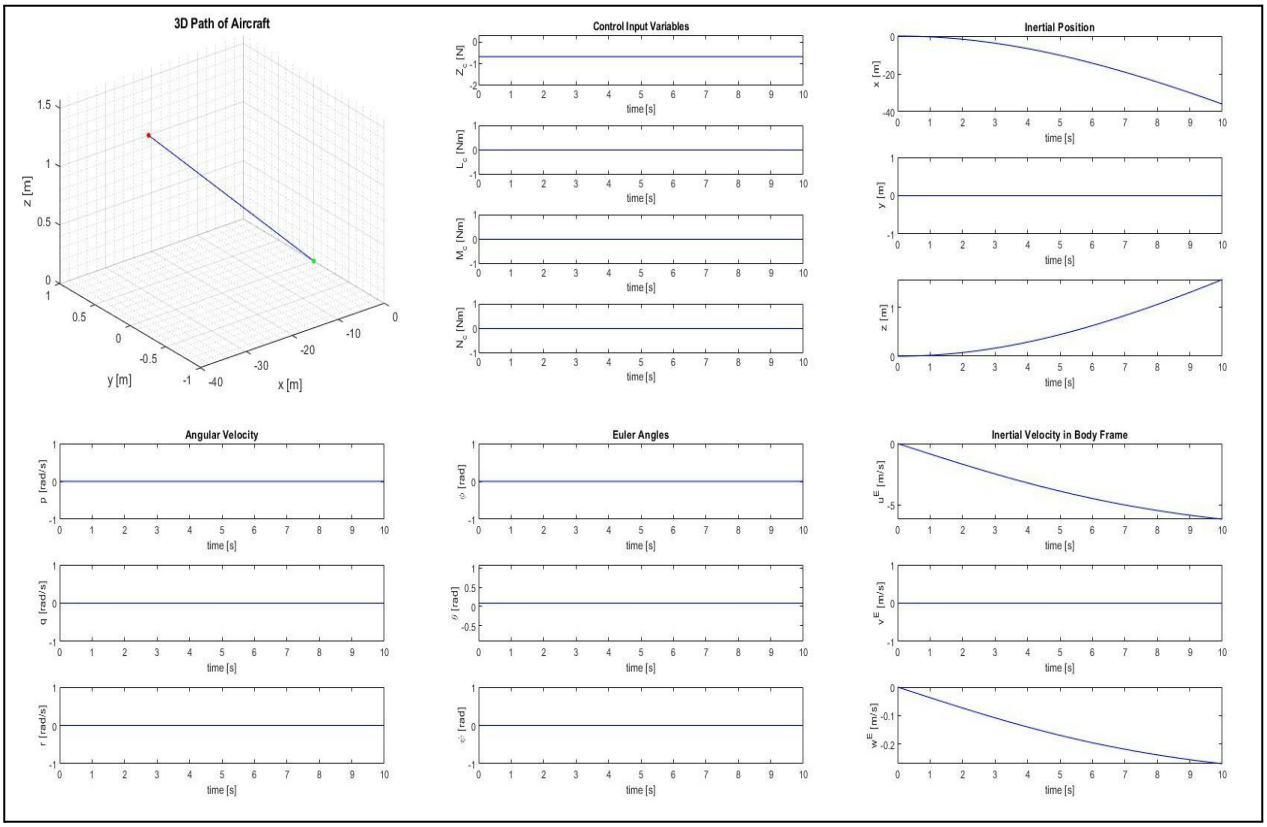

Figure 7: Problem 2.1 Deviation of +5° in Pitch

Our nonlinear equations motion tells us that a small change in pitch will cause a change in the x velocity which also changes the x position. This makes sense as the drone isn't level and therefore has an x component of lift.

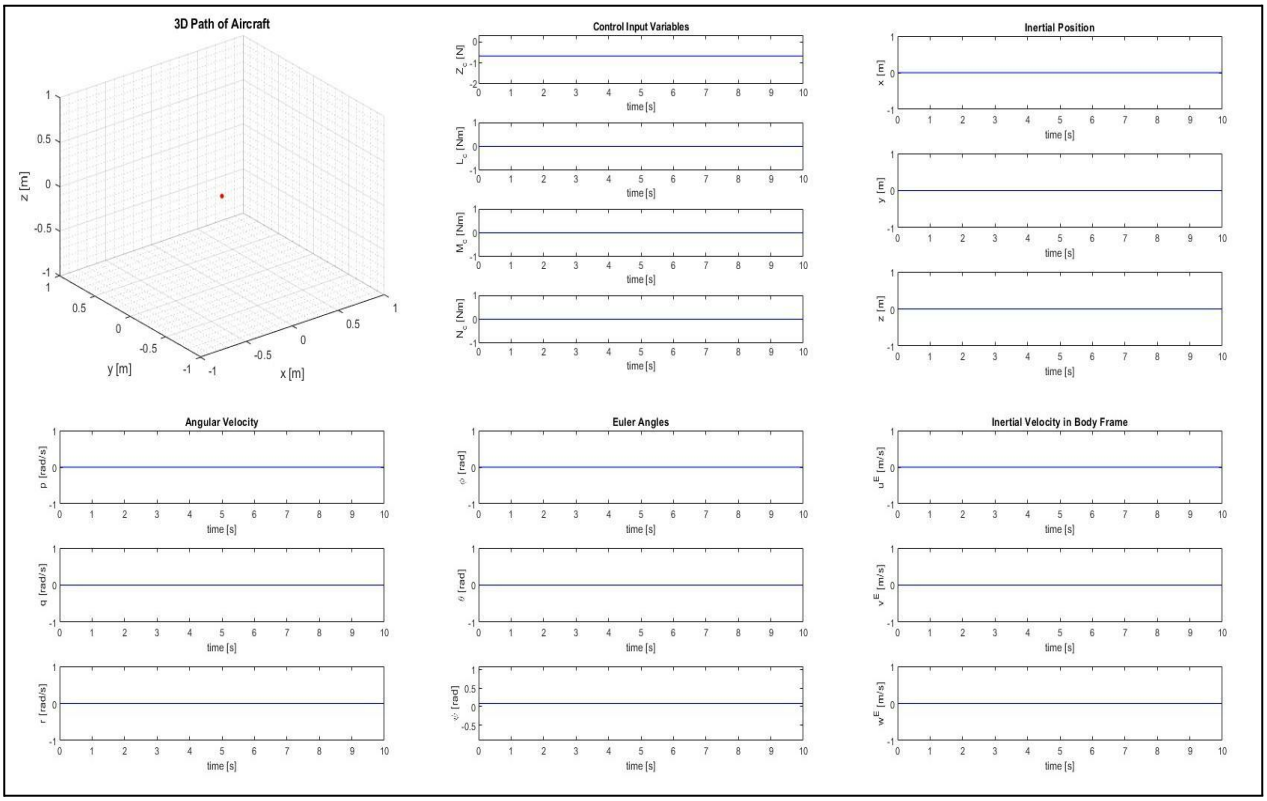

Figure 8: Problem 2.1 Deviation of +5° in Yaw

In the nonlinear equations of motion, a yaw deviation doesn't impact any other variables in the drone's motion. The drone was rotated about the z axis only and therefore is still in trim.

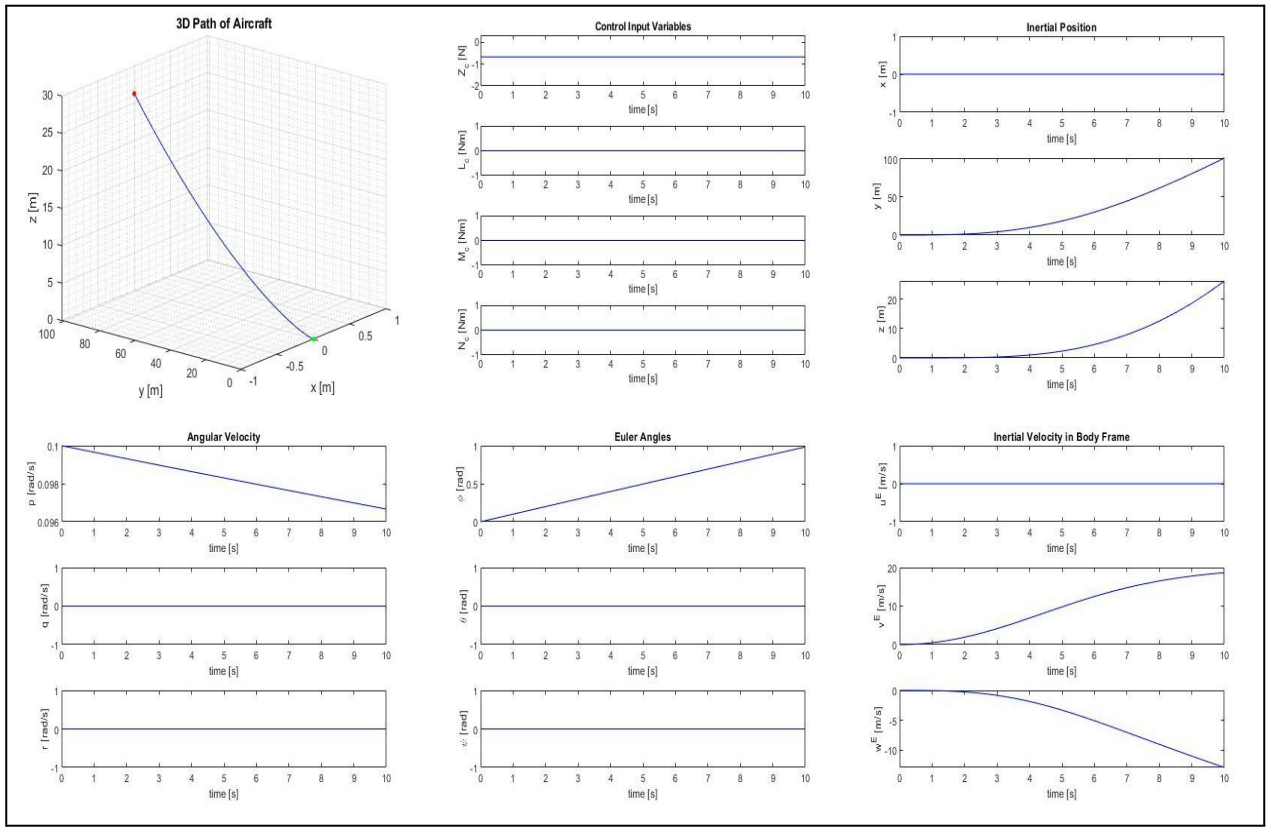

Figure 9: Problem 2.1 Deviation of +0.1 rad/s in Roll Rate

From our nonlinear equations of motion, a deviation in roll rate affects both the y and z velocities. This then causes a change in the y and z positions. This makes physical sense because if the drone is rolling, then not all of the lift points upward, causing the drone to drop. This y component of lift also causes the drone to translate in the y direction.

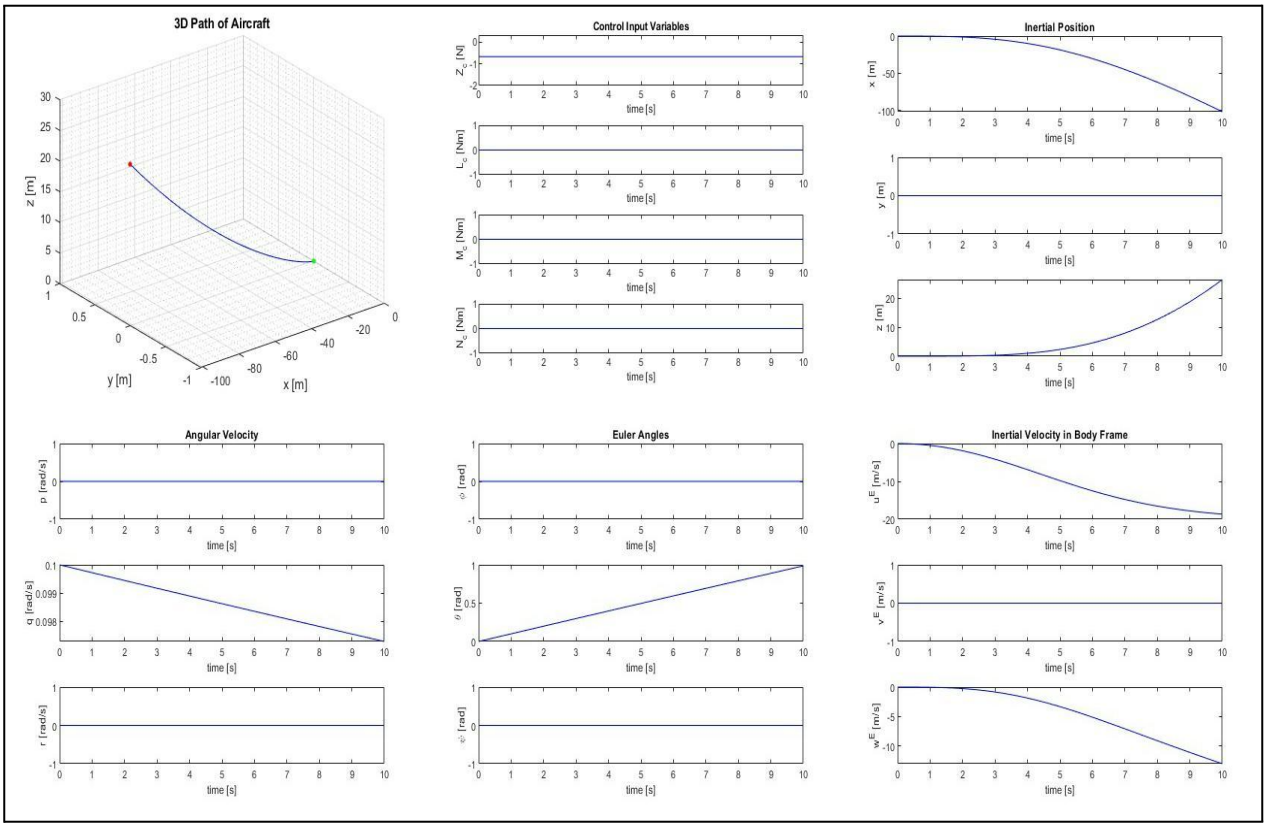

Figure 10: Problem 2.1 Deviation of +0.1 rad/s in Pitch Rate

From our nonlinear equations of motion, a deviation in roll rate affects both the x and z velocities. This then causes a change in the x and z positions. This makes physical sense because if the drone is rolling, then not all of the lift points upward, causing the drone to drop. This x component of lift also causes the drone to translate in the x direction. Physically, this is identical to a change in roll rate only it is about a different axis.

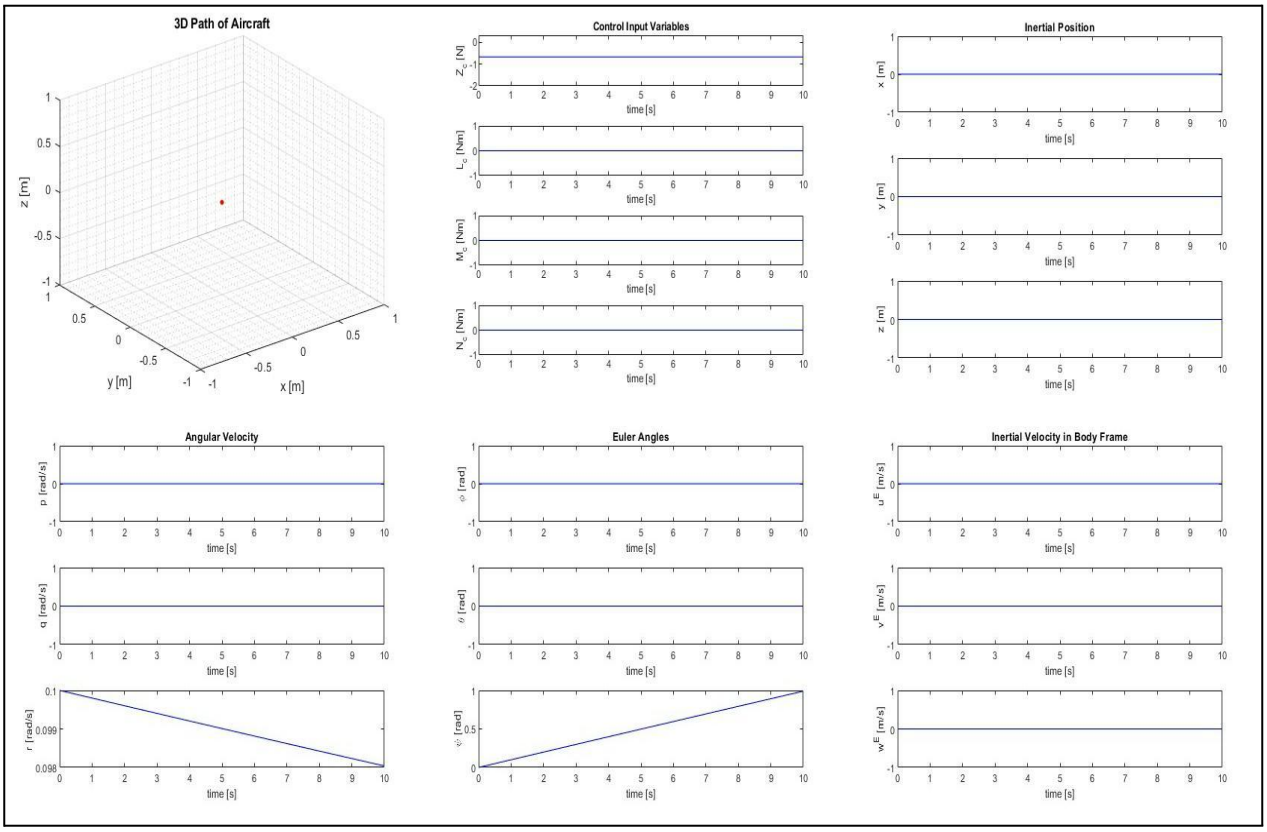

Figure 11: Problem 2.1 Deviation of +0.1 rad/s in Yaw Rate

A deviation in yaw rate doesn't impact any other state variables (other than yaw) as the drone itself is still level in hover. The only control input is a torque in the opposite direction, stopping the rotation.

The above sets of graphs for various deviations in trim conditions do make sense and also support the conclusions drawn from 1.5. With the exception of yaw, any of the perturbations from trim lead to the quadrotor being unable to return to trim. Since yaw angle is the only parameter that does not matter for trim, this makes sense. This would mean that the quadrotor is not stable in steady hover without any control systems.

2.2

Blue is nonlinear

Red dotted is linear

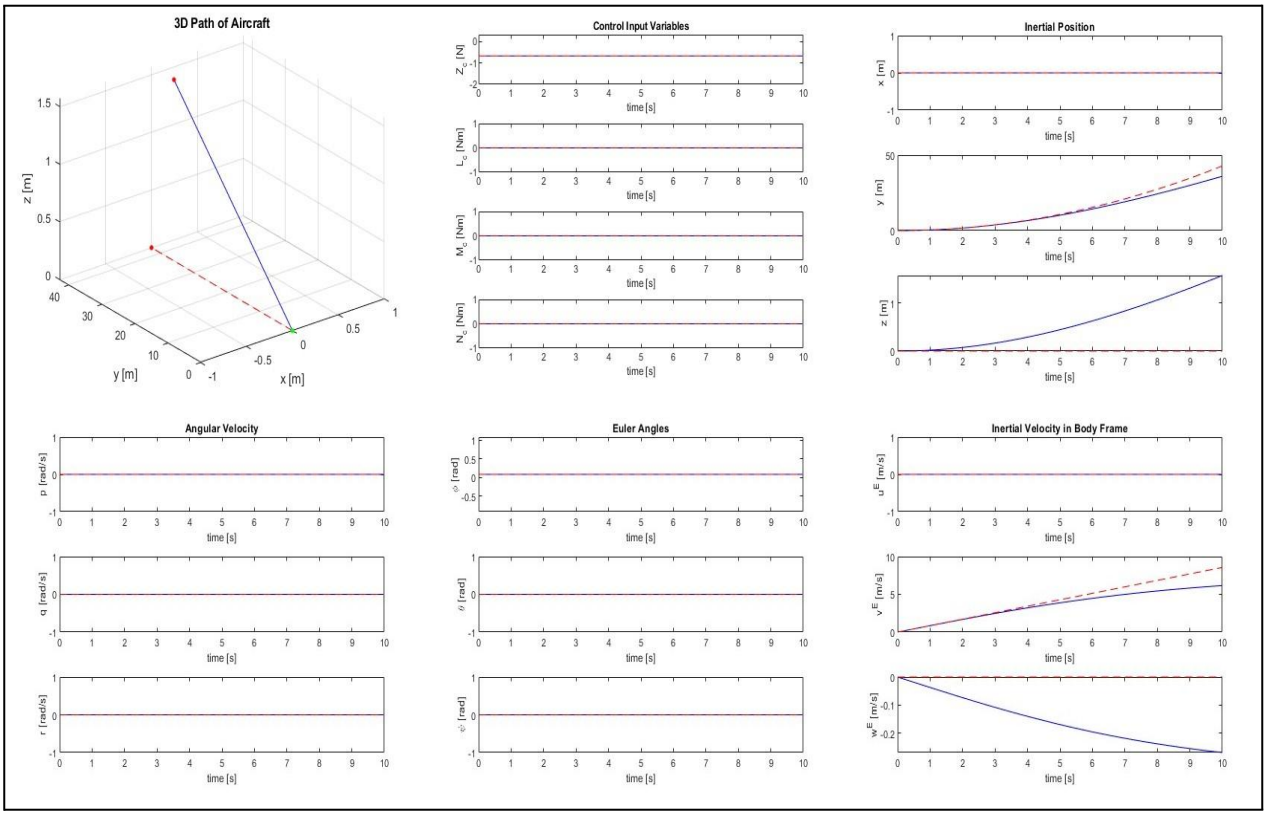

Figure 12: Problem 2.2 Deviation of +5° Roll

Above we see that everything is relatively accurate except for inertial velocity and position in the inertial y direction, which diverges around the 4 second mark because the velocity is not accurately calculated by the linearized equations, as well as vertical velocity and position, as the linearized equations do not result in values for them at all.

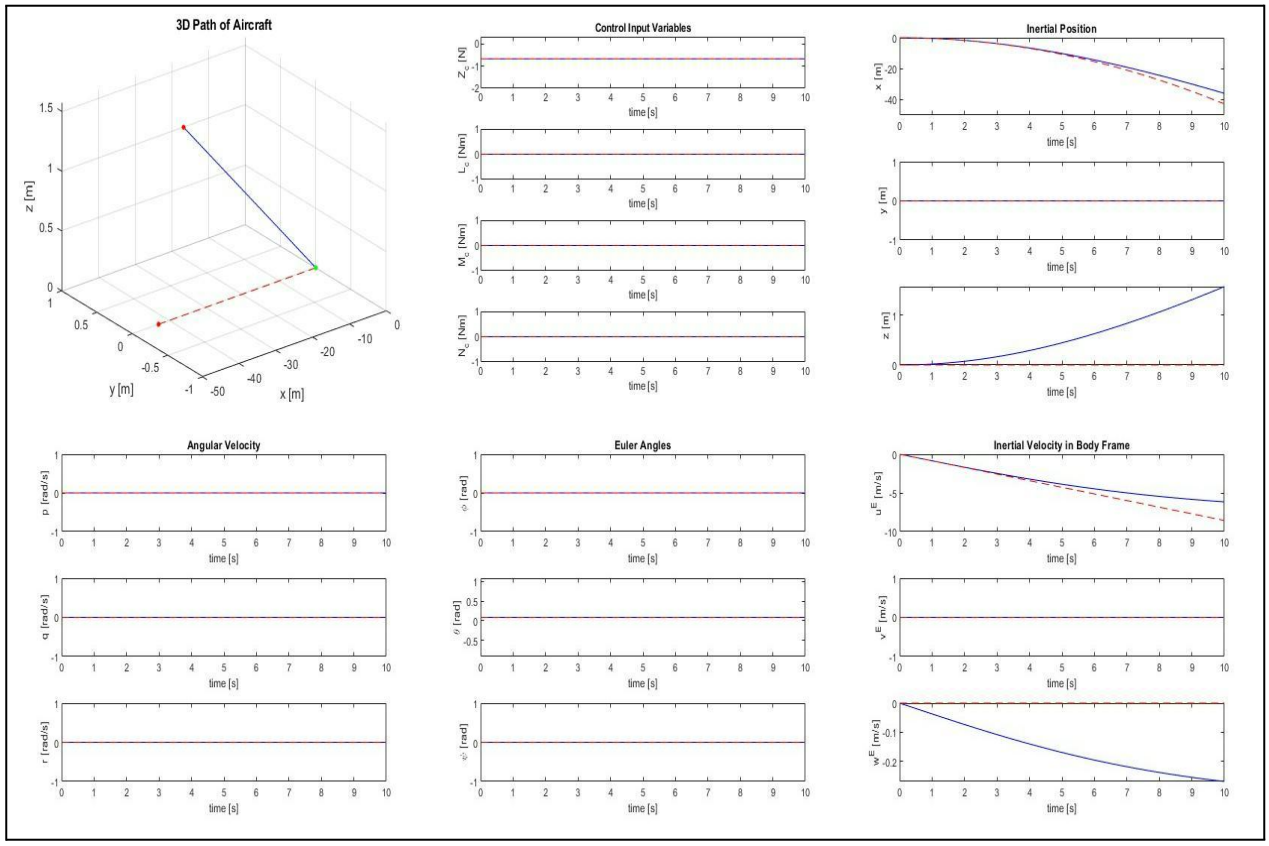

Figure 13: Problem 2.2 Deviation of +5° Pitch

The story is similar here, in which everything but inertial positions and velocities in the x and z directions are represented the same by both linear and nonlinear equations. In the inertial x direction, we see accurate results until around the 4 second mark, due to the linear equations not adjusting over time accurately. In the inertial z direction, we see no response in the linear equations.

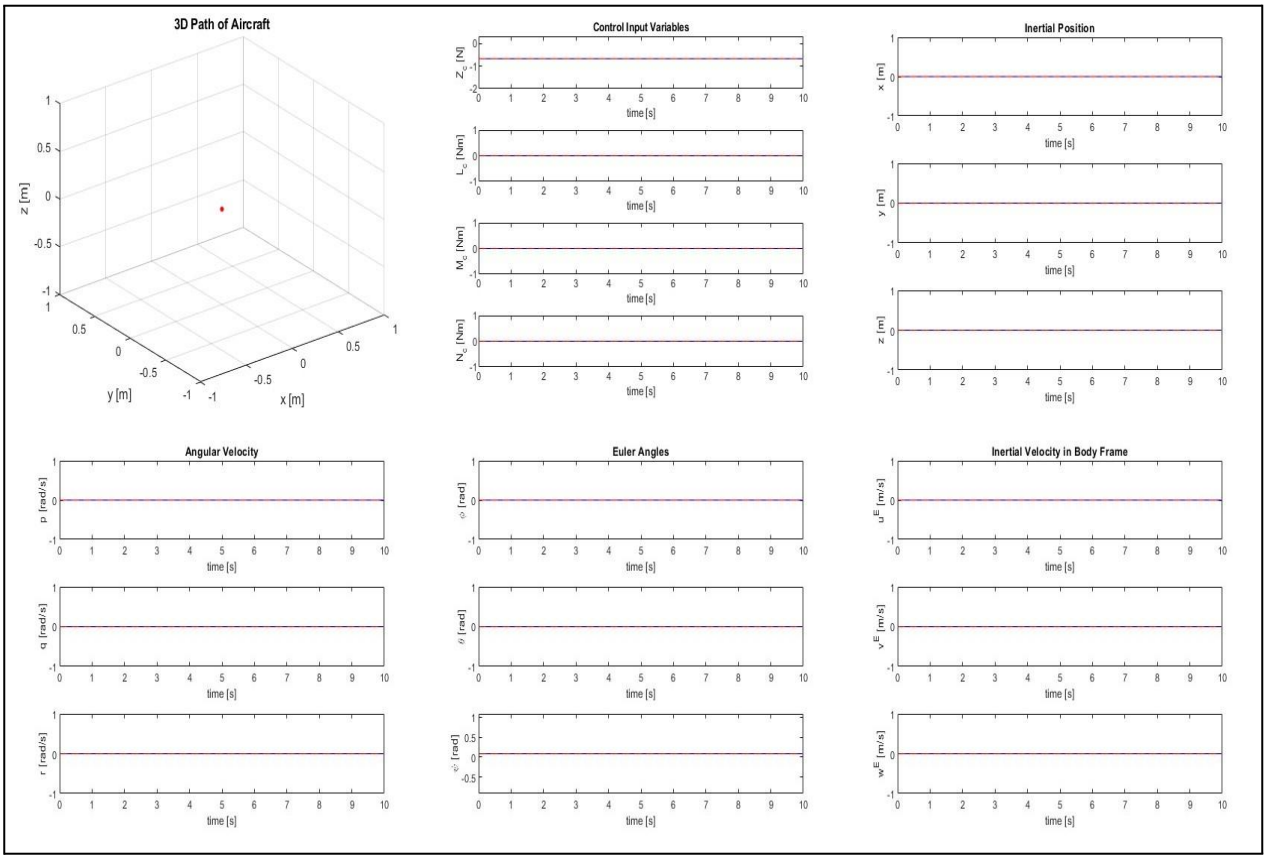

Figure 14: Problem 2.2 Deviation of +5° Yaw

The impacts of yaw deviations on nonlinear and linearized motion are accurate when they are the only changes, because it does not involve velocity changes. When velocity changes occur the linearized equations rapidly fall out of accuracy, especially in the inertial z direction, while in the inertial x and y directions the linearized equations are accurate for around the first 4 seconds.

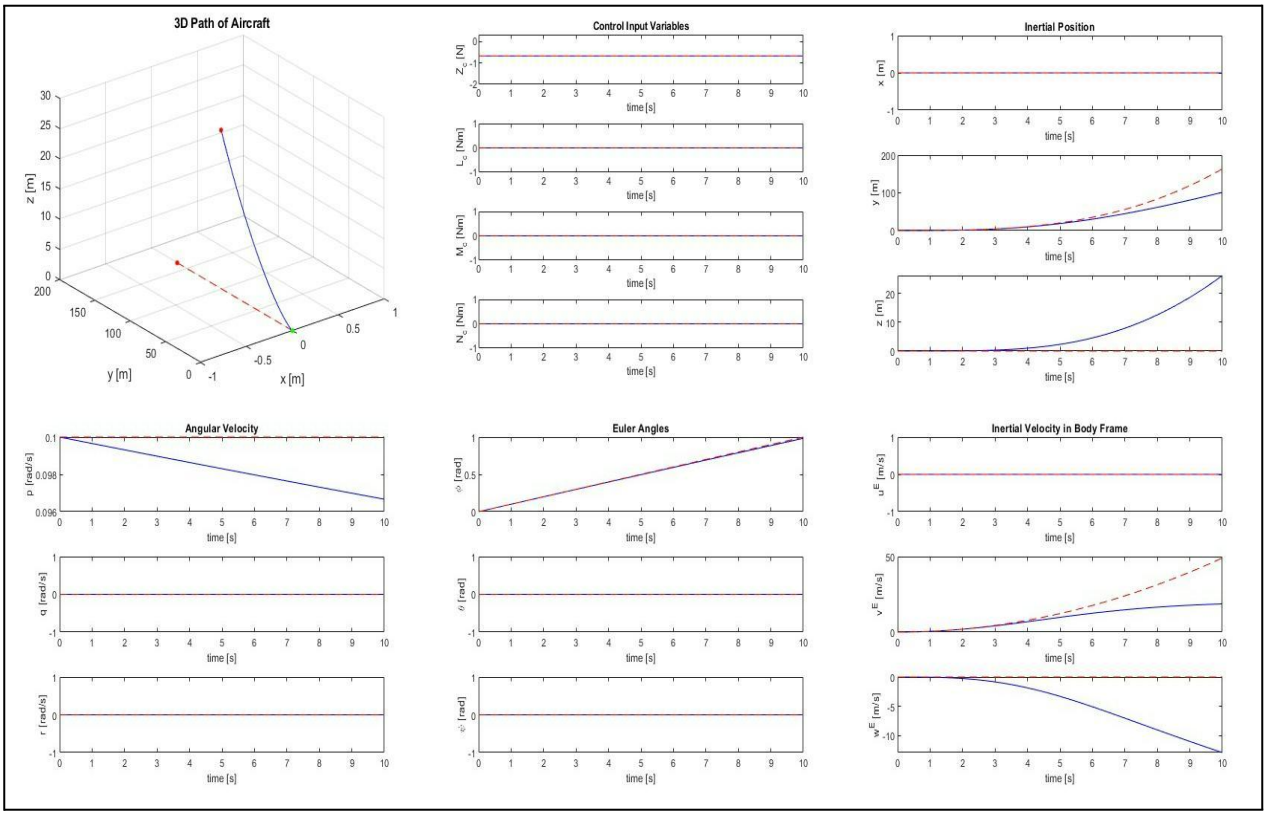

Figure 15: Problem 2.2 Deviation of +0.1 rad/s Roll Rate

In roll rate deviation, we see a very small difference between linear and nonlinear equations in the roll rate because there is no estimated drag, and because of this very little difference in euler angle value, but we see a large difference after 4 seconds in the inertial y direction velocity and position, as the linearized equations overestimate the change in velocity. In the inertial z direction velocity, once again see no change in velocity in the linearized estimations, causing a large inaccuracy after 2 seconds.

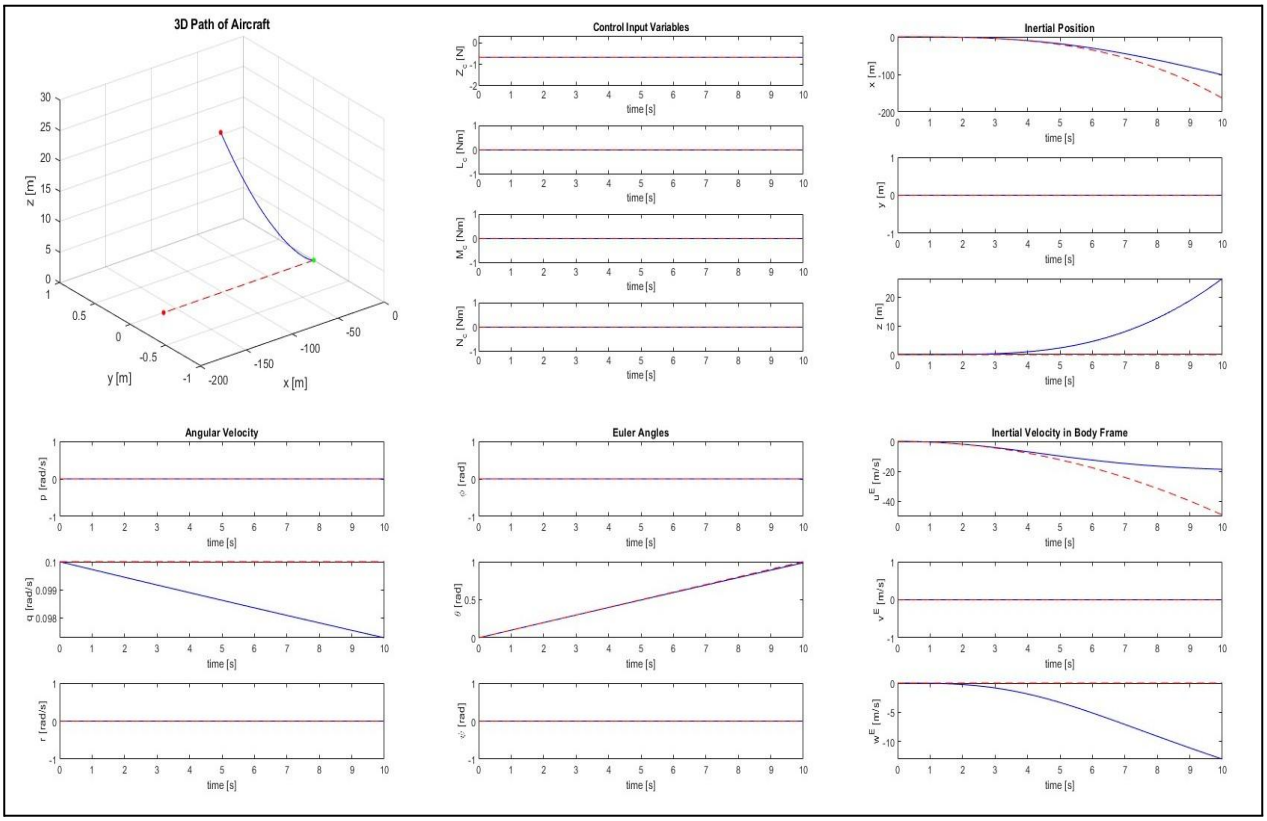

Figure 16: Problem 2.2 Deviation of +0.1 rad/s Pitch Rate

In pitch rate deviation, because of the lack of drag in the linearized equations we find very minor differences in pitch rate over time, and because of this low Euler Angle deviation, while finding large differences in inertial x velocity and position after 4 seconds. There is also large deviation in the inertial z velocity and position in the linearized equations after 2 seconds.

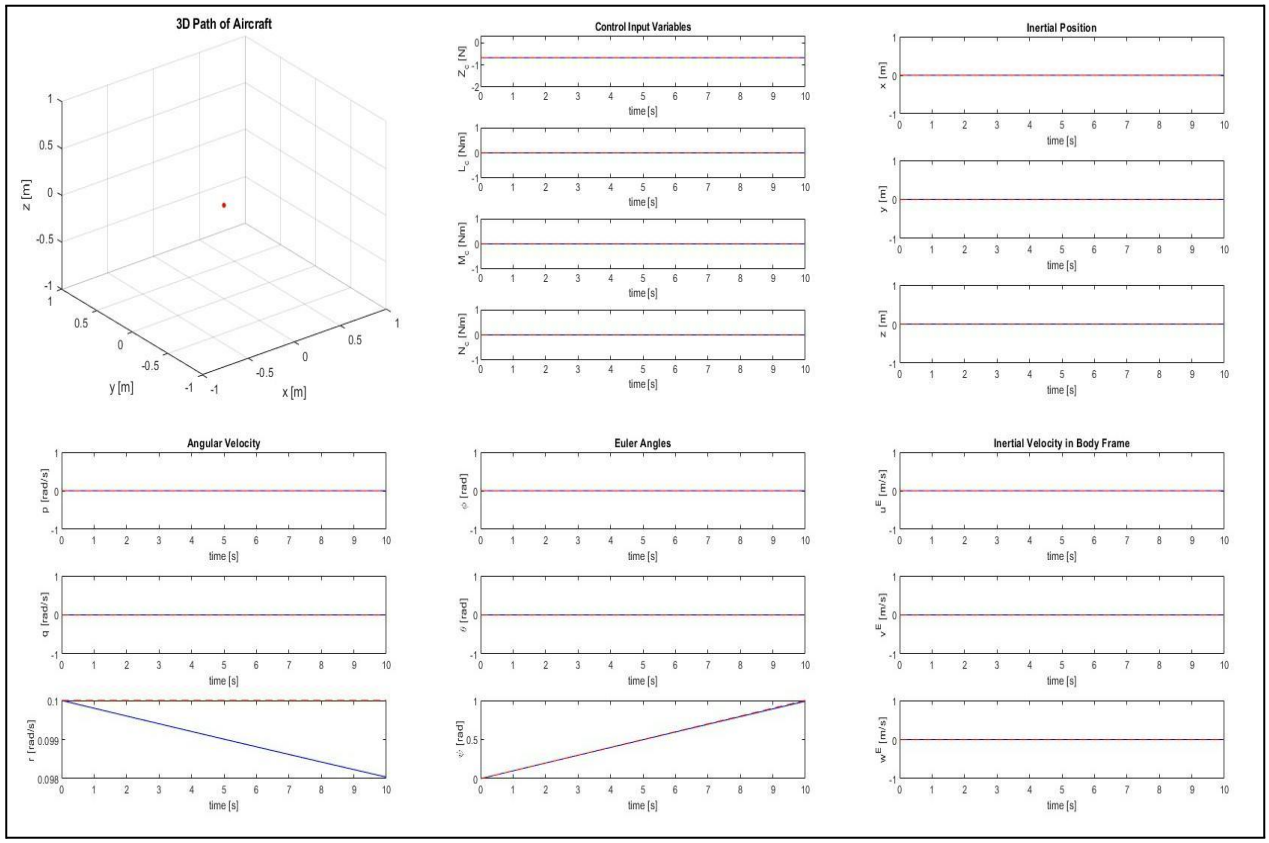

Figure 17: Problem 2.2 Deviation of +0.1 rad/s Yaw Rate

In yaw rate deviation, the same error due to poor drag estimations occurs in yaw rate and yaw angle, but otherwise values are identical due to yaw having zero impact on either linearized or nonlinear equations when it is the only deviation.

2.3

See the Appendix for RotationDerivativeFeedback function.

2.4

See the Appendix for ComputeMotorForces function.

2.5

Blue is controlled

Red dotted is uncontrolled

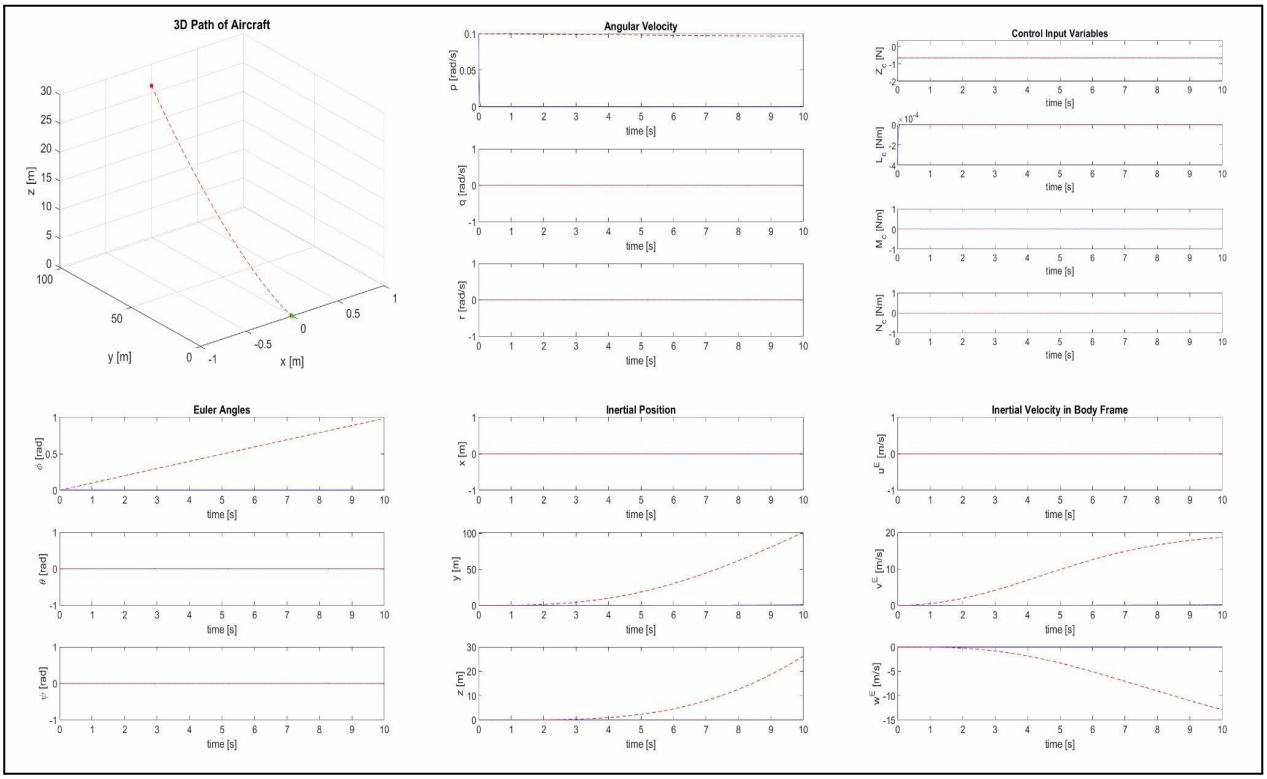

Figure 18: Problem 2.5 Deviation of +0.1 rad/s in Roll Rate

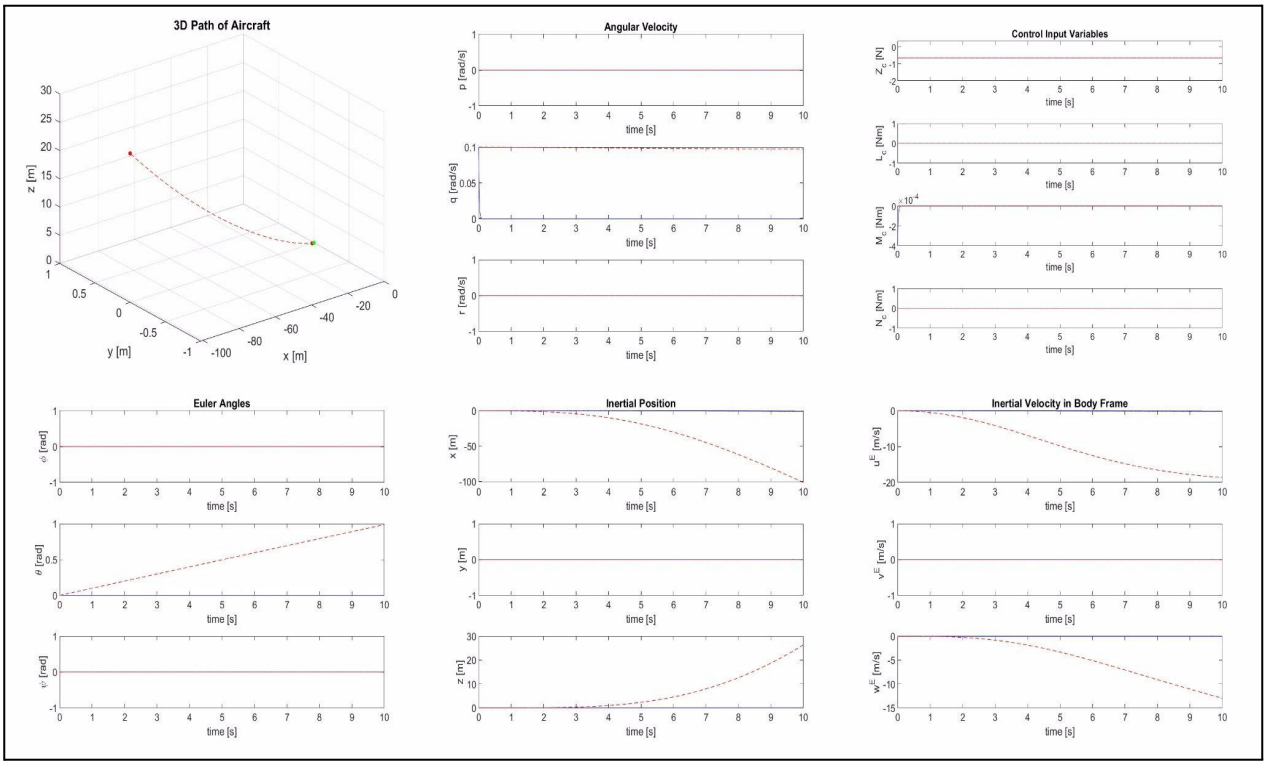

Figure 19: Problem 2.5 Deviation of +0.1 rad/s in Pitch Rate

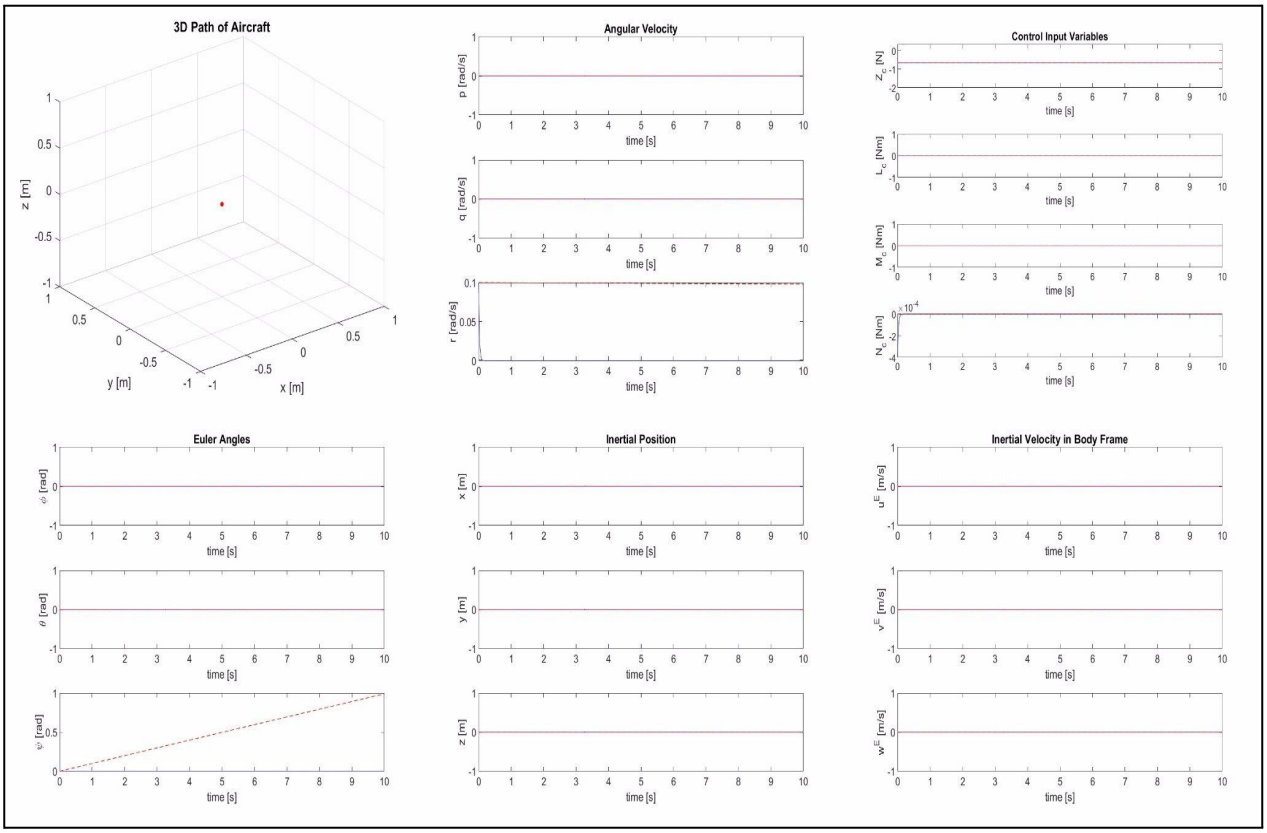

Figure 20: Problem 2.5 Deviation of +0.1 rad/s in Yaw Rate

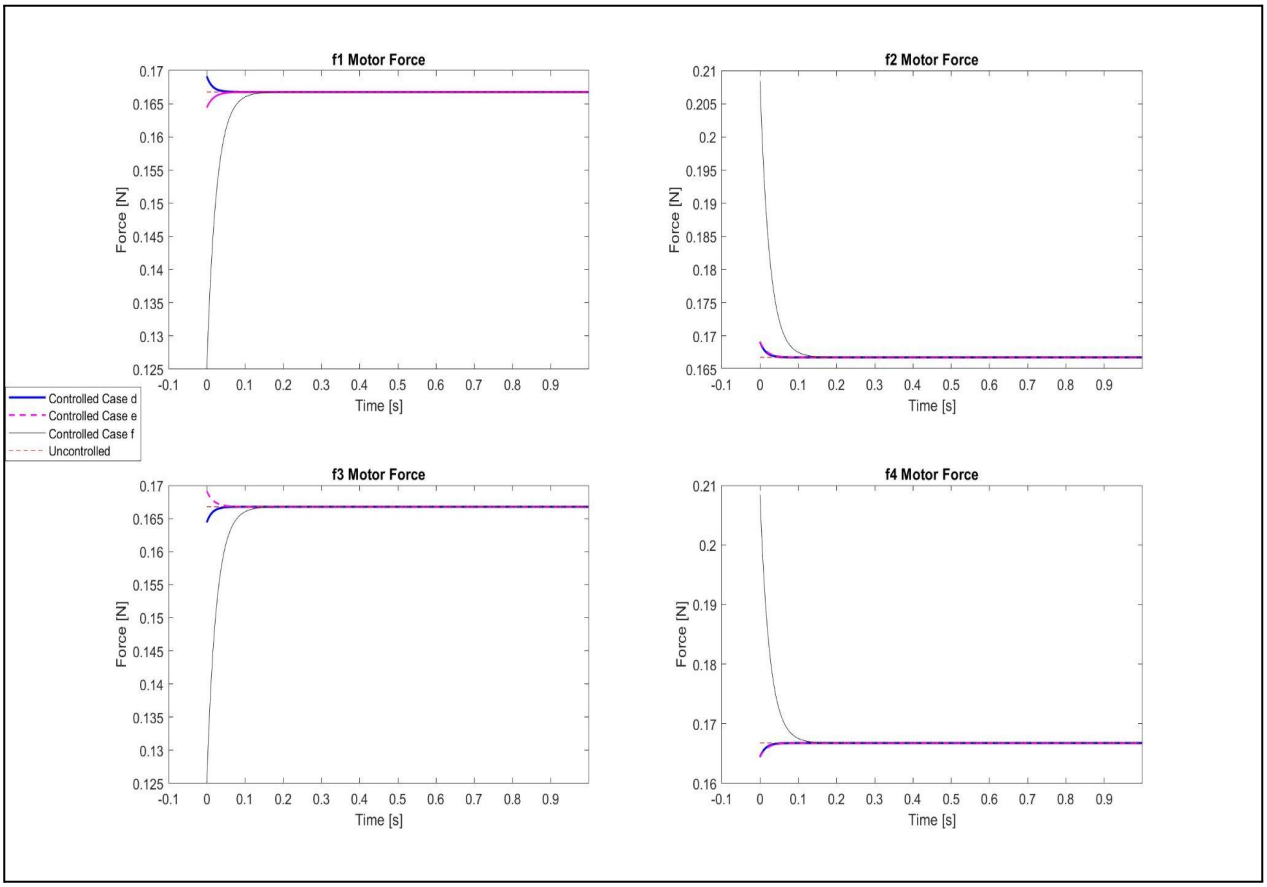

Figure 21: Problem 2.5 Motor Thrust Values

What we can see from the controlled vs uncontrolled systems is that quadcopters are inherently unstable without a control law. This was already seen in Part 1.5 but is reiterated here. With a deviation in roll, pitch or yaw rate, the angular velocity is constant as there isn't a control force to oppose the rotation. Part 2.1 shows how the state variables change with perturbations but in this case, those state changes aren't corrected for either.

Figure 18 shows a perturbation in roll rate which, as already discussed, causes changes in y and z velocities. Without a control force, the y velocity does go to a constant value as drag stops the drone from accelerating. The z velocity and z position continue to increase which means that the drone is rising. It will continue to rise without control forces and moments.

Figure 19 shows a perturbation in pitch rate, which will cause changes in the x and z position and velocities. The positive perturbation to pitch rate causes the quadrotor to move backwards in the x direction as well as rise in the z direction. The x and z position and velocity will continue to increase without control forces and moments.

Figure 20 shows a yaw rate perturbation which causes the drone to spin forever without a control law. In this simulation, the drone won't fall out of the sky due to the yaw perturbation but it is still unstable. In the real world, any small air current would cause the drone to crash.

Figure 21 shows the motor forces for each of the above situations. When the drone is uncontrolled, the total force from the motors is equal to the weight of the drone with no deviations over time. When controlled, the drone produces asymmetric force in order to counteract the perturbations.

Lab Task 3: Feedback Control Design and Implementation

3.1

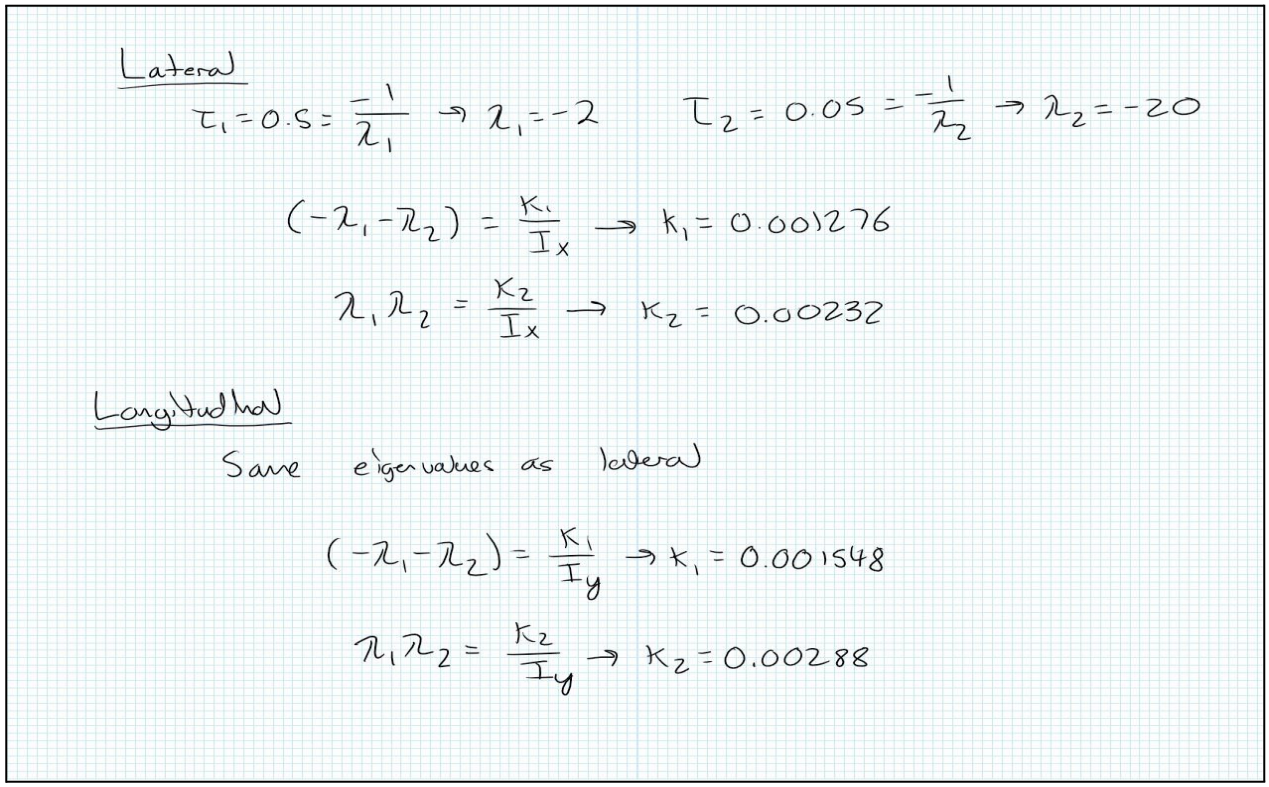

Figure 22: Problem 3.1 Lateral and Longitudinal Gains Derivation

By setting the parts of the desired and actual characteristic equations equal as seen in figure 22, as well as having one time constant dominate by a whole magnitude, feedback gains for both lateral and longitudinal were determined.

3.2

See the Appendix for InnerLoopFeedback function.

3.3

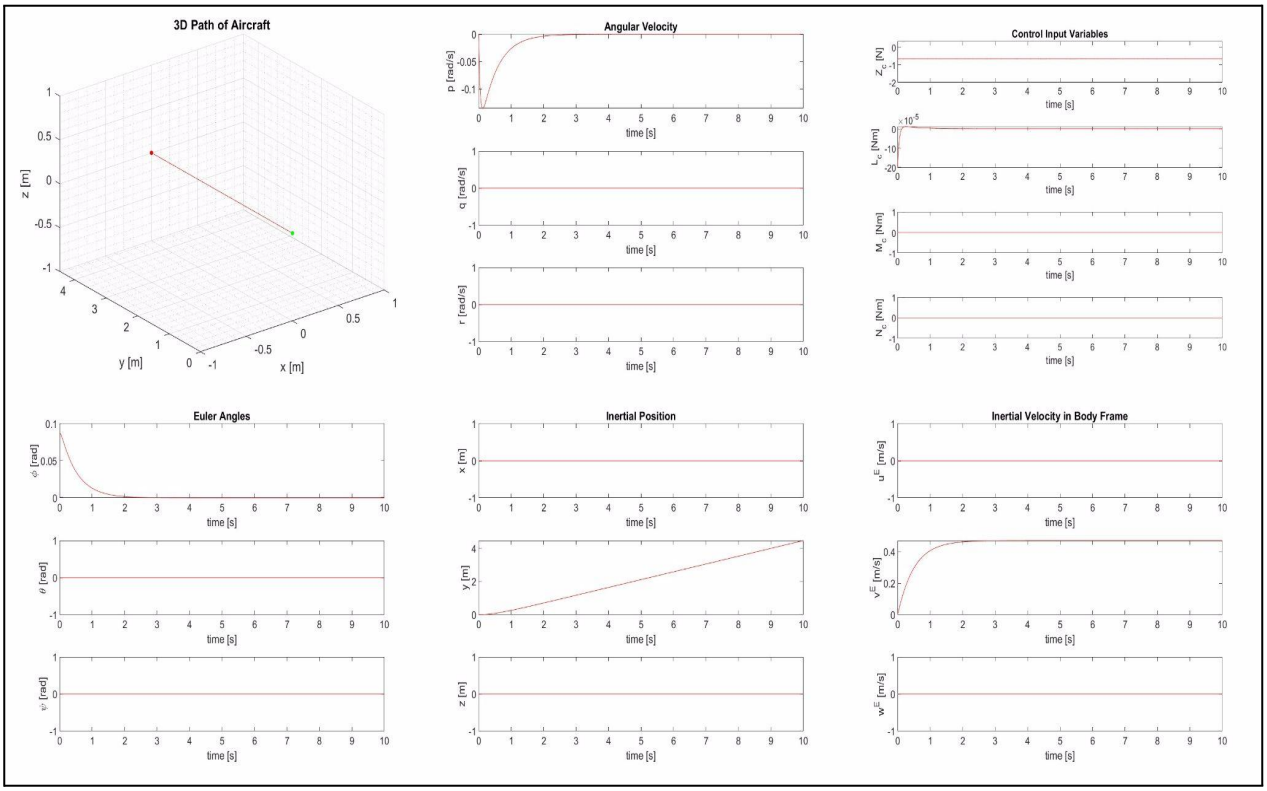

Figure 23: Problem 3.3 Deviation of +5° Roll

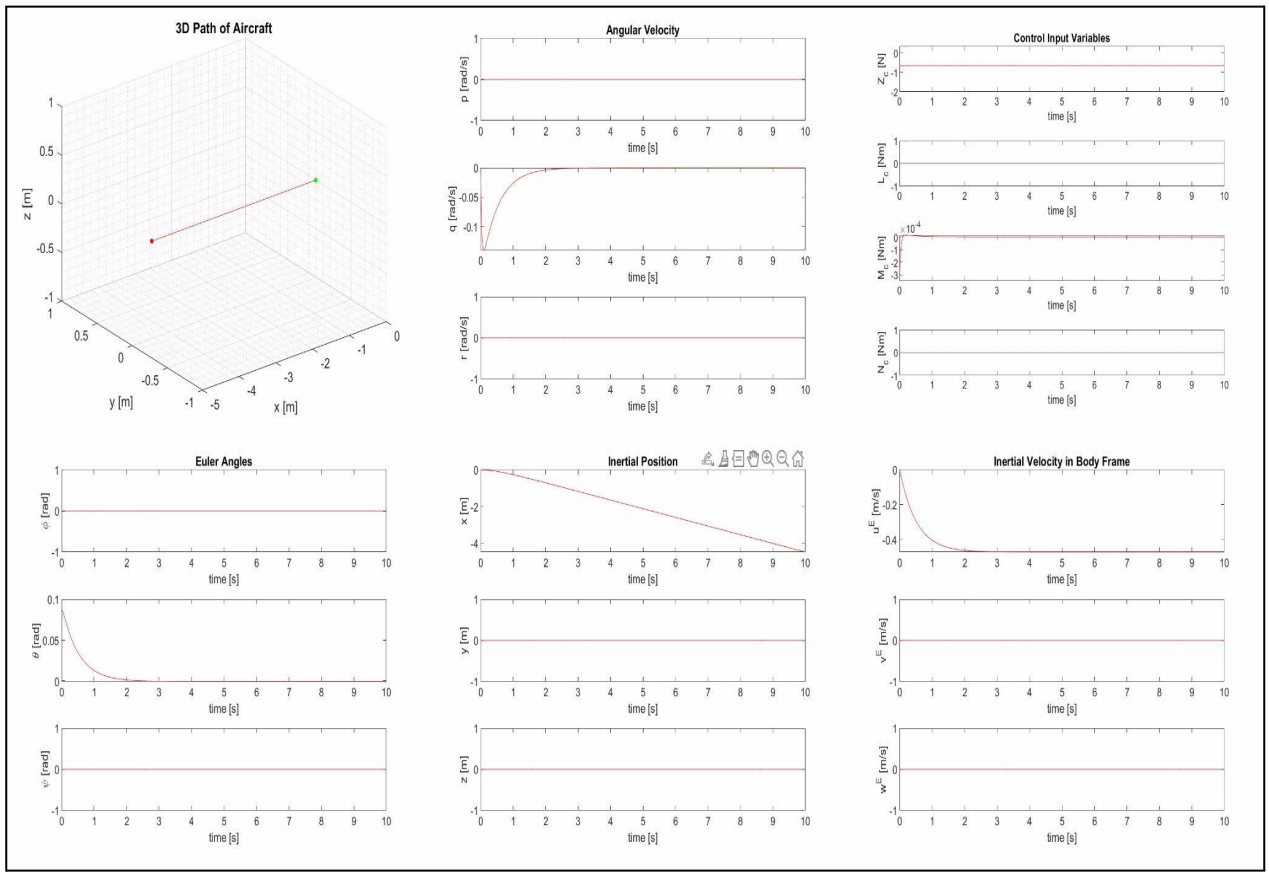

Figure 24: Problem 3.3 Deviation of +5° Pitch

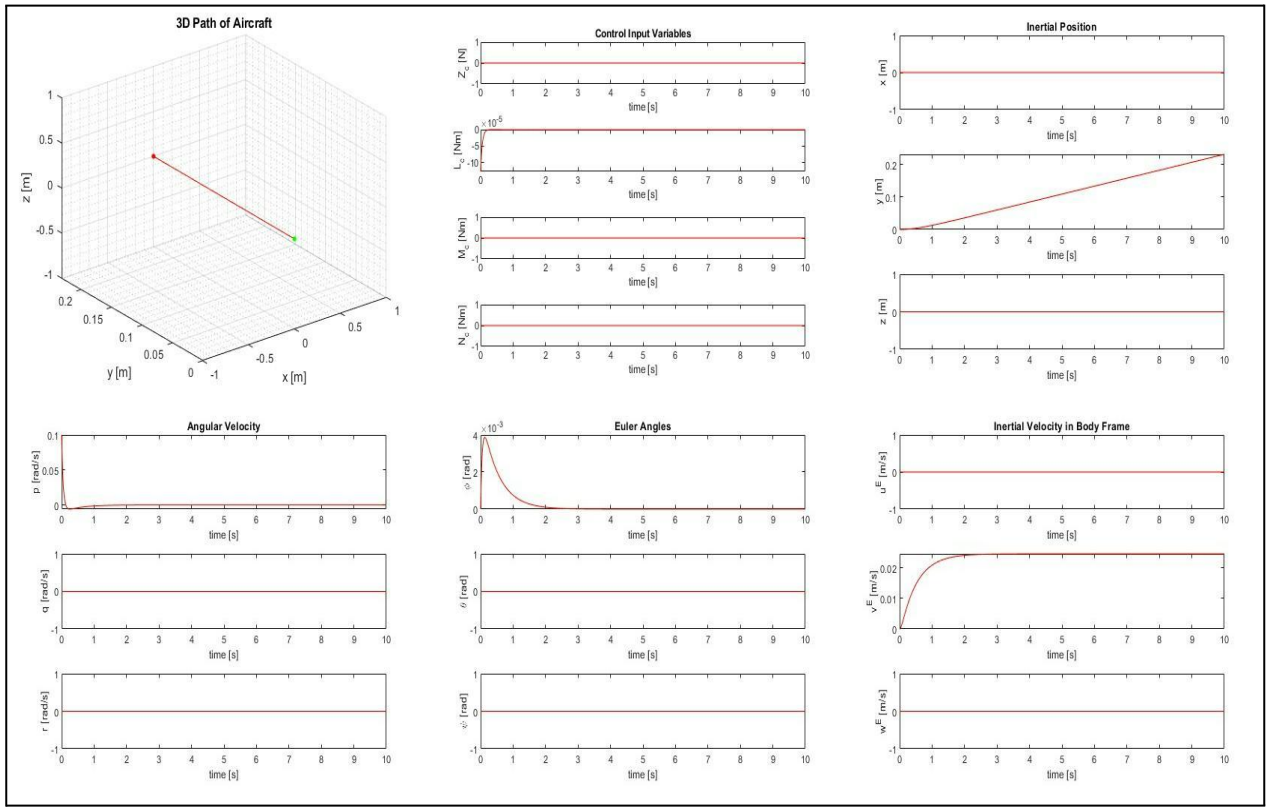

Figure 25: Problem 3.3 Deviation of +0.1 rad/s Roll Rate

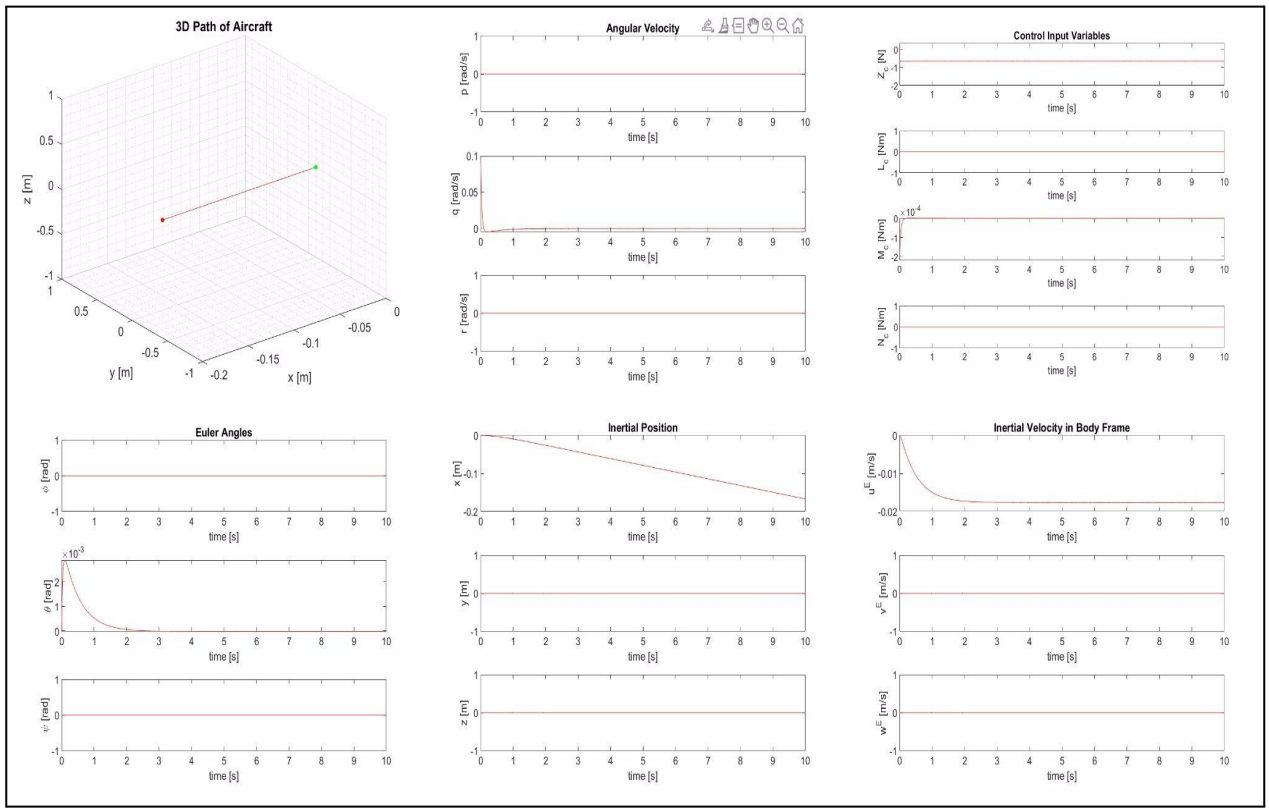

Figure 26: Problem 3.3 Deviation of +0.1 rad/s Pitch Rate

These plots do make sense according to the linear response equations. The perturbation in roll causes a change in the y velocity and by extension the y position. The perturbation in pitch causes a change in the x velocity and by extension the x position. No other state variables are impacted by these perturbations. From our linearized equations of motion, a deviation in roll rate affects both the y and z velocities. This then causes a change in the y and z positions. This makes physical sense because if the drone is rolling, then not all of the lift points upward, causing the drone to drop. This y component of lift also causes the drone to translate in the y direction. A perturbation in pitch rate is almost identical, with the x velocity and x position being impacted instead of the y velocity and y position.

These behaviors are nearly identical to the behaviors found in 2.1 however this is a closed loop model instead of an open one. This means that the control law isn't implemented instantaneously like in 2.1. This is most clearly reflected in the Euler angles and the angular velocity graphs. These graphs aren't linear as a result of the drone's controller continuously feeding back on itself, constantly changing its response as a result of its new state variables. The gain values were designed to be ideally damped so that the drone can return to trim as fast as possible without oscillations.

3.4

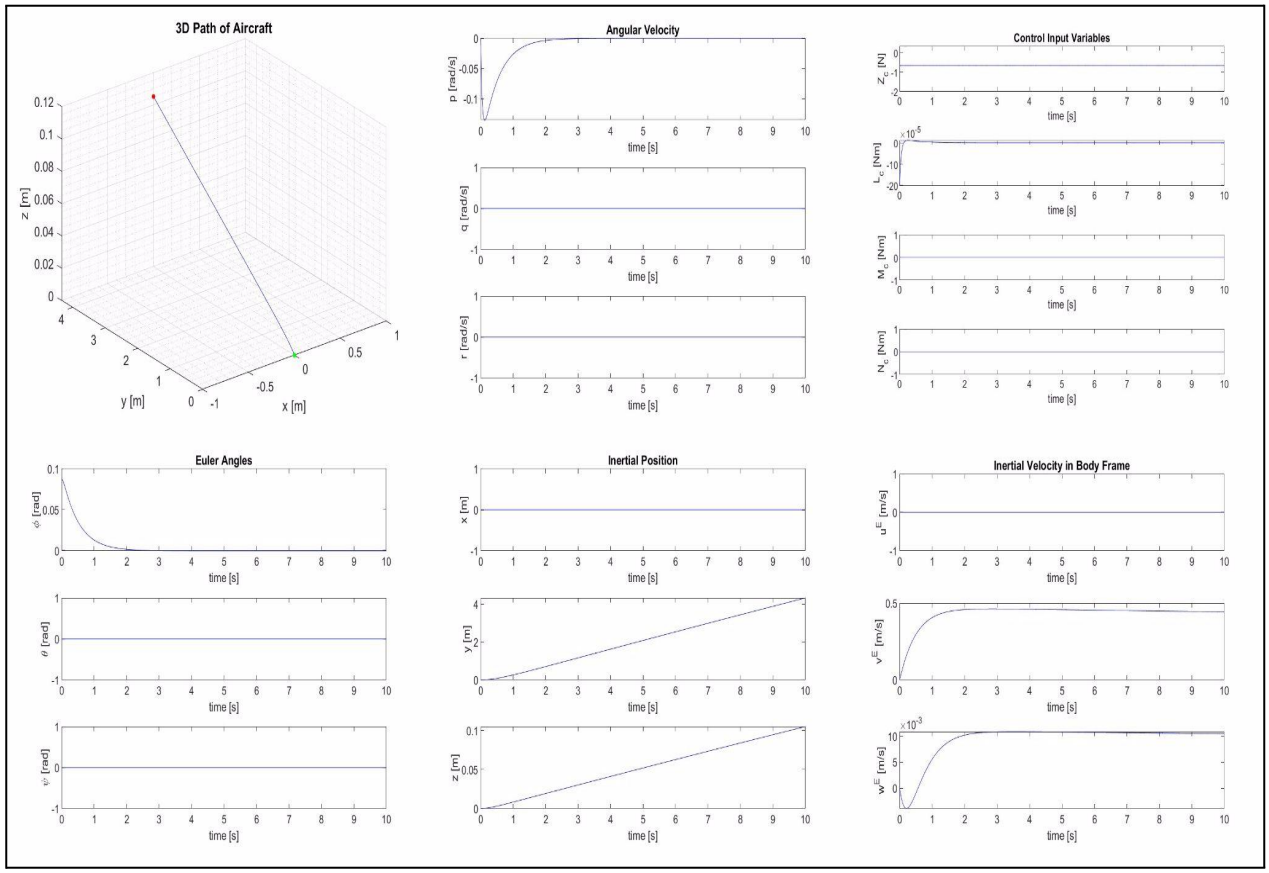

Figure 27: Problem 3.4 Deviation of +5° Roll

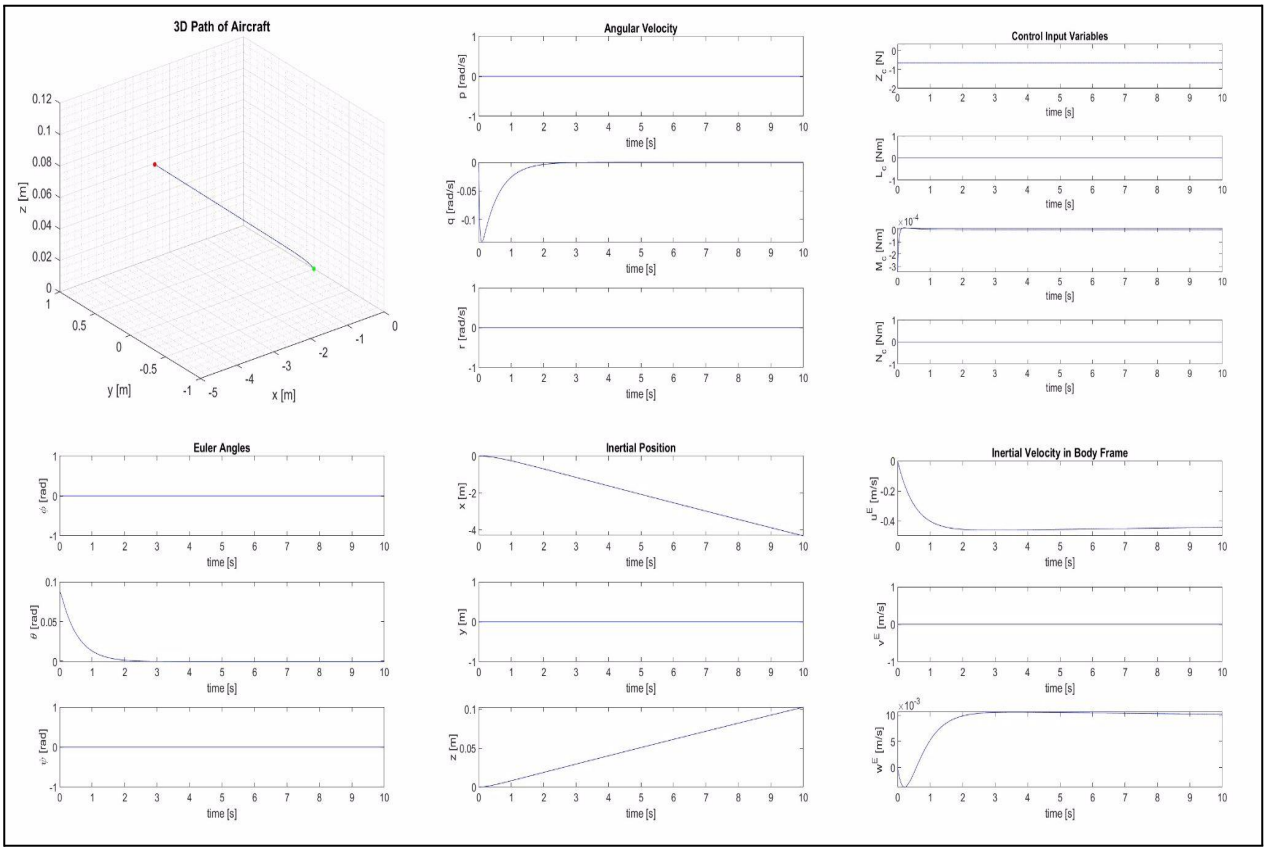

Figure 28: Problem 3.4 Deviation of +5° Pitch

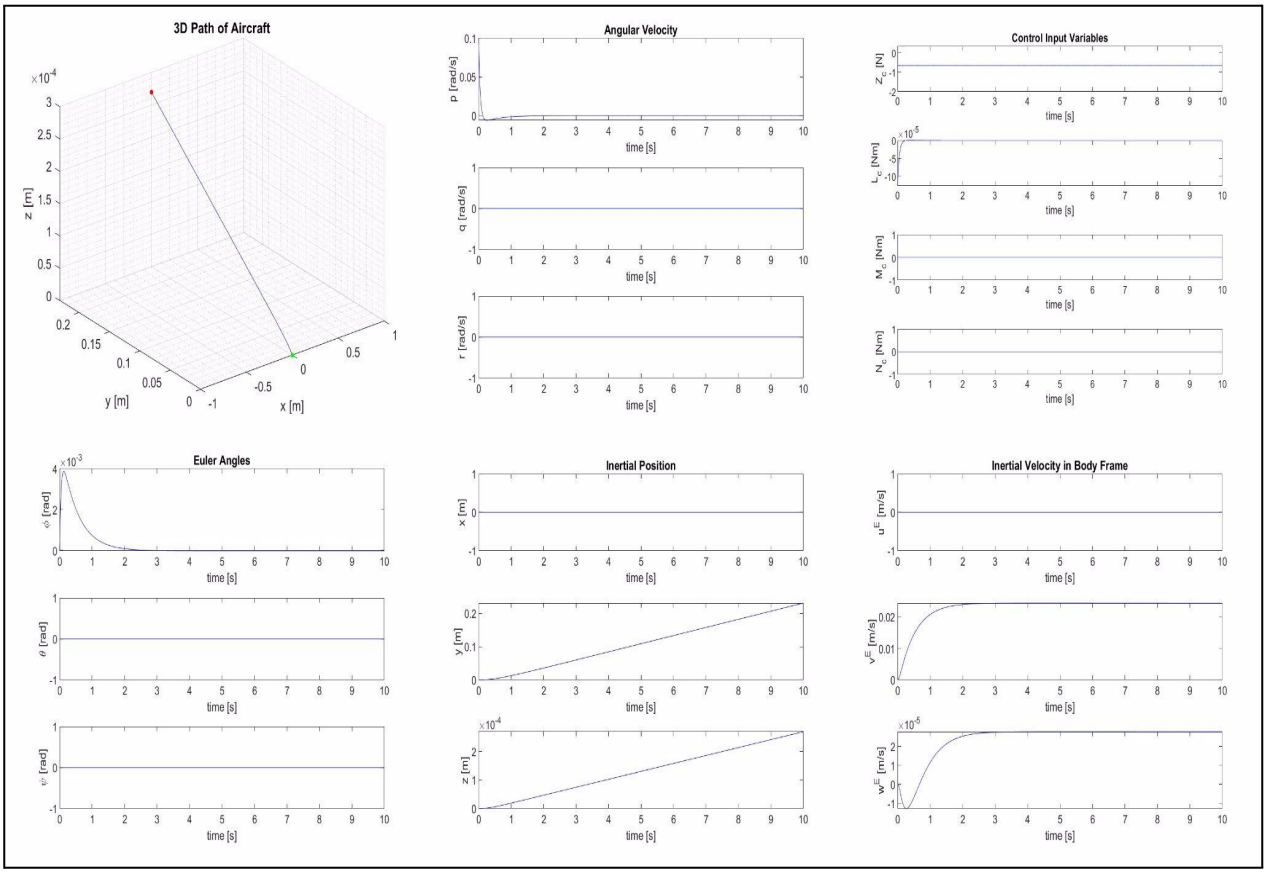

Figure 29: Problem 3.4 Deviation of +0.1 rad/s Roll Rate

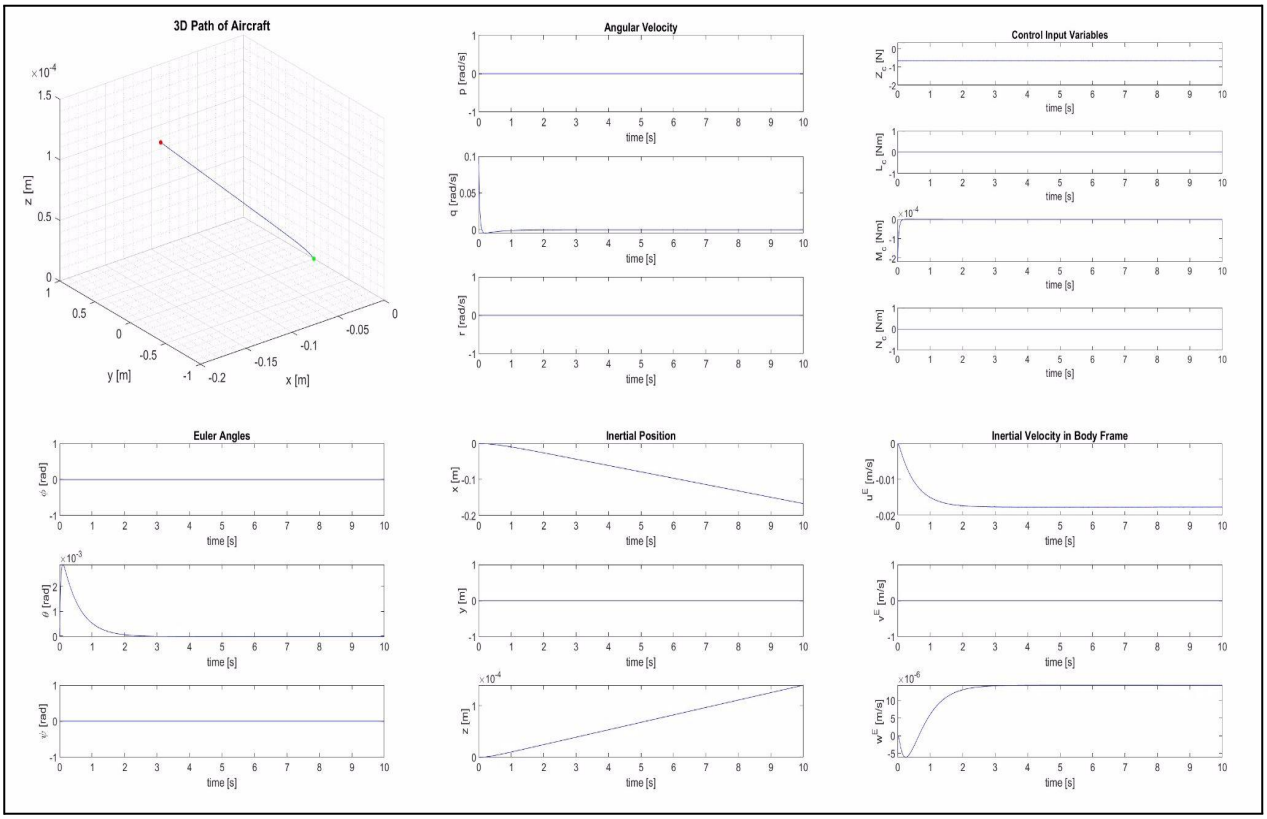

Figure 30: Problem 3.4 Deviation of +0.1 rad/s Pitch Rate

In the nonlinear response equations we see very similar responses in control due to the nonlinear equations of motion showing disturbances cause almost identical motion, with only slight differences in accuracy over time, and more generally in the inertial z direction. This results in nearly identical motion over time, with only small deviations in the inertial z velocities for the nonlinear motion in terms of a very small gain of elevation over 10 seconds, but one that should continue over time unless accounted for in a change in our modeling.

3.5

Green indicates a satisfactory eigenvalue for k3

Red dotted indicates time constant cut-off

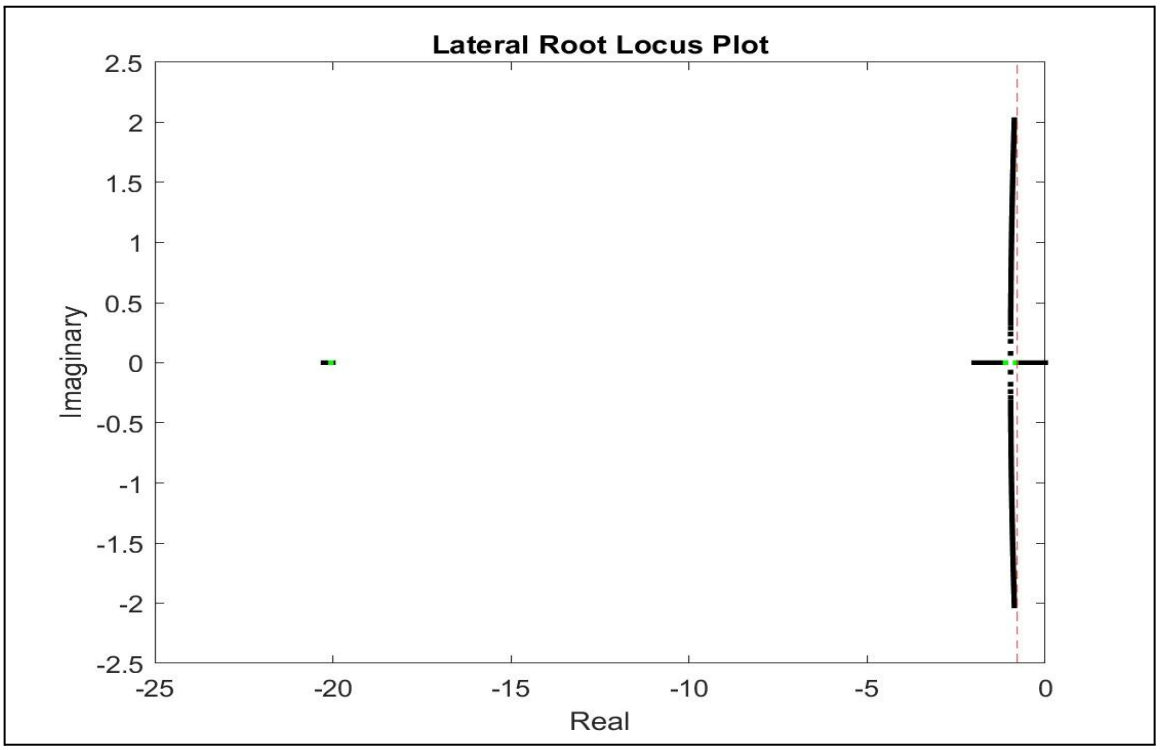

Figure 31: Problem 3.5 Lateral Root Locus Plot

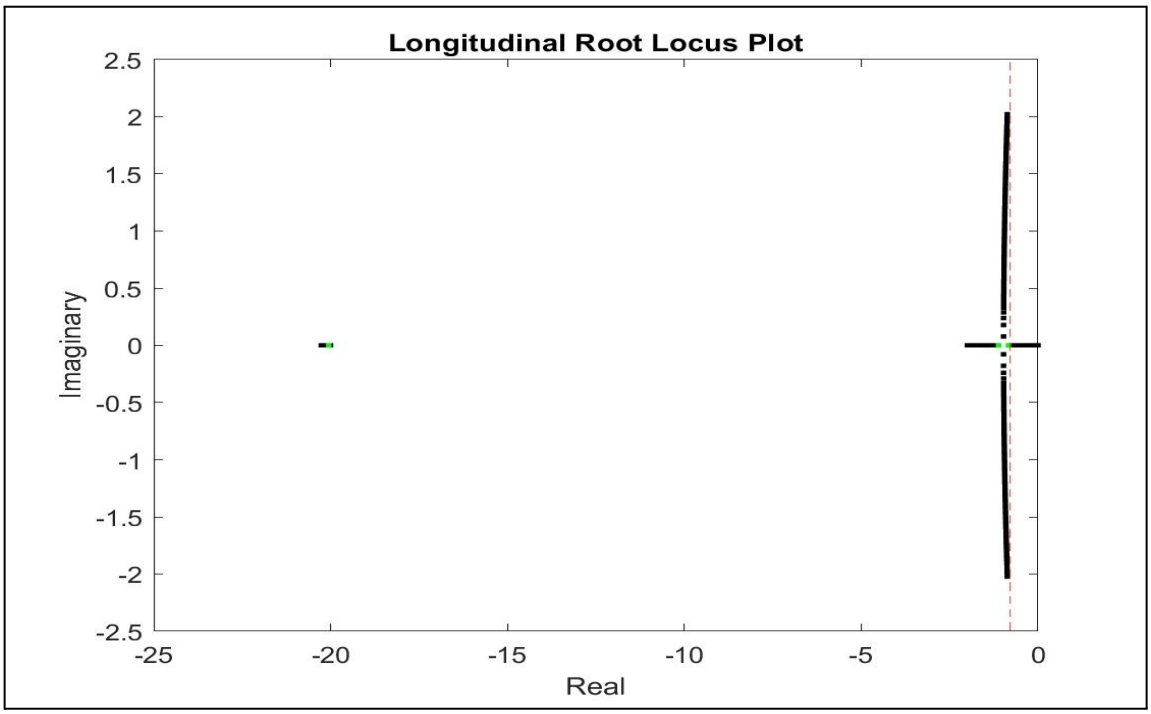

Figure 32: Longitudinal Root Locus Plot

Figure 31 and 32 show the root locus plots for the lateral and longitudinal k3 gains. For lateral gains k1 = 0.001276 $\frac{N*m}{rad}$ and k2 = 0.00232 $\frac{N*m*s}{rad}$, a gain of k3 = 0.1102 $\frac{N*s}{m}$ was found to fit the criteria of having real eigenvalues with time constants no larger than 1.25 seconds. For

longitudinal gains k1 = 0.001548 $\frac{N*m}{rad}$ and k2 = 0.00288 $\frac{N*m*s}{rad}$, a gain of k3 = -0.1360 $\frac{N*s}{m}$ was found to meet the same criteria.

3.6

See the Appendix for VelocityReferenceFeedbackLat and VelocityReferenceFeedbackLon functions.

3.7

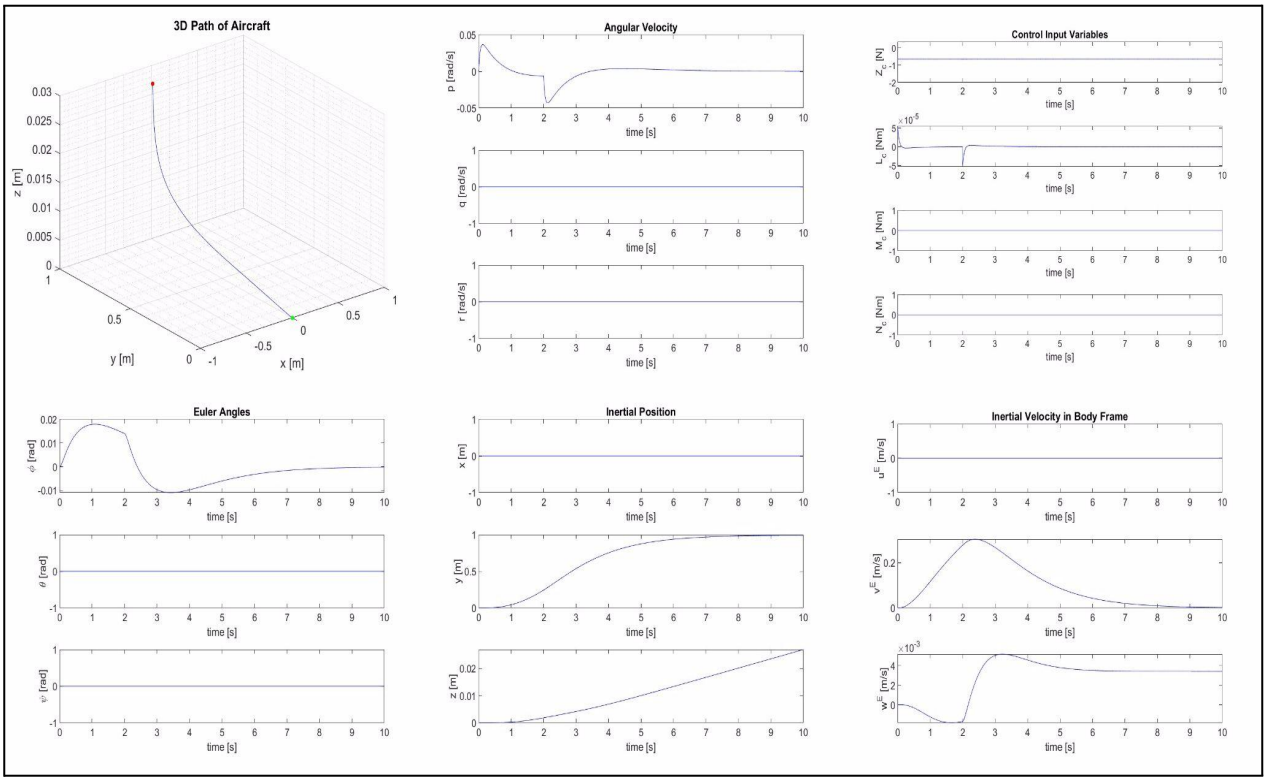

Figure 33: Problem 3.7 Feedforward Lateral Command

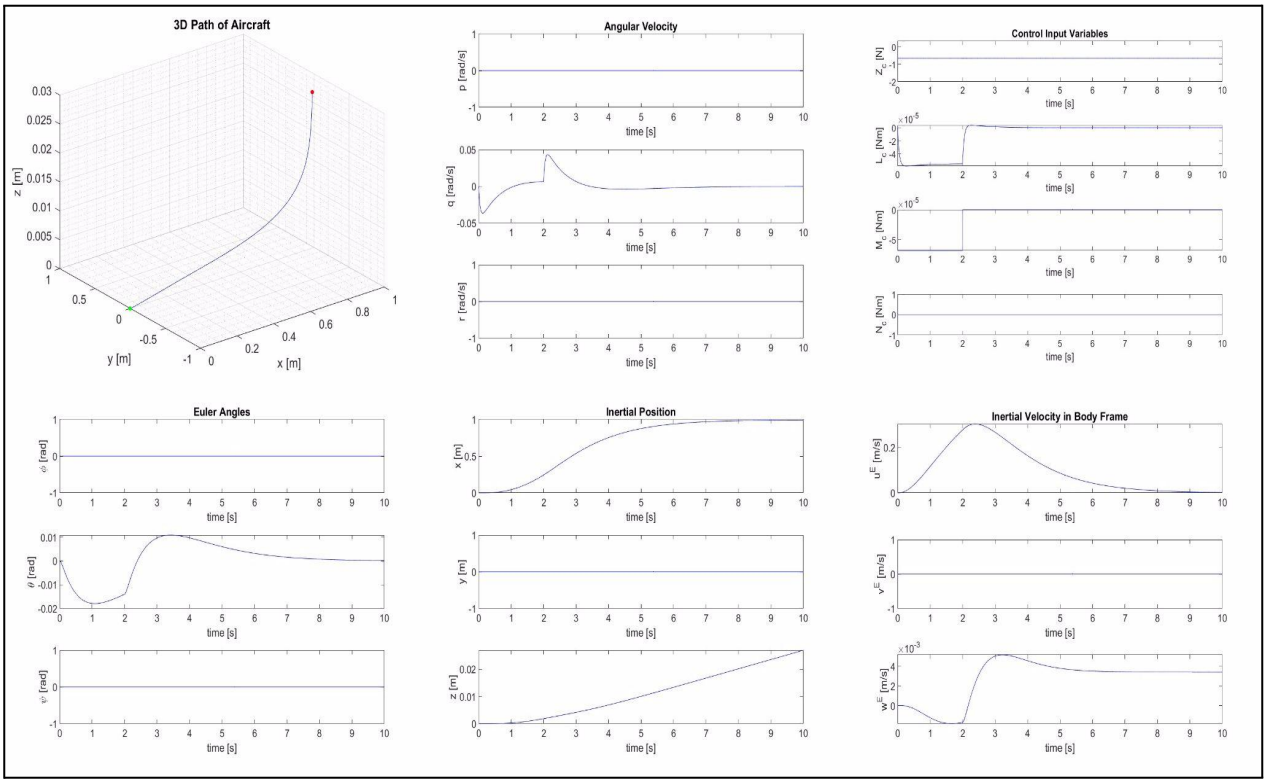

Figure 34: Problem 3.7 Feedforward Longitudinal Command

Figures 33 and 34 show the results for the feedforward commands. The expected results are the quadcopter reaching 1 meter in 2 seconds, which would indicate a velocity for u^E and v^E of 0.5 m/s. However, the actual results indicate it takes ~8 seconds to reach the 1 meter mark. This is primarily due to the fact that the quadrotor is not able to instantly go to 0.5 m/s, and in fact is not even able to reach that speed by the time the 1 meter mark is reached. This indicates that this method of open-loop command is not a good method for the required objectives of this section. If the initial condition of 0.5 m/s was fed into the equations of motion, the quadcopter would reach 1 meter in 2 seconds, but would overshoot the desired location by a full meter (not pictured above).

3.8

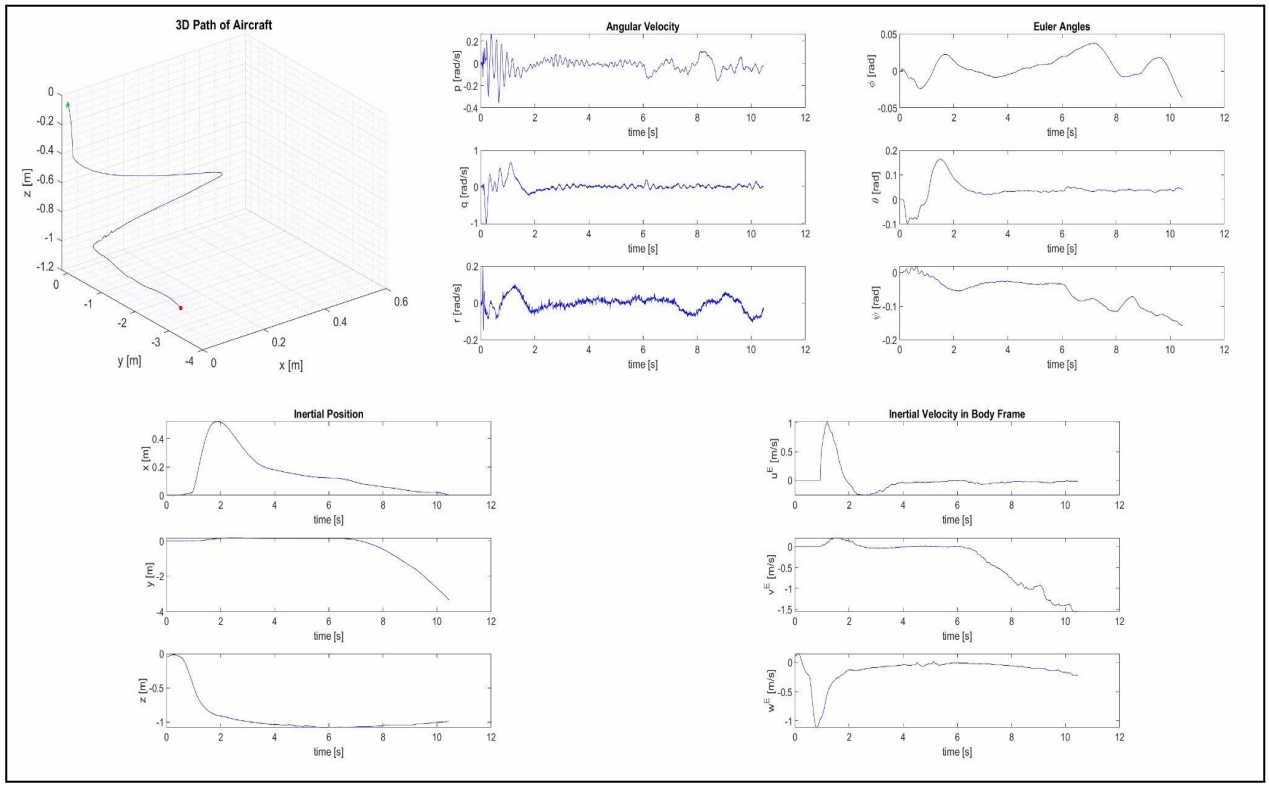

Figure 39: Problem 3.8 Tested Feedforward Lateral Command

While both the simulated and actual response of the lateral feedforward command do not match the expected response, they do have some correlations as well as differences. We see that at 6 seconds (the time the feedforward command was implemented), the actual quadrotor began to speed up, but was also not able to instantly achieve a velocity of 0.5 m/s in the body y axis. It took time for the quadrotor to be able to get to that speed. However, the simulated response was never actually able to reach that speed. The actual quadrotor also was not able to stop at exactly one meter like the simulated response was. It goes ~2.5 meters past the desired 1 meter distance. This is most likely due to the fact that just like how the quadrotor is not able to instantly achieve 0.5 m/s, it is also not able to instantly go from 0.5 m/s to 0 m/s. In fact, the data indicates that the quadrotor does not even attempt to slow down after reaching the 1 meter distance. We are, however, unsure if this is due to how the experiment is carried out or if our feedforward command was incorrect. One similarity that is seen between the simulated and actual responses is the angular velocity, euler angles, and inertial x velocity in the body frame look similar between the two tests. The parameters of u^E, q, r, and θ stay relatively close to zero, while the parameters of p and Φ at the beginning and end of the 2 second feedforward command have similar trends.

Team Participation Table

Name	Plan	Model	Experiment	Results	Report	Code	ACK
Matthew Bradford	1	0	0	1	1	1	X
Lauren Christenson	1	1	1	1	1	1	X
Reece Fountain	1	0	0	1	1	1	X
Jared Steffen	2	2	1	2	1	2	X

Appendix – MATLAB Functions

All Sections: Main

```
clc
clear
close all
%% Constants
% Quadrotor Constants
m = 0.068; %[kg]
d = 0.060; %[m]
km = 0.0024; %[Nm/N]
Ix = 5.8 * 10^-5; %[kgm^2]
Iy = 7.2 * 10^-5; %[kgm^2]
Iz = 1.0 * 10^-4; %[kgm^2]
nu = 1.0 * 10^-3; %[N/(m/s)^2]
mu = 2.0 * 10^-6; %[Nm/(rad/s)^2]
g = 9.81; %[m/s^2]
% Inertia Matrix
I = [Ix;Iy;Iz];
%% 1.2/1.3
% Initial Conditions
var_a = [0;0;0;0;0;0;0;0;0;0;0;0];
% Time
tspan = [0 10]; %[s]
% Motor Forces/Figure Vectors
W = m*g; %[N]
Zc_a = -W;
motor_forces_a = [-Zc_a/4; -Zc_a/4; -Zc_a/4; -Zc_a/4];
controls_a = [Zc_a,0,0,0];
Gc_a = [0;0;0];
fig_a = 1:6;
% ode45 Call
[ta,var_dot_a] = ode45(@(tspan,var_a)
QuadrotorEOM(tspan,var_a,g,m,I,d,km,nu,mu,motor_forces_a),tspan,var_a);
controls_a = [Zc_a;0;0;0] .* ones(length(ta),1)';
% PlotAircraftSim Call
PlotAircraftSim(ta,var_dot_a,controls_a,fig_a,'-b')
%% 1.4a
% Calculate Drag
Va = 5; %[m/s]
drag = nu*Va^2;
% Calculate Phi
phi = deg2rad(atand(-m*g/drag) + 90);
% Calculate Zc
Zc_b = -m*g/cos(phi);
% Calculate v_E and w_E
theta = 0;
psi = 0;
inert_vel = [0;5;0];
cPH = cos(phi);
cT = cos(theta);
cPS = cos(psi);
sPH = sin(phi);
sT = sin(theta);
sPS = sin(psi);
Rot321 = [cT*cPS,sPH*sT*cPS-cPH*sPS,cPH*sT*cPS+sPH*sPS;
          cT*sPS,sPH*sT*sPS+cPH*cPS,cPH*sT*sPS-sPH*cPS;
```

```
            -sT,sPH*cT,cPH*cT];
inert_vel_bb = Rot321' * inert_vel;
% Initial Conditions
var_b = [0;0;0;phi;0;0;inert_vel_bb(1);inert_vel_bb(2);inert_vel_bb(3);0;0;0];
% Motor Forces/Figure Vectors
motor_forces_b = [-Zc_b/4; -Zc_b/4; -Zc_b/4; -Zc_b/4];
fig_b = 7:12;
% ode45 Call
[tb,var_dot_b] = ode45(@(tspan,var_b)
QuadrotorEOM(tspan,var_b,g,m,I,d,km,nu,mu,motor_forces_b),tspan,var_b);
% Controls Vector
controls_b = [Zc_b;0;0;0] .* ones(length(tb),1)';
% PlotAircraftSim Call
PlotAircraftSim(tb,var_dot_b,controls_b,fig_b,'-b')
%% 1.4b
% Calculate Theta
theta = -deg2rad(atand(-m*g/drag) + 90);
% Calculate Zc
Zc_c = -m*g/cos(theta);
% Calculate u_E and w_E
phi = 0;
psi = deg2rad(90);
cPH = cos(phi);
cT = cos(theta);
cPS = cos(psi);
sPH = sin(phi);
sT = sin(theta);
sPS = sin(psi);
Rot321 = [cT*cPS,sPH*sT*cPS-cPH*sPS,cPH*sT*cPS+sPH*sPS;
            cT*sPS,sPH*sT*sPS+cPH*cPS,cPH*sT*sPS-sPH*cPS;
            -sT,sPH*cT,cPH*cT];
inert_vel_bc = Rot321' * inert_vel;
% Initial Conditions
var_c =
[0;0;0;0;theta;psi;inert_vel_bc(1);inert_vel_bc(2);inert_vel_bc(3);0;0;0];
% Motor Forces/Control/Figure Vectors
motor_forces_c = [-Zc_c/4; -Zc_c/4; -Zc_c/4; -Zc_c/4];
fig_c = 13:18;
% ode45 Call
[tc,var_dot_c] = ode45(@(tspan,var_c)
QuadrotorEOM(tspan,var_c,g,m,I,d,km,nu,mu,motor_forces_c),tspan,var_c);
% Controls Vector
controls_c = [Zc_c;0;0;0].* ones(length(tc),1)';
% PlotAircraftSim Call
PlotAircraftSim(tc,var_dot_c,controls_c,fig_c,'-b')
%% 1.5
% Import Hardware Data
load("RSdata_nocontrol.mat")
% Figure Vector
fig = 19:24;
% PlotAircraftSim Call
PlotAircraftSim(rt_estim.time,rt_estim.signals.values,sqrt(abs(rt_motor.signals
.values)*13840.4)',fig,'-b')
% Note: Not sure how to get Zc, Lc, Mc, Nc from data, so those graphs can be
ignored
%% 2.1/2.2
% Initial Conditions
var_d = [0;0;0;deg2rad(5);0;0;0;0;0;0;0;0];
var_e = [0;0;0;0;deg2rad(5);0;0;0;0;0;0;0];
```

```
var_f = [0;0;0;0;0;deg2rad(5);0;0;0;0;0;0];
var_g = [0;0;0;0;0;0;0;0;0;0.1;0;0];
var_h = [0;0;0;0;0;0;0;0;0;0;0.1;0];
var_i = [0;0;0;0;0;0;0;0;0;0;0;0.1];
% Motor Forces and Moments
deltaFc = 0;
deltaGc = [0;0;0];
% Nonlinear ode45 Call
[td,var_dot_d] = ode45(@(tspan,var_d)
QuadrotorEOM(tspan,var_d,g,m,I,d,km,nu,mu,motor_forces_a),tspan,var_d);
[te,var_dot_e] = ode45(@(tspan,var_e)
QuadrotorEOM(tspan,var_e,g,m,I,d,km,nu,mu,motor_forces_a),tspan,var_e);
[tf,var_dot_f] = ode45(@(tspan,var_f)
QuadrotorEOM(tspan,var_f,g,m,I,d,km,nu,mu,motor_forces_a),tspan,var_f);
[tg,var_dot_g] = ode45(@(tspan,var_g)
QuadrotorEOM(tspan,var_g,g,m,I,d,km,nu,mu,motor_forces_a),tspan,var_g);
[th,var_dot_h] = ode45(@(tspan,var_h)
QuadrotorEOM(tspan,var_h,g,m,I,d,km,nu,mu,motor_forces_a),tspan,var_h);
[ti,var_dot_i] = ode45(@(tspan,var_i)
QuadrotorEOM(tspan,var_i,g,m,I,d,km,nu,mu,motor_forces_a),tspan,var_i);
% Linear ode45 Call
[tdL,var_dot_dL] = ode45(@(tspan,var_d)
QuadrotorEOM_Linearized(tspan,var_d,g,m,I,deltaFc,deltaGc),tspan,var_d);
[teL,var_dot_eL] = ode45(@(tspan,var_e)
QuadrotorEOM_Linearized(tspan,var_e,g,m,I,deltaFc,deltaGc),tspan,var_e);
[tfL,var_dot_fL] = ode45(@(tspan,var_f)
QuadrotorEOM_Linearized(tspan,var_f,g,m,I,deltaFc,deltaGc),tspan,var_f);
[tgL,var_dot_gL] = ode45(@(tspan,var_g)
QuadrotorEOM_Linearized(tspan,var_g,g,m,I,deltaFc,deltaGc),tspan,var_g);
[thL,var_dot_hL] = ode45(@(tspan,var_h)
QuadrotorEOM_Linearized(tspan,var_h,g,m,I,deltaFc,deltaGc),tspan,var_h);
[tiL,var_dot_iL] = ode45(@(tspan,var_i)
QuadrotorEOM_Linearized(tspan,var_i,g,m,I,deltaFc,deltaGc),tspan,var_i);
% Figure Vector
fig_d = 25:30;
fig_e = 31:36;
fig_f = 37:42;
fig_g = 43:48;
fig_h = 49:54;
fig_i = 55:60;
% Control Vectors
controls_d = [Zc_a;0;0;0] .* ones(length(td),1)';
controls_e = [Zc_a;0;0;0] .* ones(length(te),1)';
controls_f = [Zc_a;0;0;0] .* ones(length(tf),1)';
controls_g = [Zc_a;0;0;0] .* ones(length(tg),1)';
controls_h = [Zc_a;0;0;0] .* ones(length(th),1)';
controls_i = [Zc_a;0;0;0] .* ones(length(ti),1)';
controls_dL = [Zc_a;0;0;0] .* ones(length(tdL),1)';
controls_eL = [Zc_a;0;0;0] .* ones(length(teL),1)';
controls_fL = [Zc_a;0;0;0] .* ones(length(tfL),1)';
controls_gL = [Zc_a;0;0;0] .* ones(length(tgL),1)';
controls_hL = [Zc_a;0;0;0] .* ones(length(thL),1)';
controls_iL = [Zc_a;0;0;0] .* ones(length(tiL),1)';
% PlotAircraftSim Calls
% Nonlinear
PlotAircraftSim(td,var_dot_d,controls_d,fig_d,'-b')
PlotAircraftSim(te,var_dot_e,controls_e,fig_e,'-b')
PlotAircraftSim(tf,var_dot_f,controls_f,fig_f,'-b')
PlotAircraftSim(tg,var_dot_g,controls_g,fig_g,'-b')
```

```
PlotAircraftSim(th,var_dot_h,controls_h,fig_h,'-b')
PlotAircraftSim(ti,var_dot_i,controls_i,fig_i,'-b')
% Linear
PlotAircraftSim(tdL,var_dot_dL,controls_dL,fig_d,'--r')
PlotAircraftSim(teL,var_dot_eL,controls_eL,fig_e,'--r')
PlotAircraftSim(tfL,var_dot_fL,controls_fL,fig_f,'--r')
PlotAircraftSim(tgL,var_dot_gL,controls_gL,fig_g,'--r')
PlotAircraftSim(thL,var_dot_hL,controls_hL,fig_h,'--r')
PlotAircraftSim(tiL,var_dot_iL,controls_iL,fig_i,'--r')
%% 2.5
% RotationDerivativeFeedback Calls
[Fc_g,Gc_g] = RotationDerivativeFeedback(var_g,m,g);
[Fc_h,Gc_h] = RotationDerivativeFeedback(var_h,m,g);
[Fc_i,Gc_i] = RotationDerivativeFeedback(var_i,m,g);
% ode45 Calls
[tgRF,var_dot_gRF] = ode45(@(tspan,var_g)
QuadrotorEOMwithRateFeedback(tspan,var_g,g,m,I,nu,mu),tspan,var_g);
[thRF,var_dot_hRF] = ode45(@(tspan,var_h)
QuadrotorEOMwithRateFeedback(tspan,var_h,g,m,I,nu,mu),tspan,var_h);
[tiRF,var_dot_iRF] = ode45(@(tspan,var_i)
QuadrotorEOMwithRateFeedback(tspan,var_i,g,m,I,nu,mu),tspan,var_i);
% Control Moment and Force Vectors
for k = 1:numel(tgRF)
   [~,Lc_g(:,k),Mc_g(:,k),Nc_g(:,k)] =
QuadrotorEOMwithRateFeedback(tgRF(k),var_dot_gRF(k,:), g, m, I, nu, mu);
end
for k = 1:numel(thRF)
   [~,Lc_h(:,k),Mc_h(:,k),Nc_h(:,k)] =
QuadrotorEOMwithRateFeedback(thRF(k),var_dot_hRF(k,:), g, m, I, nu, mu);
end
for k = 1:numel(tiRF)
   [~,Lc_i(:,k),Mc_i(:,k),Nc_i(:,k)] =
QuadrotorEOMwithRateFeedback(tiRF(k),var_dot_iRF(k,:), g, m, I, nu, mu);
end
Fc_g = (Fc_g * ones(length(Lc_g),1))';
Fc_h = (Fc_h * ones(length(Lc_h),1))';
Fc_i = (Fc_i * ones(length(Lc_i),1))';
Gc_g = [Lc_g;Mc_g;Nc_g];
Gc_h = [Lc_h;Mc_h;Nc_h];
Gc_i = [Lc_i;Mc_i;Nc_i];
% Control/Figure Vectors
controls_gRF = [Fc_g;Lc_g;Mc_g;Nc_g];
controls_hRF = [Fc_h;Lc_h;Mc_h;Nc_h];
controls_iRF = [Fc_i;Lc_i;Mc_i;Nc_i];
fig_gRF = 61:66;
fig_hRF = 67:72;
fig_iRF = 73:78;
% PlotAircraftSim Calls
% Controlled
PlotAircraftSim(tgRF,var_dot_gRF,controls_gRF,fig_gRF,'-b')
PlotAircraftSim(thRF,var_dot_hRF,controls_hRF,fig_hRF,'-b')
PlotAircraftSim(tiRF,var_dot_iRF,controls_iRF,fig_iRF,'-b')
% % Uncontrolled
PlotAircraftSim(tg,var_dot_g,controls_g,fig_gRF,'--r')
PlotAircraftSim(th,var_dot_h,controls_h,fig_hRF,'--r')
PlotAircraftSim(ti,var_dot_i,controls_i,fig_iRF,'--r')
% ComputeMotorForces Call for Controlled and Uncontrolled Systems
% Controlled
motor_forces_gRF = ComputeMotorForces(Fc_g,Gc_g,km,d);
```

```
motor_forces_hRF = ComputeMotorForces(Fc_h,Gc_h,km,d);
motor_forces_iRF = ComputeMotorForces(Fc_i,Gc_i,km,d);
% Uncontrolled
motor_forces_ghi = ComputeMotorForces(Zc_a,Gc_a,km,d);
motor_forces_ghi = motor_forces_ghi .*
ones(length(motor_forces_ghi),length(tg));
% Plot Motor Forces for Controlled and Uncontrolled Systems
figure(79);
subplot(221)
plot(tgRF,motor_forces_gRF(1,:),'-b','LineWidth',1.5); hold on;
plot(thRF,motor_forces_hRF(1,:),'-m','LineWidth',1.2);
plot(tiRF,motor_forces_iRF(1,:),'-k');
plot(tg,motor_forces_ghi(1,:),'--r');
title('f1 Motor Force')
xlabel('Time [s]')
ylabel('Force [N]')
xlim([-0.1 1])
subplot(222)
plot(tgRF,motor_forces_gRF(2,:),'-b','LineWidth',1.5); hold on;
plot(thRF,motor_forces_hRF(2,:),'--m','LineWidth',1.2);
plot(tiRF,motor_forces_iRF(2,:),'-k');
plot(tg,motor_forces_ghi(2,:),'--r');
title('f2 Motor Force')
xlabel('Time [s]')
ylabel('Force [N]')
xlim([-0.1 1])
subplot(223)
plot(tgRF,motor_forces_gRF(3,:),'-b','LineWidth',1.5); hold on;
plot(thRF,motor_forces_hRF(3,:),'--m','LineWidth',1.2);
plot(tiRF,motor_forces_iRF(3,:),'-k');
plot(tg,motor_forces_ghi(3,:),'--r')
title('f3 Motor Force')
xlabel('Time [s]')
ylabel('Force [N]')
xlim([-0.1 1])
subplot(224)
plot(tgRF,motor_forces_gRF(4,:),'-b','LineWidth',1.5); hold on;
plot(thRF,motor_forces_hRF(4,:),'--m','LineWidth',1.2);
plot(tiRF,motor_forces_iRF(4,:),'-k');
plot(tg,motor_forces_ghi(4,:),'--r')
title('f4 Motor Force')
xlabel('Time [s]')
ylabel('Force [N]')
xlim([-0.1 1])
Lgnd = legend('Controlled Case d','Controlled Case e','Controlled Case
f','Uncontrolled','Location','bestoutside');
Lgnd.Position(1) = -0.06;
Lgnd.Position(2) = 0.45;
%% 3.1
% Gains were calculated by hand for 3.1
%% 3.3
% ode45 Calls
[tdILF,var_dot_dILF] = ode45(@(tspan,var_d)
QuadrotorEOM_LinearizedWithILFeedback(tspan,var_d,g,m,I),tspan,var_d);
[teILF,var_dot_eILF] = ode45(@(tspan,var_e)
QuadrotorEOM_LinearizedWithILFeedback(tspan,var_e,g,m,I),tspan,var_e);
[tgILF,var_dot_gILF] = ode45(@(tspan,var_g)
QuadrotorEOM_LinearizedWithILFeedback(tspan,var_g,g,m,I),tspan,var_g);
[thILF,var_dot_hILF] = ode45(@(tspan,var_h)
```

```
QuadrotorEOM_LinearizedWithILFeedback(tspan,var_h,g,m,I),tspan,var_h);
% Control Moment Vectors
for k = 1:numel(tdILF)
   [~,Lc_dILF(:,k),Mc_dILF(:,k),Nc_dILF(:,k)] =
QuadrotorEOM_LinearizedWithILFeedback(tdILF(k),var_dot_dILF(k,:), g, m, I);
end
for k = 1:numel(teILF)
   [~,Lc_eILF(:,k),Mc_eILF(:,k),Nc_eILF(:,k)] =
QuadrotorEOM_LinearizedWithILFeedback(teILF(k),var_dot_eILF(k,:), g, m, I);
end
for k = 1:numel(tgILF)
   [~,Lc_gILF(:,k),Mc_gILF(:,k),Nc_gILF(:,k)] =
QuadrotorEOM_LinearizedWithILFeedback(tgILF(k),var_dot_gILF(k,:), g, m, I);
end
for k = 1:numel(thILF)
   [~,Lc_hILF(:,k),Mc_hILF(:,k),Nc_hILF(:,k)] =
QuadrotorEOM_LinearizedWithILFeedback(thILF(k),var_dot_hILF(k,:), g, m, I);
end
% InnerLoopFeedback Calls
[Fc_dILF,Gc_dILF] = InnerLoopFeedback(var_d);
[Fc_eILF,Gc_eILF] = InnerLoopFeedback(var_e);
[Fc_gILF,Gc_gILF] = InnerLoopFeedback(var_g);
[Fc_hILF,Gc_hILF] = InnerLoopFeedback(var_h);
% Control Force Vectors
Fc_dILF = (0 * ones(length(Lc_dILF),1))';
Fc_eILF = (0 * ones(length(Lc_eILF),1))';
Fc_gILF = (0 * ones(length(Lc_gILF),1))';
Fc_hILF = (0 * ones(length(Lc_hILF),1))';
% Control/Figure Vectors
controls_dILF = [Fc_dILF;Lc_dILF;Mc_dILF;Nc_dILF];
controls_eILF = [Fc_eILF;Lc_eILF;Mc_eILF;Nc_eILF];
controls_gILF = [Fc_gILF;Lc_gILF;Mc_gILF;Nc_gILF];
controls_hILF = [Fc_hILF;Lc_hILF;Mc_hILF;Nc_hILF];
fig_dILF = 80:85;
fig_eILF = 86:91;
fig_gILF = 92:97;
fig_hILF = 98:103;
% PlotAircraftSim Calls
PlotAircraftSim(tdILF,var_dot_dILF,controls_dILF,fig_dILF,'-r')
PlotAircraftSim(teILF,var_dot_eILF,controls_eILF,fig_eILF,'-r')
PlotAircraftSim(tgILF,var_dot_gILF,controls_gILF,fig_gILF,'-r')
PlotAircraftSim(thILF,var_dot_hILF,controls_hILF,fig_hILF,'-r')
%% 3.4
% ode45 Calls
[tdILF2,var_dot_dILF2] = ode45(@(tspan,var_d)
QuadrotorEOMWithILFeedback(tspan,var_d,g,m,I,nu,mu),tspan,var_d);
[teILF2,var_dot_eILF2] = ode45(@(tspan,var_e)
QuadrotorEOMWithILFeedback(tspan,var_e,g,m,I,nu,mu),tspan,var_e);
[tgILF2,var_dot_gILF2] = ode45(@(tspan,var_g)
QuadrotorEOMWithILFeedback(tspan,var_g,g,m,I,nu,mu),tspan,var_g);
[thILF2,var_dot_hILF2] = ode45(@(tspan,var_h)
QuadrotorEOMWithILFeedback(tspan,var_h,g,m,I,nu,mu),tspan,var_h);
% Control Moment Vectors
for k = 1:numel(tdILF2)
   [~,Lc_dILF2(:,k),Mc_dILF2(:,k),Nc_dILF2(:,k)] =
QuadrotorEOMWithILFeedback(tdILF2(k),var_dot_dILF2(k,:), g, m, I, nu, mu);
end
for k = 1:numel(teILF2)
   [~,Lc_eILF2(:,k),Mc_eILF2(:,k),Nc_eILF2(:,k)] =
```

```
QuadrotorEOMWithILFeedback(teILF2(k),var_dot_eILF2(k,:), g, m, I, nu, mu);
end
for k = 1:numel(tgILF2)
   [~,Lc_gILF2(:,k),Mc_gILF2(:,k),Nc_gILF2(:,k)] =
QuadrotorEOMWithILFeedback(tgILF2(k),var_dot_gILF2(k,:), g, m, I, nu, mu);
end
for k = 1:numel(thILF2)
   [~,Lc_hILF2(:,k),Mc_hILF2(:,k),Nc_hILF2(:,k)] =
QuadrotorEOMWithILFeedback(thILF2(k),var_dot_hILF2(k,:), g, m, I, nu, mu);
end
% InnerLoopFeedback Calls
[Fc_dILF2,Gc_dILF2] = InnerLoopFeedback(var_d);
[Fc_eILF2,Gc_eILF2] = InnerLoopFeedback(var_d);
[Fc_gILF2,Gc_gILF2] = InnerLoopFeedback(var_d);
[Fc_hILF2,Gc_hILF2] = InnerLoopFeedback(var_d);
% Control Force Vectors
Fc_dILF2 = (Fc_dILF2 * ones(length(Lc_dILF2),1))';
Fc_eILF2 = (Fc_eILF2 * ones(length(Lc_eILF2),1))';
Fc_gILF2 = (Fc_gILF2 * ones(length(Lc_gILF2),1))';
Fc_hILF2 = (Fc_hILF2 * ones(length(Lc_hILF2),1))';
% Control Vectors
controls_dILF2 = [Fc_dILF2;Lc_dILF2;Mc_dILF2;Nc_dILF2];
controls_eILF2 = [Fc_eILF2;Lc_eILF2;Mc_eILF2;Nc_eILF2];
controls_gILF2 = [Fc_gILF2;Lc_gILF2;Mc_gILF2;Nc_gILF2];
controls_hILF2 = [Fc_hILF2;Lc_hILF2;Mc_hILF2;Nc_hILF2];
% PlotAircraftSim Calls
PlotAircraftSim(tdILF2,var_dot_dILF2,controls_dILF2,fig_dILF,'-b')
PlotAircraftSim(teILF2,var_dot_eILF2,controls_eILF2,fig_eILF,'-b')
PlotAircraftSim(tgILF2,var_dot_gILF2,controls_gILF2,fig_gILF,'-b')
PlotAircraftSim(thILF2,var_dot_hILF2,controls_hILF2,fig_hILF,'-b')
%% 3.5
% Lateral
% Gains
k1_lat = 0.001276;
k2_lat = 0.00232;
k3_lat = (0:0.05:10)*Ix;
% Find k3_lat for Tau = 1.25s
eigA_lat = zeros(length(k3_lat),3);
j = 1;
for i = 1:length(k3_lat)
   A = [0 g 0;
        0 0 1;
        -k3_lat(i)/Ix -k2_lat/Ix -k1_lat/Ix];
   eigA_lat(j,:) = eig(A);
   figure(104);
   plot(real(eigA_lat),imag(eigA_lat),'.k')
   hold on
   j = j + 1;
end
plot(real(eigA_lat(39,:)),imag(eigA_lat(39,:)),'.g')
xline(-0.8,'--r')
xlabel('Real')
ylabel('Imaginary')
title('Lateral Root Locus Plot')
hold off
k3_lat = k3_lat(39);
% Longitudinal
% Gains
k1_lon = 0.001584;
```

```
k2_lon = 0.00288;
k3_lon = (0:0.05:10) * -Iy;
% Find k3_lon for Tau = 1.25s
eigA_lon = zeros(length(k3_lon),3);
j = 1;
for i = 1:length(k3_lon)
   A = [0 -g 0;
        0 0 1
        -k3_lon(i)/Iy -k2_lon/Iy -k1_lon/Iy];
   eigA_lon(j,:)= eig(A);
   figure(105);
   plot(real(eigA_lon), imag(eigA_lon), '.k');
   hold on
   j = j + 1;
end
plot(real(eigA_lon(39,:)), imag(eigA_lon(39,:)), '.g');
xline(-.8,'--r');
xlabel('Real')
ylabel('Imaginary')
title('Longitudinal Root Locus Plot')
hold off
k3_lon = k3_lon(39);
%% 3.7
% Lateral
% Initial State Vector
var_j = [0;0;0;0;0;0;0;0;0;0;0;0];
% ode45 Call
[tj,var_dot_j] = ode45(@(tspan,var_j)
QuadrotorEOMTranslation(tspan,var_j,g,m,I,nu,mu,'Lateral'),tspan,var_j);
% Control Forces/Moments
for k = 1:numel(tj)
   [~,Lc_j(:,k),Mc_j(:,k),Nc_j(:,k)] =
QuadrotorEOMTranslation(tj(k),var_dot_j(k,:), g, m, I, nu, mu, 'Lateral');
end
[Fc_j,Gc_j] = VelocityReferenceFeedbackLat(tspan,var_j);
Fc_j = (Fc_j * ones(length(Lc_j),1))';
% Control/Figures Vector
controls_j = [Fc_j;Lc_j;Mc_j;Nc_j];
fig_j = 106:111;
% PlotAircraftSim Call
PlotAircraftSim(tj,var_dot_j,controls_j,fig_j,'-b')
%--------------------------------------------------------------------------
------
% Longitudinal
% ode45 Call
[tk,var_dot_k] = ode45(@(tspan,var_j)
QuadrotorEOMTranslation(tspan,var_j,g,m,I,nu,mu,'Longitudinal'),tspan,var_j);
% Control Forces/Moments
for k = 1:numel(tj)
   [~,Lc_k(:,k),Mc_k(:,k),Nc_k(:,k)] =
QuadrotorEOMTranslation(tj(k),var_dot_j(k,:), g, m, I, nu, mu, 'Longitudinal');
end
[Fc_k,Gc_k] = VelocityReferenceFeedbackLon(tspan,var_j);
Fc_k = (Fc_k * ones(length(Lc_j),1))';
% Control/Figure Vectors
controls_k = [Fc_k;Lc_k;Mc_k;Nc_k];
fig_k = 112:117;
% PlotAircraftSim Call
PlotAircraftSim(tk,var_dot_k,controls_k,fig_k,'-b')
```

```
%% 3.8
% Tested Gains
% Load Data
gains_data = load('RSdata_10_26.mat');
% Figure Vector
fig_1 = 118:123;
% PlotAircraftSim Call
PlotAircraftSim(gains_data.rt_estim.time,gains_data.rt_estim.signals.values,sqr
t(abs(gains_data.rt_motor.signals.values)*13840.4)',fig_1,'-b')
% Note: Not sure how to get Zc, Lc, Mc, Nc from data, so those graphs can be
ignored
```

1.1: PlotAircraftSim

```
function PlotAircraftSim(time, aircraft_state_array, control_input_array, fig,
col)
%-----------------------------------------------------------------
% Inputs: length n time of nth set of variables
%         12xn aircraft states
%         4xn control inputs (Zc,Lc,Mc,Nc)
%         6x1 figure numbers to plot over
%         string col
% Outputs: 6 figures
%                 4 figs w/ 3 subplot (inertial pos, Euler angle, interial
%                       velocity in body, angular velocity)
%                 1 fig w/ 4 subplot (each control input variable)
%                 1 fig w/ 3D path (pos height up, start green, finish red)
% Methodology: plot full aircraft states and control inputs
%-----------------------------------------------------------------
% Inertial Position
figure(fig(1));
subplot(311); % x direction
plot(time, aircraft_state_array(:,1), col); hold on;
title('Inertial Position')
xlabel('time [s]'); ylabel('x [m]');
subplot(312); % y direction
plot(time, aircraft_state_array(:,2), col); hold on;
xlabel('time [s]'); ylabel('y [m]')
subplot(313); % z direction
plot(time, aircraft_state_array(:,3), col); hold on;
xlabel('time [s]'); ylabel('z [m]');
% Euler Angles
figure(fig(2));
subplot(311);
plot(time, aircraft_state_array(:,4), col); hold on;
title('Euler Angles')
xlabel('time [s]'); ylabel('\phi [rad]');
subplot(312);
plot(time, aircraft_state_array(:,5), col); hold on;
xlabel('time [s]'); ylabel('\theta [rad]');
subplot(313);
plot(time, aircraft_state_array(:,6), col); hold on;
xlabel('time [s]'); ylabel('\psi [rad]');
% Inertial Velocity in Body Frame
figure(fig(3));
subplot(311);
plot(time, aircraft_state_array(:,7), col); hold on;
title('Inertial Velocity in Body Frame')
xlabel('time [s]'); ylabel('u^{E} [m/s]');
subplot(312);
```

```
plot(time, aircraft_state_array(:,8), col); hold on;
xlabel('time [s]'); ylabel('v^{E} [m/s]');
subplot(313);
plot(time, aircraft_state_array(:,9), col); hold on;
xlabel('time [s]'); ylabel('w^{E} [m/s]');
% Angular Velocity
figure(fig(4));
subplot(311);
plot(time, aircraft_state_array(:,10), col); hold on;
title('Angular Velocity')
xlabel('time [s]'); ylabel('p [rad/s]');
subplot(312);
plot(time, aircraft_state_array(:,11), col); hold on;
xlabel('time [s]'); ylabel('q [rad/s]');
subplot(313);
plot(time, aircraft_state_array(:,12), col); hold on;
xlabel('time [s]'); ylabel('r [rad/s]');
% Control Input Variables
Zc = control_input_array(1,:);
Lc = control_input_array(2,:);
Mc = control_input_array(3,:);
Nc = control_input_array(4,:);
figure(fig(5));
subplot(411);
plot(time, Zc, col); hold on;
title('Control Input Variables')
xlabel('time [s]'); ylabel('Z_{c} [N]');
subplot(412);
plot(time, Lc, col); hold on;
xlabel('time [s]'); ylabel('L_{c} [Nm]');
% ylim([-.001 0.001])
subplot(413);
plot(time, Mc, col); hold on;
xlabel('time [s]'); ylabel('M_{c} [Nm]');
% ylim([-.001 0.001])
subplot(414);
plot(time, Nc, col); hold on;
xlabel('time [s]'); ylabel('N_{c} [Nm]');
% ylim([-.001 0.001])
% 3D Path of Aircraft
len = length(time);
figure(fig(6))
plot3(aircraft_state_array(:,1),aircraft_state_array(:,2),aircraft_state_array(
:,3),col)
hold on
grid on; grid minor;
title('3D Path of Aircraft')
xlabel('x [m]'); ylabel('y [m]'); zlabel('z [m]');
plot3(aircraft_state_array(1,1),aircraft_state_array(1,2),aircraft_state_array(
1,3),'.g','MarkerSize',12); hold on;
plot3(aircraft_state_array(len,1),aircraft_state_array(len,2),aircraft_state_ar
ray(len,3),'.r','MarkerSize',12); hold on;
end
```

1.2: QuadrotorEOM

```
function [var_dot] = QuadrotorEOM(~, var, g, m, I, d, km, nu, mu, motor_forces)
%-------------------------------------------------------------
% Inputs:
%   var: 12x1 state vector
```

```
%   g: gravity
%   m: quadcopter mass
%   I: inertia matrix
%   d: motor distance from center of gravity
%   km: control moment coefficient
%   nu: aerodynamic force coefficient
%   mu: aerodynamic moment coefficient
%   motor_forces: 4x1 vector of force generated by each motor
% Outputs:
%   var_dot: 12x1 state vector derivative
%-------------------------------------------------------------------
% Extract var/motor_forces/I data
phi = var(4);
theta = var(5);
psi = var(6);
uE = var(7);
vE = var(8);
wE = var(9);
inert_vel_b = [uE;vE;wE];
p = var(10);
q = var(11);
r = var(12);
ang_vel = [p;q;r];
Ix = I(1);
Iy = I(2);
Iz = I(3);
% Calculate Lc,Mc,Nc,Zc from motor_forces
controlFM_Mat = [-1,-1,-1,-1;
             -d/sqrt(2),-d/sqrt(2),d/sqrt(2),d/sqrt(2);
             d/sqrt(2),-d/sqrt(2),-d/sqrt(2),d/sqrt(2);
             km,-km,km,-km];
control_FM = controlFM_Mat * motor_forces;
Zc = control_FM(1);
Lc = control_FM(2);
Mc = control_FM(3);
Nc = control_FM(4);
% Moments Calculations
moments = -mu * norm(ang_vel) * ang_vel;
L = moments(1);
M = moments(2);
N = moments(3);
% Aerodynamic Forces Calculation
aero_f = -nu * norm(inert_vel_b) * inert_vel_b;
X = aero_f(1);
Y = aero_f(2);
Z = aero_f(3);
% Trigonometric Evaluated Angles
cPH = cos(phi);
cT = cos(theta);
cPS = cos(psi);
sPH = sin(phi);
sT = sin(theta);
sPS = sin(psi);
tT = tan(theta);
secT = sec(theta);
% Rotation Matrix
Rot321 = [cT*cPS,sPH*sT*cPS-cPH*sPS,cPH*sT*cPS+sPH*sPS;
            cT*sPS,sPH*sT*sPS+cPH*cPS,cPH*sT*sPS-sPH*cPS;
            -sT,sPH*cT,cPH*cT];
```

```
% Euler Angle Rates Matrix
EulMat = [1,sPH*tT,cPH*tT;
          0,cPH,-sPH
          0,sPH*secT,cPH*secT];
% Inertial Position Derivatives
inerital_pos_dot = Rot321 * [uE;vE;wE];
% Euler Angle Derivatives
euler_angle_rates = EulMat * ang_vel;
% Inertial Velocity in Body Derivatives
uE_dot = r*vE - q*wE - g*sT + X/m;
vE_dot = p*wE - r*uE + g*cT*sPH + Y/m;
wE_dot = q*uE - p*vE + g*cT*cPH + Z/m + (Zc/m);
% Roll Rate Derivatives
p_dot = (Iy - Iz)/Ix*q*r + L/Ix + Lc/Ix;
q_dot = (Iz - Ix)/Iy*p*r + M/Iy + Mc/Iy;
r_dot = (Ix - Iy)/Iz*p*q + N/Iz + Nc/Iz;
% Final State Derivative Vector
var_dot =
[inerital_pos_dot;euler_angle_rates;uE_dot;vE_dot;wE_dot;p_dot;q_dot;r_dot];
end
```

2.2: QuadrotorEOM_Linearized

```
function var_dot = QuadrotorEOM_Linearized(~, var, g, m, I, deltaFc, deltaGc)
%-----------------------------------------------------------------
% Inputs: t = time
%         var = 12x1 column of state deviations from steady hover
%         g = gravity
%         m = quadcopter mass
%         I = inertia matrix
%         deltaFc = deviation from steady hover trim
%         deltaGc = deviation from steady hover trim
% Output: var_dot = 12x1 state vector derivative
%-----------------------------------------------------------------
% Extract var Components
delta_phi = var(4);
delta_theta = var(5);
delta_u = var(7);
delta_v = var(8);
delta_w = var(9);
delta_p = var(10);
delta_q = var(11);
delta_r = var(12);
deltaZc = deltaFc;
deltaLc = deltaGc(1);
deltaMc = deltaGc(2);
deltaNc = deltaGc(3);
% Extract I Components
Ix = I(1);
Iy = I(2);
Iz = I(3);
% Linearized EOMs
delta_xE_dot = delta_u;
delta_yE_dot = delta_v;
delta_zE_dot = delta_w;
delta_phi_dot = delta_p;
delta_theta_dot = delta_q;
delta_psi_dot = delta_r;
delta_u_dot = - g * delta_theta;
delta_v_dot = g * delta_phi;
```

```
delta_w_dot = deltaZc / m;
delta_p_dot = deltaLc / Ix;
delta_q_dot = deltaMc / Iy;
delta_r_dot = deltaNc / Iz;
% Final State Derivative Vector
var_dot = [delta_xE_dot;delta_yE_dot;delta_zE_dot
   delta_phi_dot;delta_theta_dot;delta_psi_dot
   delta_u_dot;delta_v_dot;delta_w_dot
   delta_p_dot;delta_q_dot;delta_r_dot];
end
```

2.3: RotationDerivativeFeedback

```
function [Fc,Gc] = RotationDerivativeFeedback(var,m,g)
%-------------------------------------------------
% Inputs:
%   var: 12x1 quadrotor state vector
%   m: quadcopter mass
%   g: gravity
% Outputs:
%   Fc: control force
%   Gc: 3x1 control moments vector
%-------------------------------------------------
p = var(10);
q = var(11);
r = var(12);
k = 0.004; % given gain
Zc = -m*g;
Lc = k*-p;
Mc = k*-q;
Nc = k*-r;
Fc = Zc;
Gc = [Lc;Mc;Nc];
end
```

2.4: ComputeMotorForces

```
function motor_forces = ComputeMotorForces(Fc,Gc,km,d)
%-------------------------------------------------
% Inputs:
%   Fc: control force
%   Gc: 3x1 control moments vector
%   d: motor distance from center of gravity
%   km: control moment coefficient
% Outputs:
%   motor_forces: 4x1 vector of force generated by each motor
%-------------------------------------------------
control_force_moments = [Fc;Gc];
CM = [-1,-1,-1,-1;
    -d/sqrt(2),-d/sqrt(2),d/sqrt(2),d/sqrt(2);
     d/sqrt(2),-d/sqrt(2),-d/sqrt(2),d/sqrt(2);
     km,-km,km,-km];
motor_forces = CM^-1 * control_force_moments;
end
```

2.5: QuadrotorEOMwithRateFeedback

```
function [var_dot,Lc,Mc,Nc] = QuadrotorEOMwithRateFeedback(~,var,g,m,I,nu,mu)
%-----------------------------------------------------------------
% Inputs:
%   var: 12x1 state vector
```

```
%   g: gravity
%   m: quadcopter mass
%   I: inertia matrix
%   nu: aerodynamic force coefficient
%   mu: aerodynamic moment coefficient
% Outputs:
%   var_dot: 12xN state vector derivative
%   Lc: X control moment
%   Mc: Y control moment
%   Nc: Z control moment
%-------------------------------------------------------------------
% Extract var/motor_forces/I Data
phi = var(4);
theta = var(5);
psi = var(6);
uE = var(7);
vE = var(8);
wE = var(9);
inert_vel_b = [uE;vE;wE];
p = var(10);
q = var(11);
r = var(12);
ang_vel = [p;q;r];
Ix = I(1);
Iy = I(2);
Iz = I(3);
airspeed = norm(inert_vel_b);
omega_mag = norm(ang_vel);
% Angular Control
[Fc,Gc] = RotationDerivativeFeedback(var,m,g);
Lc = Gc(1);
Mc = Gc(2);
Nc = Gc(3);
% Trigonometric Evaluated Angles
cPH = cos(phi);
cT = cos(theta);
cPS = cos(psi);
sPH = sin(phi);
sT = sin(theta);
sPS = sin(psi);
tT = tan(theta);
secT = sec(theta);
% Rotation Matrix
Rot321 = [cT*cPS,sPH*sT*cPS-cPH*sPS,cPH*sT*cPS+sPH*sPS;
         cT*sPS,sPH*sT*sPS+cPH*cPS,cPH*sT*sPS-sPH*cPS;
         -sT,sPH*cT,cPH*cT];
% Inertial Position Derivatives
inerital_pos_dot = Rot321 * inert_vel_b;
% Euler Angle Rates Matrix
EulMat = [1,sPH*tT,cPH*tT;
         0,cPH,-sPH
         0,sPH*secT,cPH*secT];

% Euler Angle Derivatives
euler_angle_rates = EulMat * ang_vel;
% Calculate velocity dynamics
vel_dot1 =    [ang_vel(3)*inert_vel_b(2)-ang_vel(2)*inert_vel_b(3);
               ang_vel(1)*inert_vel_b(3)-ang_vel(3)*inert_vel_b(1);
               ang_vel(2)*inert_vel_b(1)-ang_vel(1)*inert_vel_b(2)];
```

```
vel_dot2 =    g * [-sT;
                   cT*sPH;
                   cT*cPH];

vel_dot3 =    -nu*airspeed/m * inert_vel_b;

vel_dot4 =    [0;0;Fc/m];
vel_dot = vel_dot1 + vel_dot2 + vel_dot3 + vel_dot4;
% Roll Rate Derivatives
ang_vel_dot1 =    [(Iy-Iz)*ang_vel(2)*ang_vel(3)/Ix;
                   (Iz-Ix)*ang_vel(1)*ang_vel(3)/Iy;
                   (Ix-Iy)*ang_vel(1)*ang_vel(2)/Iz];

ang_vel_dot2 =    -mu*omega_mag*ang_vel./I;

ang_vel_dot3 =    Gc./I;
angvel_dot = ang_vel_dot1 + ang_vel_dot2 + ang_vel_dot3;
% Final State Derivative Vector
var_dot = [inerital_pos_dot;euler_angle_rates;vel_dot;angvel_dot];
end
```

3.2: InnerLoopFeedback

```
function [Fc,Gc] = InnerLoopFeedback(var)
%-------------------------------------------------
% Inputs:
%   var: 12x1 quadrotor state vector
% Outputs:
%   Fc: control force
%   Gc: 3x1 control moments vector
%-------------------------------------------------
% Extract var Parameters
phi = var(4);
theta = var(5);
p = var(10);
q = var(11);
r = var(12);
% Constants
m = 0.068; %[kg]
g = 9.81; %[m/s^2]
% Gains
k1_lat = 0.001276;
k2_lat = 0.00232;
k1_lon = 0.0022;
k2_lon = 0.004;
k_spin = 0.004;
% Fc Calculation
Fc = -m*g;
% Gc Calculations
Lc = -k1_lat*p - k2_lat*phi;
Mc = -k1_lon*q - k2_lon*theta;
Nc = -k_spin*r;
Gc = [Lc;Mc;Nc];
end
```

3.3: QuadrotorEOM_LinearizedWithILFeedback

```
function [var_dot,Lc,Mc,Nc] = QuadrotorEOM_LinearizedWithILFeedback(~, var, g,
```

```
m, I)
%-------------------------------------------------------------------
% Inputs: t = time
%         var = 12x1 column of state deviations from steady hover
%         g = gravity
%         m = quadcopter mass
%         I = inertia matrix
%
% Output: var_dot = 12x1 state vector derivative
%         Lc: X control moment
%         Mc: Y control moment
%         Nc: Z control moment
%-------------------------------------------------------------------
% Extract var Components
delta_phi = var(4);
delta_theta = var(5);
delta_u = var(7);
delta_v = var(8);
delta_w = var(9);
delta_p = var(10);
delta_q = var(11);
delta_r = var(12);
% Extract I Components
Ix = I(1);
Iy = I(2);
Iz = I(3);
% InnerLoopFeedBack Call
[Fc,Gc] = InnerLoopFeedback(var);
deltaFc = 0;
Fc = deltaFc;
Lc = Gc(1);
Mc = Gc(2);
Nc = Gc(3);
% Linearized EOMs
delta_xE_dot = delta_u;
delta_yE_dot = delta_v;
delta_zE_dot = delta_w;
delta_phi_dot = delta_p;
delta_theta_dot = delta_q;
delta_psi_dot = delta_r;
delta_u_dot = - g * delta_theta;
delta_v_dot = g * delta_phi;
delta_w_dot = Fc / m;
delta_p_dot = Gc(1) / Ix;
delta_q_dot = Gc(2) / Iy;
delta_r_dot = Gc(3) / Iz;
% Final State Derivative Vector
var_dot = [delta_xE_dot;delta_yE_dot;delta_zE_dot
   delta_phi_dot;delta_theta_dot;delta_psi_dot
   delta_u_dot;delta_v_dot;delta_w_dot
   delta_p_dot;delta_q_dot;delta_r_dot];
end
```

3.4: QuadrotorEOMWithILFeedback

```
function [var_dot,Lc,Mc,Nc] = QuadrotorEOMWithILFeedback(~, var, g, m, I, nu,
mu)
%-------------------------------------------------------------------
% Inputs:
%   var: 12x1 state vector
```

```
%   g: gravity
%   m: quadcopter mass
%   I: inertia matrix
%   nu: aerodynamic force coefficient
%   mu: aerodynamic moment coefficient
% Outputs:
%   var_dot: 12xN state vector derivative
%   Lc: X control moment
%   Mc: Y control moment
%   Nc: Z control moment
%-------------------------------------------------------------------
% Extract var/motor_forces/I data
phi = var(4);
theta = var(5);
psi = var(6);
uE = var(7);
vE = var(8);
wE = var(9);
inert_vel_b = [uE;vE;wE];
p = var(10);
q = var(11);
r = var(12);
ang_vel = [p;q;r];
Ix = I(1);
Iy = I(2);
Iz = I(3);
% Calculate Lc, Mc, Nc, Zc
[Zc,Gc] = InnerLoopFeedback(var);
Lc = Gc(1);
Mc = Gc(2);
Nc = Gc(3);
% Moments Calculations
moments = -mu * norm(ang_vel) * ang_vel;
L = moments(1);
M = moments(2);
N = moments(3);
% Aerodynamic Forces Calculation
aero_f = -nu * norm(inert_vel_b) * inert_vel_b;
X = aero_f(1);
Y = aero_f(2);
Z = aero_f(3);
% Trigonometric Evaluated Angles
cPH = cos(phi);
cT = cos(theta);
cPS = cos(psi);
sPH = sin(phi);
sT = sin(theta);
sPS = sin(psi);
tT = tan(theta);
secT = sec(theta);
% Rotation Matrix
Rot321 = [cT*cPS,sPH*sT*cPS-cPH*sPS,cPH*sT*cPS+sPH*sPS;
            cT*sPS,sPH*sT*sPS+cPH*cPS,cPH*sT*sPS-sPH*cPS;
            -sT,sPH*cT,cPH*cT];
% Euler Angle Rates Matrix
EulMat = [1,sPH*tT,cPH*tT;
         0,cPH,-sPH
         0,sPH*secT,cPH*secT];
% Inertial Position Derivatives
```

```
inerital_pos_dot = Rot321 * [uE;vE;wE];
% Euler Angle Derivatives
euler_angle_rates = EulMat * ang_vel;
% Inertial Velocity in Body Derivatives
uE_dot = r*vE - q*wE - g*sT + X/m;
vE_dot = p*wE - r*uE + g*cT*sPH + Y/m;
wE_dot = q*uE - p*vE + g*cT*cPH + Z/m + (Zc/m);
% Roll Rate Derivatives
p_dot = (Iy - Iz)/Ix*q*r + L/Ix + Lc/Ix;
q_dot = (Iz - Ix)/Iy*p*r + M/Iy + Mc/Iy;
r_dot = (Ix - Iy)/Iz*p*q + N/Iz + Nc/Iz;
% Final State Derivative Vector
var_dot =
[inerital_pos_dot;euler_angle_rates;uE_dot;vE_dot;wE_dot;p_dot;q_dot;r_dot];
end
```

3.6: VelocityReferenceFeedbackLat

```
function [Fc,Gc] = VelocityReferenceFeedbackLat(t,var)
%-------------------------------------------------
% Inputs:
%   t: time vector
%   var: 12x1 quadrotor state vector
% Outputs:
%   Fc: control force
%   Gc: 3x1 control moments vector
%-------------------------------------------------
% Constants
m = 0.068; %[kg]
g = 9.81; %[m/s^2]
% Gain Values
k1_lat = 0.001276;
k2_lat = 0.00232;
k3_lat = 1.102E-4;
k1_lon = 0.001584;
k2_lon = 0.00288;
k3_lon = -1.368E-4;
k_spin = 0.004;
% var Values Needed
x = var(1);
y = var(2);
phi = var(4);
theta = var(5);
u = var(7);
v = var(8);
p = var(10);
q = var(11);
r = var(12);
% Determine vr, ur From t
if t <= 2
   vr = 0.5;
   ur = 0;
else
   vr = 0;
   ur = 0;
end
% Calculate Lc, Mc, Nc
Lc = -k1_lat*p - k2_lat*phi + k3_lat*(vr - v);
Mc = -k1_lon*q - k2_lon*theta + k3_lon*(ur - u);
Nc = -k_spin*r;
```

```
% Fc and Gc
Fc = -m*g;
Gc = [Lc;Mc;Nc];
end
```

3.6: VelocityReferenceFeedbackLon

```
function [Fc,Gc] = VelocityReferenceFeedbackLon(t,var)
%-------------------------------------------------
% Inputs:
%   t: time vector
%   var: 12x1 quadrotor state vector
% Outputs:
%   Fc: control force
%   Gc: 3x1 control moments vector
%-------------------------------------------------
% Constants
m = 0.068; %[kg]
g = 9.81; %[m/s^2]
% Gain Values
k1_lat = 0.001276;
k2_lat = 0.00232;
k3_lat = 1.102E-4;
k1_lon = 0.001584;
k2_lon = 0.00288;
k3_lon = -1.368E-4;
k_spin = 0.004;
% var Values Needed
phi = var(4);
theta = var(5);
u = var(7);
v = var(8);
p = var(10);
q = var(11);
r = var(12);
% Determine vr, ur From Time
if t <= 2
   vr = 0;
   ur = 0.5;
else
   vr = 0;
   ur = 0;
end
% Calculate Lc, Mc, Nc
Lc = -k1_lat*p - k2_lat*phi + k3_lat*(vr - v);
Mc = -k1_lon*q - k2_lon*theta + k3_lon*(ur - u);
Nc = -k_spin*r;
% Fc and Gc
Fc = -m*g;
Gc = [Lc;Mc;Nc];
end
```

3.7: QuadrotorEOMTranslation

```
function [var_dot,Lc,Mc,Nc] = QuadrotorEOMTranslation(t, var, g, m, I, nu, mu,
mode)
%-----------------------------------------------------------------
% Inputs:
%   var: 12x1 state vector
%   g: gravity
%   m: quadcopter mass
```

```
%   I: inertia matrix
%   nu: aerodynamic force coefficient
%   mu: aerodynamic moment coefficient
%   mode: string that is either 'Lateral' or 'Longitudinal'
%         *Note: Capitalization doesn't matter
% Outputs:
%   var_dot: 12xN state vector derivative
%   Lc: X control moment
%   Mc: Y control moment
%   Nc: Z control moment
%-------------------------------------------------------------------
% Extract var/motor_forces/I data
phi = var(4);
theta = var(5);
psi = var(6);
uE = var(7);
vE = var(8);
wE = var(9);
inert_vel_b = [uE;vE;wE];
p = var(10);
q = var(11);
r = var(12);
ang_vel = [p;q;r];
Ix = I(1);
Iy = I(2);
Iz = I(3);
% Calculate Lc, Mc, Nc, Zc
M1 = 'Lateral';
M2 = 'Longitudinal';
Mcomp1 = strcmpi(M1,mode);
Mcomp2 = strcmpi(M2,mode);
if Mcomp1 == 1 && Mcomp2 == 0
   [Zc,Gc] = VelocityReferenceFeedbackLat(t,var);
   Lc = Gc(1);
   Mc = Gc(2);
   Nc = Gc(3);
elseif Mcomp1 == 0 && Mcomp2 == 1
   [Zc,Gc] = VelocityReferenceFeedbackLon(t,var);
   Lc = Gc(1);
   Mc = Gc(2);
   Nc = Gc(3);
else
   disp('Invalid Mode Entered');
end
% Moments Calculations
moments = -mu * norm(ang_vel) * ang_vel;
L = moments(1);
M = moments(2);
N = moments(3);
% Aerodynamic Forces Calculation
aero_f = -nu * norm(inert_vel_b) * inert_vel_b;
X = aero_f(1);
Y = aero_f(2);
Z = aero_f(3);
% Trigonometric Evaluated Angles
cPH = cos(phi);
cT = cos(theta);
cPS = cos(psi);
sPH = sin(phi);
```

```
sT = sin(theta);
sPS = sin(psi);
tT = tan(theta);
secT = sec(theta);
% Rotation Matrix
Rot321 = [cT*cPS,sPH*sT*cPS-cPH*sPS,cPH*sT*cPS+sPH*sPS;
            cT*sPS,sPH*sT*sPS+cPH*cPS,cPH*sT*sPS-sPH*cPS;
            -sT,sPH*cT,cPH*cT];
% Euler Angle Rates Matrix
EulMat = [1,sPH*tT,cPH*tT;
         0,cPH,-sPH
         0,sPH*secT,cPH*secT];
% Inertial Position Derivatives
inerital_pos_dot = Rot321 * [uE;vE;wE];
% Euler Angle Derivatives
euler_angle_rates = EulMat * ang_vel;
% Inertial Velocity in Body Derivatives
uE_dot = r*vE - q*wE - g*sT + X/m;
vE_dot = p*wE - r*uE + g*cT*sPH + Y/m;
wE_dot = q*uE - p*vE + g*cT*cPH + Z/m + (Zc/m);
% Roll Rate Derivatives
p_dot = (Iy - Iz)/Ix*q*r + L/Ix + Lc/Ix;
q_dot = (Iz - Ix)/Iy*p*r + M/Iy + Mc/Iy;
r_dot = (Ix - Iy)/Iz*p*q + N/Iz + Nc/Iz;
% Final State Derivative Vector
var_dot =
[inerital_pos_dot;euler_angle_rates;uE_dot;vE_dot;wE_dot;p_dot;q_dot;r_dot];
end
```