University of Colorado - Boulder

ASEN 3128 : Lab 3

Submitted October 9th, 2020

Lab 3: Quadrotor Controls

Author:

Ryan Collins[a]

Author:

Gio Borrani[b]

Author:

Connor O'Reilly[c]

Author:

Thyme Zuschlag[d]

Professor:

Professor Frew

[a] 104768223
[b] 109216456
[c] 107054811
[d] 109221211

College of Engineering & Applied Science
UNIVERSITY OF COLORADO BOULDER

Table of Contents

Page

I Plots . . . 2

II Problem 3 . . . 7

III Problem 4 . . . 9

IV Problem 5 . . . 10

Participation . . . 14

Appendices . . . 14

Code . . . 37

Derivations . . . 37

I. Plots

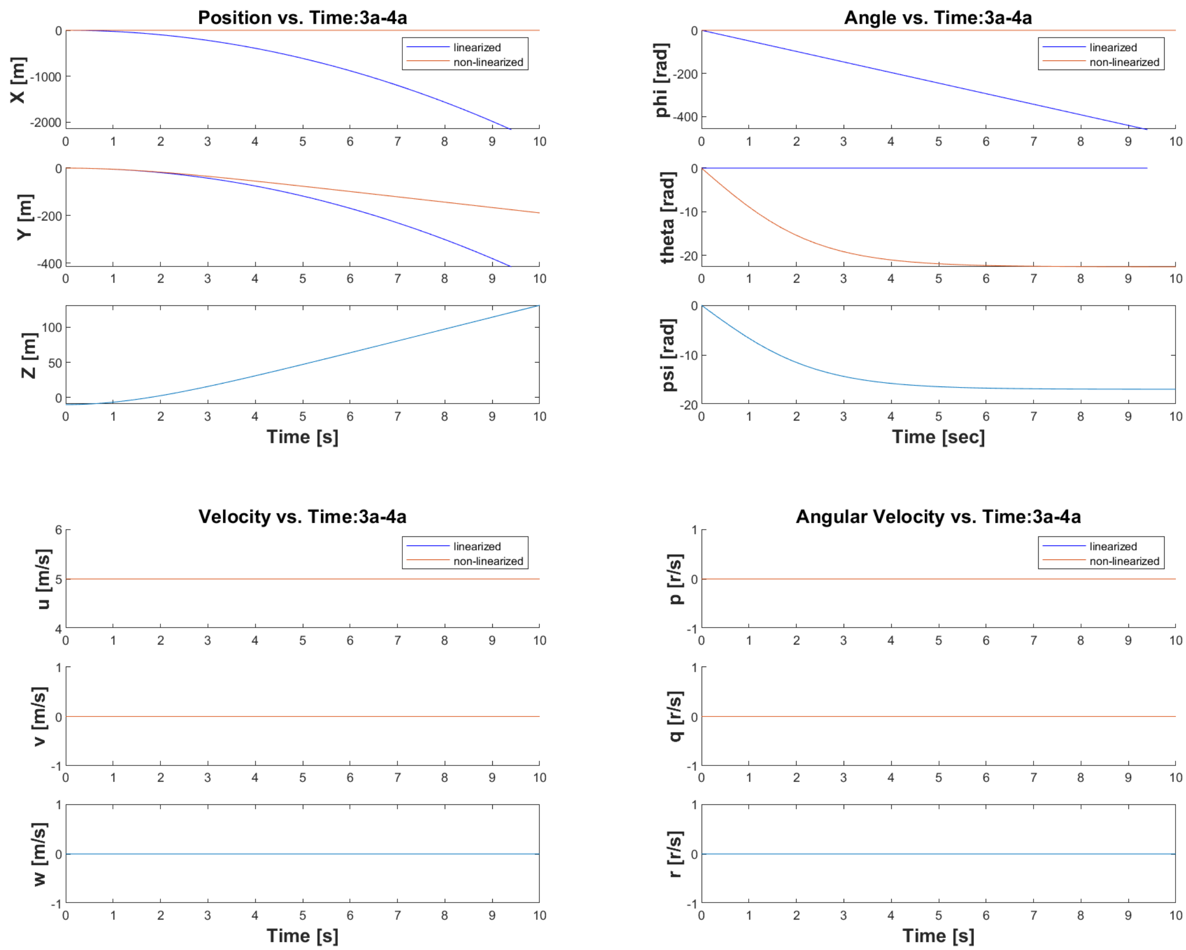


Fig. 1 5 deg initial roll deviation

The above figure denotes the first task in which a 5 degree roll deviation is introduced. While the simulation suggests that there would be a change in position among all three axes, a change in Angle among three axes, and no change to the velocity, we believe that there would be a positive change in the x direction and a converging change (to zero) in the z direction. Additionally, the angle would only change about phi (which should remain relatively constant), whereas the velocity should increase for w and u should be a constant as time increases.

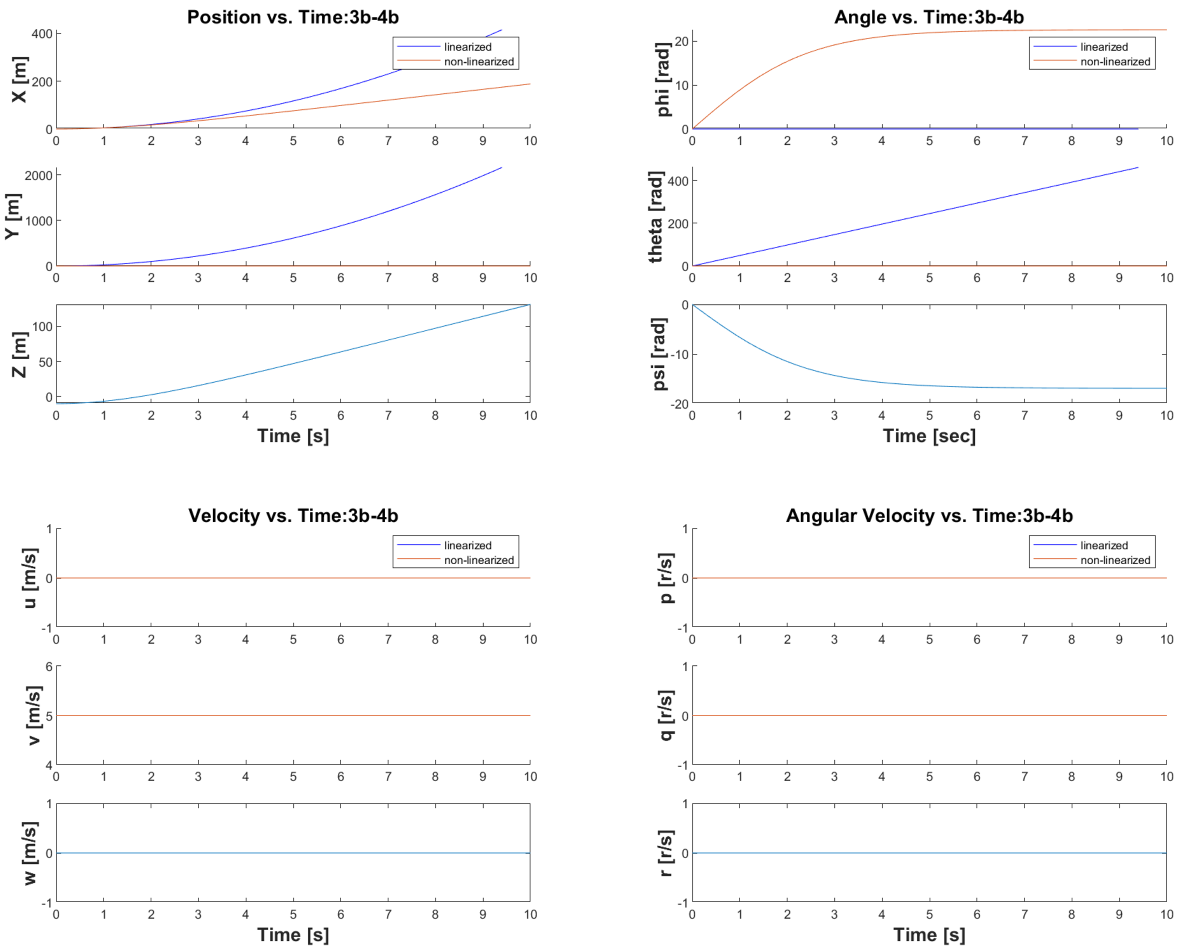


Fig. 2

While we observe that there is an increasing position in all three dimensions, we were anticipating to see a change in the Y and Z direction, (where z is decreasing to 0). We only expected to see changes in the theta direction (not in the psi or phi position). Additionally, if there were to be any change in velocity it would be seen in the positive direction for both v and w. That aside the angular velocity is as expected.

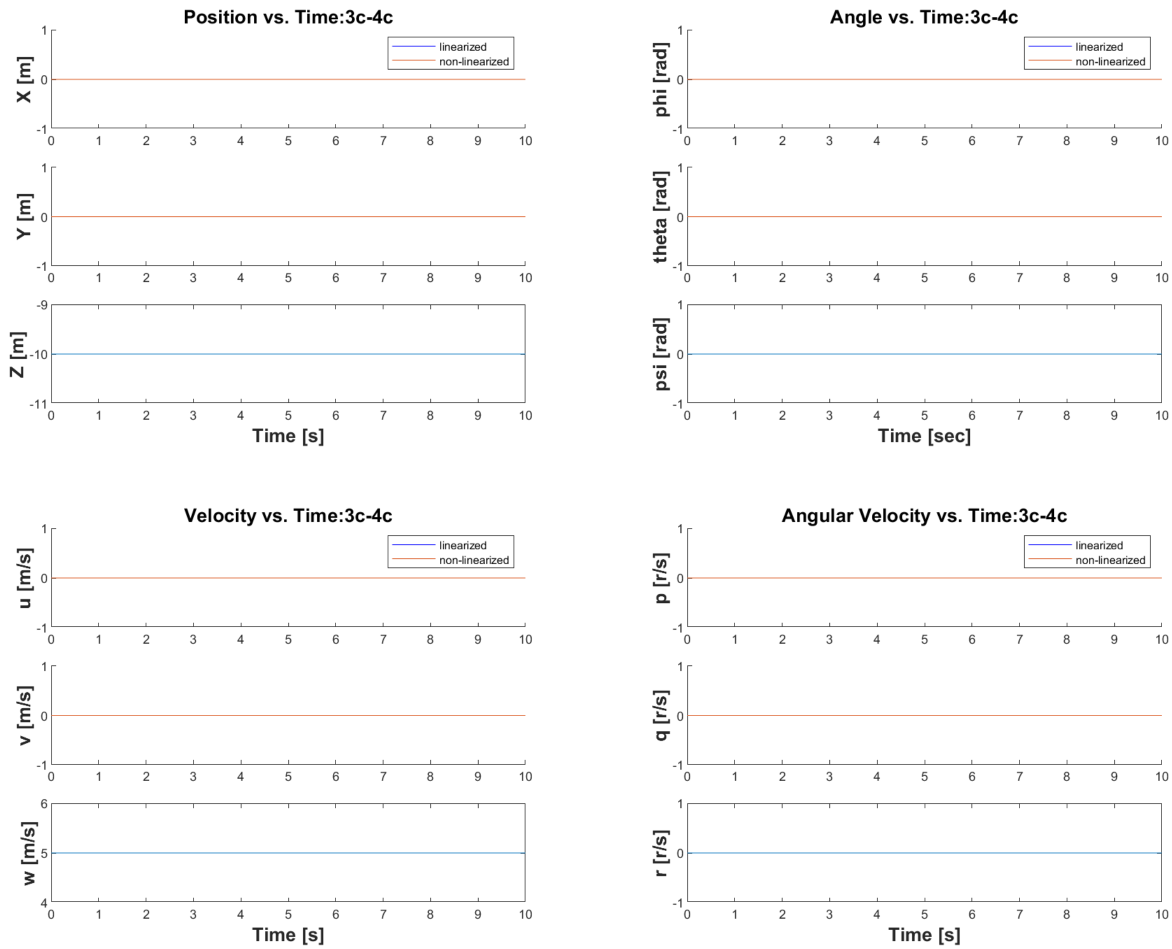


Fig. 3

Given the rotation about the orientation 5 degrees it can be observed that no change in steady level flight occurs through out the timespan. We do anticipate that an impulsive change the yaw would occur at t=0, although it still should remain constant through the duration of flight.

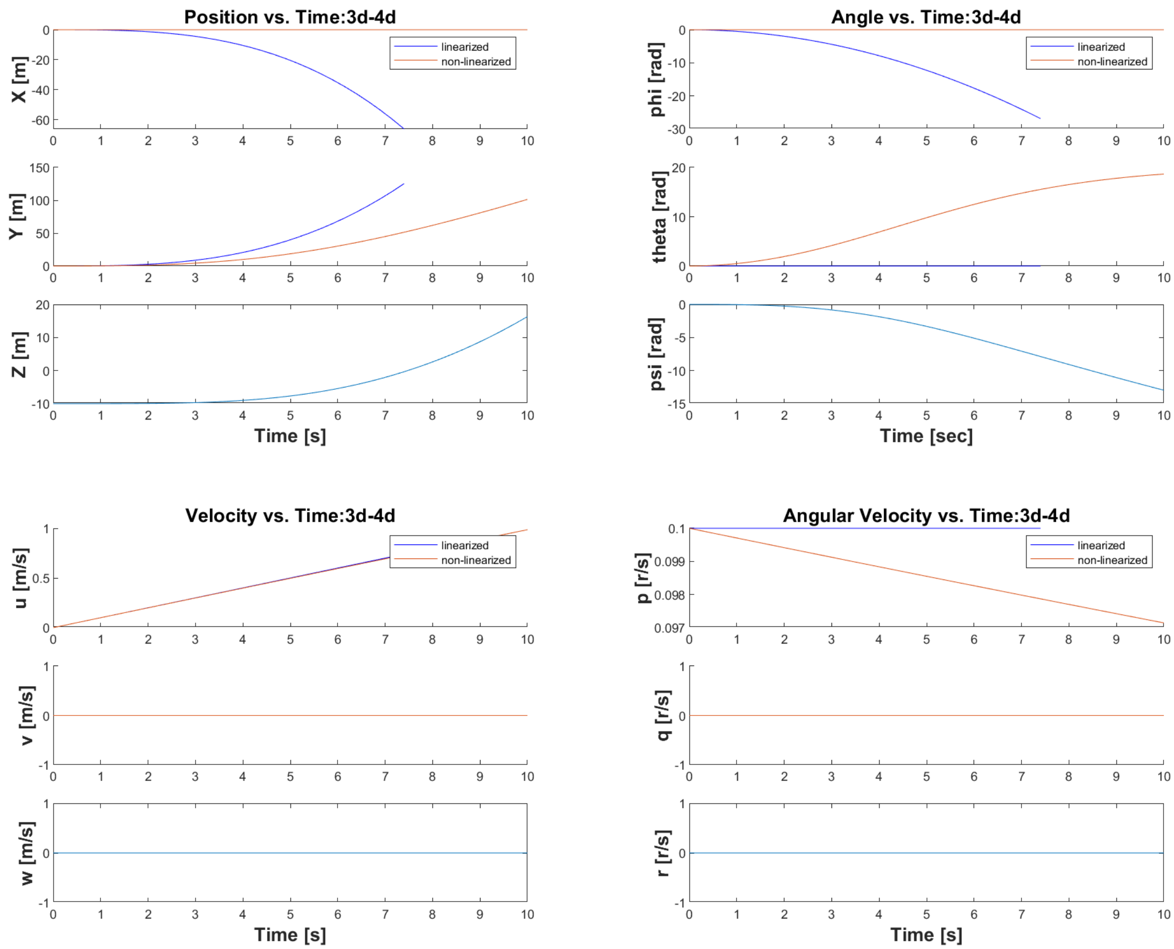


Fig. 4

For task D the roll rate is not 0, in the graphs we are observing that it will begin translating in the u direction. We were anticipating that a change in the pitch "rate" would result in a change in the w and u (velocity) in the positive direction. Additionally, we were anticipating for the X and Z positions to be changing but not the Y component. Finally, we were only anticipating for the phi value to change with respect to time, there is no reason for the theta and psi graphs to be changing.

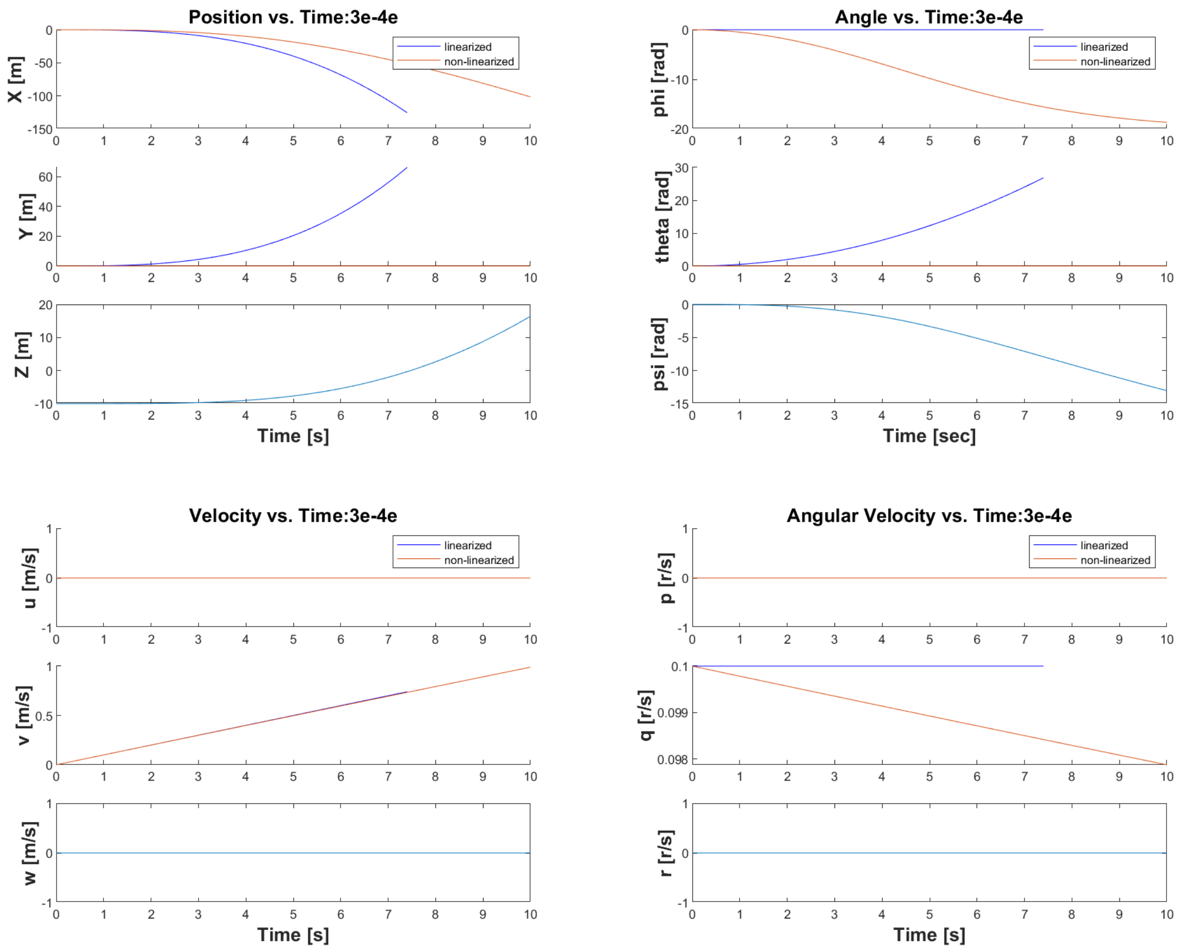


Fig. 5

Similarly to Figure 4 we anticipated to see movement in the Y and Z planes, angle change in just the theta position, the Velocity should have changed in just the V and W directions, and the angular velocity should be changing in the q (as shown). This make sense because the z-input force is no longer strong enough to balance the force of gravity.

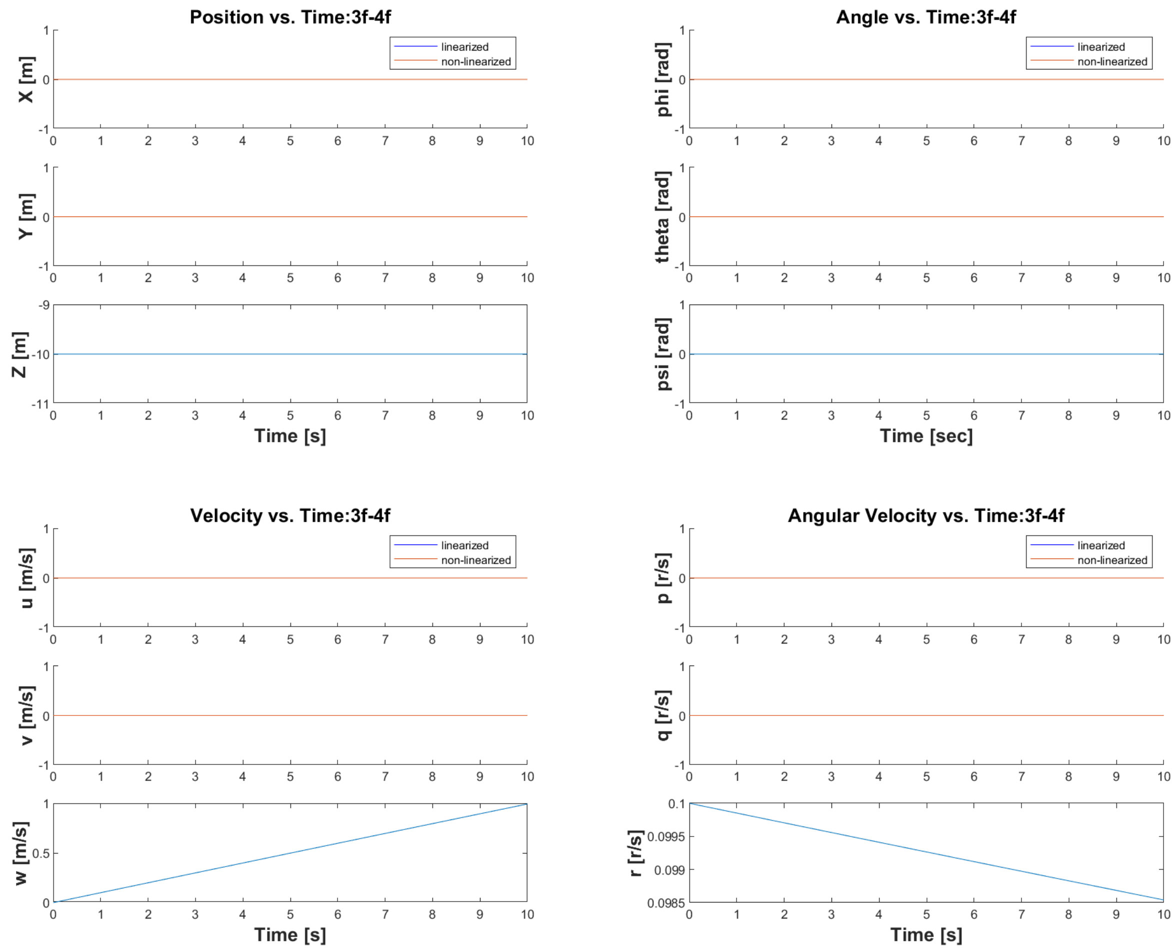


Fig. 6

For Task F (Figure 6) the angular rate changes about the z-axis, otherwise known as the yaw angle. We were not anticipating to see a change in the z-component for velocity, as a change in angular velocity should have no effect on the velocity.

II. Problem 3

In the plots presented in the previous section, the red lines represent the non-linearized model while the blue represent the linearized model. Most all of the calculated plots make physical sense given the quadrotor equations of motion. Some trends that we expect to see are that a change in the angle rate results in a change in p, q, and r. A change in position causes a change in the velocity and often time in the acceleration as well.

Figure 1 depicts the effect of a +5°deviation in roll. For example, when the roll is deviated, the quadrotor should accelerate in the negative y direction and fall. However in our plots, the x position remains constant while the y position decreases and the z position increases. There is a change in position which shows a slight change in velocity and therefore no change in acceleration is depicted in our plots. This shows the roll did not have the affect we were assuming. This could be due to the slight deviation or an error in our model causing this particular simulation to have this discrepancy. For example, the z-position should be decreasing but it is increasing, this is due a to a sign error in the code.

Figure 2 shows the effect on the quadrotor given a pitch deviation of +5°. As pitch deviates, the quadrotor should accelerate in the positive x and z directions, and should fall. This is all expected, because the motor forces do not change during this time, so when the roll and pitch angles of the quadrotor change, these forces no longer directly oppose gravity and in turn cause the quadrotor to accelerate. However in our plots, no velocity is shown and there is an increase in the z direction.

Figure 3 simulates a +5°deviation in yaw. The aircraft stays level because when the yaw angle is deviated, the quadrotor is stable because the lift forces remain aligned with gravity, and the net vertical force on the quadrotor are still going to be zero.

In Figure 4, there is a +0.1 rad/sec deviation in roll rate. This causes the quadrotor to experience an increase in pitch angle and decrease in roll and yaw angles. There is also a change in position in the positive y and z direction and an increase in velocity of the quadrotor's body x-direction. This is inconsistent with theory.

Figure 5 shows the effects of a +0.1 rad/sec deviation in pitch rate. The quadrotor should experience acceleration in the negative x-direction. It does move in the a and z planes as expected, it also changes velocity in the v and q directions as expected.

Figure 6 shows the effects of a +0.1 red/sec deviation in yaw rate. The angles do not change which is to be expected but r changes which is also expected.

Overall, these plots do not correspond to theory as much as the group would like to see. It can possibly be traced back to when an error in the code starting by when position was calculated as some of initial positions are off. This would in turn affect the rest of the plots.

Steady hover is the only stable flight condition if it is undisturbed. Without feedback control, it does not return to this state after experiencing a disruption. This is shown in Figure 1-6 as they never return to the initial positions. When feedback is implemented, they do. (The plots, if correct, would back this theory up.) Quadrotor flight stability is achieved when the aircraft can return to trim after a disturbance occurs. When there are slight alterations as shown in the plots, this can lead to drastic changes in position, angle, velocities, and accelerations. This would not be a stable flight condition.

However, there is an exception with yaw angle and yaw rate. It is shown that in Figure 3 for yaw angle, everything remains constant over the time span. For Figure 6, everything but w and r remain constant over the time span, meaning it will remain in steady hover. This part of our simulation is consistent with theory.

III. Problem 4

The blue plots show the deviations developed from the linearized dynamics model. A change in the roll angle will affect lateral variables of motion, a change in pitch will affect the longitudinal variables, and a deviation in yaw will affect the spin variables. As all this is expected, our model does not exactly match with theory. This could be due to the small deviations or due to another source of error in our code.

When comparing the linear model to non linear model, it is important to compare the time frame. As the times both remain consistent, the graphs begin to deviate. This is due to the simplifying assumptions for a linear model which include the rotor gyroscopic effects as well as inertial cross-coupling are assumed to be negligible. These assumptions describe the discrepancy between the two models.

For example, looking at Figure 5, for a deviation in pitch rate, the positions change in all directions. And the difference in the model predictions is quite obvious. A change in pitch rate will affect the longitudinal variables, so in this case this would be q and theta. Both these variable are changing with respect to time. A similar analysis can be done on the other plots and the overall trend is that our plots are slightly inconsistent with theory. This can be traced back to a discrepancy within the non-linear model.

Figure 1, +5°deviation in roll: does not experience the same pitching as the non-linearized simulation and both the x and y positions decrease over time.

Figure 2, +5°deviation in pitch: all three components of position increase drastically where all but x did not with the non-linearized equations, there is also a change in the psi angle but this does not affect the aircraft's stability

Figure 3, a +5°deviation in yaw: once again, yaw deviation allows the quadrotor to remain in steady hover and in the linearized equations this translates to no change in any of the graphs

Figure 4, a +0.1 rad/sec deviation in roll rate: the x-component of position decreases while the y and z increase. The quadrotor's velocity in the u direction increases at the same rate as the non-linear model, and phi and psi are seen decreasing over time

Figure 5, a +0.1 rad/sec deviation in pitch rate: x position decreases while y and z increase, v increases at the same rate as the non-linearized simulation, pitching angle increases over time while yaw decreases

Figure 6, a +0.1 red/sec deviation in yaw rate: because a change yaw doesn't interfere with steady hover, the quadrotor remains stable but does experience an increase in the z-component of velocity w.

IV. Problem 5

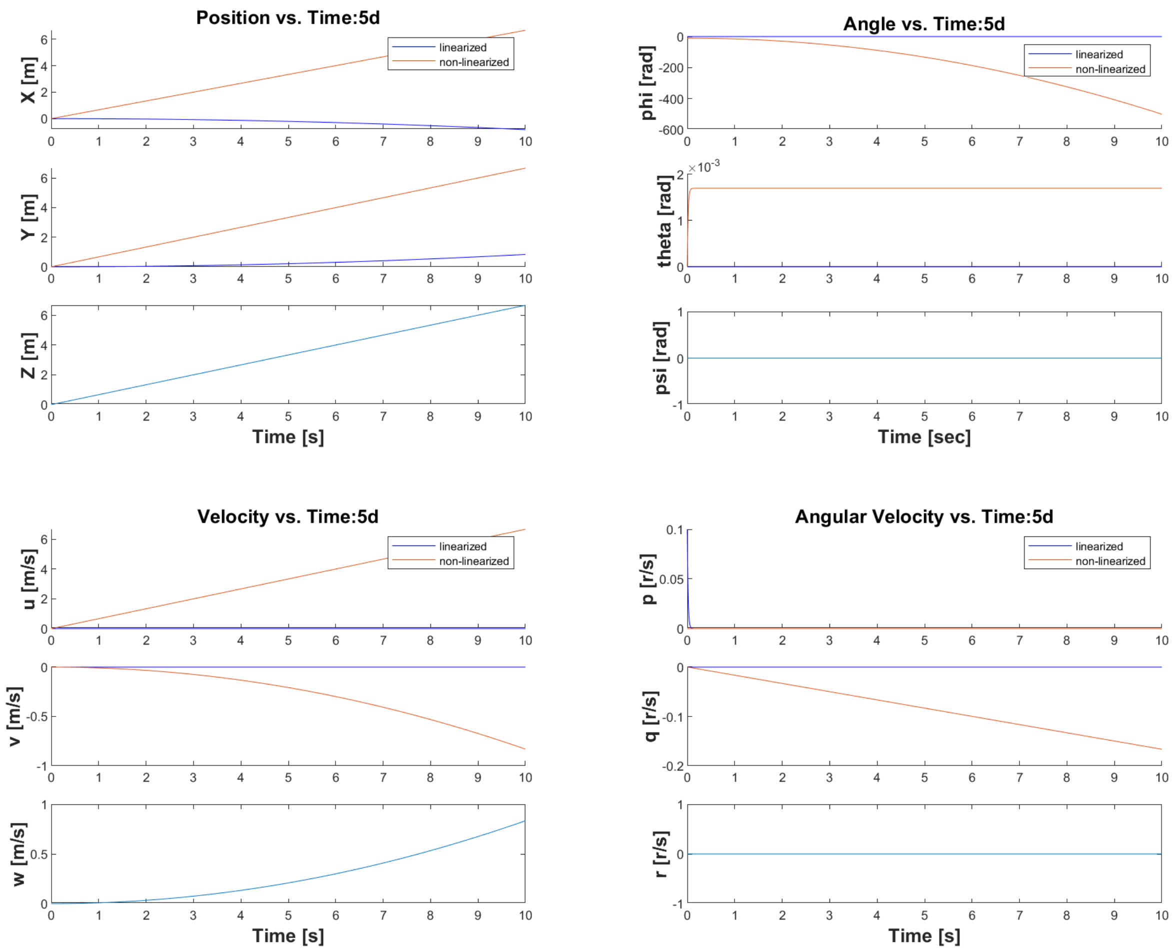
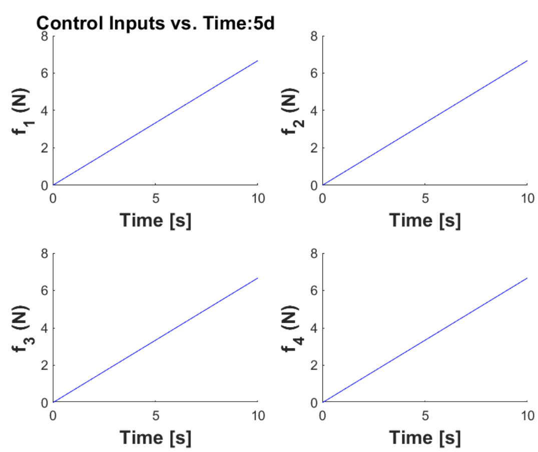


Fig. 7



The quadrotor has no inherent stability without the addition of a feedback control loop. Without feedback control, an initial deviation will result in diverging paths from steady hover flight. When there if feedback control, you should expect to see slight divergence then returning to the original steady hover flight.

Within task 5d, the control method was compared with the non-linearized method (in which the control is denoted in blue (contrary to the legend)). We observe in Figure 7 that the position increases from zero linearly across the timespan, furthermore we see a decrease in phi and a constant theta. Additionally, we observe an increasing velocity in the w and u, as well as a decreasing velocity in the v. Finally, we observe a decreasing q angular velocity. These plots do not correspond the theory.

We were anticipating to see that the angles, and velocity should have slowly converged to a steady level state (zero). We overall see much less drastic changes in position, attitude, as well as angular velocity.

Additionally, the control law is effected by having a proportional gain in the way of a tendency to converge towards a steady level state. This is possible due to the control law adjusting the "input forces" to try and simulate a level state. From task 5, we can observe the control inputs are increasing as time increases. We logically expect for input force to decrease to that of a sustained steady level system, as opposed to increasing more drastically. Although these are not the plots we were expecting, it can be traced back to an error in the linearized code which provided some plots that were not expected for the linearized portion as well.

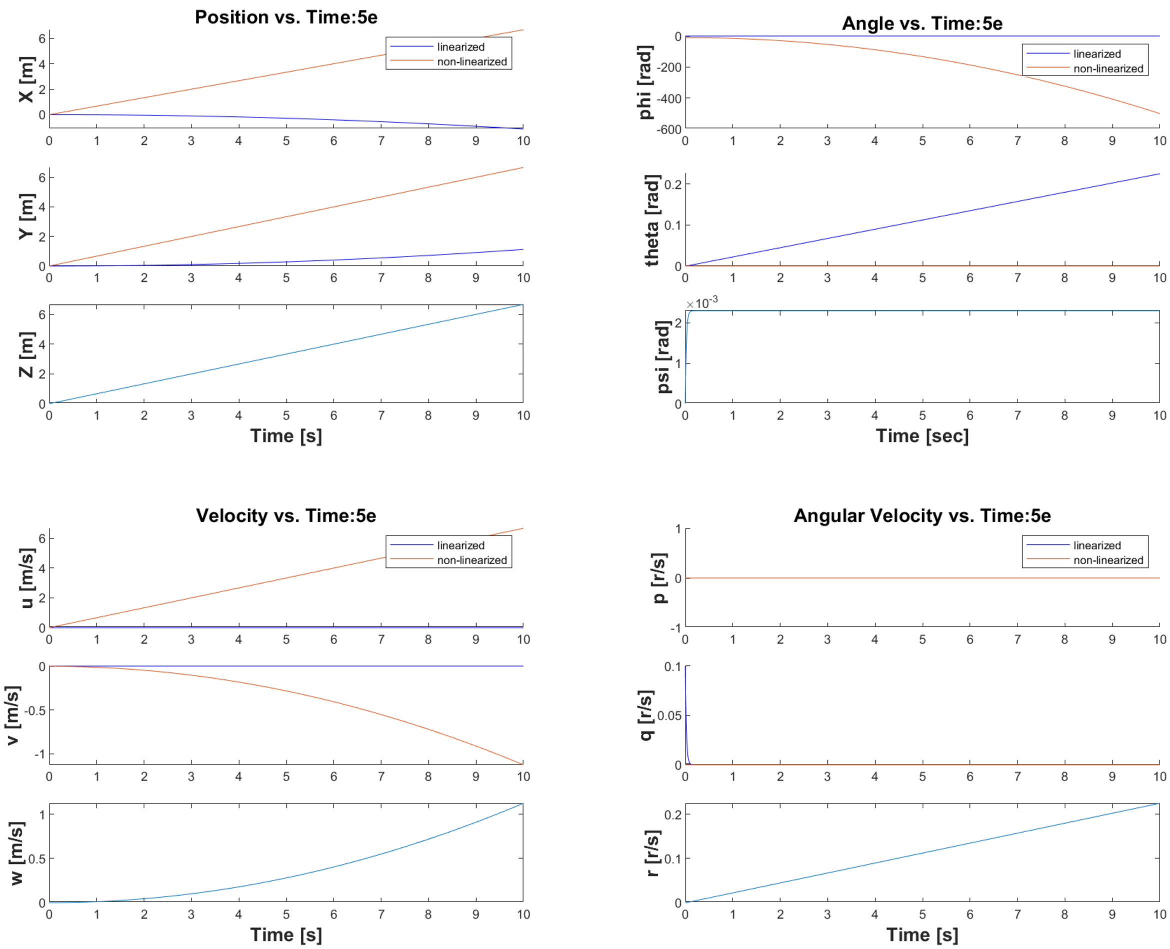
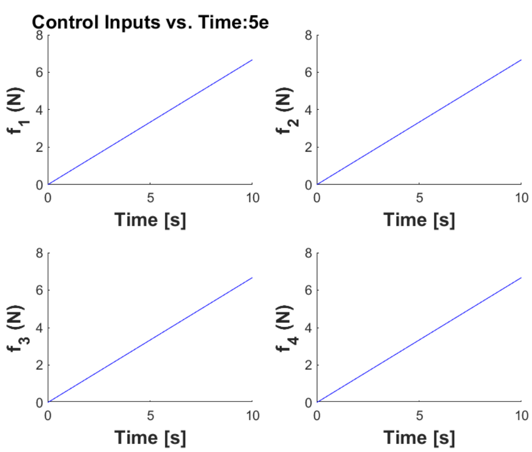
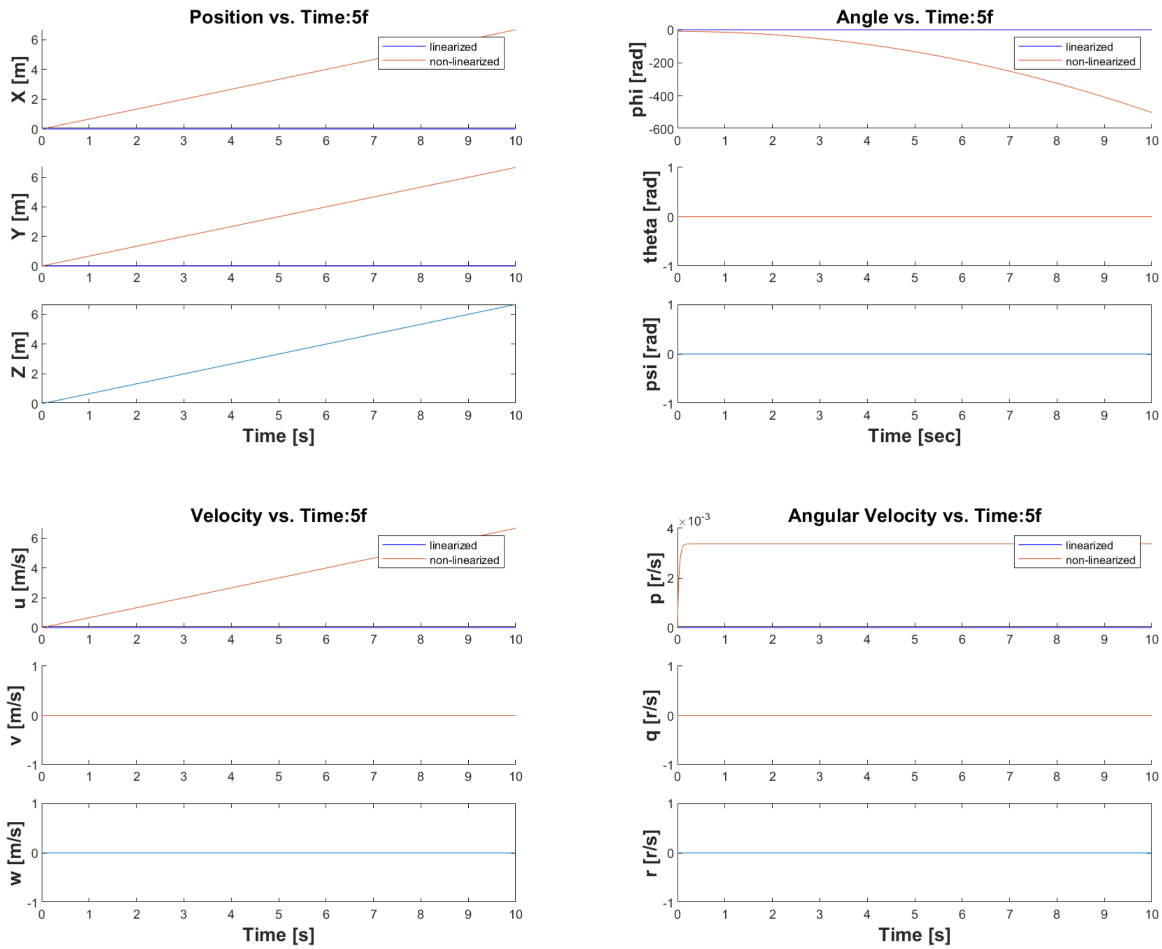

Position vs. Time:5e
X [m]
Y [m]
Z [m]
Time [s]
linearized
non-linearized
Angle vs. Time:5e
phi [rad]
theta [rad]
psi [rad]
Time [sec]
Velocity vs. Time:5e
u [m/s]
v [m/s]
w [m/s]
Time [s]
Angular Velocity vs. Time:5e
p [r/s]
q [r/s]
r [r/s]
Time [s]

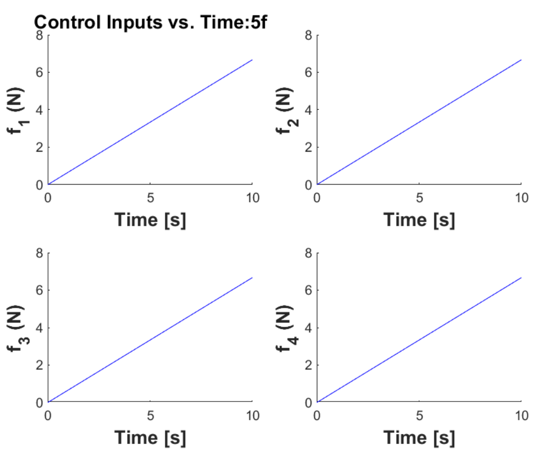
Fig. 8


Control Inputs vs. Time:5e
f1 (N)
f2 (N)
f3 (N)
f4 (N)
Time [s]



Position vs. Time:5f
X [m]
Y [m]
Z [m]
Time [s]
linearized
non-linearized
Angle vs. Time:5f
phi [rad]
theta [rad]
psi [rad]
Time [sec]
Velocity vs. Time:5f
u [m/s]
v [m/s]
w [m/s]
Angular Velocity vs. Time:5f
p [r/s]
q [r/s]
r [r/s]
×10-3

Fig. 9



Control Inputs vs. Time:5f
f1 (N)
f2 (N)
f3 (N)
f4 (N)
Time [s]

Participation Table

	Plan	Model	Experiment	Results	Report	Code	ACK
Gio Borrani	1	1	N/A	2	1	1	GB
Connor O'Reilly	1	1	N/A	1	1	2	CTO
Ryan Collins	1	2	N/A	2	2	1	RAC
Thyme Zuschlag	2	1	N/A	2	1	1	TZ

2 = Lead, 1 = Participate, 0 = Not involved, for each element.

Appendix A: MATLAB Code

```
%% 3218 Lab 3
% Authors: Connor O'Reilly, Gio Borrani, Ryan Collins, Thyme Zuschlag
% ASEN 3128
% L3_Master.m
% Created: 09/25/2020
% Last Edited: 09/25/2020

%% Purpose:
   %The purpose is to answer the following Questions:
      %Graph 6-plots denoted by the state vector and input controls
      %Develop Initial input forces
      %Test several simulation cases
      %etc...

%% Housekeeping
   clear all;
   clc;
   clf;
   close all;

%% Define Variables
   %givens
    m = .068; %kg
```

```
radius = .060; % meters
k =.0024; % Nm/N ?
g = 9.81;% m/s^2
 R = 0.060; %Radial distance from CG to propeller
 km = 0.0024; % moment coefficient
 Ix = 6.8E-5; %x-axis Moment of Inertia [kg*m^2]
 Iy = 9.2E-5; %Bodyy-axis Moment of Inertia [kg*m^2]
 Iz = 1.35E-4; %Body z-axis Moment of Inertia [kg*m^2]
 n = 1E-3; %Aerodynamic force coefficient [N/(m/s)^2]
 u = 2E-6; %mu, Aerodynamic moment coefficient [N*m/(rad/s)^2]

%state vector
xE = 0; % inertial x posn (Does not matter)
yE = 0; % inertial y posn (Does not matter)
zE = 0; % inertial z posn (Does not matter)
phi = 0; % roll angle (Must be zero)
theta = 0; % pitch angle (Must be zero)
psi = 0.5; % yaw angle (Does not matter)
uE = 0; % inertial vel x (Must be zero)
vE = 0; % inertial vel y (Must be zero)
wE = 0; % inertial vel z (Must be zero)
p = 0; % angular vel x (roll rate)
q = 0; % angular vel y (pitch rate)
r = 0; % angular vel z (yaw rate)

%control forces and moments
f1 = m*g/4;
f2 = m*g/4;
f3 = m*g/4;
f4 = m*g/4;

radius=0.060;%meters
Zc = -f1-f2-f3-f4;
Lc = (radius/sqrt(2))*(-f1-f2+f3+f4);
Mc = (radius/sqrt(2))*(f1-f2-f3+f4);
```

```
  Nc = k*((f1-f2+f3-f4));

  M_Aero=[0 0 0];

  tspan = [0 10];

%% Initialize State Vectors & Plot Prep

  aircraft_state_array = [xE,yE,zE,phi,theta,psi,uE,vE,wE,p,q,r]'; %start with
      values from last lab
  control_input_array = ComputeMotorForces(Zc, Lc, Mc, Nc, R, km); %Convert control
      moments into controlable forces.




  %% 3a & 4a
  aircraft_state_array=TestCase(1); %Pull state vector w/ 5 rad deflection from test
      database
  f = [f1 f2 f3 f4]; %Initialize initial force matrix
  [tx,x1] = ode45(@(t,x) quadCopODE_lin(t,x,f),tspan,aircraft_state_array); %Run ODE
      for 3a
  [tx2,x2] = ode45(@(t,x)
      objectEOM_nonlin(t,x,m,R,km,Ix,Iy,Iz,n,u,f1,f2,f3,f4),tspan,aircraft_state_array);
      %Run ODE for 4a
  x1=x1(1:length(x2) , :); % Match vector length for X1 & X2
  tx=tx(1:length(tx2) , :); %"
  tx=[tx, tx2]; % condition time vector for plots
  x1=[x1 , x2]; %"
  col=['b-','k-'];
  PlotAircraftSim_2(tx,x1,control_input_array,col, '3a-4a',(false)); %Plot
 %% 3b - 4b
  aircraft_state_array=TestCase(2);
  [tx,x1] = ode45(@(t,x) quadCopODE_lin(t,x,f),tspan,aircraft_state_array);
  [tx2,x2] = ode45(@(t,x)
```

```
    objectEOM_nonlin(t,x,m,R,km,Ix,Iy,Iz,n,u,f1,f2,f3,f4),tspan,aircraft_state_array);
  x1=x1(1:length(x2) , :);
  tx=tx(1:length(tx2) , :);
  tx=[tx, tx2];
  x1=[x1 , x2];
  col=['b-','k-'];
  PlotAircraftSim_2(tx,x1,control_input_array,'b-','3b-4b',(false));
%% 3c - 4c
  aircraft_state_array=TestCase(3);
 [tx,x1] = ode45(@(t,x) quadCopODE_lin(t,x,f),tspan,aircraft_state_array);
  [tx2,x2] = ode45(@(t,x)
    objectEOM_nonlin(t,x,m,R,km,Ix,Iy,Iz,n,u,f1,f2,f3,f4),tspan,aircraft_state_array);
  x1=x1(1:length(x2) , :);
  tx=tx(1:length(tx2) , :);
  tx=[tx, tx2];
  x1=[x1 , x2];
  col=['b-','k-'];
  PlotAircraftSim_2(tx,x1,control_input_array,'b-','3c-4c',(false));
%% 3d - 4d
  aircraft_state_array=TestCase(4);
  [tx,x1] = ode45(@(t,x) quadCopODE_lin(t,x,f),tspan,aircraft_state_array);
  [tx2,x2] = ode45(@(t,x)
    objectEOM_nonlin(t,x,m,R,km,Ix,Iy,Iz,n,u,f1,f2,f3,f4),tspan,aircraft_state_array);
  x1=x1(1:length(x2) , :);
  tx=tx(1:length(tx2) , :);
  tx=[tx, tx2];
  x1=[x1 , x2];
  col=['b-','k-'];
  PlotAircraftSim_2(tx,x1,control_input_array,col, '3d-4d',(false)); %Plot Function
%% 3e - 4e
  aircraft_state_array=TestCase(5);
  [tx,x1] = ode45(@(t,x) quadCopODE_lin(t,x,f),tspan,aircraft_state_array);
  [tx2,x2] = ode45(@(t,x)
    objectEOM_nonlin(t,x,m,R,km,Ix,Iy,Iz,n,u,f1,f2,f3,f4),tspan,aircraft_state_array);
  x1=x1(1:length(x2) , :);
```

```
  tx=tx(1:length(tx2) , :);
  tx=[tx, tx2];
  x1=[x1 , x2];
  col=['b-','k-'];
 PlotAircraftSim_2(tx,x1,control_input_array,col, '3e-4e',(false));
%% 3f - 4f
  aircraft_state_array=TestCase(6);
  [tx,x1] = ode45(@(t,x) quadCopODE_lin(t,x,f),tspan,aircraft_state_array);
  [tx2,x2] = ode45(@(t,x)
      objectEOM_nonlin(t,x,m,R,km,Ix,Iy,Iz,n,u,f1,f2,f3,f4),tspan,aircraft_state_array);
  x1=x1(1:length(x2) , :);
  tx=tx(1:length(tx2) , :);
  tx=[tx, tx2];
  x1=[x1 , x2];
  col=['b-','k-'];
  PlotAircraftSim_2(tx,x1,control_input_array,col, '3f-4f',(false));

   %% 5d
  aircraft_state_array=TestCase(4);
  aircraft_state_array=[aircraft_state_array , 0, 0, 0, 0];
  f = [f1 f2 f3 f4];
  [tx,x1] = ode45(@(t,x) quadCopODE_FBC(t,x, f),tspan,aircraft_state_array);
  [tx2,x2] = ode45(@(t,x) quadCopODE_FBC(t,x, f),tspan,aircraft_state_array);
  x1=x1(1:length(x2) , :);
  tx=tx(1:length(tx2) , :);
  cntrl = [x2(:,13),x2(:,14),x2(:,15),x2(:,16)];
  tx=[tx, tx2];
  x1=[x1 , x2];
  col=['b-','k-'];
  PlotAircraftSim_2(tx,x1,cntrl,col, '5d',(true));
%% 5e
aircraft_state_array=TestCase(5);
  aircraft_state_array=[aircraft_state_array , 0, 0, 0, 0];
  f = [f1 f2 f3 f4];
  [tx,x1] = ode45(@(t,x) quadCopODE_FBC(t,x, f),tspan,aircraft_state_array);
```

```
    [tx2,x2] = ode45(@(t,x) quadCopODE_FBC(t,x, f),tspan,aircraft_state_array);
    x1=x1(1:length(x2) , :);
    tx=tx(1:length(tx2) , :);
    cntrl = [x2(:,13),x2(:,14),x2(:,15),x2(:,16)];
    tx=[tx, tx2];
    x1=[x1 , x2];
    col=['b-','k-'];

    PlotAircraftSim_2(tx,x1,cntrl,col, '5e',(true));
 %% 5f
aircraft_state_array=TestCase(6);
    aircraft_state_array=[aircraft_state_array , 0, 0, 0, 0];
    f = [f1 f2 f3 f4];
    [tx,x1] = ode45(@(t,x) quadCopODE_FBC(t,x, f),tspan,aircraft_state_array);
    [tx2,x2] = ode45(@(t,x) quadCopODE_FBC(t,x, f),tspan,aircraft_state_array);
    x1=x1(1:length(x2) , :);
    tx=tx(1:length(tx2) , :);
    cntrl = [x2(:,13),x2(:,14),x2(:,15),x2(:,16)];
    tx=[tx, tx2];
    x1=[x1 , x2];
    col=['b-','k-'];

    PlotAircraftSim_2(tx,x1,cntrl,col, '5f',(true));
```

```
function [xstate] = objectEOM_nonlin(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
%
%Inputs:
% t = time
% x = 12-dimension state vector includes the inertial velocity in
%  inertial coordinates and the inertial position in inertial coordinates
%  [x; y; z; phi; theta; psi; u; v; w; p; q; r]
% m = mass of drone (kg)
% r = radius of frame from motor to cg
% km = control moment coefficient
% Ix,Iy,Iz = moments about axis
```

```
% v = drag coefficient
% mu = moment coefficient
% f1,f2,f3,f4 = forces from motors
%
%Outputs:
% xdot = 12-dimension state vector includes inertial velocity in inertial
%   coordinates and the inertial acceleration in inertial coordinates
%   [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
%   pdot; qdot; rdot]
%
%Methodology: Use Newton's second law F=ma to calculate the acceleration
%  and velocity at each point in time for ode45 to integrate to find
%  position. Drag, gravity and motor thrust are only forces acting on drone.
%

g = 9.81;

%Get IV's
x1 = x(1);
y1 = x(2);
z1 = x(3);
phi1 = x(4);
theta1 = x(5);
psi1 = x(6);
u1 = x(7);
v1 = x(8);
w1 = x(9);
p1 = x(10);
q1 = x(11);
r1 = x(12);
Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction

%Find Pdot(xdot ydot zdot) (earth fixed)
Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];

```

```
%Find Odot(thetadot phidot psidot)
Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];

%Get magnitude of velocity (B)
V_a = sqrt(u1^2 + v1^2 + w1^2);

%Calculate acceleration (body: Vb = [udot; vdot; wdot] components
%Vb = -w_b*V_b+f_b/m
Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1))+1/m*(-v*V_a*u1);
Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1))+1/m*(-v*V_a*v1);
Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(-v*V_a*w1)+1/m*(Zc);

%Calculate L, M, N
m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];

%Calculate m_ctl
m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);

%Calculate omega_dot(pdot,qdot,rdot)
omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);

%Put back into ode45
xstate(1) = Pdot(1); %xdot
xstate(2) = Pdot(2); %ydot
xstate(3) = Pdot(3); %zdot
xstate(4) = Odot(1); %phidot
xstate(5) = Odot(2); %thetadot
xstate(6) = Odot(3); %psidot
xstate(7) = Vb(1); %udot
xstate(8) = Vb(2); %vdot
xstate(9) = Vb(3); %wdot
```

```
xstate(10) = omega_dot(1); %pdot
xstate(11) = omega_dot(2); %qdot
xstate(12) = omega_dot(3); %rdot
xstate = xstate';
end
```

```
function
    PlotAircraftSim_2(time,aircraft_state_array,control_input_array,col,name,pass)
%% Prelude
% Written By: Ryan Collins
% Written On: 9/25
% Scriptname: PlotAircraftSim

%Inputs:
   %time: 1xn time vector
   %aircraft_state_array: 12xn array of aircraft states
      %aircraft_state_array(1:3)=position (x,y,z)
      %aircraft_state_array(4:6)=Angle (phi, theta, psi)
      %aircraft_state_array(7:9)=Velocity (u , v, w)
      %aircraft_state_array(10:12)=Angle_Rate (p,q,r)
   %control_input_array: 4xn array of control inputs
      %control_input_array=
   %col: string denoting the color to be used for each plot

%Outputs: none


%% Disect initial Parameters

pos1 = aircraft_state_array(:,1:3);
Vel1 = aircraft_state_array(:,4:6);
angle1 = aircraft_state_array(:,7:9);
omega1 = aircraft_state_array(:,10:12);

pos2 = aircraft_state_array(:,13:15);
```

```
Vel2 = aircraft_state_array(:,16:18);
angle2 = aircraft_state_array(:,19:21);
omega2 = aircraft_state_array(:,22:24);

CIA=control_input_array.';

%% State Output subplots - Fig 1-4 (3 subplots per)
   % Plot Position vs. Time
   figure ()

   subplot(3,1,1)
   hold on;
   plot(time(:,1),pos1(:,1),col(1))
   plot(time(:,2),pos2(:,1),col(2))
   legend('linearized','non-linearized')
   title(strcat('Position vs. Time: ',name), 'fontsize',13,'fontweight','bold')
   ylabel('X [m]','fontsize',13,'fontweight','bold')

   subplot(3,1,2)
   hold on;
   plot(time(:,1),pos1(:,2),col(1))
   plot(time(:,2),pos2(:,2),col(2))
   ylabel('Y [m]','fontsize',13,'fontweight','bold')

   subplot(3,1,3)
   plot(time(:,1),pos1(:,3),col(1))
   plot(time(:,2),pos2(:,3),col(2))
   ylabel('Z [m]','fontsize',13,'fontweight','bold')
   xlabel('Time [s]','fontsize',13,'fontweight','bold')
  % pause(1)
  saveas(gcf,strcat(name, '1','.png'))

% Plot Euler Angles vs. Time
   figure ()
     subplot(3,1,1)
```

```
    hold on;
    plot(time(:,1),angle1(:,1),col(1))
    plot(time(:,2),angle2(:,1),col(2))
    legend('linearized','non-linearized')
    title(strcat('Angle vs. Time: ',name), 'fontsize',13,'fontweight','bold')
    ylabel('phi [rad]','fontsize',13,'fontweight','bold')

    subplot(3,1,2)
    hold on;
    plot(time(:,1),angle1(:,2),col(1))
    plot(time(:,2),angle2(:,2),col(2))
    ylabel('theta [rad]','fontsize',13,'fontweight','bold')

    subplot(3,1,3)
    plot(time(:,1),angle1(:,3),col(1))
    plot(time(:,2),angle2(:,3),col(2))
    ylabel('psi [rad]','fontsize',13,'fontweight','bold')
    xlabel('Time [sec]','fontsize',13,'fontweight','bold')
 %  pause(1)
    saveas(gcf,strcat(name, '2','.png'))

% Plot Velocity vs. Time
    figure ()
     subplot(3,1,1)
    hold on;
    plot(time(:,1),Vel1(:,1),col(1))
    plot(time(:,2),Vel2(:,1),col(2))
    legend('linearized','non-linearized')
    title(strcat('Velocity vs. Time: ',name), 'fontsize',13,'fontweight','bold')
    ylabel('u [m/s]','fontsize',13,'fontweight','bold')

    subplot(3,1,2)
    hold on;
    plot(time(:,1),Vel1(:,2),col(1))
    plot(time(:,2),Vel2(:,2),col(2))
```

```
    ylabel('v [m/s]','fontsize',13,'fontweight','bold')

    subplot(3,1,3)
    plot(time(:,1),Vel1(:,3),col(1))
    plot(time(:,2),Vel2(:,3),col(2))
    ylabel('w [m/s]','fontsize',13,'fontweight','bold')
    xlabel('Time [s]','fontsize',13,'fontweight','bold')
    %pause(1)
    saveas(gcf,strcat(name, '3','.png'))

% Plot Angular Velocity vs. Time
    figure ()
    subplot(3,1,1)
    hold on;
    plot(time(:,1),omega1(:,1),col(1))
    plot(time(:,2),omega2(:,1),col(2))
    legend('linearized','non-linearized')
    title(strcat('Angular Velocity vs. Time: ',name),
        'fontsize',13,'fontweight','bold')
    ylabel('p [r/s]','fontsize',13,'fontweight','bold')

    subplot(3,1,2)
    hold on;
    plot(time(:,1),omega1(:,2),col(1))
    plot(time(:,2),omega2(:,2),col(2))
    ylabel('q [r/s]','fontsize',13,'fontweight','bold')

    subplot(3,1,3)
    plot(time(:,1),omega1(:,3),col(1))
    plot(time(:,2),omega2(:,3),col(2))
    ylabel('r [r/s]','fontsize',13,'fontweight','bold')
    xlabel('Time [s]','fontsize',13,'fontweight','bold')
    %pause(1)
    saveas(gcf,strcat(name, '4','.png'))
```

```
%% 3D Plot - Fig 5

figure()
   hold on;
   plot3(pos1(:,1),pos1(:,2),pos1(:,3))
   plot3(pos2(:,1),pos2(:,2),pos2(:,3))
%    plot3(pos1(1,end),pos1(2,end),pos(3,end),'r.')%Plot last point in red
%    plot3(pos1(1,1),pos1(1,2),pos(1,3),'g.')%Plot first point in green
%    plot3(pos2(end,1),pos2(end,2),pos(end,3),'r.')%Plot last point in red
%    plot3(pos2(1,1),pos2(1,2),pos(1,3),'g.')%Plot first point in green
   %hold on;
   title(strcat('Flight Profile: ',name), 'fontsize',13,'fontweight','bold')
   legend('linearized','non-linearized')
   ylabel('y position - (m)','fontsize',13,'fontweight','bold')
   xlabel('x position - (m)','fontsize',13,'fontweight','bold')
   zlabel('z position - (m)','fontsize',13,'fontweight','bold')
   %pause(1)
   saveas(gcf,strcat(name, '5','.png'))


   %% Control Input Variables - Fig 6
%
if (pass==true)
   figure ()
   hold on;

   subplot(2,2,1)
   hold on;
   title(strcat('Control Inputs vs. Time:',name), 'fontsize',13,'fontweight','bold')
   ylabel('f_1 (N)','fontsize',13,'fontweight','bold')
   xlabel('Time [s]','fontsize',13,'fontweight','bold')
   plot(time(:,2),CIA(1,:),col(1))

   subplot(2,2,2)
```

```
    hold on;
    ylabel('f_2 (N)','fontsize',13,'fontweight','bold')
    xlabel('Time [s]','fontsize',13,'fontweight','bold')
    plot(time(:,2),CIA(2,:),col(1))

    subplot(2,2,3)
    hold on;
    ylabel('f_3 (N)','fontsize',13,'fontweight','bold')
    xlabel('Time [s]','fontsize',13,'fontweight','bold')
    plot(time(:,2),CIA(3,:),col(1))

    subplot(2,2,4)
    hold on;
    ylabel('f_4 (N)','fontsize',13,'fontweight','bold')
    xlabel('Time [s]','fontsize',13,'fontweight','bold')
   plot(time(:,2),CIA(4,:),col(1))
   saveas(gcf,strcat(name, '6','.png'))
end

end
```

```
function [xstate] = quadCopODE_lin(t,x,f)
%
%Inputs:
% t = time
% x = 12x1 state vector includes the inertial velocity in
%  inertial coordinates and the inertial position in inertial coordinates
%  [x; y; z; phi; theta; psi; u; v; w; p; q; r]
% f = 1x4 force vector from motors
%
%Outputs:
% xstate = 12x1 state vector includes inertial velocity in inertial
%   coordinates and the inertial acceleration in inertial coordinates
%   [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
%   pdot; qdot; rdot]
```

```
%
%Methodology: Use Newton's second law F=ma to calculate the acceleration
%  and velocity at each point in time for ode45 to integrate to find
%  position. Drag, gravity and motor thrust are only forces acting on drone.
%

%% Given Constants
m = 0.068; %kg
R = 0.060; %m

k_m = 0.0024; %N*m/(N)
I_x = 0.000068; %kg*m^2
I_y = 0.000092; %kg*m^2
I_z = 0.000135; %kg*m^2

g = 9.81;

%% Pulling from init conditions
position_inert = x(1:3); %
euler_angles = x(4:6);
body_vel = x(7:9);
body_omega = x(10:12);


%% Transform
%body to inertial coordinates (xdot,ydot,zdot)
pos_dot =
    R_eb(euler_angles(1),euler_angles(2),euler_angles(3),'rad')*[body_vel(1);body_vel(2);body_vel(3)]


%% Control Forces and Moments
%motor_forces = .25*g*m;
L_c = R/sqrt(2) * ( -f(1) - f(2) + f(3) + f(4) );
M_c = R/sqrt(2) * ( f(1) - f(2) - f(3) + f(4) );
N_c = k_m * ( f(1) - f(2) + f(3) - f(4) );
```

```
Z_c = -f(1)-f(2)-f(3)-f(4);

%[fa,fb,fc,fd] = ComputeMotorForces(Z_c,L_c,M_c,N_c,R,k_m);

%% Roll Pitch Yaw rates
O_dot(1) = L_c/I_x;
O_dot(2) = M_c/I_y;
O_dot(3) = N_c/I_z;

%% u v w rates
Vb(1) = g * -euler_angles(1);
Vb(2) = g * euler_angles(2);
Vb(3) = Z_c/m;

%% State derivative
xstate(1) = pos_dot(1); %xdot
xstate(2) = pos_dot(2); %ydot
xstate(3) = pos_dot(3); %zdot
xstate(4) = body_omega(1); %phidot
xstate(5) = body_omega(2); %thetadot
xstate(6) = body_omega(3); %psidot
xstate(7) = Vb(1); %udot
xstate(8) = Vb(2); %vdot
xstate(9) = Vb(3); %wdot
xstate(10) = O_dot(1); %pdot
xstate(11) = O_dot(2); %qdot
xstate(12) = O_dot(3); %rdot
xstate = xstate';
end
```

```
function [xstate] = quadCopODE_FBC(t,x,f)
%
%Inputs:
% t = time
% x = 12x1 state vector includes the inertial velocity in
```

```
%  inertial coordinates and the inertial position in inertial coordinates
%  [x; y; z; phi; theta; psi; u; v; w; p; q; r]
% f = 1x4 force vector from motors
%
%Outputs:
% xstate = 12x1 state vector includes inertial velocity in inertial
%   coordinates and the inertial acceleration in inertial coordinates
%   [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
%   pdot; qdot; rdot]
%
%Methodology: Use Newton's second law F=ma to calculate the acceleration
%  and velocity at each point in time for ode45 to integrate to find
%  position. Drag, gravity and motor thrust are only forces acting on drone.
%

%% Given Constants
m = 0.068; %kg
R = 0.060; %m
k1= .004; %Nmsec/rad

k_m = 0.0024; %N*m/(N)
I_x = 0.000068; %kg*m^2
I_y = 0.000092; %kg*m^2
I_z = 0.000135; %kg*m^2

g = 9.81;

%% Pulling from init conditions
position_inert = x(1:3); %
euler_angles = x(4:6);
body_vel = x(7:9);
body_omega = x(10:12);

%% Transform
%body to inertial coordinates (xdot,ydot,zdot)
```

```
pos_dot =
    R_eb(euler_angles(1),euler_angles(2),euler_angles(3),'rad')*[body_vel(1);body_vel(2);body_vel(3)]


%% Control Forces and Moments
%motor_forces = .25*g*m;
L_c = -k1*body_omega(1);
M_c = -k1*body_omega(2);
N_c = -k1*body_omega(3);
Z_c = -f(1)-f(2)-f(3)-f(4);


F = ComputeMotorForces(Z_c,L_c,M_c,N_c,R,k_m);


%% Roll Pitch Yaw rates
O_dot(1) = L_c/I_x;
O_dot(2) = M_c/I_y;
O_dot(3) = N_c/I_z;

%% u v w rates
Vb(1) = g * -euler_angles(1);
Vb(2) = g * euler_angles(2);
Vb(3) = Z_c/m;

%% State derivative
xstate(1) = pos_dot(1); %xdot
xstate(2) = pos_dot(2); %ydot
xstate(3) = pos_dot(3); %zdot
xstate(4) = body_omega(1); %phidot
xstate(5) = body_omega(2); %thetadot
xstate(6) = body_omega(3); %psidot
xstate(7) = Vb(1); %udot
xstate(8) = Vb(2); %vdot
xstate(9) = Vb(3); %wdot
```

```
xstate(10) = O_dot(1); %pdot
xstate(11) = O_dot(2); %qdot
xstate(12) = O_dot(3); %rdot

xstate(13) = F(1);
xstate(14) = F(2);
xstate(15) = F(3);
xstate(16) = F(4);

xstate = xstate';

end
```

```
function [xstate] = objectEOM_nonlin(t,x,m,r,km,Ix,Iy,Iz,v,mu,f1,f2,f3,f4)
%
%Inputs:
% t = time
% x = 12-dimension state vector includes the inertial velocity in
%  inertial coordinates and the inertial position in inertial coordinates
%  [x; y; z; phi; theta; psi; u; v; w; p; q; r]
% m = mass of drone (kg)
% r = radius of frame from motor to cg
% km = control moment coefficient
% Ix,Iy,Iz = moments about axis
% v = drag coefficient
% mu = moment coefficient
% f1,f2,f3,f4 = forces from motors
%
%Outputs:
% xdot = 12-dimension state vector includes inertial velocity in inertial
%   coordinates and the inertial acceleration in inertial coordinates
%   [xdot; ydot; zdot; phidot; thetadot; psidot; udot; vdot; wdot;
%   pdot; qdot; rdot]
%
%Methodology: Use Newton's second law F=ma to calculate the acceleration
```

```
%  and velocity at each point in time for ode45 to integrate to find
%  position. Drag, gravity and motor thrust are only forces acting on drone.
%

g = 9.81;

%Get IV's
x1 = x(1);
y1 = x(2);
z1 = x(3);
phi1 = x(4);
theta1 = x(5);
psi1 = x(6);
u1 = x(7);
v1 = x(8);
w1 = x(9);
p1 = x(10);
q1 = x(11);
r1 = x(12);
Zc = -f1 - f2 - f3 - f4; %sum of 4 motor thrusts in -z direction

%Find Pdot(xdot ydot zdot) (earth fixed)
Pdot = R_eb(phi1,theta1,psi1,'rad')*[u1;v1;w1];

%Find Odot(thetadot phidot psidot)
Odot = T(phi1,theta1,psi1,'rad')*[p1;q1;r1];

%Get magnitude of velocity (B)
V_a = sqrt(u1^2 + v1^2 + w1^2);

%Calculate acceleration (body: Vb = [udot; vdot; wdot] components
%Vb = -w_b*V_b+f_b/m
Vb(1) = (r1*v1-q1*w1)+g*(-sin(theta1))+1/m*(-v*V_a*u1);
Vb(2) = (p1*w1-r1*u1)+g*(cos(theta1)*sin(phi1))+1/m*(-v*V_a*v1);
Vb(3) = (q1*u1-p1*v1)+g*(cos(theta1)*cos(phi1))+1/m*(-v*V_a*w1)+1/m*(Zc);
```

```
%Calculate L, M, N
m_a = -mu*sqrt(p1^2+q1^2+r1^2)*[p1; q1; r1];

%Calculate m_ctl
m_ctl(1) = (r/sqrt(2))*(-f1-f2+f3+f4);
m_ctl(2) = (r/sqrt(2))*(f1-f2-f3+f4);
m_ctl(3) = (r/sqrt(2))*(f1-f2+f3-f4);

%Calculate omega_dot(pdot,qdot,rdot)
omega_dot(1) = (Iy-Iz)/(Ix)*q1*r1 + 1/Ix*m_a(1) + 1/Ix*m_ctl(1);
omega_dot(2) = (Iz-Ix)/(Iy)*p1*r1 + 1/Iy*m_a(2) + 1/Iy*m_ctl(2);
omega_dot(3) = (Ix-Iy)/(Iz)*p1*q1 + 1/Iz*m_a(3) + 1/Iz*m_ctl(3);

%Put back into ode45
xstate(1) = Pdot(1); %xdot
xstate(2) = Pdot(2); %ydot
xstate(3) = Pdot(3); %zdot
xstate(4) = Odot(1); %phidot
xstate(5) = Odot(2); %thetadot
xstate(6) = Odot(3); %psidot
xstate(7) = Vb(1); %udot
xstate(8) = Vb(2); %vdot
xstate(9) = Vb(3); %wdot
xstate(10) = omega_dot(1); %pdot
xstate(11) = omega_dot(2); %qdot
xstate(12) = omega_dot(3); %rdot
xstate = xstate';
end
```

```
function [REB] = R_eb(phi,theta,psi,units)
%switch if input is either rad or deg
switch units
   case 'deg'
      REB(1,1) = cosd(theta)*cosd(psi);
```

```
        REB(1,2) = cosd(theta)*sind(psi);
        REB(1,3) = -sind(theta);
        REB(2,1) = sind(phi)*sind(theta)*cosd(psi)-cosd(phi)*sind(psi);
        REB(2,2) = sind(phi)*sind(theta)*sind(psi)+cosd(phi)*cosd(psi);
        REB(2,3) = sind(phi)*cosd(theta);
        REB(3,1) = cosd(phi)*sind(theta)*cosd(psi)+sind(phi)*sind(psi);
        REB(3,2) = cosd(phi)*sind(theta)*sind(psi)-sind(phi)*cosd(psi);
        REB(3,3) = cosd(phi)*cosd(theta);
    case 'rad'
        phi = phi * (180/pi);
        theta = theta * (180/pi);
        psi = psi * (180/pi);
        REB(1,1) = cosd(theta)*cosd(psi);
        REB(1,2) = cosd(theta)*sind(psi);
        REB(1,3) = -sind(theta);
        REB(2,1) = sind(phi)*sind(theta)*cosd(psi)-cosd(phi)*sind(psi);
        REB(2,2) = sind(phi)*sind(theta)*sind(psi)+cosd(phi)*cosd(psi);
        REB(2,3) = sind(phi)*cosd(theta);
        REB(3,1) = cosd(phi)*sind(theta)*cosd(psi)+sind(phi)*sind(psi);
        REB(3,2) = cosd(phi)*sind(theta)*sind(psi)-sind(phi)*cosd(psi);
        REB(3,3) = cosd(phi)*cosd(theta);
end
REB = inv(REB);

end
```

```
function [Tmat] = T(phi,theta,psi,units)
%T Summary of this function goes here
%  Detailed explanation goes here
switch units
    case 'deg'
        Tmat(1,1) = 1;
        Tmat(1,2) = sind(phi)*tand(theta);
        Tmat(1,3) = cosd(phi)*tand(theta);
        Tmat(2,1) = 0;
```

```
        Tmat(2,2) = cosd(phi);
        Tmat(2,3) = -sind(phi);
        Tmat(3,1) = 0;
        Tmat(3,2) = sind(phi)*secd(theta);
        Tmat(3,3) = cosd(phi)*secd(theta);

    case 'rad'
        phi = phi * (180/pi);
        theta = theta * (180/pi);
        psi = psi * (180/pi);
        Tmat(1,1) = 1;
        Tmat(1,2) = sind(phi)*tand(theta);
        Tmat(1,3) = cosd(phi)*tand(theta);
        Tmat(2,1) = 0;
        Tmat(2,2) = cosd(phi);
        Tmat(2,3) = -sind(phi);
        Tmat(3,1) = 0;
        Tmat(3,2) = sind(phi)*secd(theta);
        Tmat(3,3) = cosd(phi)*secd(theta);
end
```

```
function [F] = ComputeMotorForces(Zc, Lc, Mc, Nc, R, km)
%{
    ASEN 3128 Lab 3
    Authors: Connor O'Reilly, Gio Borrani
             Ryan Collins, Thyme Zuschlag
    Inputs:
        Lc, Mc, Nc: Control Moments
        Zc: Control Force
        R: Radial distance from CG to propeller [m]
        km: Control moment coefficient
    Outputs:
        F: vector containing 4 motor thrusts

%}
```

```matlab

giv = [Zc Lc Mc Nc].';
r = (R/sqrt(2));
mat = [-1 -1 -1 -1;...
      -r -r r r;...
       r -r -r r;...
       km -km km -km];
mat = mat.';
F = mat*giv;
end
```

```matlab
function [vec] = TestCase(numb)
   %Ryan Collins
   %Takes in a number and gives back test case state vectors between 1-7
if (numb==1)
   vec=[0 0 -10 5/180*pi 0 0 0 0 0 0 0 0];
elseif (numb==2)
   vec=[0 0 -10 0 5/180*pi 0 0 0 0 0 0 0];
elseif (numb==3)
   vec=[0 0 -10 0 0 5/180*pi 0 0 0 0 0 0 ];
elseif (numb==4)
   vec=[0 0 -10 0 0 0 0 0 0 0.1 0 0];
elseif (numb==5)
   vec=[0 0 -10 0 0 0 0 0 0 0 0.1 0];
elseif (numb==6)
   vec=[0 0 -10 0 0 0 0 0 0 0 0 0.1];
elseif (numb==7)
   vec=[0 0 0 5 5 5 0 0 0 0.1 0.1 0.1];
end
end
```

Appendix B: Derivations