ASEN 3128 - Assignment 1

Seth Hill
Gregory Kondor
Connor Ott

Group 14 - Station 7b
February 12, 2018
University of Colorado - Boulder

Assignment 5 | Connor Ott

Linearized Model:

$$\Delta\dot{p} \approx \frac{R}{I_x}(\Delta f_2 + \Delta f_3 - \Delta f_1 - \Delta f_4) \qquad R = \frac{d}{\sqrt{2}}$$

$$\Delta\dot{q} \approx \frac{R}{I_y}(\Delta f_3 + \Delta f_4 - \Delta f_1 - \Delta f_2)$$

$$\Delta\dot{r} \approx \frac{k}{I_z}(\Delta f_2 + \Delta f_4 - \Delta f_1 - \Delta f_3)$$

$$\Delta\dot{u}^E = -g\Delta\theta$$

$$\Delta\dot{v}^E \approx g\Delta\phi$$

$$\Delta\dot{w}^E \approx \frac{1}{m}(-\Delta f_1 - \Delta f_2 - \Delta f_3 - \Delta f_4)$$

$$\Delta\dot{\phi} \approx \Delta p$$

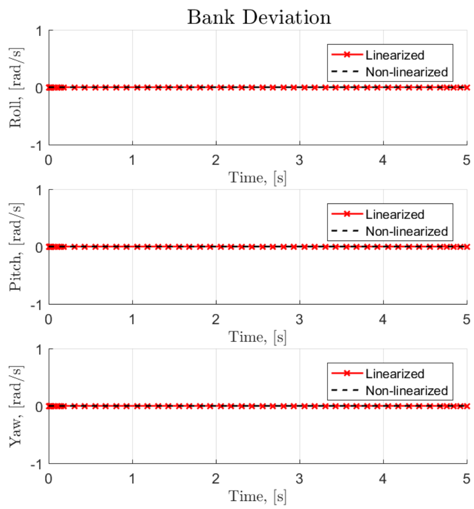
$$\Delta\dot{\theta} \approx \Delta q$$

$$\Delta\dot{\psi} \approx \Delta r$$

Problem 2 - 3

The following plots indicate where the linear and nonlinear models deviate from each other when the quad copter strays from the nominal trim state.

Part a) - 5° Bank Deviation



;

Figure 1: Body coordinate angular velocity as a function of time.

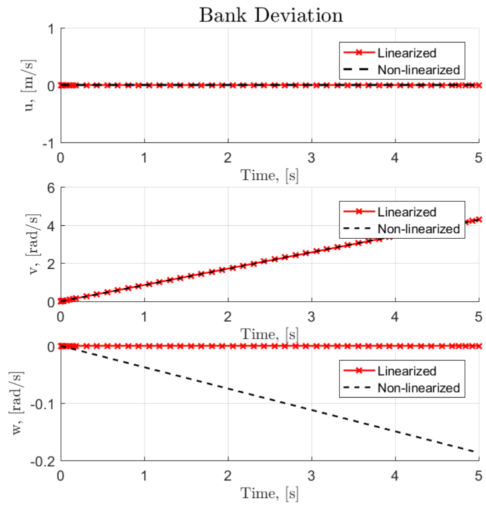
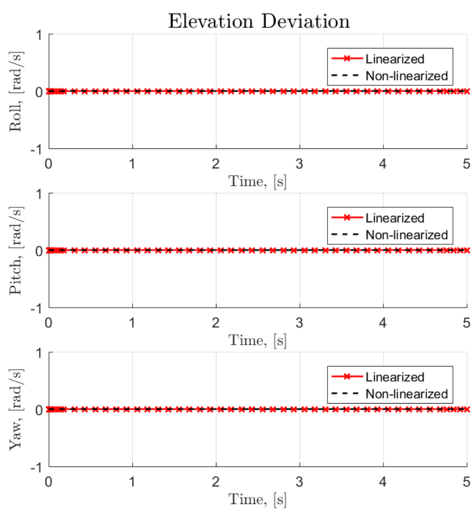


Figure 2: Body coordinate velocity as a function of time.

Part b) - 5° Elevation Deviation



;

Figure 3: Body coordinate angular velocity as a function of time.

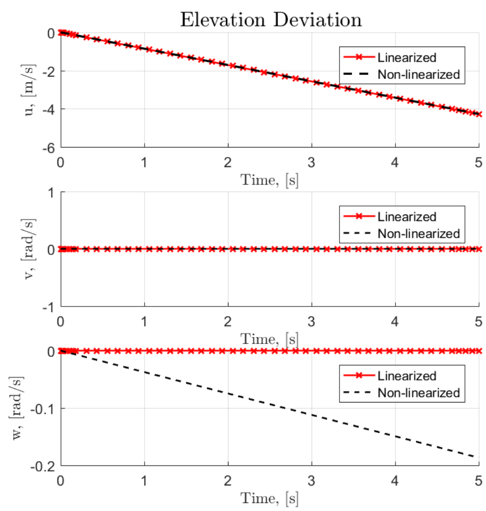
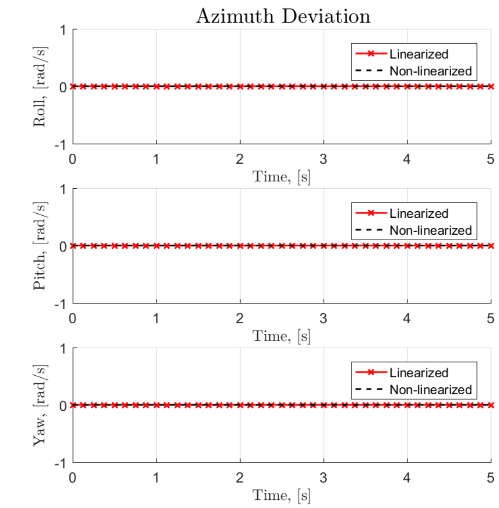


Figure 4: Body coordinate velocity as a function of time.

Part c) - 5° Azimuth Deviation



;

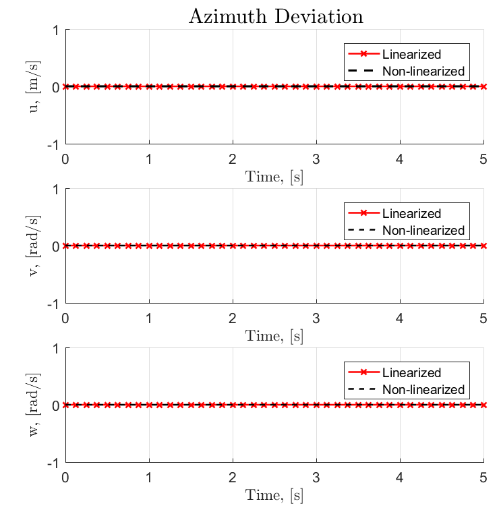
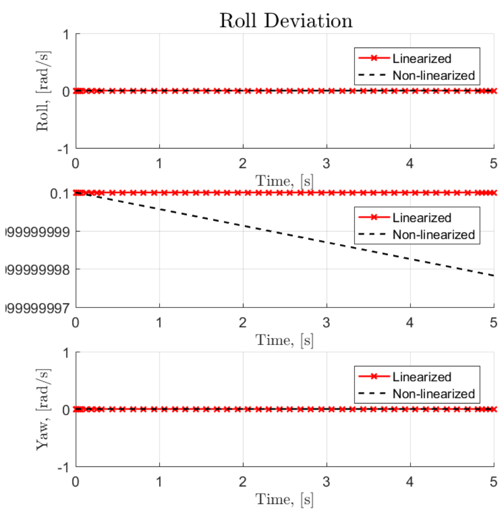


Figure 5: Body coordinate angular velocity as a function of time.

Figure 6: Body coordinate velocity as a function of time.

Part d) - 0.1 rad/s Roll Deviation



;

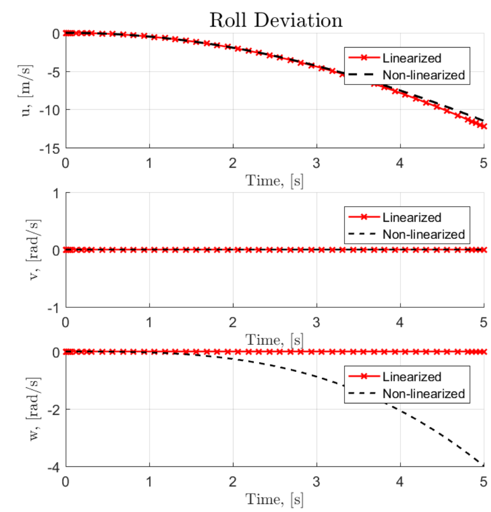
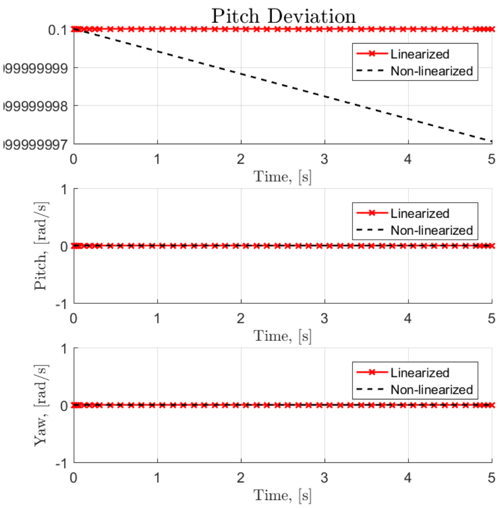


Figure 7: Body coordinate angular velocity as a function of time.

Figure 8: Body coordinate velocity as a function of time.

Part e) - 0.1 rad/s Pitch Deviation



;

Figure 9: Body coordinate angular velocity as a function of time.

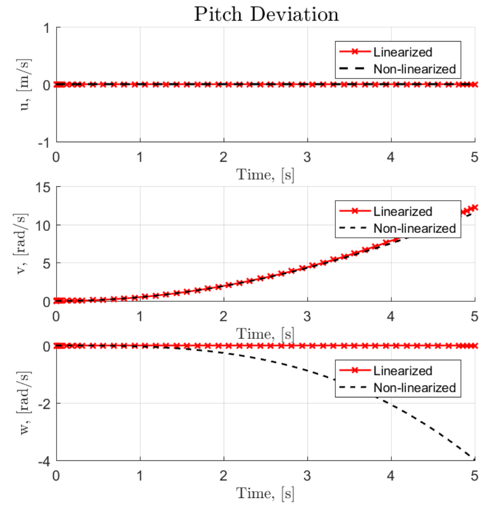
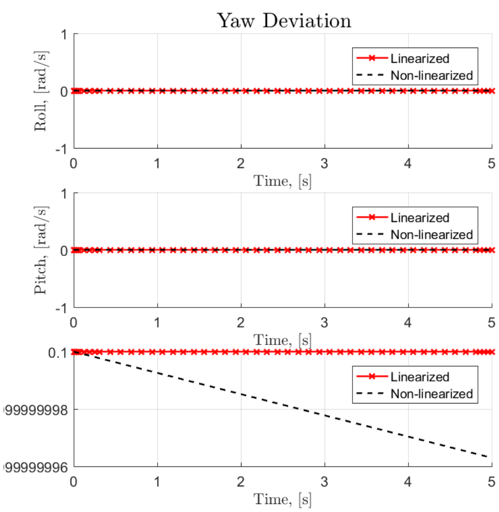


Figure 10: Body coordinate velocity as a function of time.

Part f) - 0.1 rad/s Yaw Deviation



;

Figure 11: Body coordinate angular velocity as a function of time.

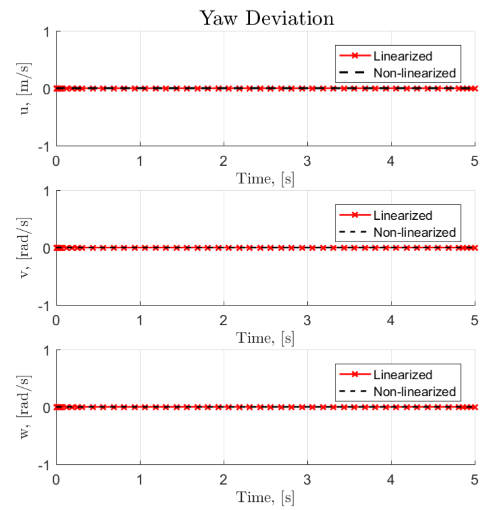


Figure 12: Body coordinate velocity as a function of time.

Problem 4

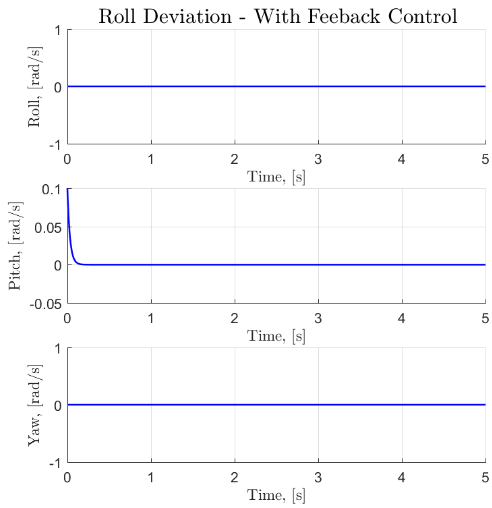
This next problem introduces control moments for the quad copter based on angular rates, roll, pitch, and yaw. The gains for these are as follows.

1. $K_p = 0.003\ [N \cdot /(rad/s)]$
2. $K_q = 0.003\ [N \cdot /(rad/s)]$
3. $K_r = 0.0012\ [N \cdot /(rad/s)]$

These gains were then implemented as follows:

1. $L_c =$ - $k_p \cdot p$
2. $M_c =$ - $k_q \cdot q$
3. $N_c =$ - $k_r \cdot r$

This was tested with the **non-linear** model by implementing deviations in roll, pitch, and yaw rate. The results of this are shown below.



;

Figure 13: Body coordinate angular velocity as a function of time.

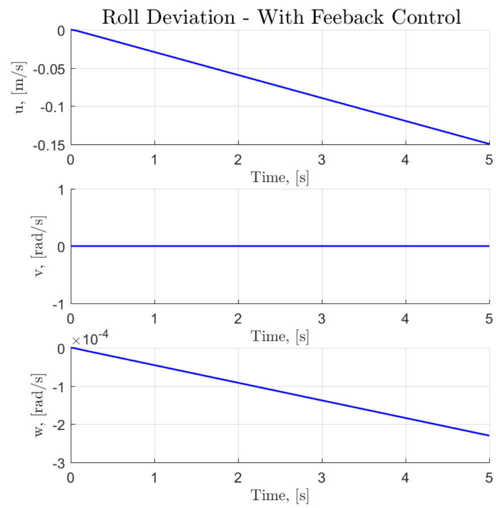
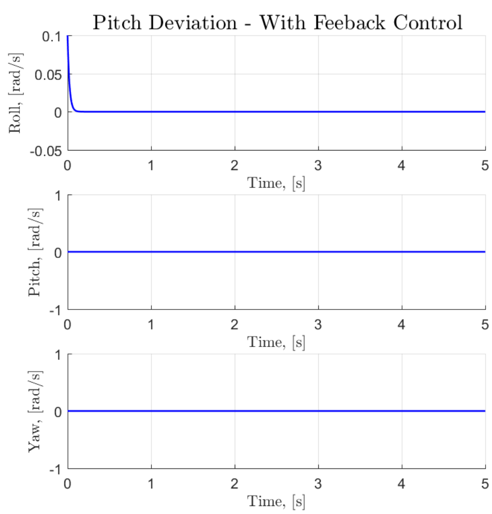


Figure 14: Body coordinate velocity as a function of time.



;

Figure 15: Body coordinate angular velocity as a function of time.

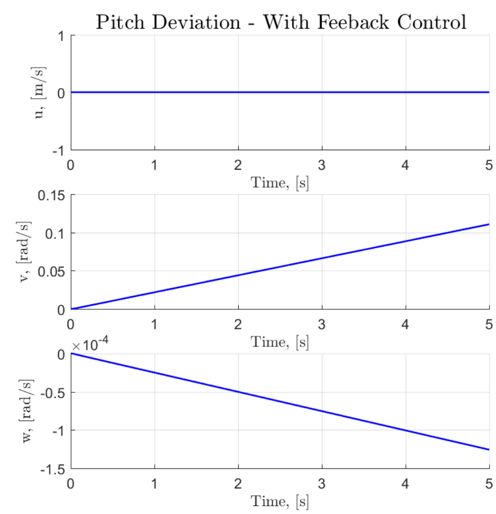
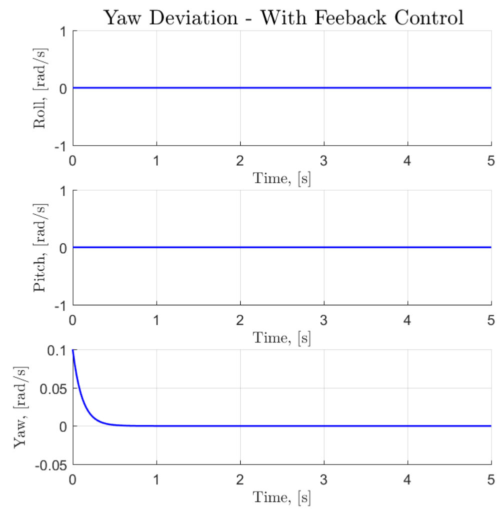
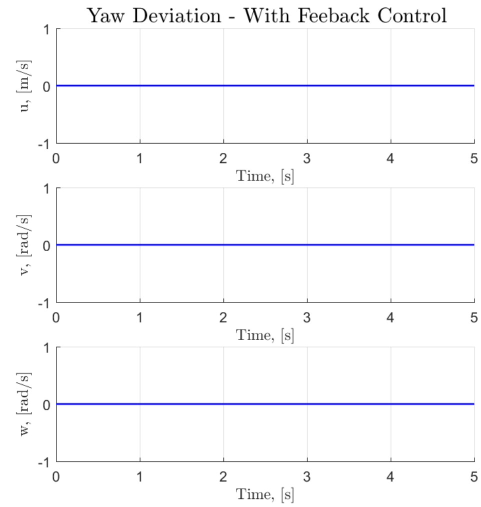


Figure 16: Body coordinate velocity as a function of time.





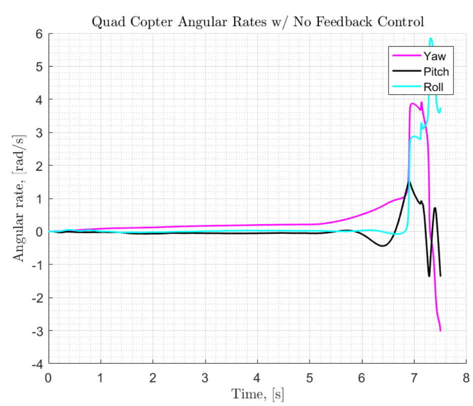
;

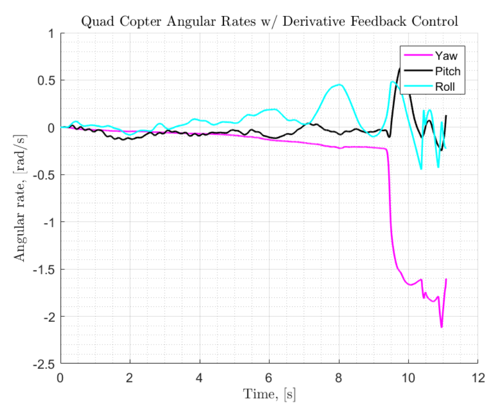
Figure 17: Body coordinate angular velocity as a function of time.

Figure 18: Body coordinate velocity as a function of time.

Problem 5

This problem looked into the difference between zero control and derivative control for the physical quad copter. The difference between the two in terms of state variables roll, pitch, and yaw can be seen in the following figures. From a more qualitative point of view, it was seen that the scenario with no control tumbled to the ground almost immediately. Conversely, the scenario with derivative feedback control drifted like a lovely little leaf for some time before entering a tumble and crashing.





;

Figure 19: Body coordinate angular velocity as a function of time.

Figure 20: Body coordinate angular velocity as a function of time.

Note here that control was removed approximately 5 seconds in to the experiment for both scenarios, and while the case with no feedback control tumbled almost immediately, the case with derivative feedback control is able to maintain some semblance of hover for approximately 3-4 seconds after moving to derivative control.

MATLAB Code

```
%------------------------------------------------------------------------
% quadCopterSim_Ass3 simulates the dynamics of a small quad copter using
% numeical integration of Euler's Moment Equations
%
% Created: 1/30/18 - Connor Ott
% Last Modified: 1/30/18 - Connor Ott
%------------------------------------------------------------------------

clc; clear; close all;
set(0, 'defaulttextinterpreter', 'latex');

%% Hover Trim
ti = 0;
tf = 5; % s
analyticalTrim = 0 * ones(1, 4);
initConds = zeros(1, 12);
initConds(12) = -1;

options = odeset('Events', @termEvents, 'RelTol', 1e-8);
[t, F] = ode45(@(t, F)quadCopterODE_lin(t, F, analyticalTrim), ...
                                        [ti, tf], initConds, options);

figure('visible', 'off')
hold on; grid on; axis equal;
plot3(F(:, 10), F(:, 11), F(:, 12), 'ro')
set(gca, 'zdir', 'reverse')
zlabel('Down Position, [m]')
xlabel('East Position, [m]')
ylabel('North Position, [m]')
title('Steady Hover Trim - Linearized')
set(gca, 'ticklabelinterpreter', 'latex', ...
         'fontsize', 12);
view(-59, 17)

%% Part 2
% a) - f) Deviations in different parameters and comparision between linear
% and nonlinear models.
ti = 0;
tf = 5; % s
linTrim = 0 * ones(1, 4);
nLTrim = ones(1, 4)*0.068*9.81 / 4;


condMat = zeros(6, 12);
condMat(:, 12) = -4; % initial height
condMat(1, 7) = deg2rad(5); % [rad]
condMat(2, 8) = deg2rad(5); % [rad]
condMat(3, 9) = deg2rad(5); % [rad]
condMat(4, 4) = 0.1; % [rad/s]
condMat(5, 5) = 0.1; % [rad/s]
condMat(6, 6) = 0.1; % [rad/s]

titleCell = {'Bank Deviation', 'Elevation Deviation', ...
```

```matlab
                'Azimuth Deviation', 'Pitch Deviation', ...
                'Roll Deviation', 'Yaw Deviation'};
options = odeset('Events', @termEvents, 'RelTol', 1e-8);

[r, ~] = size(condMat);
sSize = get(0, 'screensize');
for i = 1:r

    tspan = [0, 5]; %s
    initConds = condMat(i, :);

    [t_lin, F_lin] = ode45(@(t, F)quadCopterODE_lin(t, F, linTrim), ...
                                tspan, initConds);
    [t_nL, F_nL] = ode45(@(t, F)quadCopterODE(t, F, nLTrim), ...
                                tspan, initConds);

    %% pqr Plots
    figure('pos', [sSize(3)*0.25, sSize(4)*0.20, ...
                   sSize(3)*0.40, sSize(4)*0.70]);
    subplot(3, 1, 1)
    hold on; grid on;
    plot(t_lin, F_lin(:, 4), 'rx-', 'linewidth', 1.2)
    plot(t_nL, F_nL(:, 4), 'k--', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('Roll, [rad/s]')
    legend('Linearized', 'Non-linearized')

    subplot(3, 1, 2)
    hold on; grid on;
    plot(t_lin, F_lin(:, 5), 'rx-', 'linewidth', 1.2)
    plot(t_nL, F_nL(:, 5), 'k--', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('Pitch, [rad/s]')
    legend('Linearized', 'Non-linearized')

    subplot(3, 1, 3)
    hold on; grid on;
    plot(t_lin, F_lin(:, 6), 'rx-', 'linewidth', 1.2)
    plot(t_nL, F_nL(:, 6), 'k--', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('Yaw, [rad/s]')
    legend('Linearized', 'Non-linearized')

    [~, t] = suplabel(titleCell{i}, 't', [.1 .1 .84 .84]);
    set(t, 'fontsize', 15)
    set(gcf,'Visible','off')
%     if exist(['./Ass3figs/',titleCell{i}, '_pqr.png'], 'file') ~=2
        saveas(gcf, ['./Ass3figs/',titleCell{i}, '_pqr.png']);
%     end


    %% uvw Plots
    figure('pos', [sSize(3)*0.25, sSize(4)*0.20, ...
                   sSize(3)*0.40, sSize(4)*0.70]);
    suplabel(titleCell{i}, 't');
```

```
    subplot(3, 1, 1)
    hold on; grid on;
    plot(t_lin, F_lin(:, 1), 'rx-', 'linewidth', 1.2)
    plot(t_nL, F_nL(:, 1), 'k--', 'linewidth', 1.5)
    xlabel('Time, [s]')
    ylabel('u, [m/s]')
    legend('Linearized', 'Non-linearized')

    subplot(3, 1, 2)
    hold on; grid on;
    plot(t_lin, F_lin(:, 2), 'rx-', 'linewidth', 1.2)
    plot(t_nL, F_nL(:, 2), 'k--', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('v, [rad/s]')
    legend('Linearized', 'Non-linearized')

    subplot(3, 1, 3)
    hold on; grid on;
    plot(t_lin, F_lin(:, 3), 'rx-', 'linewidth', 1.2)
    plot(t_nL, F_nL(:, 3), 'k--', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('w, [rad/s]')
    legend('Linearized', 'Non-linearized')

    [~, t] = suplabel(titleCell{i}, 't', [.1 .1 .84 .84]);
    set(t, 'fontsize', 15)
    set(gcf,'Visible','off')
%     if exist(['./Ass3figs/',titleCell{i}, '_uvw.png'], 'file') ~=2
        saveas(gcf, ['./Ass3figs/',titleCell{i}, '_uvw.png']);
%     end
end

%% Plotting experimental quad copter data
load('RSdata_Drone01_1330.mat');
times = rt_estim.time(:);
pdata = rt_estim.signals.values(:, 4);
qdata = rt_estim.signals.values(:, 5);
rdata = rt_estim.signals.values(:, 6);

load('RSdata_Wed_1315.mat')
time_d = rt_estim.time(:);
pdata_d = rt_estim.signals.values(:, 4);
qdata_d = rt_estim.signals.values(:, 5);
rdata_d = rt_estim.signals.values(:, 6);


figure
hold on; grid on; grid minor;
plot(times, pdata, 'm-', 'linewidth', 1.3)
plot(times, qdata, 'k-', 'linewidth', 1.3)
plot(times, rdata, 'c-', 'linewidth', 1.3)
title('Quad Copter Angular Rates w/ No Feedback Control')
xlabel('Time, [s]')
ylabel('Angular rate, [rad/s]')
legend('Yaw', 'Pitch', 'Roll')
```

```

figure
hold on; grid on; grid minor;
plot(time_d, pdata_d, 'm-', 'linewidth', 1.3)
plot(time_d, qdata_d, 'k-', 'linewidth', 1.3)
plot(time_d, rdata_d, 'c-', 'linewidth', 1.3)
title('Quad Copter Angular Rates w/ Derivative Feedback Control')
xlabel('Time, [s]')
ylabel('Angular rate, [rad/s]')
legend('Yaw', 'Pitch', 'Roll')
```

```
function dfdt = quadCopterODE(t, F, trim)
% Defines dynamics of quad copter


%% Physical properties and constants
alpha = 2e-6;   % [N/(m/s)^2]
beta = 1e-6;    % [N/(m/s)^2]
heta = 1e-3;    % [N/(rad/s)^2]
xsi = 3e-3;     % [N/(rad/s)^2]

I_x = 6.8e-5;   % [kg m^2]
I_y = 9.2e-5;   % ['']
I_z = 1.35e-4;  % ['']

m = 0.068;      % [kg]
d = 0.06;       % [m]
k = 0.0024;     % [~]
rad = d/sqrt(2);  % [m]
g = 9.81;       % [m/s^2]

%% Pulling from input, F and trim
% Inertial velocity in body coords
u = F(1);
v = F(2);
w = F(3);

% Inertial angular velocity in body coords
p = F(4);
q = F(5);
r = F(6);

% Euler angles
phi = F(7);
theta = F(8);
psi = F(9);

% Position vector from origin to COM in inertial coords
x_E = F(10);
y_E = F(11);
z_e = F(12);

f1 = trim(1); % motor forces
f2 = trim(2);
f3 = trim(3);
f4 = trim(4);
```

```
%% Aerodynamic and Control Moments and Forces

% Moments
L_a = - alpha^2 * p^2 * sign(p);
M_a = - alpha^2 * q^2 * sign(q);
N_a = - beta^2 * r^2 * sign(r);

L_c = ((f2 + f3) - (f1 + f4)) * rad;
M_c = ((f3 + f4) - (f1 + f2)) * rad;
N_c = (f2 + f4 - (f1 + f2)) * k;

L = L_a + L_c;
M = M_a + M_c;
N = N_a + N_c;

% Forces
X_a = - heta^2 * u^2 * sign(u);
Y_a = - heta^2 * v^2 * sign(v);
Z_a = - xsi^2 * w^2 * sign(w);

X_c = 0;
Y_c = 0;
Z_c = -sum(trim);

X = X_a + X_c;
Y = Y_a + Y_c;
Z = Z_a + Z_c;

% Determining Roll, Pitch, and Yaw rates of change
p_dot = (I_y - I_z)/I_x * q * r + 1/I_x * L;
q_dot = (I_z - I_x)/I_y * p * r + 1/I_y * M;
r_dot = (I_x - I_y)/I_z * p * q + 1/I_z * N;
dOmega_bdt = [p_dot, q_dot, r_dot]';

% Determining translation rates
u_dot = r*v - q*w - g*sin(theta) + 1/m * X;
v_dot = p*w - r*u + g*sin(phi)*cos(theta) + 1/m * Y;
w_dot = q*u - p*v + g*cos(phi)*cos(theta) + 1/m * Z;
dV_bdt = [u_dot, v_dot, w_dot]';

% Tranlating body to inertial coordinates
x_dot =  u * cos(theta)*cos(psi) + ...
         v * (sin(phi)*sin(theta)*cos(psi) - cos(phi)*sin(psi)) + ...
         w * (cos(phi)*sin(theta)*cos(psi) + sin(phi)*sin(psi));
y_dot =  u * cos(theta)*sin(psi) + ...
         v * (sin(phi)*sin(theta)*sin(psi) + cos(phi)*cos(psi)) + ...
         w * (cos(phi)*sin(theta)*sin(psi) - sin(phi)*cos(psi));
z_dot = -u * sin(theta) + ...
         v * sin(phi)*cos(theta) + ...
         w * cos(phi)*cos(theta);
dV_Edt = [x_dot, y_dot, z_dot]';

% Translating Angular Momentum to Euler angles
phi_dot   = p + (q*sin(phi)+ r*cos(phi))*tan(theta);
```

```
theta_dot = q*cos(phi) - r*sin(phi);
psi_dot   = q*sin(phi)*sec(theta) + r*cos(phi)*sec(theta);
dEuldt = [phi_dot, theta_dot, psi_dot]';

% Concatenating for output
dfdt = [dV_bdt; dOmega_bdt; dEuldt; dV_Edt];



end
```

```
function dfdt = quadCopterODE_lin(t, F, trim)
% Defines dynamics of quad copter


%% Physical properties and constants
alpha = 2e-6;    % [N/(m/s)^2]
beta = 1e-6;     % [N/(m/s)^2]
heta = 1e-3;     % [N/(rad/s)^2]
xsi = 3e-3;      % [N/(rad/s)^2]

I_x = 6.8e-5;    % [kg m^2]
I_y = 9.2e-5;    % ['']
I_z = 1.35e-4;   % ['']

m = 0.068;       % [kg]
d = 0.06;        % [m]
k = 0.0024;      % [~]
rad = d/sqrt(2); % [m]
g = 9.81;        % [m/s^2]

%% Pulling from input, F and trim
% Inertial velocity in body coords
delta_u = F(1);
delta_v = F(2);
delta_w = F(3);

% Inertial angular velocity in body coords
delta_p = F(4);
delta_q = F(5);
delta_r = F(6);

% Euler angles
delta_phi = F(7);
delta_theta = F(8);
delta_psi = F(9);

% Position vector from origin to COM in inertial coords
x_E = F(10);
y_E = F(11);
z_e = F(12);

delta_f1 = trim(1); % motor forces
delta_f2 = trim(2);
delta_f3 = trim(3);
delta_f4 = trim(4);
```

```
%% Aerodynamic and Control Moments and Forces

% Moments
L_a = 0; %- alpha^2 * p^2 * sign(p);
M_a = 0; %- alpha^2 * q^2 * sign(q);
N_a = 0; %- beta^2 * r^2 * sign(r);

L_c = ((delta_f2 + delta_f3) - (delta_f1 + delta_f4)) * rad;
M_c = ((delta_f3 + delta_f4) - (delta_f1 + delta_f2)) * rad;
N_c = (delta_f2 + delta_f4 - (delta_f1 + delta_f2)) * k;

L = L_a + L_c;
M = M_a + M_c;
N = N_a + N_c;

% if t > 1 && t < 1.5 % Throw a moment perturbation in for 0.2 seconds
%     L = L + momPert(1);
%     M = M + momPert(2);
%     N = N + momPert(3);
% end

% Forces
X_a = 0; %- heta^2 * delta_u^2 * sign(delta_u);
Y_a = 0; %- heta^2 * delta_v^2 * sign(delta_v);
Z_a = 0; %- xsi^2 * delta_w^2 * sign(delta_w);

X_c = 0;
Y_c = 0;
Z_c = -sum(trim);

X = X_a + X_c;
Y = Y_a + Y_c;
Z = Z_a + Z_c;

% Determining Roll, Pitch, and Yaw rates of change
deltap_dot = rad/I_x * (delta_f2 + delta_f3 - delta_f1 - delta_f4);
deltaq_dot = rad/I_y * (delta_f3 + delta_f4 - delta_f1 - delta_f2);
deltar_dot = k/I_z * (delta_f2 + delta_f4 - delta_f4 - delta_f3);
dOmega_bdt = [deltap_dot, deltaq_dot, deltar_dot]';

% Determining translation rates
deltau_dot = -g * delta_theta;
deltav_dot = g * delta_phi;
deltaw_dot = 1/m * Z;
dV_bdt = [deltau_dot, deltav_dot, deltaw_dot]';

% Tranlating body to inertial coordinates
x_dot =  delta_u * cos(delta_theta)*cos(delta_psi) + ...
         delta_v * (sin(delta_phi)*sin(delta_theta)*cos(delta_psi) - cos(
            delta_phi)*sin(delta_psi)) + ...
         delta_w * (cos(delta_phi)*sin(delta_theta)*cos(delta_psi) + sin(
            delta_phi)*sin(delta_psi));
y_dot =  delta_u * cos(delta_theta)*sin(delta_psi) + ...
         delta_v * (sin(delta_phi)*sin(delta_theta)*sin(delta_psi) + cos(
```

```
              delta_phi)*cos(delta_psi)) + ...
          delta_w * (cos(delta_phi)*sin(delta_theta)*sin(delta_psi) - sin(
              delta_phi)*cos(delta_psi));
z_dot = -delta_u * sin(delta_theta) + ...
          delta_v * sin(delta_phi)*cos(delta_theta) + ...
          delta_w * cos(delta_phi)*cos(delta_theta);
dV_Edt = [x_dot, y_dot, z_dot]';

% Translating Angular Momentum to Euler angles
deltaphi_dot   = delta_p;
deltatheta_dot = delta_q;
deltapsi_dot   = delta_r;
dEuldt = [deltaphi_dot, deltatheta_dot, deltapsi_dot]';

% Concatenating for output
dfdt = [dV_bdt; dOmega_bdt; dEuldt; dV_Edt];



end
```

```
%-------------------------------------------------------------------------
% quadCopterSim_FBC simulates the dynamics of a quad copter with
% derivative feedback control.
%
% Dependancies:
%       quadCopterODE_FBC - ODE function defining dynamics of quad copter.
%
% Created: 2/10/18 - Connor Ott
% Last Modified: 2/10/18 - Connor Ott
%-------------------------------------------------------------------------
%% Introducing Feedback control

sSize = get(0, 'screensize');
condMat = zeros(3, 12);
condMat(:, 12) = -4; % initial height
condMat(1, 4) = 0.1; % [rad/s]
condMat(2, 5) = 0.1; % [rad/s]
condMat(3, 6) = 0.1; % [rad/s]

titleCell = {'Pitch Deviation', 'Roll Deviation', 'Yaw Deviation'};
tspan = [0, 5]; %s
fbcTrim = ones(1, 4)*0.068*9.81 / 4;
for i = 1:length(titleCell)

    initConds = condMat(i, :);
    [t_FBC, F_FBC] = ode45(@(t, F)quadCopterODE_FBC(t, F, fbcTrim), ...
                                                        tspan, initConds);

    %% pqr Plots
    figure('pos', [sSize(3)*0.25, sSize(4)*0.15, ...
                   sSize(3)*0.40, sSize(4)*0.70]);
    subplot(3, 1, 1)
    hold on; grid on;
    plot(t_FBC, F_FBC(:, 4), 'b-', 'linewidth', 1.2)
    xlabel('Time, [s]')
```

```matlab
    ylabel('Roll, [rad/s]')

    subplot(3, 1, 2)
    hold on; grid on;
    plot(t_FBC, F_FBC(:, 5), 'b-', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('Pitch, [rad/s]')

    subplot(3, 1, 3)
    hold on; grid on;
    plot(t_FBC, F_FBC(:, 6), 'b-', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('Yaw, [rad/s]')

    [~, t] = suplabel([titleCell{i}, ' - With Feeback Control'],...
                        't', [.1 .1 .84 .84]);
    set(t, 'fontsize', 15)
    set(gcf,'Visible','off')
    if exist(['./Ass3figs_fbc/',titleCell{i}, '_pqrFBC.png'], 'file') ~=2
        saveas(gcf, ['./Ass3figs_fbc/',titleCell{i}, '_pqrFBC.png']);
    end


    %% uvw Plots
    figure('pos', [sSize(3)*0.25, sSize(4)*0.15, ...
                   sSize(3)*0.40, sSize(4)*0.70]);
    suplabel(titleCell{i}, 't');
    subplot(3, 1, 1)
    hold on; grid on;
    plot(t_FBC, F_FBC(:, 1), 'b-', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('u, [m/s]')

    subplot(3, 1, 2)
    hold on; grid on;
    plot(t_FBC, F_FBC(:, 2), 'b-', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('v, [rad/s]')

    subplot(3, 1, 3)
    hold on; grid on;
    plot(t_FBC, F_FBC(:, 3), 'b-', 'linewidth', 1.2)
    xlabel('Time, [s]')
    ylabel('w, [rad/s]')

    [~, t] = suplabel([titleCell{i}, ' - With Feeback Control'], ...
                                    't', [.1 .1 .84 .84]);
    set(t, 'fontsize', 15)
    set(gcf,'Visible','off')
    if exist(['./Ass3figs_fbc/',titleCell{i}, '_uvwFBC.png'], 'file') ~=2
        saveas(gcf, ['./Ass3figs_fbc/',titleCell{i}, '_uvwFBC.png']);
    end
end
```

```matlab
function dfdt = quadCopterODE_FBC(t, F, trim)
% Defines dynamics of quad copter
```

```
%% Physical properties and constants
alpha = 2e-6;     % [N/(m/s)^2]
beta = 1e-6;      % [N/(m/s)^2]
heta = 1e-3;      % [N/(rad/s)^2]
xsi = 3e-3;       % [N/(rad/s)^2]

I_x = 6.8e-5;     % [kg m^2]
I_y = 9.2e-5;     % ['']
I_z = 1.35e-4;    % ['']

m = 0.068;        % [kg]
d = 0.06;         % [m]
k = 0.0024;       % [~]
rad = d/sqrt(2);  % [m]
g = 9.81;         % [m/s^2]

% Gains
k_p = 0.003;      % [Nm/(rad/s)]
k_q = 0.003;      % [Nm/(rad/s)]
k_r = 0.0012;     % [Nm/(rad/s)]

%% Pulling from input, F and trim
% Inertial velocity in body coords
u = F(1);
v = F(2);
w = F(3);

% Inertial angular velocity in body coords
p = F(4);
q = F(5);
r = F(6);

% Euler angles
phi = F(7);
theta = F(8);
psi = F(9);

% Position vector from origin to COM in inertial coords
x_E = F(10);
y_E = F(11);
z_e = F(12);

f1 = trim(1); % motor forces
f2 = trim(2);
f3 = trim(3);
f4 = trim(4);

%% Aerodynamic and Control Moments and Forces

% Moments
L_a = - alpha^2 * p^2 * sign(p);
M_a = - alpha^2 * q^2 * sign(q);
N_a = - beta^2 * r^2 * sign(r);

```

```
L_c = -k_p * p;
M_c = -k_q * q;
N_c = -k_r * r;

L = L_a + L_c;
M = M_a + M_c;
N = N_a + N_c;

% Forces
X_a = - heta^2 * u^2 * sign(u);
Y_a = - heta^2 * v^2 * sign(v);
Z_a = - xsi^2 * w^2 * sign(w);

X_c = 0;
Y_c = 0;
Z_c = -sum(trim);

X = X_a + X_c;
Y = Y_a + Y_c;
Z = Z_a + Z_c;

% Determining Roll, Pitch, and Yaw rates of change
p_dot = (I_y - I_z)/I_x * q * r + 1/I_x * L;
q_dot = (I_z - I_x)/I_y * p * r + 1/I_y * M;
r_dot = (I_x - I_y)/I_z * p * q + 1/I_z * N;
dOmega_bdt = [p_dot, q_dot, r_dot]';

% Determining translation rates
u_dot = r*v - q*w - g*sin(theta) + 1/m * X;
v_dot = p*w - r*u + g*sin(phi)*cos(theta) + 1/m * Y;
w_dot = q*u - p*v + g*cos(phi)*cos(theta) + 1/m * Z;
dV_bdt = [u_dot, v_dot, w_dot]';

% Tranlating body to inertial coordinates
x_dot =  u * cos(theta)*cos(psi) + ...
         v * (sin(phi)*sin(theta)*cos(psi) - cos(phi)*sin(psi)) + ...
         w * (cos(phi)*sin(theta)*cos(psi) + sin(phi)*sin(psi));
y_dot =  u * cos(theta)*sin(psi) + ...
         v * (sin(phi)*sin(theta)*sin(psi) + cos(phi)*cos(psi)) + ...
         w * (cos(phi)*sin(theta)*sin(psi) - sin(phi)*cos(psi));
z_dot = -u * sin(theta) + ...
         v * sin(phi)*cos(theta) + ...
         w * cos(phi)*cos(theta);
dV_Edt = [x_dot, y_dot, z_dot]';

% Translating Angular Momentum to Euler angles
phi_dot   = p + (q*sin(phi)+ r*cos(phi))*tan(theta);
theta_dot = q*cos(phi) - r*sin(phi);
psi_dot   = q*sin(phi)*sec(theta) + r*cos(phi)*sec(theta);
dEuldt = [phi_dot, theta_dot, psi_dot]';

% Concatenating for output
dfdt = [dV_bdt; dOmega_bdt; dEuldt; dV_Edt];

end
```

```matlab
function [value, isTerm, direction] = termEvents(t, F)
% Define events for quad copter ODE
value = [F(12), F(12) + 100]';
isTerm = [1,1]';
direction = [];
end
```