University of Colorado Boulder

ASEN 3801: Aerospace Vehicle Dynamics and Controls Lab

September 29, 2023

ASEN 3801 Lab 4: Quadrotor Simulation and Control

Group 28

Abbitt Holland[1]

Austin Hunter[2]

Drew Kane[3]

Rishab Pally[4]

Professor: Eric Frew

[1]SID: 108756154
[2]SID: 110266023
[3]SID:
[4]SID: 109519936

Ann and H.J. Smead Aerospace Engineering Sciences
UNIVERSITY OF COLORADO BOULDER

Introduction

Lab 4, Quadrotor Simulation and Control, focused on the fundamental relationship between characteristics and flight dynamics in both feedback-controlled and uncontrolled aircraft cases. By computationally integrating the aircraft's derivative state vector, we can better understand the equations of motion in the simulation of a quadrotor. This includes both trim states as well as with initial perturbations/values. Therefore with and without aerodynamic forces and moments using Matlab's ODE45 function. There are multiple plots/figures showing a state variable against time as well as forces and moments. In Lab Task 2, different variations in the aircraft state array were used to simulate both the linear and nonlinear equations of motion over time. To highlight the differences between the two methods linear and nonlinear procedures have been plotted and examined. To simulate the equations of motion with rate feedback over time, the force and moment control vectors were also computed in addition to the motor forces. In Lab Task 3, a feedback circuit with control gains was analyzed in order to stabilize a quadcopter during its predetermined flight path. In order to approximate initial condition variations, the feedback control laws were then applied in a linearized and nonlinear closed-loop system.

Task 1

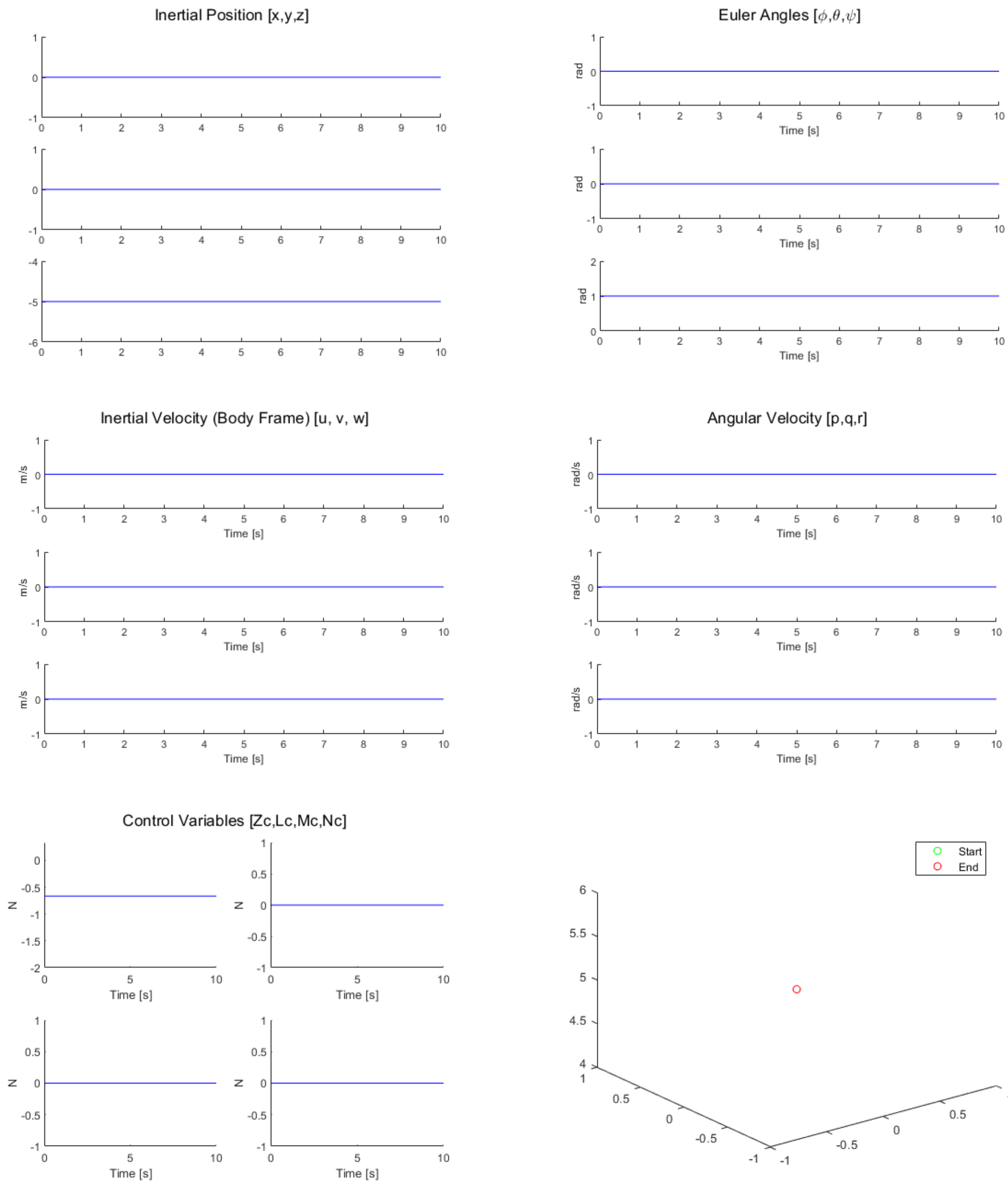
1.1

See Appendix for PlotAircraftSim.m

1.5

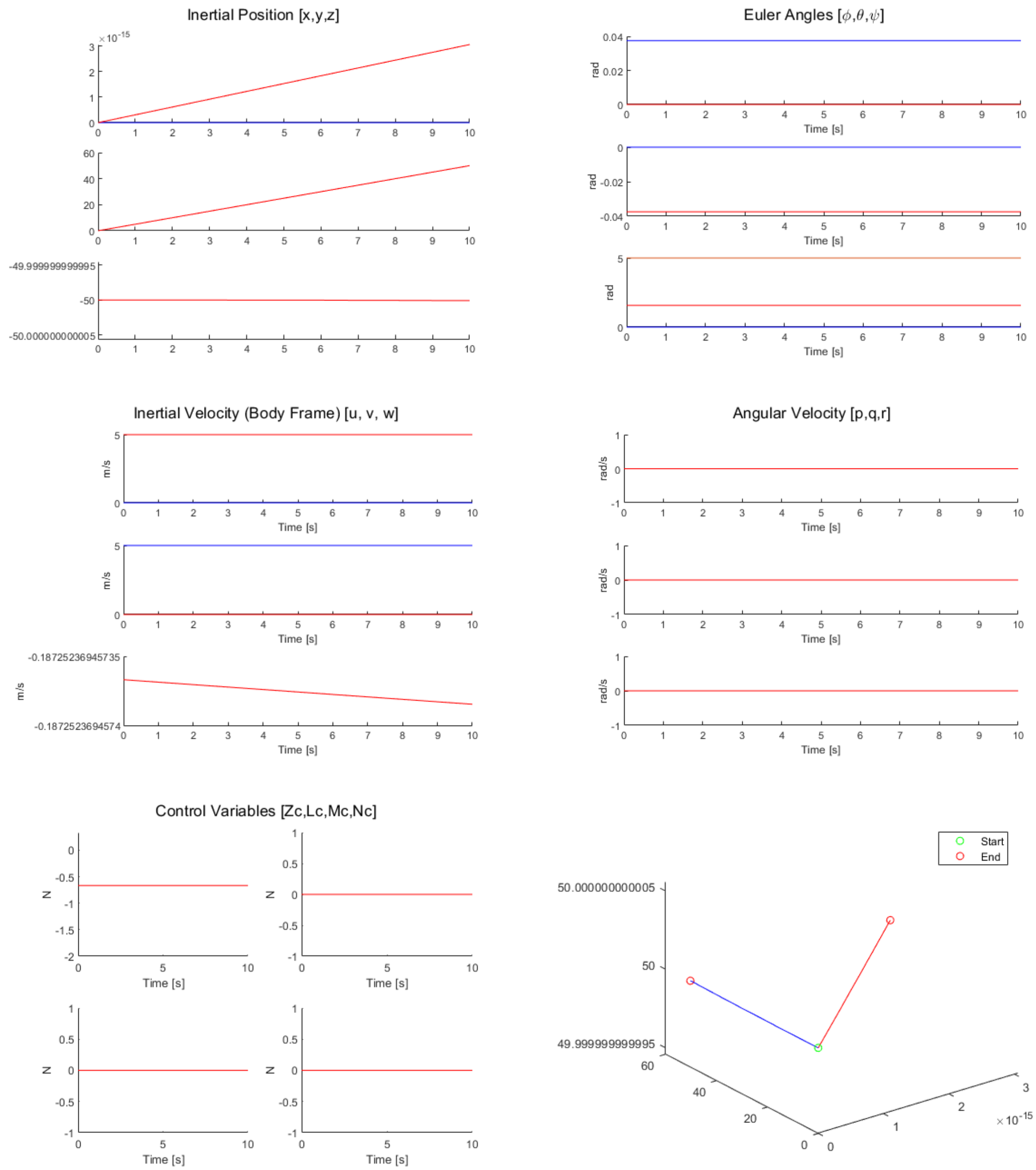
After analyzing using Matlab and plotting the simulation we can determine that the drone should be able to maintain a steady hover flight over without external forces/disturbances. The inertial position, velocity, Euler angles and rotational angular velocity all remain constant with a value of 0(approximately 0). This response is shown the figures above. Moreover, in stable, hovering flight, the body moments and Z force both output a value of 0. Using the Matlab prediction, the quadcopter demonstration is not stable during its initial steady hovering flight, as shown by the plots, video, and data. This results in differences between simulation and the experiment. The Matlab simulation makes the assumption that there are no outside disturbances, such as wind pressure variances and that the flight conditions are ideal. Due to these variables plus the lack of a feedback control law in the quadcopter used for this experiment, little perturbations cause unstable flying, which worsens until the aircraft loses all control over its flight characteristics and results in a crash. The respective flight plots/figures that are shown with the test video further demonstrate this pattern. All plots should maintain a value of 0 upon initialization at t = 0 if there are no external variables present during the test flight. Nevertheless, all plots visibly represent disturbance cases in position, velocity, Euler angles and angular velocity. Due to the coupling of these variables with other flight variables, perturbations increase, resulting in a decrease in the inertial y value and a rapid increase in the x. Additionally, u, w, p and q increase with θ and ψ whereas v decreases. The oscillations represents the loss of control the aircraft faces as it drops.

1.2-3



Inertial Position [x,y,z]
Euler Angles [ϕ,θ,ψ]
rad
Time [s]
Inertial Velocity (Body Frame) [u, v, w]
m/s
Angular Velocity [p,q,r]
rad/s
Control Variables [Zc,Lc,Mc,Nc]
N
Start
End

1.4

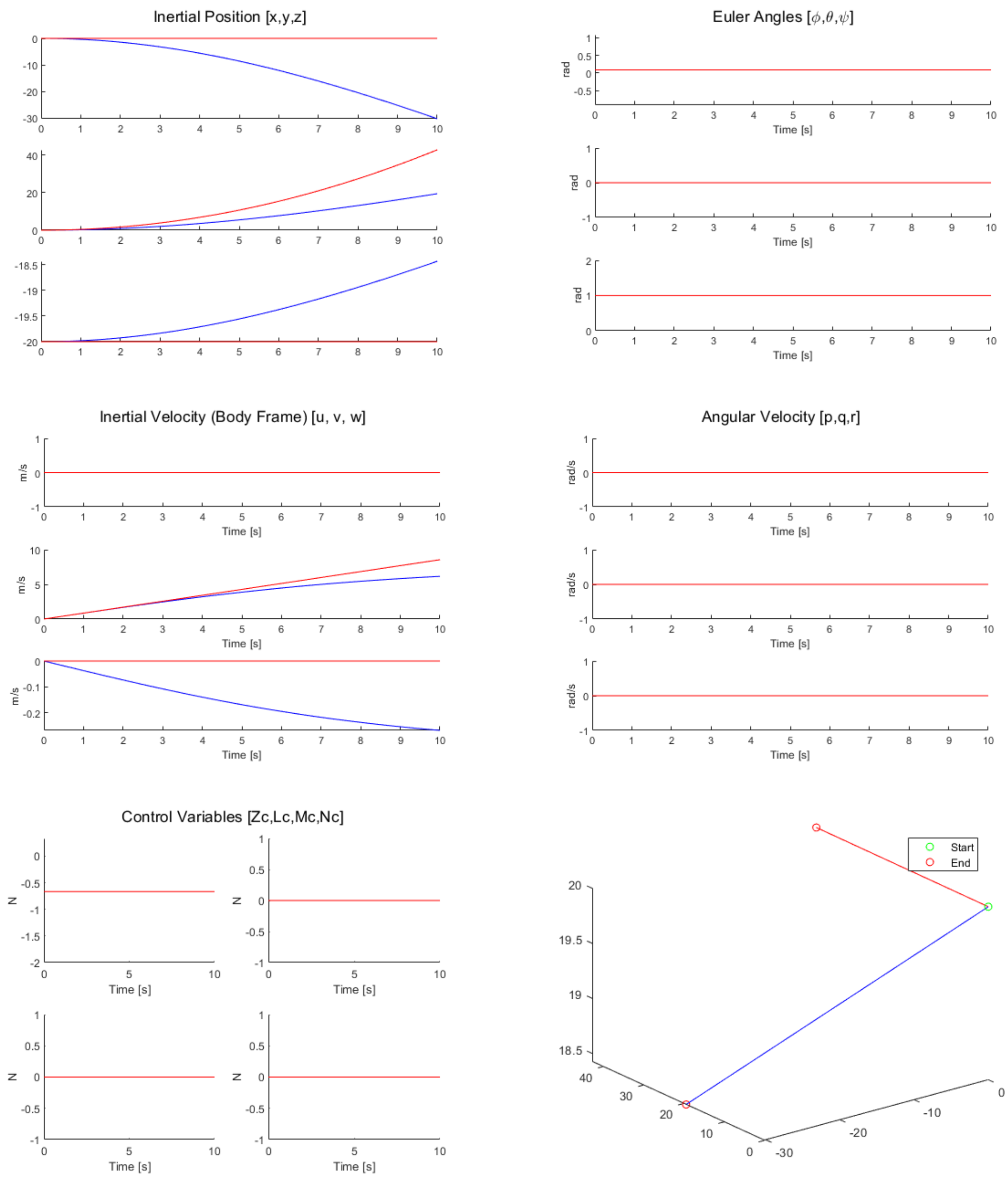


Task 2

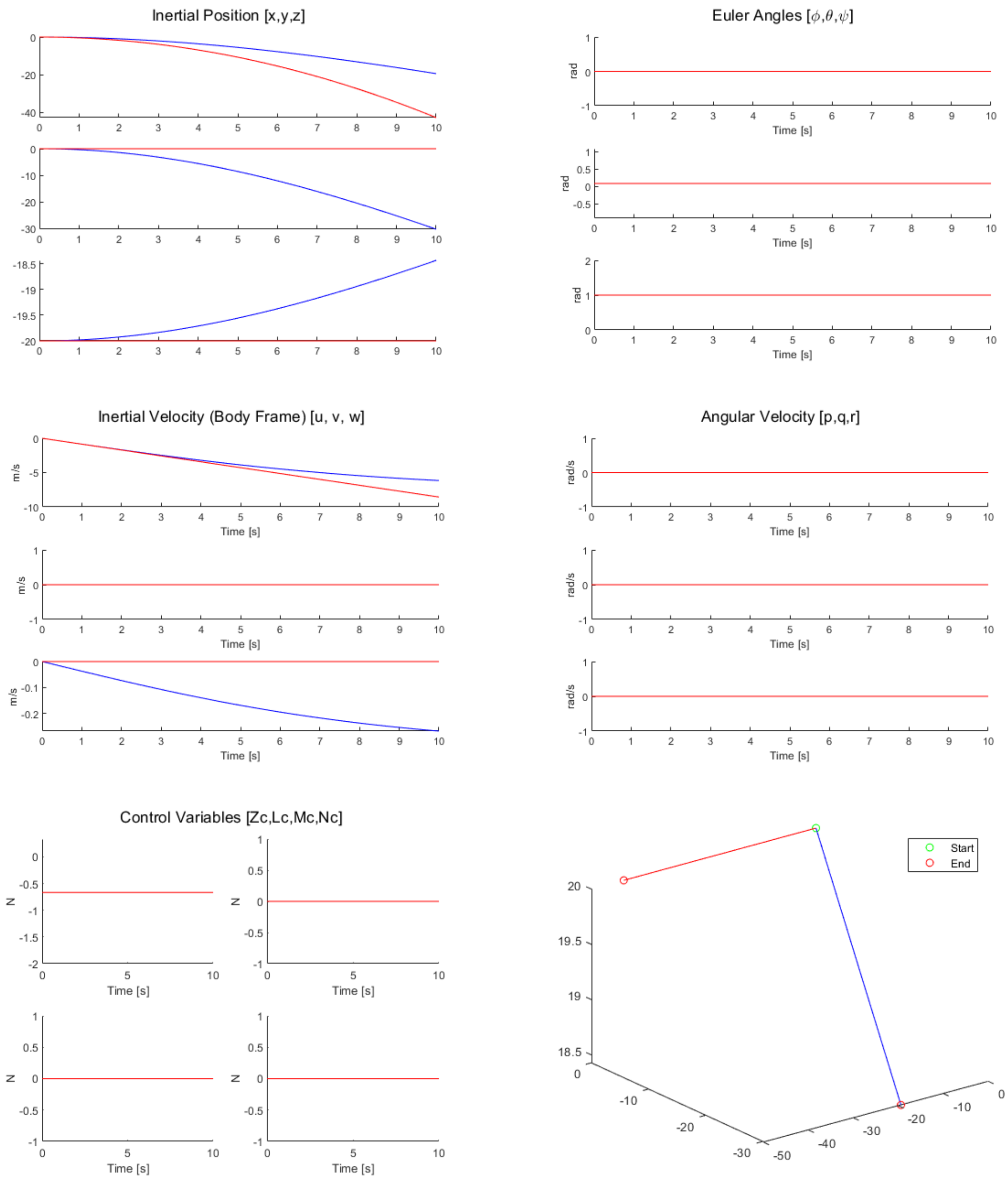
The nonlinearized model is represented by the blue lines in the plots and the linearized case is represented by the red lines. In Figure 2.2, the difference between the two models can be seen with a positive z direction pointing down on the plots. This is due to the assumption that roll has no impact on vertical force, the linearized model shows the movement as having no change in the aircraft's height. The nonlinear model contradicts the same assumption by demonstrating that a roll disturbance causes the quadcoptor to lose some of its force in the vertical direction, which causes it to start descending. It does not make sense for a quadrotor in a state of roll to maintain its current altitude unless the rotors altered speed to account for the rolling variations in upward force. Therefore the non-linearized version in this case displays an error that was resulted due to linearization. The errors stated in part 1.5 are mathematically proven by this result as a linearized model assumes a small angle, it would seem to be able to maintain constant hover and eliminate external effects. Yet the small values over time have a much larger affect.

Blue: Nonlinear model **Task 2.1**
Red: Linearized model **Task 2.2**
Green: EOM Model with rate feedback **Task 2.5**

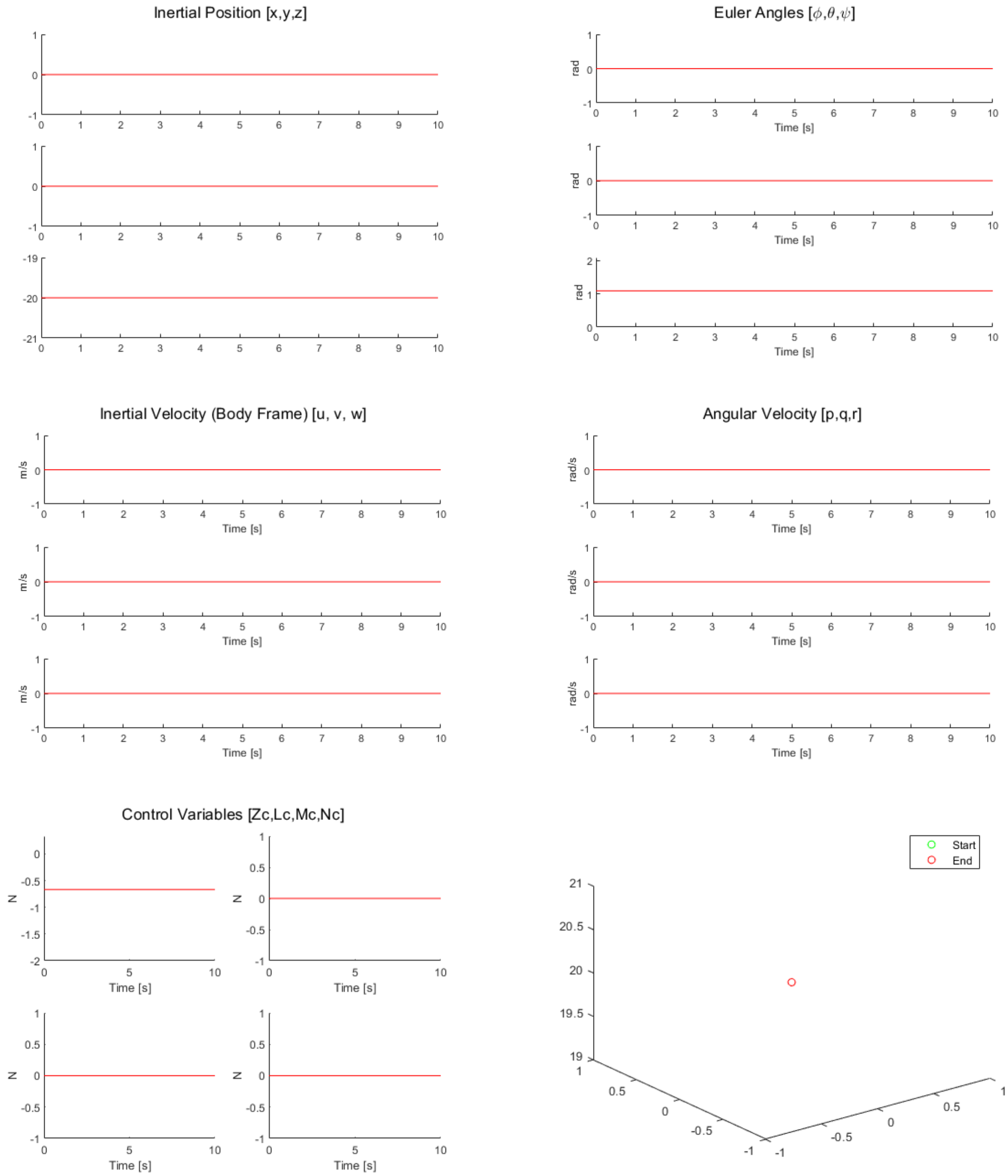
Deviation by +5 deg in roll



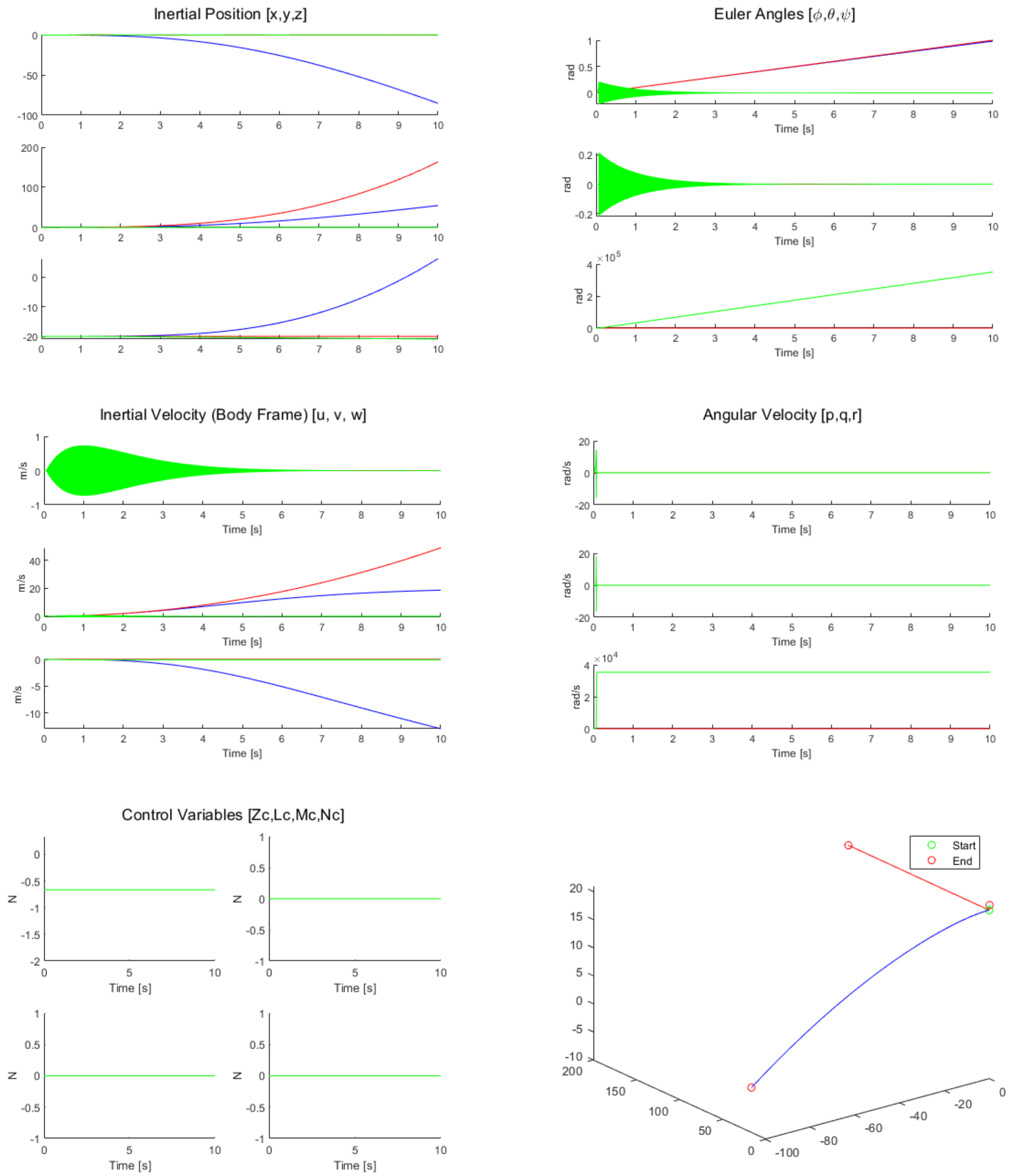
Deviation by +5 deg in pitch



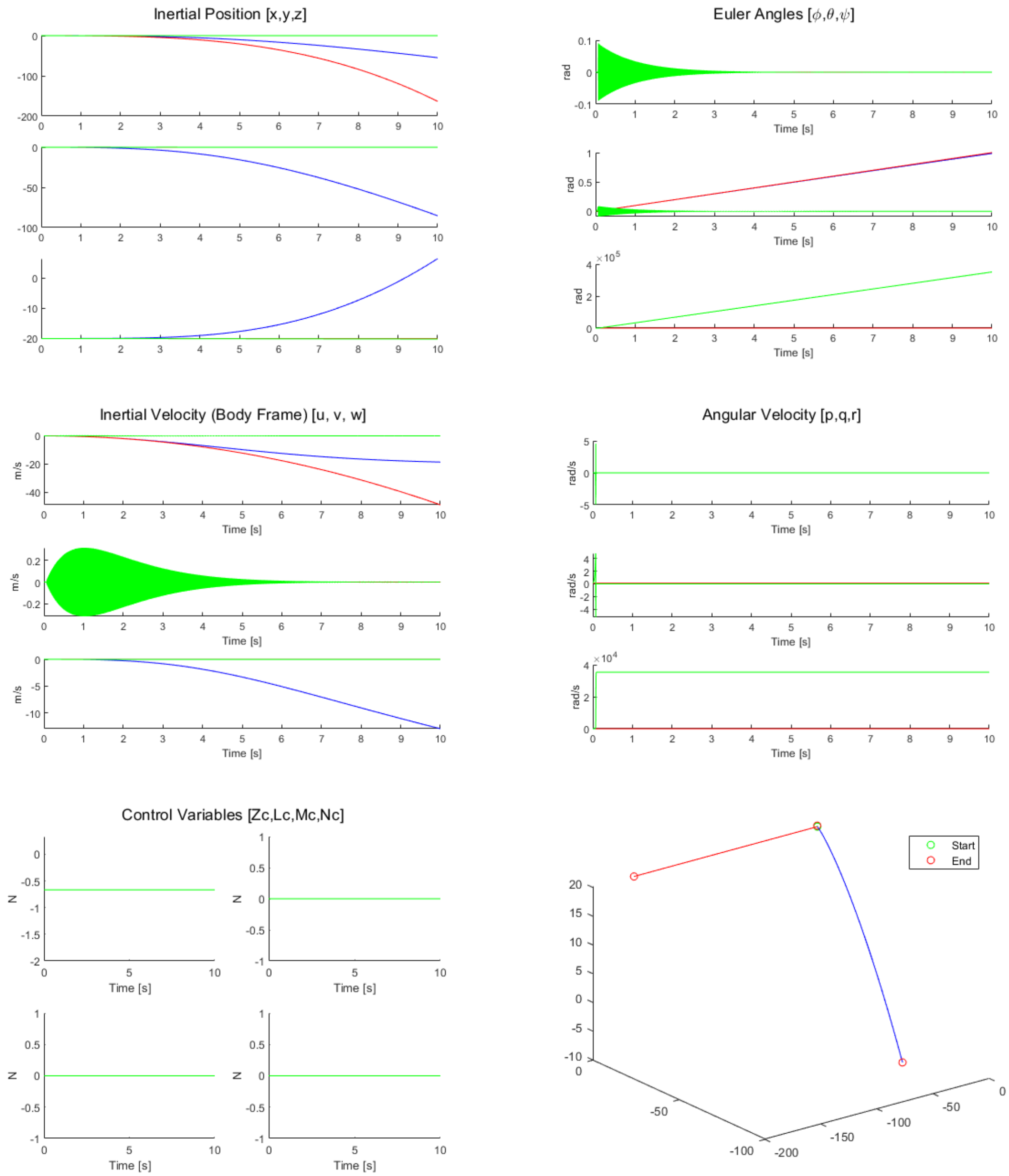
Deviation by +5 deg in yaw



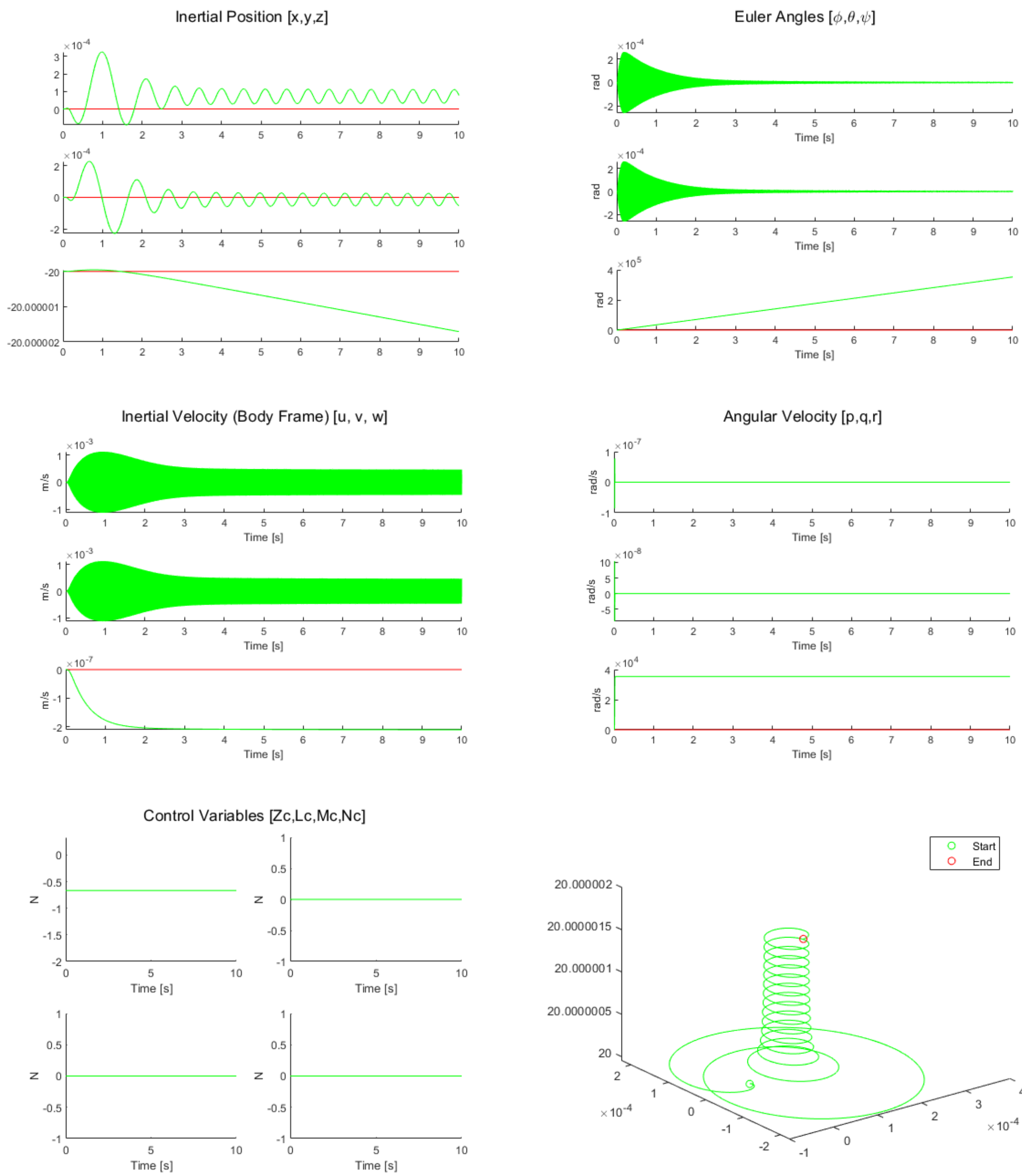
Deviation by +0.1 rad/s in roll rate



Deviation by +0.1 rad/s in pitch rate



Deviation by +0.1 rad/s in yaw rate



2.1

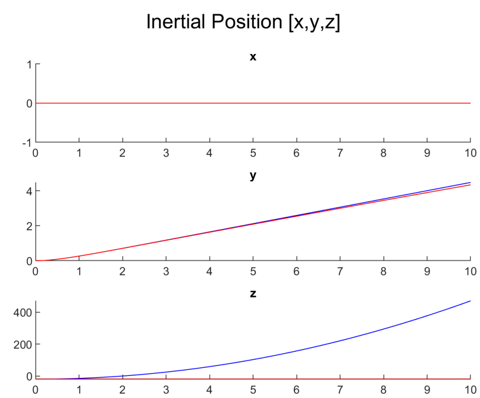
Comparing the linear and nonlinear graphs, the linear graphs tend to be straight lines without changes in rate. With the nonlinear graphs, there were changes in slope which are consistent with a non-linear system. In general then, these results do agree with what was recorded for lab task 1.5.

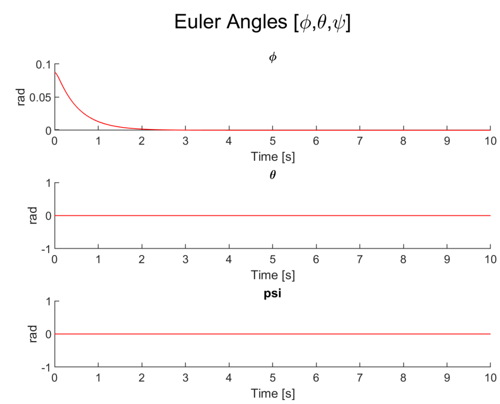
2.5

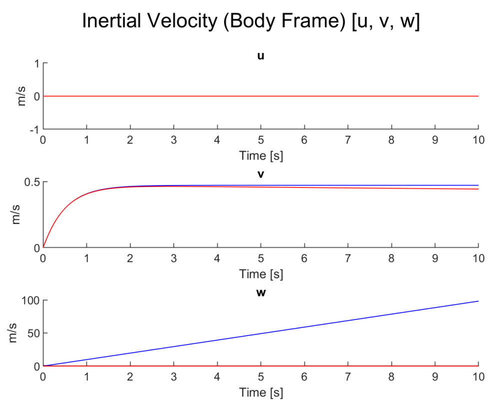
The effect of the control law is the quadrotor's motion is stabilized and regulated around the hover trim condition/state. The quadrotor responds to disturbances and keeps its stability due to the feedback controller, which then modifies control inputs based on the difference between the desired and actual states.

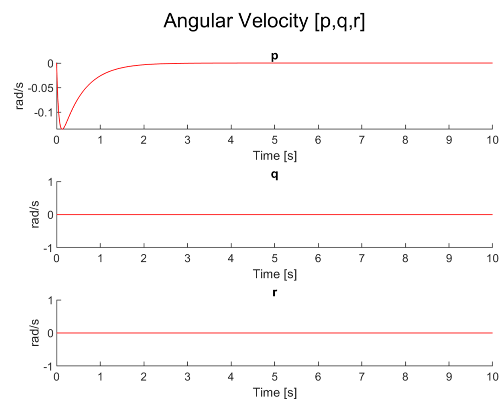
Task 3

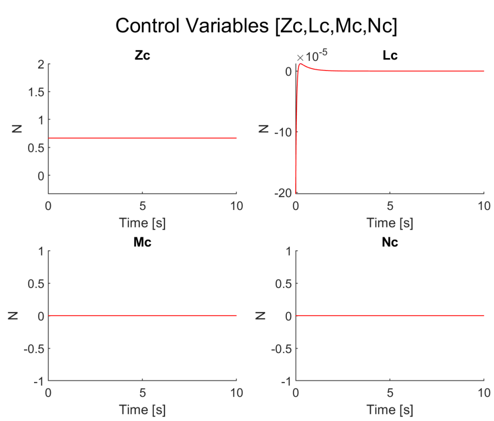
Blue: Linearized Response
Red: Feedback Response
Deviation by +5 deg in roll

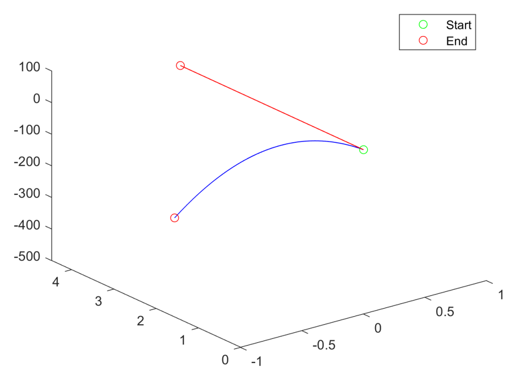




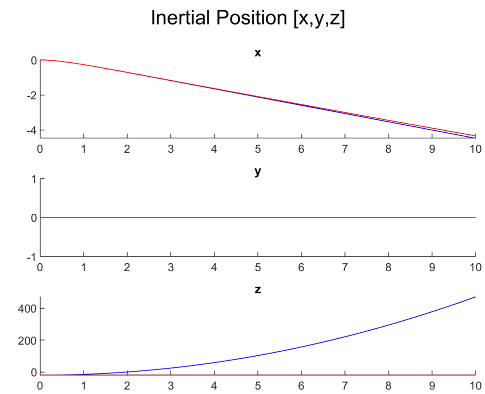


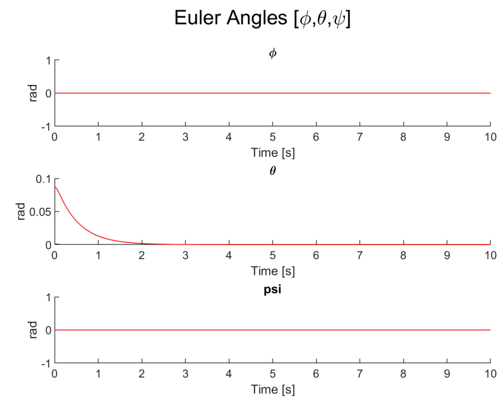


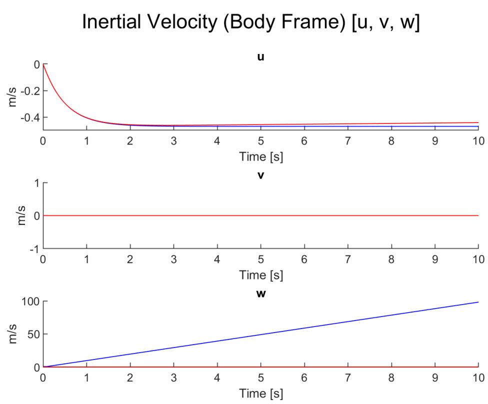


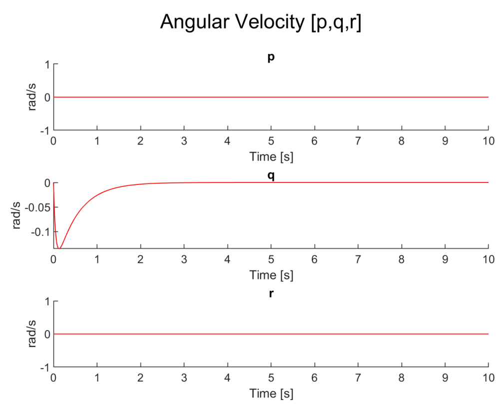


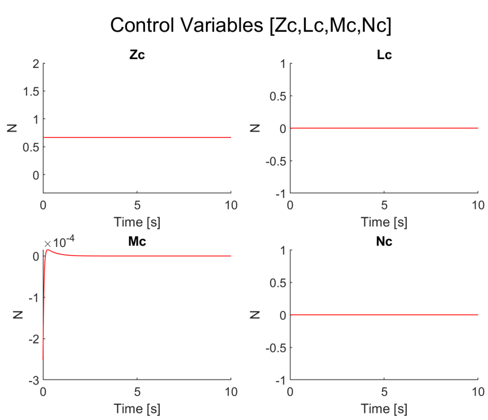
Deviation by +5 deg in pitch

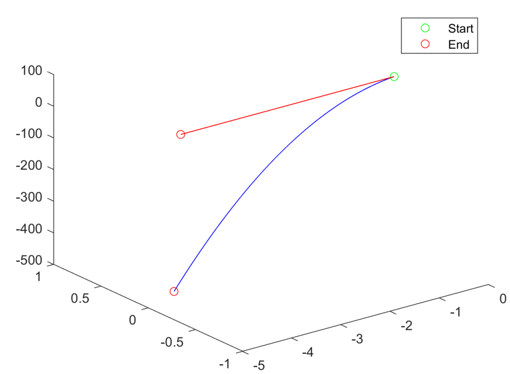




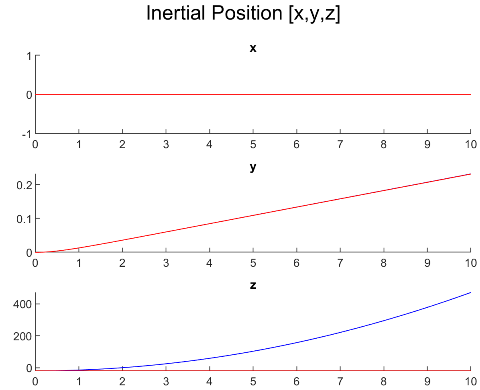


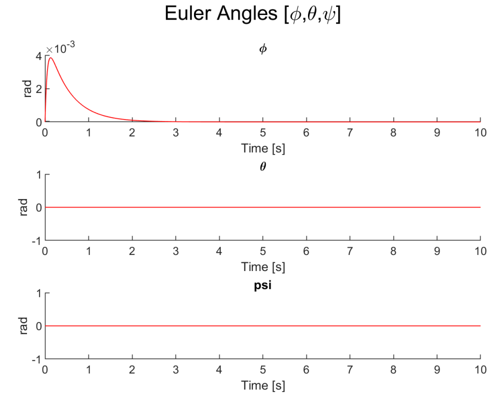


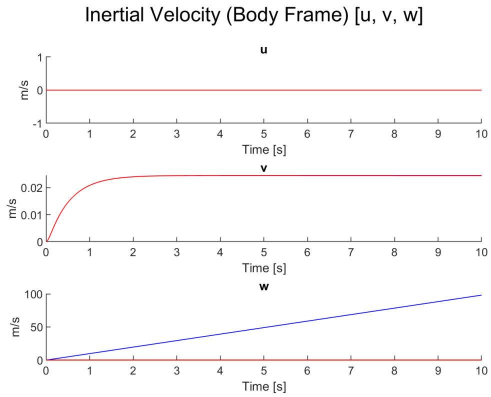


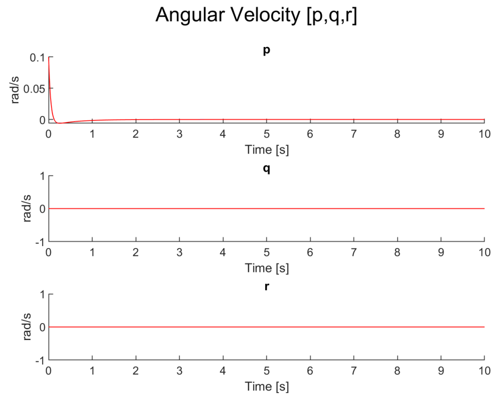


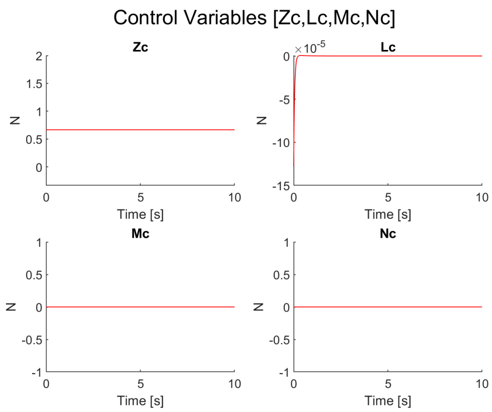
Deviation by +0.1 rad/s in roll rate

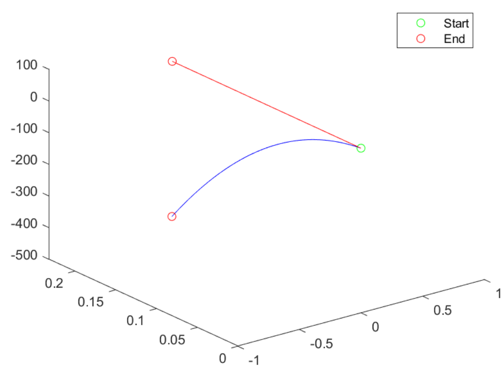




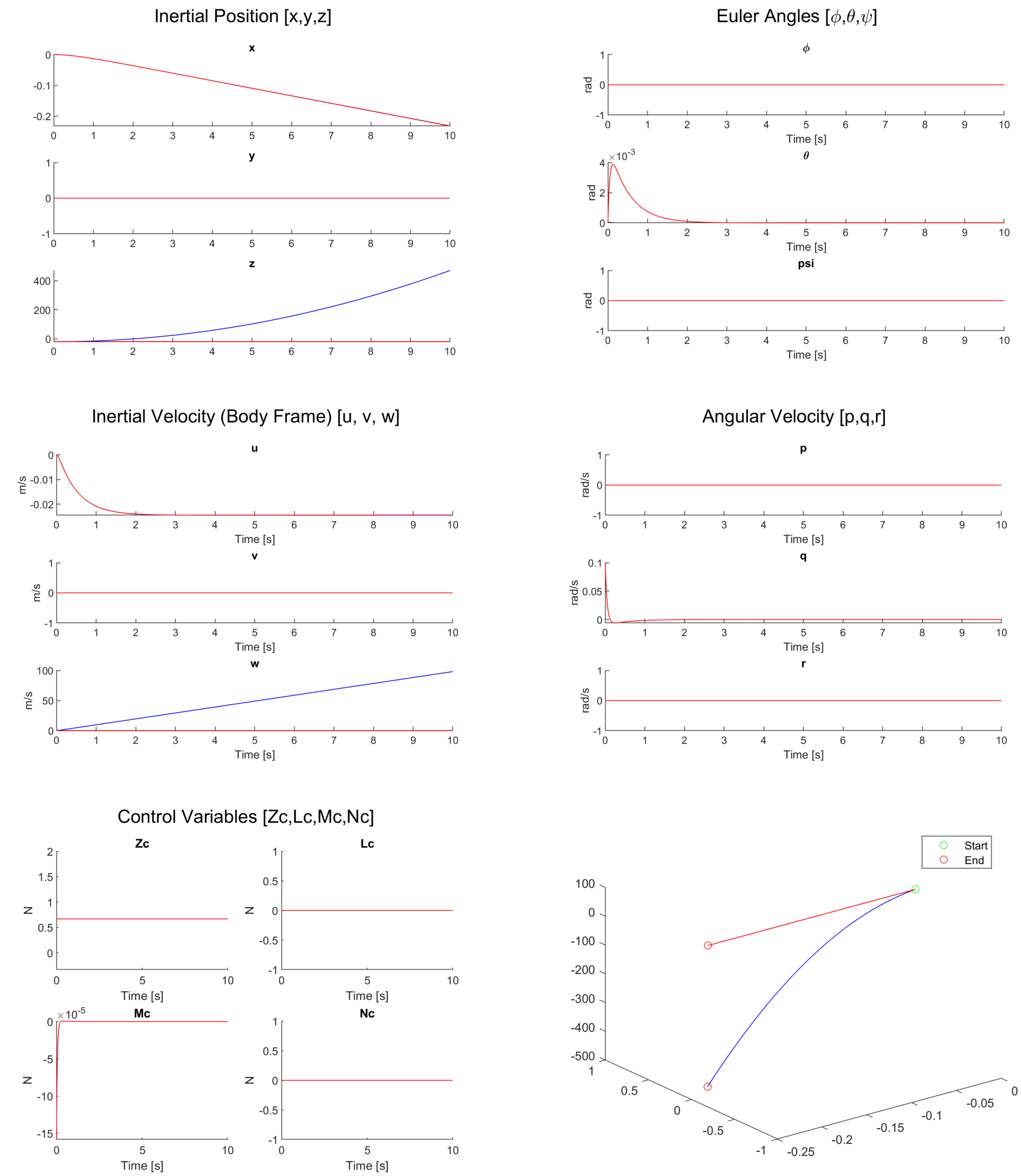






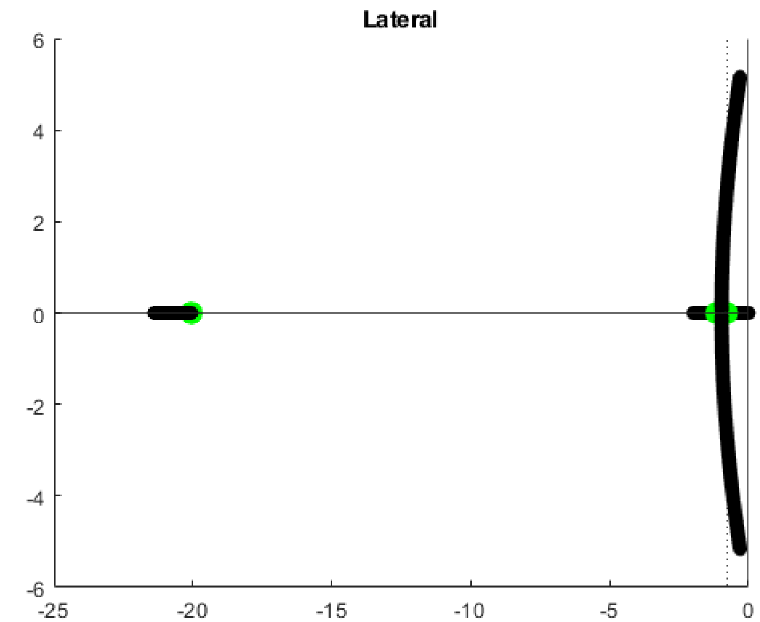


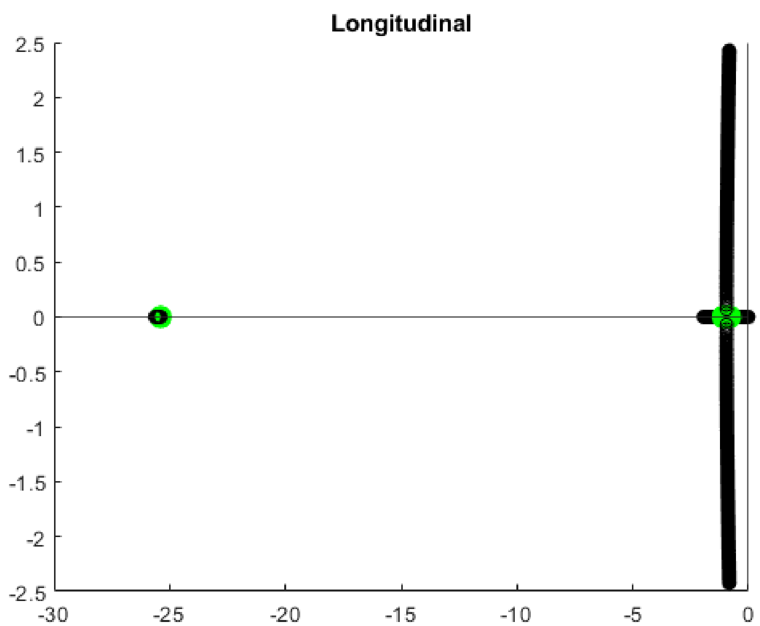
Deviation by +0.1 rad/s in pitch rate



The linear closed loop response is very clearly different than the non-linear response. This can be explained as our deviations are somewhat large and therefore have a distinct impact on the operation of the quadrotor - especially over a period of 10 seconds. This causes aggressive loss of altitude in all four cases in the linear system while the non-linear system is able to maintain a proper height.

3.5



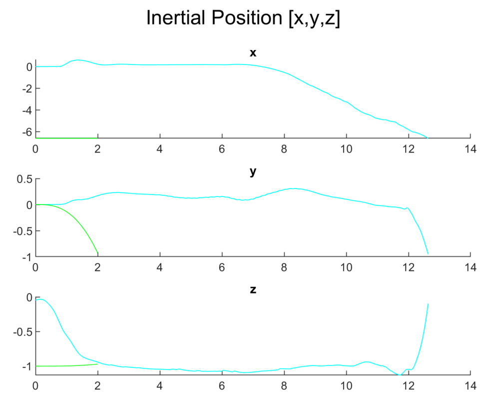


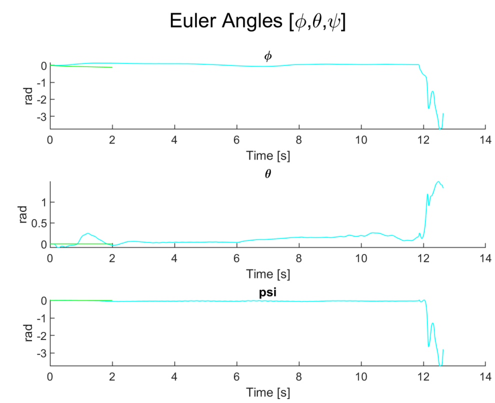
Final k values:

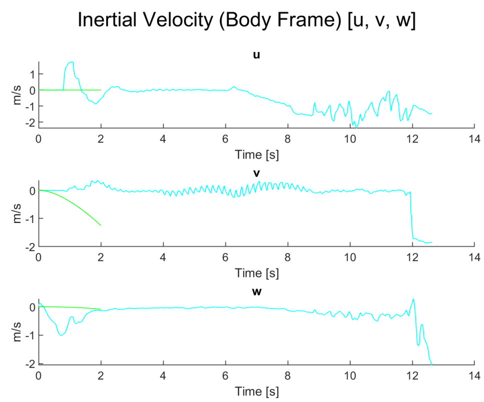
	k1	k2	k3
Lateral	0.0013	0.0023	1.1200
Longitudinal	0.0016	0.0029	-1.3800

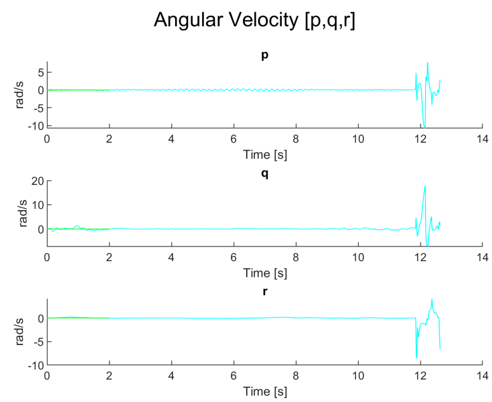
3.7-8

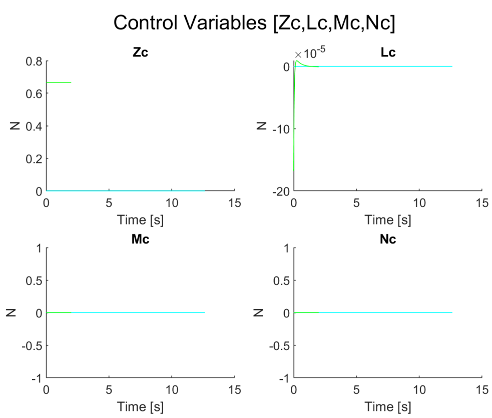
Green: Nonlinear simulation with Velocity Feedback
Cyan: Gain implementation (actual response)
Lateral Control

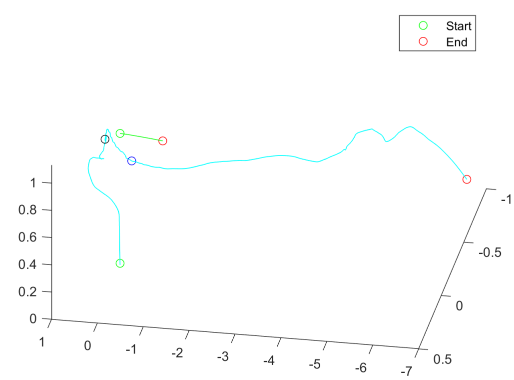






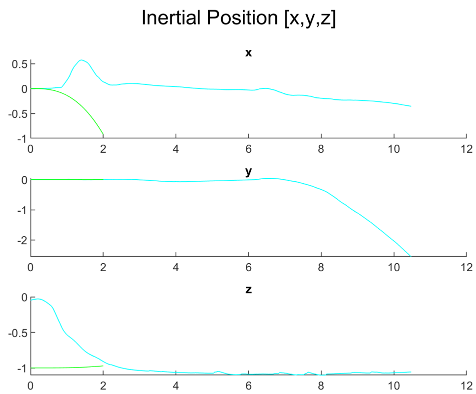


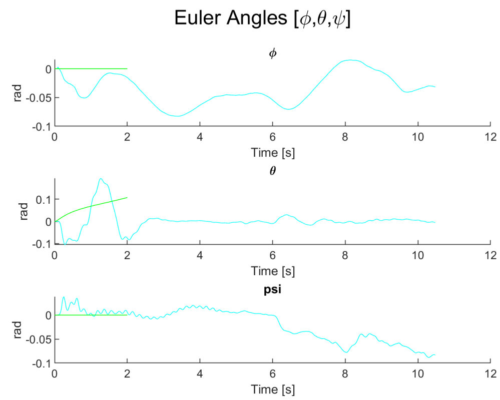


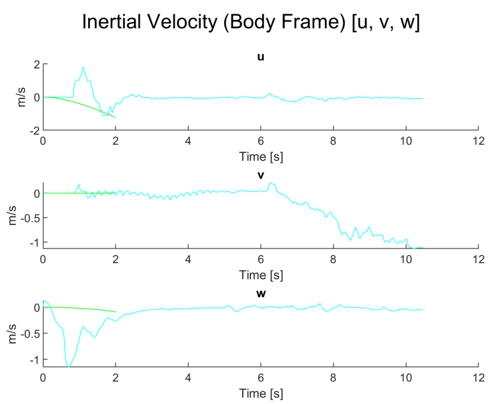


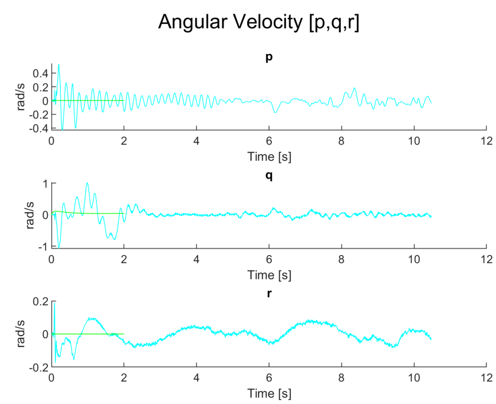
The Lateral control appears to work well in the simulated case. Maintaining altitude and translating almost exactly 1m with some adjustment of the reference velocity. The real implementation moves a little under 1m over the 2-second command period before continuing to shift and then losing altitude.

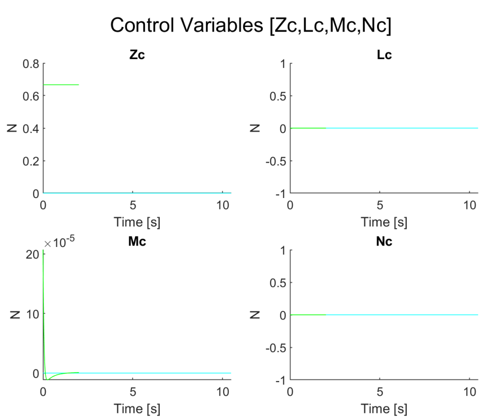
Longitudinal Control

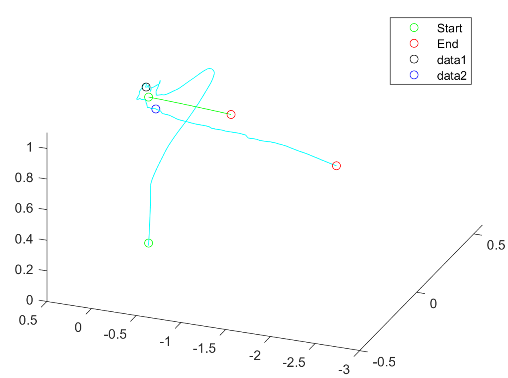




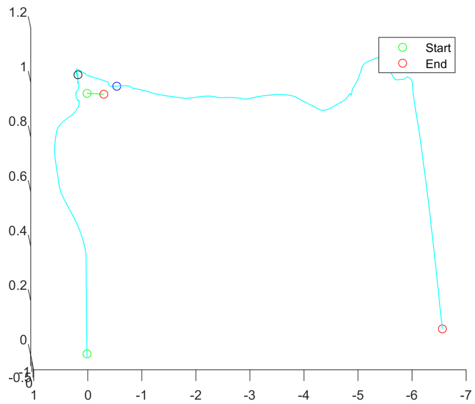


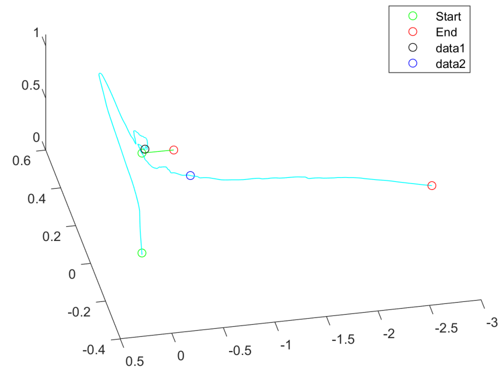






The Longitudinal control appears to work well in the simulated case. Maintaining altitude and translating almost exactly 1m with some adjustment of the reference velocity. The real implementation moves a little well under 1m over the 2-second command period before continuing to shift but maintaining altitude in contrast with the Lateral control case.





Using the same reference velocity we supplied to the TA team, (0.5 m/s) we can see that the displacement by the model lines up very well with that of the physical implementation. With futher development of the command, we could have adjusted the reference command velocity to achieve a displacement of 1m over the two seconds.

Member Contributions

Team Participation	Plan	Model	Experiment	Results	Report	Code	ACK
Abbitt Holland	2	1	1	1	2	1	x
Austin Hunter	2	1	1	1	1	2	x
Drew Kane	1	2	1	1	1	1	x
Rishab Pally	2	1	1	1	2	1	x

Appendix

```
function [motor_forces] = ComputeMotorForces(Fc,Gc, d,km)
% a function to calculate the motor thrust forces given the control force
    and moments
%   motor_forces is a 4x1 column vector [f1; f2; f3; f4]
M = [-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];
controls = [Fc(3);Gc];
motor_forces = -1*(M\controls);

end

function [Fc, Gc] = InnerLoopFeedback(t,var,lat,long)
%INNERLOOPFEEDBACK Summary of this function goes here
%   Detailed explanation goes here
%   Detailed explanation goes here
g = 9.81;
m = 0.068; %kg
Fc = [0; 0; m*g];


% Control moments about each body axis is proportional to the rotational
% rates about their respective axes

% define gain
k = -0.004; %Nm/(rad/s)
Lc = -(lat.k1)*var(10) - (lat.k2)*var(4);
Mc = -(long.k1)*var(11) - (long.k2)*var(5);
Nc = k*var(12);
Gc = [Lc; Mc; Nc];
end

clc; clear; close all;

%% ASEN 3801 Lab 4 Group 28

fig = 1:6;

% Givens
g = 9.81;
m = 0.068; %kg
d = 0.060; %m
km = 0.0024; %Nm/N
I_x = 5.8*10^-5; %kgm^2
I_y = 7.2*10^-5; %kgm^2
I_z = 1.0*10^-4; %kgm^2
nu = 1*10^-3; %N/(m/s)^2
mu = 2*10^-6; %N*m/(m/s)^2
motor_forces = m*g/4.*ones(4,1);
I = [I_x 0 0; 0 I_y 0; 0 0 I_z];

quad = @(t,var)QuadrotorEOM(t,var,g,m,I,d,km,nu,mu,motor_forces);
var = [0;0;-5;  0;0;1;  0;0;0; 0;0;0];
```

```
t = [0 10];
[time, aircraft_state] = ode45(quad,t,var);
%%

Z_c = -sum(motor_forces);
L_c = d/sqrt(2)*(-motor_forces(1)-motor_forces(2)+motor_forces(3)+
    motor_forces(4));
M_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)-motor_forces(3)+
    motor_forces(4));
N_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)+motor_forces(3)-
    motor_forces(4));

control_moments = [Z_c, L_c, M_c, N_c].*ones(numel(time),4);

PlotAircraftSim(time,aircraft_state,control_moments,fig,'-b')

%% 1.4
fig = fig +6;
V_a = 5;
phi = atan2(V_a^2*nu,m*g);

Z_c = -nu*V_a^2/sin(phi);

motor_forces = -Z_c/4.*ones(4,1);
quad = @(t,var)QuadrotorEOM(t,var,g,m,I,d,km,nu,mu,motor_forces);
var = [0;0;-50;  phi;0;0;  0;cos(phi)*V_a;-sin(phi)*V_a; 0;0;0];
t = [0 10];
[time, aircraft_state] = ode45(quad,t,var);


L_c = d/sqrt(2)*(-motor_forces(1)-motor_forces(2)+motor_forces(3)+
    motor_forces(4));
M_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)-motor_forces(3)+
    motor_forces(4));
N_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)+motor_forces(3)-
    motor_forces(4));

control_moments = [Z_c, L_c, M_c, N_c].*ones(numel(time),4);
PlotAircraftSim(time,aircraft_state,control_moments,fig,'-b')

%
quad = @(t,var)QuadrotorEOM(t,var,g,m,I,d,km,nu,mu,motor_forces);
var = [0;0;-50;  0;-phi;pi/2;  cos(phi)*V_a;0;-sin(phi)*V_a; 0;0;0];
t = [0 10];
[time, aircraft_state] = ode45(quad,t,var);
for i = 1:length(time)
V(i) = norm(aircraft_state(i,7:9));
end
figure(8)
plot(time, V)

L_c = d/sqrt(2)*(-motor_forces(1)-motor_forces(2)+motor_forces(3)+
    motor_forces(4));
M_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)-motor_forces(3)+
```

```
    motor_forces(4));
N_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)+motor_forces(3)-
    motor_forces(4));

control_moments = [Z_c, L_c, M_c, N_c].*ones(numel(time),4);
PlotAircraftSim(time,aircraft_state,control_moments,fig,'-r')

close all; clear; clc;
%% Variables
% Givens
g = 9.81;
m = 0.068; %kg
d = 0.060; %m
km = 0.0024; %Nm/N
I_x = 5.8*10^-5; %kgm^2
I_y = 7.2*10^-5; %kgm^2
I_z = 1.0*10^-4; %kgm^2
nu = 1*10^-3; %N/(m/s)^2
mu = 2*10^-6; %N*m/(m/s)^2
I = [I_x 0 0; 0 I_y 0; 0 0 I_z]; % Moment of Inertias
motor_forces = m*g/4.*ones(4,1);
t = [0 10];


%% Figure Nonsense
fig1 = (1:6)'; % For 1a
fig2 = (7:12)'; % For 1b
fig3 = (13:18)'; % For 1c
fig4 = (19:24)'; % For 1d
fig5 = (25:30)'; % For 1e
fig6 = (31:36)';  % For 1f
figures = [fig1, fig2, fig3, fig4, fig5, fig6]; % Combining into an array
col = ['b', 'r','g']; % Choice of color
titles = ["Problem 1a (+5 degree roll deviation)", "Problem 1b (+5 degree
    pitch deviation)", "Problem 1c (+5 degree yaw deviation)", "Problem 1d
    (+0.1 [rad/s] roll rate deviation)", "Problem 1e (+0.1 [rad/s] pitch
    rate deviation)", "Problem 1f (+0.1 [rad/s] yaw rate deviation)"];

% Deviatons
for i = 1:6

     deviations = [0;0;0; 0;0;0; 0;0;0; 0;0;0;]; % Initiallizing 12x1
        vector of zeros for state vector
     var = [0;0;-20;  0;0;1;  0;0;0; 0;0;0]; % state vector from Lab Task 1
     if i <4
        deviations(i+3) = 5 * (pi/180); % [degrees]
     else
        deviations(i+6) = 0.1; % [rad/s]
     end
     var = var + deviations;
     quad = @(t,var)QuadrotorEOM(t,var,g,m,I,d,km,nu,mu,motor_forces);
     [time, aircraft_state] = ode45(quad,t,var);

     Zc = -sum(motor_forces);
```

```
    controls = [Zc,0,0,0] .* ones(length(time),4);
    PlotAircraftSim(time,aircraft_state,controls,figures(:,i),col(1));
    % Linearizeed
    deltaFc = [0;0;0]; deltaGc = [0;0;0];
    quadLinear = @(t,var)QuadrotorEOM_Linearized(t,var,g,m,I,deltaFc,
       deltaGc);
    [timeL, aircraft_state_Linearized] = ode45(quadLinear,t,var);
    PlotAircraftSim(timeL,aircraft_state_Linearized,controls,figures(:,i),
       col(2));
    % Feedback
    if i > 3
    quadFeedback = @(t,v)QuadrotorEOMwithRateFeedback(t,v, g, m, I, nu, mu
       );
    [timeF,aircraft_state_Feedback] = ode45(quadFeedback,t,var);

    controls = zeros(length(aircraft_state_Feedback),4);
    for j = 1:length(aircraft_state_Feedback)
    [Fc, Gc] = RotationDerivativeFeedback(aircraft_state_Feedback(j,:),m,g
       );
    %motor_forces = ComputeMotorForces(Fc, Gc, d, km);
    controls(j,:) = [Fc(3),Gc];
    end
    PlotAircraftSim(timeF,aircraft_state_Feedback,controls,figures(:,i),
       col(3))
    end

end

close all; clear; clc;
%% Variables
% Givens
g = 9.81;
m = 0.068; %kg
d = 0.060; %m
km = 0.0024; %Nm/N
I_x = 5.8*10^-5; %kgm^2
I_y = 7.2*10^-5; %kgm^2
I_z = 1.0*10^-4; %kgm^2
nu = 1*10^-3; %N/(m/s)^2
mu = 2*10^-6; %N*m/(m/s)^2
I = [I_x 0 0; 0 I_y 0; 0 0 I_z]; % Moment of Inertias
t = [0 10];

%% Design Feedback control system 3.1

T1 = 0.5;
T2 = 0.05;

lam1 = -1/T1;
lam2 = -1/T2;

lat.k1 = -(lam1+lam2)*I_x;
lat.k2 = lam1*lam2*I_x;
```

```
long.k1 = -(lam1+lam2)*I_y;
long.k2 = lam1*lam2*I_y;

%% 3.5

% Lateral Control: find k3
figure(101)
title("Lateral")
hold on;
xline(0); yline(0); xline(-0.8,':k');
for idx = 1:15^3
k3 = idx * 0.000001;

A = [0 g 0; 0 0 1; -k3/I_x -lat.k2/I_x -lat.k1/I_x];

[V,D] = eig(A);

for i = 1:3
x1(i) = real(D(i,i));
y1(i) = imag(D(i,i));
end
if (all(x1 < -0.8)) && (all(y1 == 0))
plot(x1,y1,'go',LineWidth=5)
lat.k3 = k3;
else
plot(x1,y1,'ko')
end
end

% Longitudinal Control: find k3
figure(102)
title('Longitudinal')
hold on;
xline(0); yline(0); xline(-0.8,':k');
for idx = 1:10^3
k3 = -idx * 0.000001;

A = [0 -g 0; 0 0 1; -k3/I_x -long.k2/I_x -long.k1/I_x];

[V,D] = eig(A);

for i = 1:3
x1(i) = real(D(i,i));
y1(i) = imag(D(i,i));
end
if (all(x1 < -0.8)) && (all(y1 == 0))
plot(x1,y1,'go',LineWidth=5)
long.k3 = k3;
else
plot(x1,y1,'ko')
end
end

%% 3.3 & 3.4 & 3.7
```

```
% Figure Nonsense
fig1 = (1:6)'; % For 1a
fig2 = (7:12)'; % For 1b
fig3 = (13:18)'; % For 1c
fig4 = (19:24)'; % For 1d
fig5 = (25:30)'; % For 1e
fig6 = (31:36)';  % For 1f
figures = [fig1, fig2, fig3, fig4, fig5, fig6]; % Combining into an array
col = ['b', 'r','g']; % Choice of color
titles = ["Problem 1a (+5 degree roll deviation)", "Problem 1b (+5 degree
   pitch deviation)", "Problem 1c (+5 degree yaw deviation)", "Problem 1d
   (+0.1 [rad/s] roll rate deviation)", "Problem 1e (+0.1 [rad/s] pitch
   rate deviation)", "Problem 1f (+0.1 [rad/s] yaw rate deviation)"];

% Deviatons
for i = 1:4

    deviations = [0;0;0; 0;0;0; 0;0;0; 0;0;0;]; % Initiallizing 12x1
       vector of zeros for state vector
    var = [0;0;-20;  0;0;0;  0;0;0; 0;0;0]; % trim state vector from Lab
       Task 1
    if i <3
       deviations(i+3) = 5 * (pi/180); % [degrees]
    else
       deviations(i+7) = 0.1; % [rad/s]
    end
    var = var + deviations;

    % Linearized
    quadLinear = @(t,var)QuadrotorEOM_CL_Linearized(t,var,g,m,I,lat,long);
    [timeL, aircraft_state_Linearized] = ode45(quadLinear,t,var);

    controls = zeros(length(aircraft_state_Linearized),4);
    for j = 1:length(aircraft_state_Linearized)
    [Fc, Gc] = InnerLoopFeedback(t,aircraft_state_Linearized(j,:),lat,long
       );
    %motor_forces = ComputeMotorForces(Fc, Gc, d, km);
    controls(j,:) = [Fc(3),Gc'];
    end
    PlotAircraftSim(timeL,aircraft_state_Linearized,controls,figures(:,i),
       col(1));

    % Feedback
    quadFeedback = @(t,v)QuadrotorEOMwithRateFeedback_CL(t,v, g, m, I, nu,
        mu,lat,long);
    [timeF,aircraft_state_Feedback] = ode45(quadFeedback,t,var);

    controls = zeros(length(aircraft_state_Feedback),4);
    for j = 1:length(aircraft_state_Feedback)
    [Fc, Gc] = InnerLoopFeedback(t,aircraft_state_Feedback(j,:),lat,long);
    %motor_forces = ComputeMotorForces(Fc, Gc, d, km);
    controls(j,:) = [Fc(3),Gc'];
    end
```

```
    PlotAircraftSim(timeF,aircraft_state_Feedback,controls,figures(:,i),
        col(2))

    % % Velocity Feedback
    % quadFeedback = @(t,v)QuadrotorEOMwithRateFeedback_CL_Velocity(t,v, g
        , m, I, nu, mu,lat,long);
    % [timeF,aircraft_state_Feedback_V] = ode45(quadFeedback,t,var);
    %
    % controls = zeros(length(aircraft_state_Feedback_V),4);
    % for j = 1:length(aircraft_state_Feedback_V)
    % [Fc, Gc] = VelocityReferenceFeedback(t,aircraft_state_Feedback_V(j
        ,:),lat,long);
    % %motor_forces = ComputeMotorForces(Fc, Gc, d, km);
    % controls(j,:) = [Fc(3),Gc'];
    % end
    %
    % PlotAircraftSim(timeF,aircraft_state_Feedback_V,controls,figures(:,i
        ),col(3))

end



%% 3.7 & 3.8


t = [0 2];
% Lateral
load("RSdata_11_16.mat")
times = rt_estim.time(:);
stateEstim = rt_estim.signals.values(:,:);
controls = zeros(length(times),4);
PlotAircraftSim(times,stateEstim,controls,figures(:,5),'c');
figure(30)
hold on;
[~,i] = min(abs(times-6));
[~,idx] = min(abs(times-8));
plot3(stateEstim(i,1),stateEstim(i,2),-stateEstim(i,3),'ko')
plot3(stateEstim(idx,1),stateEstim(idx,2),-stateEstim(idx,3),'bo')

type = 1;
var = [stateEstim(end,1);0;-rt_cmd.signals.values(4);  0;0;0;  0;0;0;
    0;0;0]; % trim state vector
quadFeedback = @(t,var)QuadrotorEOMwithRateFeedback_CL_Velocity(t, var, g,
     m, I, nu, mu,lat,long,type);
[timeF,aircraft_state_Feedback_V] = ode45(quadFeedback,t,var);

    controls = zeros(length(aircraft_state_Feedback_V),4);
    for j = 1:length(aircraft_state_Feedback_V)
    [Fc, Gc] = VelocityReferenceFeedback(t,aircraft_state_Feedback_V(j,:),
        lat,long,type);
    %motor_forces = ComputeMotorForces(Fc, Gc, d, km);
    controls(j,:) = [Fc(3),Gc'];
    end
```

```
    PlotAircraftSim(timeF,aircraft_state_Feedback_V,controls,figures(:,5),
       col(3))

% Longitudinal

load("RSdata_09_36.mat")
times = rt_estim.time(:);
stateEstim = rt_estim.signals.values(:,:);
controls = zeros(length(times),4);
PlotAircraftSim(times,stateEstim,controls,figures(:,6),'c');

type = 2;
var = [0;0;-rt_cmd.signals.values(4);  0;0;0;  0;0;0; 0;0;0]; % trim state
    vector
quadFeedback = @(t,var)QuadrotorEOMwithRateFeedback_CL_Velocity(t, var, g,
    m, I, nu, mu,lat,long,type);
[timeF,aircraft_state_Feedback_V] = ode45(quadFeedback,t,var);

    controls = zeros(length(aircraft_state_Feedback_V),4);
    for j = 1:length(aircraft_state_Feedback_V)
    [Fc, Gc] = VelocityReferenceFeedback(t,aircraft_state_Feedback_V(j,:),
       lat,long,type);
    %motor_forces = ComputeMotorForces(Fc, Gc, d, km);
    controls(j,:) = [Fc(3),Gc'];
    end

    PlotAircraftSim(timeF,aircraft_state_Feedback_V,controls,figures(:,6),
       col(3))

function PlotAircraftSim(time, aircraft_state_array, control_input_array,
   fig,col)
%PLOTAIRCRAFTSIM Summary of this function goes here
%   Detailed explanation goes here

%% Inertial Position
figure(fig(1));
names = ['x','y','z'];
for i = 1:3
subplot(3,1,i);
hold on;
plot(time,aircraft_state_array(:,i),col);
title(names(i))
end
sgtitle('Inertial Position [x,y,z]')
%% Euler angles
figure(fig(2));
names = ["\phi","\theta","psi"];
for i = 1:3
subplot(3,1,i);
hold on;
plot(time,aircraft_state_array(:,3+i),col);
xlabel('Time [s]')
ylabel('rad')
```

```
title(names(i))
end
sgtitle('Euler Angles [\phi,\theta,\psi]')
%% Inertial Velocity (body frame)
figure(fig(3));
names = ['u','v','w'];
for i = 1:3
subplot(3,1,i);
hold on;
plot(time,aircraft_state_array(:,6+i),col);
xlabel('Time [s]')
ylabel('m/s')
title(names(i))
end
sgtitle('Inertial Velocity (Body Frame) [u, v, w]')
%% Angular Velocity
figure(fig(4));
names = ['p','q','r'];
for i = 1:3
subplot(3,1,i);
hold on;
plot(time,aircraft_state_array(:,9+i),col);
xlabel('Time [s]')
ylabel('rad/s')
title(names(i))
end
sgtitle('Angular Velocity [p,q,r]')
%% Control Input Variables
figure(fig(5));
name = ["Zc","Lc","Mc","Nc"];
for i = 1:4
subplot(2,2,i)
hold on;
plot(time,control_input_array(:,i),col)
xlabel('Time [s]')
ylabel('N')
title(name(i))
end
sgtitle('Control Variables [Zc,Lc,Mc,Nc]')
%% 3D Path
figure(fig(6));
hold on;
plot3(aircraft_state_array(1,1),aircraft_state_array(1,2),-
    aircraft_state_array(1,3),'go');
plot3(aircraft_state_array(:,1),aircraft_state_array(:,2),-
    aircraft_state_array(:,3),col);
plot3(aircraft_state_array(end,1),aircraft_state_array(end,2),-
    aircraft_state_array(end,3),'ro');
legend('Start','','End')
% range = max([aircraft_state_array(:,1); aircraft_state_array(:,2);-
    aircraft_state_array(:,3)])+1;
% xlim([0,range])
% ylim([0,range])
% %zlim([0,-min(aircraft_state_array(:,3))])
```

```
% %axis equal;
view([-37.5 30]);


end

function [var_dot] = QuadrotorEOM(t,var,g,m,I,d,km,nu,mu,motor_forces)
%QUATDROTOREOM Summary of this function goes here
%   Detailed explanation goes here
var_dot = zeros(12,1);
phi = var(4); theta = var(5); psi = var(6);
I_x = I(1,1); I_y = I(2,2); I_z = I(3,3);
% Aerodynamic Forces
Aero_F = -nu*norm(var(7:9)).*var(7:9);

Aero_M = -mu*norm(var(10:12)).*var(10:12);

L_c = d/sqrt(2)*(-motor_forces(1)-motor_forces(2)+motor_forces(3)+
    motor_forces(4));
M_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)-motor_forces(3)+
    motor_forces(4));
N_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)+motor_forces(3)-
    motor_forces(4));
%% Position Data
R = [cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi),
    cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi); ...
     cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi),
        cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi); ...
     -sin(theta), sin(phi)*cos(theta), cos(phi)*cos(theta)];
var_dot(1:3) = R*(var(7:9));

%% Attitude
A = [1 sin(phi)*tan(theta) cos(phi)*tan(theta); 0 cos(phi) -sin(phi); 0
    sin(phi)*sec(theta) cos(phi)*sec(theta)];
var_dot(4:6) = A*var(10:12);

%% Velocity (body frame)
u = var(7); v = var(8); w = var(9); p = var(10); q = var(11); r = var(12);
var_dot(7:9) = [r*v - q*w; p*w - r*u; q*u - p*v] + g.*[-sin(theta); cos(
    theta)*sin(phi); cos(theta)*cos(phi)] + (1/m).*Aero_F + (1/m).*[0;0; -
    sum(motor_forces)];

%% Rotation Rates
var_dot(10:12) = [(I_y - I_z)/I_x*q*r; (I_z-I_x)/I_y*p*r;(I_x-I_y)/I_z*p*q
    ] +[1/I_x; 1/I_y; 1/I_z].*Aero_M +[1/I_x*L_c; 1/I_y*M_c; 1/I_z*N_c];


end

function [var_dot] = QuadrotorEOM_Linearized(t,var,g,m,I,deltaFc,deltaGc)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
var_dot = zeros(12,1);
I_x = I(1,1); I_y = I(2,2); I_z = I(3,3);
```

```
% position and attitude
var_dot(1:6) = var(7:12);

var_dot(7:9) = g.*[-var(5); var(4); 0] + 1/m.*deltaFc;

var_dot(10:12) = [deltaGc(1)/I_x; deltaGc(2)/I_y; deltaGc(3)/I_z];

end

function [var_dot] = QuadrotorEOM_CL_Linearized(t,var,g,m,I,lat,long)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
var_dot = zeros(12,1);
I_x = I(1,1); I_y = I(2,2); I_z = I(3,3);

[Fc, Gc] = InnerLoopFeedback(t,var,lat,long);
% position and attitude
var_dot(1:6) = var(7:12);

var_dot(7:9) = g.*[-var(5); var(4); 0] + 1/m.*Fc;

var_dot(10:12) = [Gc(1)/I_x; Gc(2)/I_y; Gc(3)/I_z];

end

function [var_dot] = QuadrotorEOMwithRateFeedback(t, var, g, m, I, nu, mu)
%QUADROTOREOMWITHRATEFEEDBACK Summary of this function goes here
%   Detailed explanation goes here
d = 0.060; %m
km = 0.0024; %Nm/N
var_dot = zeros(12,1);
phi = var(4); theta = var(5); psi = var(6);
I_x = I(1,1); I_y = I(2,2); I_z = I(3,3);
% Aerodynamic Forces
Aero_F = -nu*norm(var(7:9)).*var(7:9);

Aero_M = -mu*norm(var(10:12)).*var(10:12);

[Fc, Gc] = RotationDerivativeFeedback(var,m,g);
motor_forces = ComputeMotorForces(Fc, Gc, d, km);

L_c = d/sqrt(2)*(-motor_forces(1)-motor_forces(2)+motor_forces(3)+
    motor_forces(4));
M_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)-motor_forces(3)+
    motor_forces(4));
N_c = d/sqrt(2)*(motor_forces(1)-motor_forces(2)+motor_forces(3)-
    motor_forces(4));
%% Position Data
R = [cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi),
    cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi); ...
     cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi),
        cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi); ...
     -sin(theta), sin(phi)*cos(theta), cos(phi)*cos(theta)];
var_dot(1:3) = R*(var(7:9));
```

```
%% Attitude
A = [1 sin(phi)*tan(theta) cos(phi)*tan(theta); 0 cos(phi) -sin(phi); 0
   sin(phi)*sec(theta) cos(phi)*sec(theta)];
var_dot(4:6) = A*var(10:12);

%% Velocity (body frame)
u = var(7); v = var(8); w = var(9); p = var(10); q = var(11); r = var(12);
var_dot(7:9) = [r*v - q*w; p*w - r*u; q*u - p*v] + g.*[-sin(theta); cos(
   theta)*sin(phi); cos(theta)*cos(phi)] + (1/m).*Aero_F + (1/m).*[0;0; -
   sum(motor_forces)];

%% Rotation Rates
var_dot(10:12) = [(I_y - I_z)/I_x*q*r; (I_z-I_x)/I_y*p*r;(I_x-I_y)/I_z*p*q
   ] +[1/I_x; 1/I_y; 1/I_z].*Aero_M +[1/I_x*L_c; 1/I_y*M_c; 1/I_z*N_c];


end

function [var_dot] = QuadrotorEOMwithRateFeedback_CL(t, var, g, m, I, nu,
   mu,lat,long)
%QUADROTOREOMWITHRATEFEEDBACK Summary of this function goes here
%   Detailed explanation goes here
d = 0.060; %m
km = 0.0024; %Nm/N
var_dot = zeros(12,1);
phi = var(4); theta = var(5); psi = var(6);
I_x = I(1,1); I_y = I(2,2); I_z = I(3,3);
% Aerodynamic Forces
Aero_F = -nu*norm(var(7:9)).*var(7:9);

Aero_M = -mu*norm(var(10:12)).*var(10:12);

[Fc, Gc] = InnerLoopFeedback(t,var,lat,long);
motor_forces = ComputeMotorForces(Fc, Gc, d, km);

L_c = Gc(1); %d/sqrt(2)*(-motor_forces(1)-motor_forces(2)+motor_forces(3)+
   motor_forces(4));
M_c = Gc(2); %d/sqrt(2)*(motor_forces(1)-motor_forces(2)-motor_forces(3)+
   motor_forces(4));
N_c = Gc(3); %d/sqrt(2)*(motor_forces(1)-motor_forces(2)+motor_forces(3)-
   motor_forces(4));
%% Position Data
R = [cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi),
   cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi); ...
    cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi),
       cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi); ...
    -sin(theta), sin(phi)*cos(theta), cos(phi)*cos(theta)];
var_dot(1:3) = R*(var(7:9));

%% Attitude
A = [1 sin(phi)*tan(theta) cos(phi)*tan(theta); 0 cos(phi) -sin(phi); 0
   sin(phi)*sec(theta) cos(phi)*sec(theta)];
var_dot(4:6) = A*var(10:12);
```

```
%% Velocity (body frame)
u = var(7); v = var(8); w = var(9); p = var(10); q = var(11); r = var(12);
var_dot(7:9) = [r*v - q*w; p*w - r*u; q*u - p*v] + g.*[-sin(theta); cos(
    theta)*sin(phi); cos(theta)*cos(phi)] + (1/m).*Aero_F + (1/m).*[0;0; -
    sum(motor_forces)];

%% Rotation Rates
var_dot(10:12) = [(I_y - I_z)/I_x*q*r; (I_z-I_x)/I_y*p*r;(I_x-I_y)/I_z*p*q
    ] +[1/I_x; 1/I_y; 1/I_z].*Aero_M +[1/I_x*L_c; 1/I_y*M_c; 1/I_z*N_c];


end

function [var_dot] = QuadrotorEOMwithRateFeedback_CL(t, var, g, m, I, nu,
    mu,lat,long,type)
%QUADROTOREOMWITHRATEFEEDBACK Summary of this function goes here
%   Detailed explanation goes here
d = 0.060; %m
km = 0.0024; %Nm/N
var_dot = zeros(12,1);
phi = var(4); theta = var(5); psi = var(6);
I_x = I(1,1); I_y = I(2,2); I_z = I(3,3);
% Aerodynamic Forces
Aero_F = -nu*norm(var(7:9)).*var(7:9);

Aero_M = -mu*norm(var(10:12)).*var(10:12);

[Fc, Gc] = VelocityReferenceFeedback(t,var,lat,long,type);
motor_forces = ComputeMotorForces(Fc, Gc, d, km);

L_c = Gc(1); %d/sqrt(2)*(-motor_forces(1)-motor_forces(2)+motor_forces(3)+
    motor_forces(4));
M_c = Gc(2); %d/sqrt(2)*(motor_forces(1)-motor_forces(2)-motor_forces(3)+
    motor_forces(4));
N_c = Gc(3); %d/sqrt(2)*(motor_forces(1)-motor_forces(2)+motor_forces(3)-
    motor_forces(4));
%% Position Data
R = [cos(theta)*cos(psi), sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi),
    cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi); ...
     cos(theta)*sin(psi), sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi),
        cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi); ...
     -sin(theta), sin(phi)*cos(theta), cos(phi)*cos(theta)];
var_dot(1:3) = R*(var(7:9));

%% Attitude
A = [1 sin(phi)*tan(theta) cos(phi)*tan(theta); 0 cos(phi) -sin(phi); 0
    sin(phi)*sec(theta) cos(phi)*sec(theta)];
var_dot(4:6) = A*var(10:12);

%% Velocity (body frame)
u = var(7); v = var(8); w = var(9); p = var(10); q = var(11); r = var(12);
var_dot(7:9) = [r*v - q*w; p*w - r*u; q*u - p*v] + g.*[-sin(theta); cos(
    theta)*sin(phi); cos(theta)*cos(phi)] + (1/m).*Aero_F + (1/m).*[0;0; -
```

```
    sum(motor_forces)];

%% Rotation Rates
var_dot(10:12) = [(I_y - I_z)/I_x*q*r; (I_z-I_x)/I_y*p*r;(I_x-I_y)/I_z*p*q
    ] +[1/I_x; 1/I_y; 1/I_z].*Aero_M +[1/I_x*L_c; 1/I_y*M_c; 1/I_z*N_c];


end

function [Fc,Gc] = RotationDerivativeFeedback(var,m,g)
% The function takes as input the 12x1 aircraft state var, aircraft mass m
    ,
% and gravitational acceleration g.

% Control force in the body z direction is equal to the weight of the
% quadrotor
Gc = zeros(3,1);
Fc = [0; 0; m*g];


% Control moments about each body axis is proportional to the rotational
% rates about their respective axes

% define gain
k = -0.004; %Nm/(rad/s)
Gc = k.*var(10:12);
end

function [Fc, Gc] = VelocityReferenceFeedback(t,var,lat,long,type)
%VELOCITYREFERENCEFEEDBACK Summary of this function goes here
%   Detailed explanation goes here
g = 9.81;
m = 0.068; %kg
Fc = [0; 0; m*g];

if (type == 1)
v_r = 1.5; u_r = 0;
elseif (type == 2)
v_r = 0; u_r = 1.5;
end

v_r
u_r
% Control moments about each body axis is proportional to the rotational
% rates about their respective axes

% define gain
k = -0.004; %Nm/(rad/s)
Lc = -(lat.k1)*var(10) - (lat.k2)*var(4) - (lat.k3)*(v_r - var(8));
Mc = -(long.k1)*var(11) - (long.k2)*var(5) - (long.k3)*(u_r - var(7));
Nc = k*var(12);
Gc = [Lc; Mc; Nc];
end
```