

ASEN 3128 - Assignment 1

Seth Hill
Gregory Kondor
Connor Ott

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

Linearized Models

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

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

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

$$\Delta \dot{\psi}^E = -g \Delta \Theta$$

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

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

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

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

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

Problem 2 - 3

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

Part a) - 5° Bank Deviation

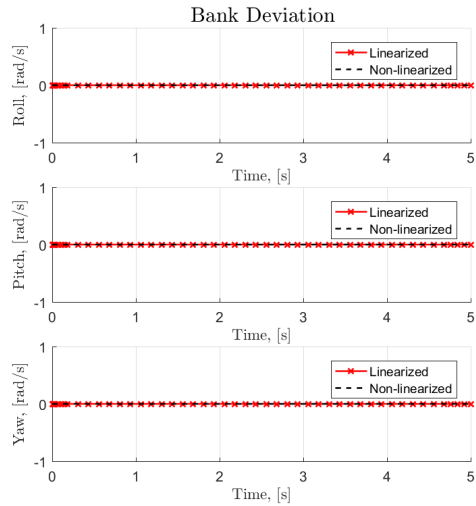


Figure 1: Body coordinate angular velocity as a function of time.

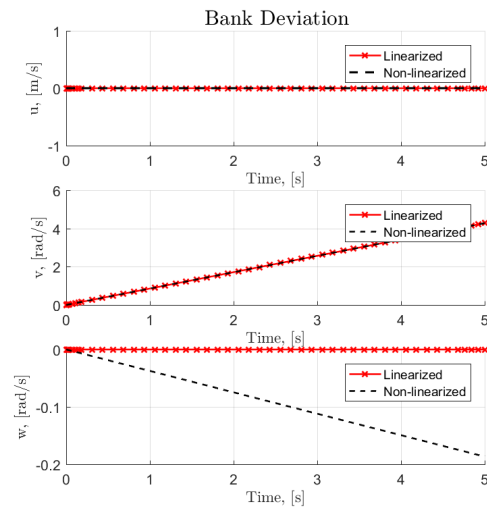


Figure 2: Body coordinate velocity as a function of time.

Part b) - 5° Elevation Deviation

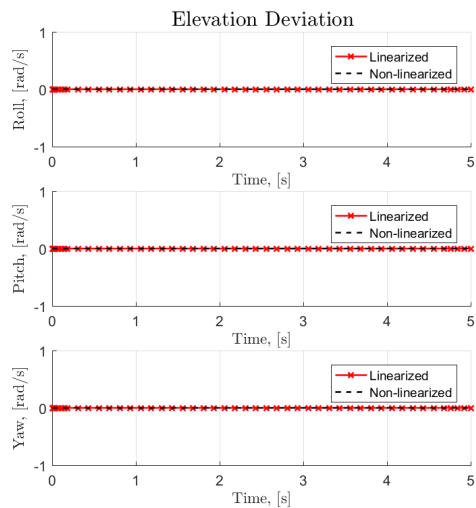


Figure 3: Body coordinate angular velocity as a function of time.

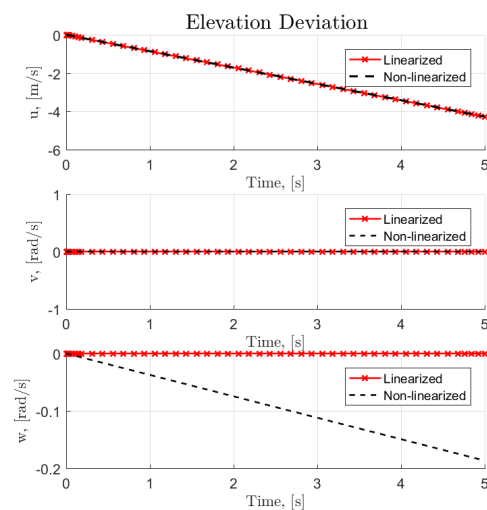


Figure 4: Body coordinate velocity as a function of time.

Part c) - 5° Azimuth Deviation

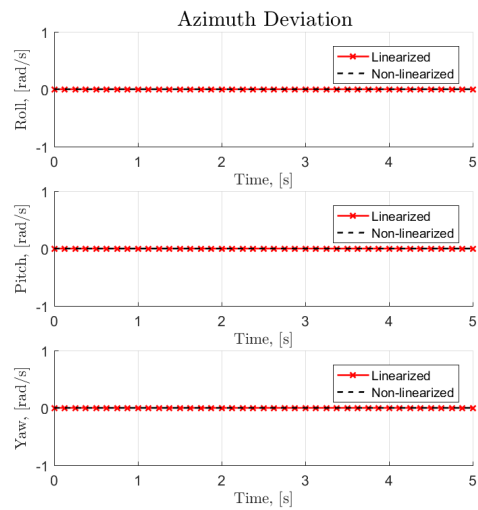


Figure 5: Body coordinate angular velocity as a function of time.

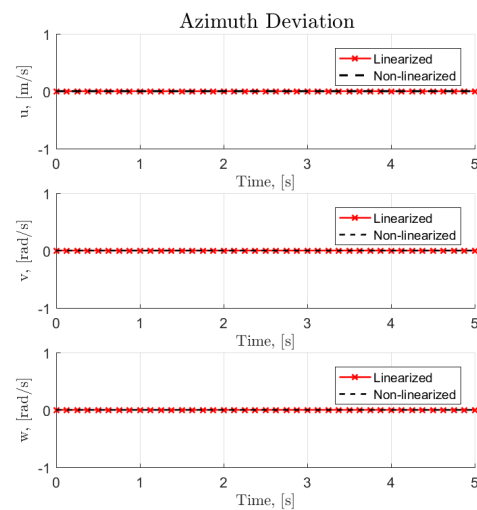


Figure 6: Body coordinate velocity as a function of time.

Part d) - 0.1 rad/s Roll Deviation

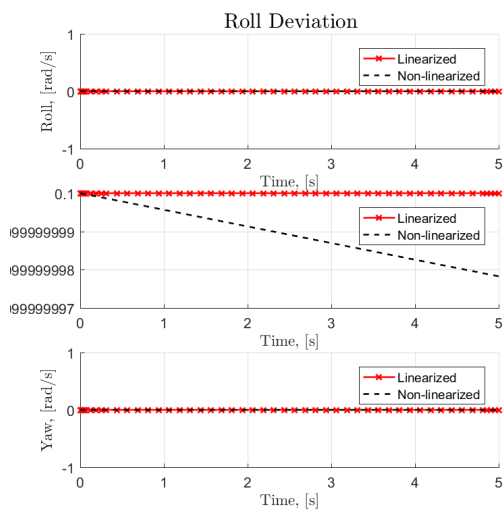


Figure 7: Body coordinate angular velocity as a function of time.

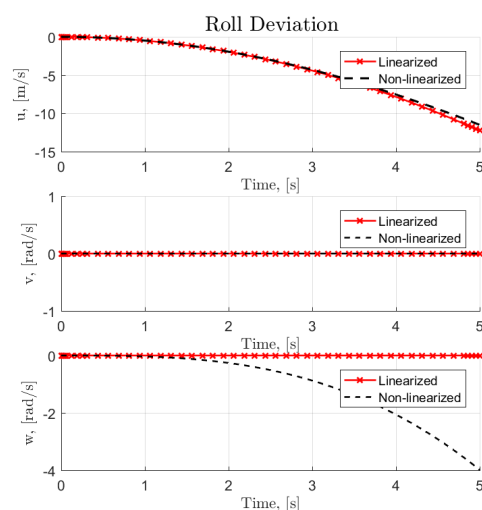


Figure 8: Body coordinate velocity as a function of time.

Part e) - 0.1 rad/s Pitch Deviation

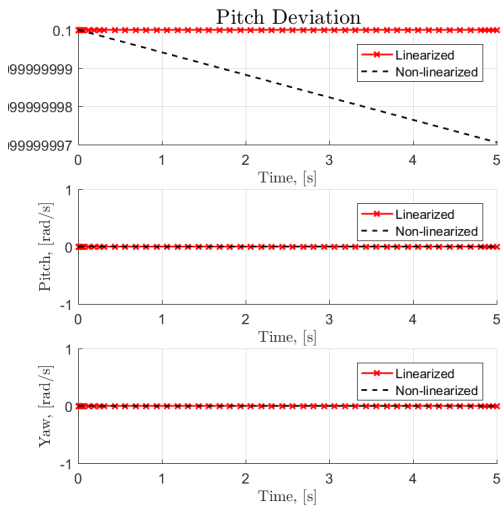


Figure 9: Body coordinate angular velocity as a function of time.

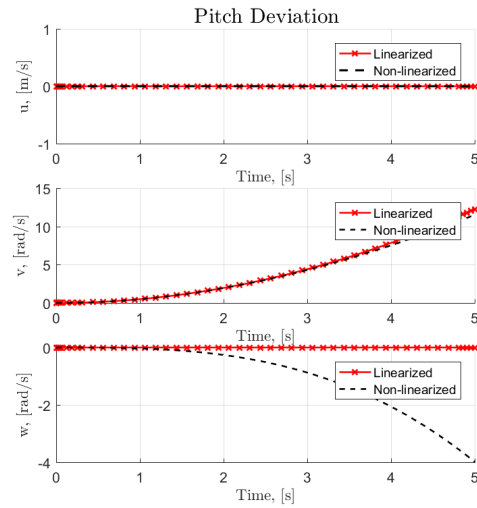


Figure 10: Body coordinate velocity as a function of time.

Part f) - 0.1 rad/s Yaw Deviation

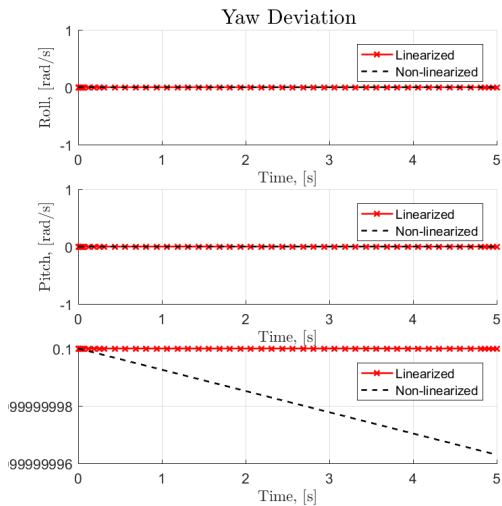


Figure 11: Body coordinate angular velocity as a function of time.

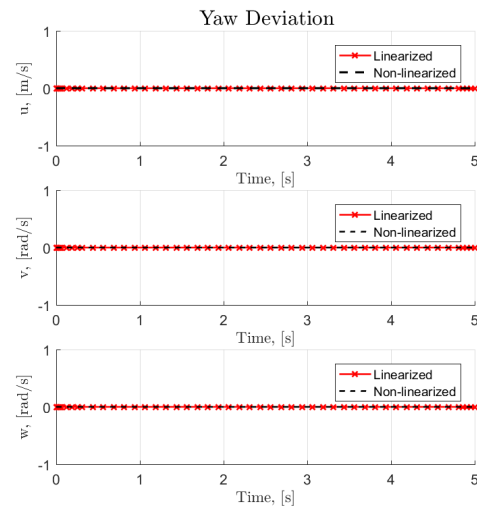


Figure 12: Body coordinate velocity as a function of time.

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.

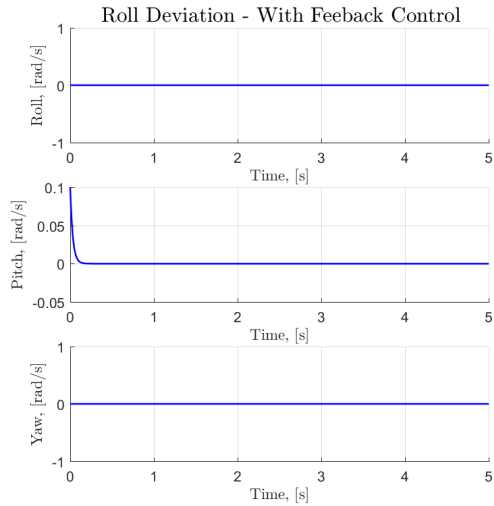


Figure 13: Body coordinate angular velocity as a function of time.

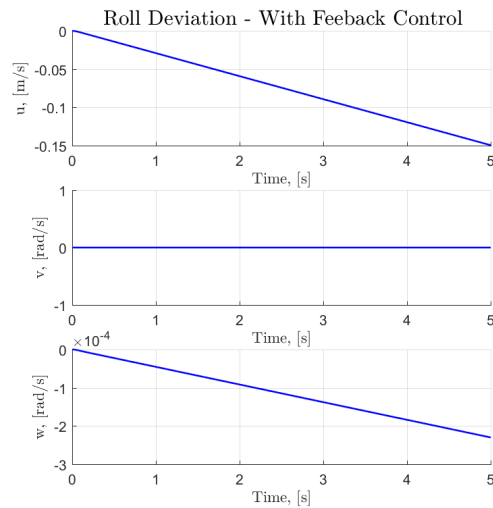


Figure 14: Body coordinate velocity as a function of time.

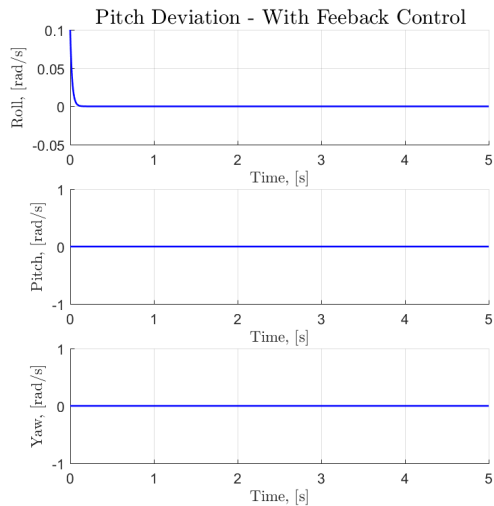


Figure 15: Body coordinate angular velocity as a function of time.

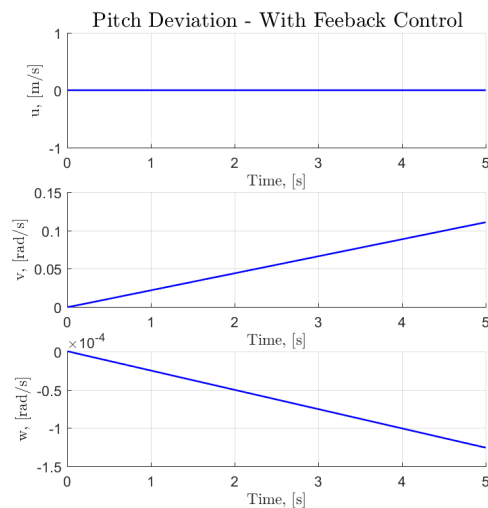


Figure 16: Body coordinate velocity as a function of time.

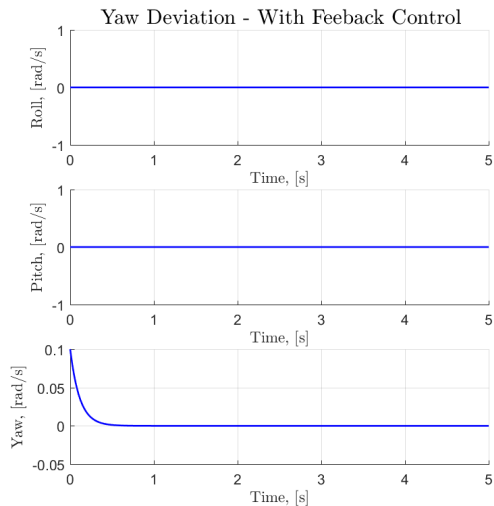


Figure 17: Body coordinate angular velocity as a function of time.

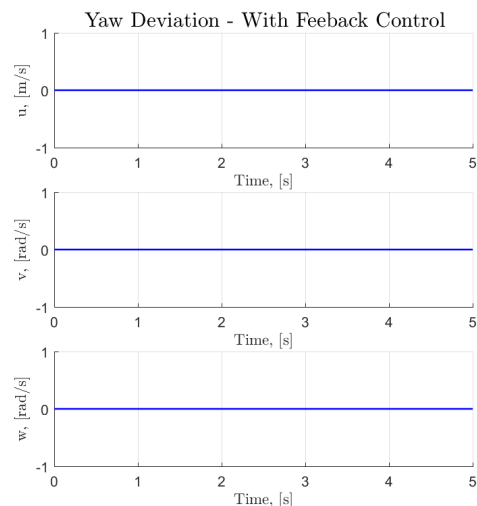


Figure 18: Body coordinate velocity as a function of time.

Problem 5

This problem looked into the difference between zero control and derivative control for the physical quadcopter. 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.

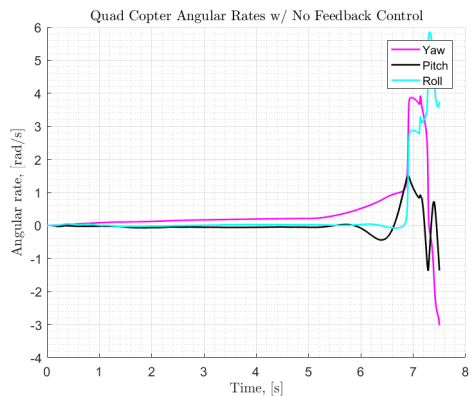


Figure 19: Body coordinate angular velocity as a function of time.

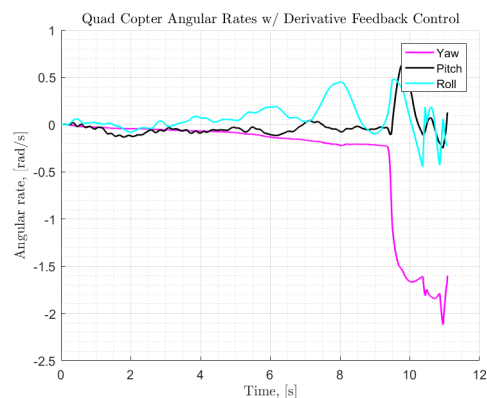


Figure 20: Body coordinate angular velocity as a function of time.

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

```
1 %-----
2 % quadCopterSim_Ass3 simulates the dynamics of a small quad copter using
3 % numerical integration of Euler's Moment Equations
4 %
5 % Created: 1/30/18 - Connor Ott
6 % Last Modified: 1/30/18 - Connor Ott
7 %-----
8
9 clc; clear; close all;
10 set(0, 'defaulttextinterpreter', 'latex');
11
12 %% Hover Trim
13 ti = 0;
14 tf = 5; % s
15 analyticalTrim = 0 * ones(1, 4);
16 initConds = zeros(1, 12);
17 initConds(12) = -1;
18
19 options = odeset('Events', @termEvents, 'RelTol', 1e-8);
20 [t, F] = ode45(@(t, F)quadCopterODE_lin(t, F, analyticalTrim), ...
21               [ti, tf], initConds, options);
22
23 figure('visible', 'off')
24 hold on; grid on; axis equal;
25 plot3(F(:, 10), F(:, 11), F(:, 12), 'ro')
26 set(gca, 'zdir', 'reverse')
27 zlabel('Down Position, [m]')
28 xlabel('East Position, [m]')
29 ylabel('North Position, [m]')
30 title('Steady Hover Trim - Linearized')
31 set(gca, 'ticklabelinterpreter', 'latex', ...
32         'fontsize', 12);
33 view(-59, 17)
34
35 %% Part 2
36 % a) - f) Deviations in different parameters and comparison between linear
37 % and nonlinear models.
38 ti = 0;
39 tf = 5; % s
40 linTrim = 0 * ones(1, 4);
41 nLTrim = ones(1, 4)*0.068*9.81 / 4;
42
43
44 condMat = zeros(6, 12);
45 condMat(:, 12) = -4; % initial height
46 condMat(1, 7) = deg2rad(5); % [rad]
47 condMat(2, 8) = deg2rad(5); % [rad]
48 condMat(3, 9) = deg2rad(5); % [rad]
49 condMat(4, 4) = 0.1; % [rad/s]
50 condMat(5, 5) = 0.1; % [rad/s]
51 condMat(6, 6) = 0.1; % [rad/s]
52
53 titleCell = {'Bank Deviation', 'Elevation Deviation', ...
```



```

54         'Azimuth Deviation', 'Pitch Deviation', ...
55         'Roll Deviation', 'Yaw Deviation'};
56 options = odeset('Events', @termEvents, 'RelTol', 1e-8);
57
58 [r, ~] = size(condMat);
59 sSize = get(0, 'screensize');
60 for i = 1:r
61
62     tspan = [0, 5]; %s
63     initConds = condMat(i, :);
64
65     [t_lin, F_lin] = ode45(@(t, F)quadCopterODE_lin(t, F, linTrim), ...
66                           tspan, initConds);
67     [t_nL, F_nL] = ode45(@(t, F)quadCopterODE(t, F, nLTrim), ...
68                           tspan, initConds);
69
70 %% pqr Plots
71 figure('pos', [sSize(3)*0.25, sSize(4)*0.20, ...
72               sSize(3)*0.40, sSize(4)*0.70]);
73 subplot(3, 1, 1)
74 hold on; grid on;
75 plot(t_lin, F_lin(:, 4), 'rx-', 'linewidth', 1.2)
76 plot(t_nL, F_nL(:, 4), 'k—', 'linewidth', 1.2)
77 xlabel('Time, [s]')
78 ylabel('Roll, [rad/s]')
79 legend('Linearized', 'Non-linearized')
80
81 subplot(3, 1, 2)
82 hold on; grid on;
83 plot(t_lin, F_lin(:, 5), 'rx-', 'linewidth', 1.2)
84 plot(t_nL, F_nL(:, 5), 'k—', 'linewidth', 1.2)
85 xlabel('Time, [s]')
86 ylabel('Pitch, [rad/s]')
87 legend('Linearized', 'Non-linearized')
88
89 subplot(3, 1, 3)
90 hold on; grid on;
91 plot(t_lin, F_lin(:, 6), 'rx-', 'linewidth', 1.2)
92 plot(t_nL, F_nL(:, 6), 'k—', 'linewidth', 1.2)
93 xlabel('Time, [s]')
94 ylabel('Yaw, [rad/s]')
95 legend('Linearized', 'Non-linearized')
96
97 [~, t] = suplabel(titleCell{i}, 't', [.1 .1 .84 .84]);
98 set(t, 'fontsize', 15)
99 set(gcf, 'Visible', 'off')
100 %     if exist(['./ Ass3figs/', titleCell{i}, '_pqr.png'], 'file') ~=2
101 %         saveas(gcf, ['./ Ass3figs/', titleCell{i}, '_pqr.png']);
102 %     end
103
104
105 %% uvw Plots
106 figure('pos', [sSize(3)*0.25, sSize(4)*0.20, ...
107               sSize(3)*0.40, sSize(4)*0.70]);
108 suplabel(titleCell{i}, 't');

```

```

109     subplot(3, 1, 1)
110     hold on; grid on;
111     plot(t_lin, F_lin(:, 1), 'rx-', 'linewidth', 1.2)
112     plot(t_nL, F_nL(:, 1), 'k—', 'linewidth', 1.5)
113     xlabel('Time, [s]')
114     ylabel('u, [m/s]')
115     legend('Linearized', 'Non-linearized')
116
117     subplot(3, 1, 2)
118     hold on; grid on;
119     plot(t_lin, F_lin(:, 2), 'rx-', 'linewidth', 1.2)
120     plot(t_nL, F_nL(:, 2), 'k—', 'linewidth', 1.2)
121     xlabel('Time, [s]')
122     ylabel('v, [rad/s]')
123     legend('Linearized', 'Non-linearized')
124
125     subplot(3, 1, 3)
126     hold on; grid on;
127     plot(t_lin, F_lin(:, 3), 'rx-', 'linewidth', 1.2)
128     plot(t_nL, F_nL(:, 3), 'k—', 'linewidth', 1.2)
129     xlabel('Time, [s]')
130     ylabel('w, [rad/s]')
131     legend('Linearized', 'Non-linearized')
132
133     [~, t] = suplabel(titleCell{i}, 't', [.1 .1 .84 .84]);
134     set(t, 'fontsize', 15)
135     set(gcf, 'Visible', 'off')
136 %     if exist(['./ Ass3figs/', titleCell{i}, '_uvw.png'], 'file') ~=2
137 %         saveas(gcf, ['./ Ass3figs/', titleCell{i}, '_uvw.png']);
138 %     end
139 end
140
141 %% Plotting experimental quad copter data
142 load('RSdata_Drone01_1330.mat');
143 times = rt_estim.time(:);
144 pdata = rt_estim.signals.values(:, 4);
145 qdata = rt_estim.signals.values(:, 5);
146 rdata = rt_estim.signals.values(:, 6);
147
148 load('RSdata_Wed_1315.mat')
149 time_d = rt_estim.time(:);
150 pdata_d = rt_estim.signals.values(:, 4);
151 qdata_d = rt_estim.signals.values(:, 5);
152 rdata_d = rt_estim.signals.values(:, 6);
153
154
155 figure
156 hold on; grid on; grid minor;
157 plot(times, pdata, 'm-', 'linewidth', 1.3)
158 plot(times, qdata, 'k-', 'linewidth', 1.3)
159 plot(times, rdata, 'c-', 'linewidth', 1.3)
160 title('Quad Copter Angular Rates w/ No Feedback Control')
161 xlabel('Time, [s]')
162 ylabel('Angular rate, [rad/s]')
163 legend('Yaw', 'Pitch', 'Roll')

```

```

164
165 figure
166 hold on; grid on; grid minor;
167 plot(time_d, pdata_d, 'm-', 'linewidth', 1.3)
168 plot(time_d, qdata_d, 'k-', 'linewidth', 1.3)
169 plot(time_d, rdata_d, 'c-', 'linewidth', 1.3)
170 title('Quad Copter Angular Rates w/ Derivative Feedback Control')
171 xlabel('Time, [s]')
172 ylabel('Angular rate, [rad/s]')
173 legend('Yaw', 'Pitch', 'Roll')

1 function dfdt = quadCopterODE(t, F, trim)
2 % Defines dynamics of quad copter
3
4
5 %% Physical properties and constants
6 alpha = 2e-6; % [N/(m/s)^2]
7 beta = 1e-6; % [N/(m/s)^2]
8 heta = 1e-3; % [N/(rad/s)^2]
9 xsi = 3e-3; % [N/(rad/s)^2]
10
11 I_x = 6.8e-5; % [kg m^2]
12 I_y = 9.2e-5; % ['']
13 I_z = 1.35e-4; % ['']
14
15 m = 0.068; % [kg]
16 d = 0.06; % [m]
17 k = 0.0024; % [~]
18 rad = d/sqrt(2); % [m]
19 g = 9.81; % [m/s^2]
20
21 %% Pulling from input, F and trim
22 % Inertial velocity in body coords
23 u = F(1);
24 v = F(2);
25 w = F(3);
26
27 % Inertial angular velocity in body coords
28 p = F(4);
29 q = F(5);
30 r = F(6);
31
32 % Euler angles
33 phi = F(7);
34 theta = F(8);
35 psi = F(9);
36
37 % Position vector from origin to COM in inertial coords
38 x_E = F(10);
39 y_E = F(11);
40 z_e = F(12);
41
42 f1 = trim(1); % motor forces
43 f2 = trim(2);
44 f3 = trim(3);
45 f4 = trim(4);

```

```

46
47 %% Aerodynamic and Control Moments and Forces
48
49 % Moments
50 L_a = - alpha^2 * p^2 * sign(p);
51 M_a = - alpha^2 * q^2 * sign(q);
52 N_a = - beta^2 * r^2 * sign(r);
53
54 L_c = ((f2 + f3) - (f1 + f4)) * rad;
55 M_c = ((f3 + f4) - (f1 + f2)) * rad;
56 N_c = (f2 + f4 - (f1 + f2)) * k;
57
58 L = L_a + L_c;
59 M = M_a + M_c;
60 N = N_a + N_c;
61
62 % Forces
63 X_a = - heta^2 * u^2 * sign(u);
64 Y_a = - heta^2 * v^2 * sign(v);
65 Z_a = - xsi^2 * w^2 * sign(w);
66
67 X_c = 0;
68 Y_c = 0;
69 Z_c = -sum(trim);
70
71 X = X_a + X_c;
72 Y = Y_a + Y_c;
73 Z = Z_a + Z_c;
74
75 % Determining Roll, Pitch, and Yaw rates of change
76 p_dot = (I_y - I_z)/I_x * q * r + 1/I_x * L;
77 q_dot = (I_z - I_x)/I_y * p * r + 1/I_y * M;
78 r_dot = (I_x - I_y)/I_z * p * q + 1/I_z * N;
79 dOmega_bdt = [p_dot, q_dot, r_dot]';
80
81 % Determining translation rates
82 u_dot = r*v - q*w - g*sin(theta) + 1/m * X;
83 v_dot = p*w - r*u + g*sin(phi)*cos(theta) + 1/m * Y;
84 w_dot = q*u - p*v + g*cos(phi)*cos(theta) + 1/m * Z;
85 dV_bdt = [u_dot, v_dot, w_dot]';
86
87 % Tranlating body to inertial coordinates
88 x_dot = u * cos(theta)*cos(psi) + ...
89         v * (sin(phi)*sin(theta)*cos(psi) - cos(phi)*sin(psi)) + ...
90         w * (cos(phi)*sin(theta)*cos(psi) + sin(phi)*sin(psi));
91 y_dot = u * cos(theta)*sin(psi) + ...
92         v * (sin(phi)*sin(theta)*sin(psi) + cos(phi)*cos(psi)) + ...
93         w * (cos(phi)*sin(theta)*sin(psi) - sin(phi)*cos(psi));
94 z_dot = -u * sin(theta) + ...
95         v * sin(phi)*cos(theta) + ...
96         w * cos(phi)*cos(theta);
97 dV_Edt = [x_dot, y_dot, z_dot]';
98
99 % Translating Angular Momentum to Euler angles
100 phi_dot = p + (q*sin(phi) + r*cos(phi))*tan(theta);

```

```

101 theta_dot = q*cos(phi) - r*sin(phi);
102 psi_dot    = q*sin(phi)*sec(theta) + r*cos(phi)*sec(theta);
103 dEuldt = [phi_dot, theta_dot, psi_dot]';
104
105 % Concatenating for output
106 dfdt = [dV_bdt; dOmega_bdt; dEuldt; dV_Edt];
107
108
109
110 end

1 function dfdt = quadCopterODE_lin(t, F, trim)
2 % Defines dynamics of quad copter
3
4
5 %% Physical properties and constants
6 alpha = 2e-6; % [N/(m/s)^2]
7 beta = 1e-6; % [N/(m/s)^2]
8 heta = 1e-3; % [N/(rad/s)^2]
9 xsi = 3e-3; % [N/(rad/s)^2]
10
11 I_x = 6.8e-5; % [kg m^2]
12 I_y = 9.2e-5; % ['']
13 I_z = 1.35e-4; % ['']
14
15 m = 0.068; % [kg]
16 d = 0.06; % [m]
17 k = 0.0024; % [~]
18 rad = d/sqrt(2); % [m]
19 g = 9.81; % [m/s^2]
20
21 %% Pulling from input, F and trim
22 % Inertial velocity in body coords
23 delta_u = F(1);
24 delta_v = F(2);
25 delta_w = F(3);
26
27 % Inertial angular velocity in body coords
28 delta_p = F(4);
29 delta_q = F(5);
30 delta_r = F(6);
31
32 % Euler angles
33 delta_phi = F(7);
34 delta_theta = F(8);
35 delta_psi = F(9);
36
37 % Position vector from origin to COM in inertial coords
38 x_E = F(10);
39 y_E = F(11);
40 z_e = F(12);
41
42 delta_f1 = trim(1); % motor forces
43 delta_f2 = trim(2);
44 delta_f3 = trim(3);
45 delta_f4 = trim(4);

```

```

46
47 %% Aerodynamic and Control Moments and Forces
48
49 % Moments
50 L_a = 0; %- alpha^2 * p^2 * sign(p);
51 M_a = 0; %- alpha^2 * q^2 * sign(q);
52 N_a = 0; %- beta^2 * r^2 * sign(r);
53
54 L_c = ((delta_f2 + delta_f3) - (delta_f1 + delta_f4)) * rad;
55 M_c = ((delta_f3 + delta_f4) - (delta_f1 + delta_f2)) * rad;
56 N_c = (delta_f2 + delta_f4 - (delta_f1 + delta_f2)) * k;
57
58 L = L_a + L_c;
59 M = M_a + M_c;
60 N = N_a + N_c;
61
62 % if t > 1 && t < 1.5 % Throw a moment perturbation in for 0.2 seconds
63 %     L = L + momPert(1);
64 %     M = M + momPert(2);
65 %     N = N + momPert(3);
66 % end
67
68 % Forces
69 X_a = 0; %- heta^2 * delta_u^2 * sign(delta_u);
70 Y_a = 0; %- heta^2 * delta_v^2 * sign(delta_v);
71 Z_a = 0; %- xsi^2 * delta_w^2 * sign(delta_w);
72
73 X_c = 0;
74 Y_c = 0;
75 Z_c = -sum(trim);
76
77 X = X_a + X_c;
78 Y = Y_a + Y_c;
79 Z = Z_a + Z_c;
80
81 % Determining Roll, Pitch, and Yaw rates of change
82 deltap_dot = rad/I_x * (delta_f2 + delta_f3 - delta_f1 - delta_f4);
83 deltaq_dot = rad/I_y * (delta_f3 + delta_f4 - delta_f1 - delta_f2);
84 deltar_dot = k/I_z * (delta_f2 + delta_f4 - delta_f1 - delta_f3);
85 dOmega_bdt = [deltap_dot, deltaq_dot, deltar_dot]';
86
87 % Determining translation rates
88 deltau_dot = -g * delta_theta;
89 deltav_dot = g * delta_phi;
90 deltaw_dot = 1/m * Z;
91 dV_bdt = [deltau_dot, deltav_dot, deltaw_dot]';
92
93 % Tranlating body to inertial coordinates
94 x_dot = delta_u * cos(delta_theta)*cos(delta_psi) + ...
95         delta_v * (sin(delta_phi)*sin(delta_theta)*cos(delta_psi) - cos(
96             delta_phi)*sin(delta_psi)) + ...
97         delta_w * (cos(delta_phi)*sin(delta_theta)*cos(delta_psi) + sin(
98             delta_phi)*sin(delta_psi));
99 y_dot = delta_u * cos(delta_theta)*sin(delta_psi) + ...
100         delta_v * (sin(delta_phi)*sin(delta_theta)*sin(delta_psi) + cos(

```

```

99         delta_phi)*cos(delta_psi)) + ...
100         delta_w * (cos(delta_phi)*sin(delta_theta)*sin(delta_psi) - sin(
101         delta_phi)*cos(delta_psi));
102     z_dot = -delta_u * sin(delta_theta) + ...
103     delta_v * sin(delta_phi)*cos(delta_theta) + ...
104     delta_w * cos(delta_phi)*cos(delta_theta);
105     dV_Edt = [x_dot, y_dot, z_dot]';
106
107 % Translating Angular Momentum to Euler angles
108 deltaphi_dot = delta_p;
109 deltatheta_dot = delta_q;
110 deltapsi_dot = delta_r;
111 dEuldt = [deltaphi_dot, deltatheta_dot, deltapsi_dot]';
112
113 % Concatenating for output
114 dfdt = [dV_bdt; dOmega_bdt; dEuldt; dV_Edt];
115
116 end

```

```

1 %-----
2 % quadCopterSim_FBC simulates the dynamics of a quad copter with
3 % derivative feedback control.
4 %
5 % Dependancies:
6 %     quadCopterODE_FBC - ODE function defining dynamics of quad copter.
7 %
8 % Created: 2/10/18 - Connor Ott
9 % Last Modified: 2/10/18 - Connor Ott
10 %-----
11 %% Introducing Feedback control
12
13 sSize = get(0, 'screensize');
14 condMat = zeros(3, 12);
15 condMat(:, 12) = -4; % initial height
16 condMat(1, 4) = 0.1; % [rad/s]
17 condMat(2, 5) = 0.1; % [rad/s]
18 condMat(3, 6) = 0.1; % [rad/s]
19
20 titleCell = {'Pitch Deviation', 'Roll Deviation', 'Yaw Deviation'};
21 tspan = [0, 5]; %s
22 fbcTrim = ones(1, 4)*0.068*9.81 / 4;
23 for i = 1:length(titleCell)
24
25     initConds = condMat(i, :);
26     [t.FBC, F.FBC] = ode45(@(t, F)quadCopterODE_FBC(t, F, fbcTrim), ...
27                             tspan, initConds);
28
29 %% pqr Plots
30 figure('pos', [sSize(3)*0.25, sSize(4)*0.15, ...
31               sSize(3)*0.40, sSize(4)*0.70]);
32 subplot(3, 1, 1)
33 hold on; grid on;
34 plot(t.FBC, F.FBC(:, 4), 'b-', 'linewidth', 1.2)
35 xlabel('Time, [s]')

```

```

36 ylabel('Roll, [rad/s]')
37
38 subplot(3, 1, 2)
39 hold on; grid on;
40 plot(t.FBC, F.FBC(:, 5), 'b-', 'linewidth', 1.2)
41 xlabel('Time, [s]')
42 ylabel('Pitch, [rad/s]')
43
44 subplot(3, 1, 3)
45 hold on; grid on;
46 plot(t.FBC, F.FBC(:, 6), 'b-', 'linewidth', 1.2)
47 xlabel('Time, [s]')
48 ylabel('Yaw, [rad/s]')
49
50 [~, t] = suplabel([titleCell{i}, ' - With Feedback Control'], ...
51                  't', [.1 .1 .84 .84]);
52 set(t, 'fontsize', 15)
53 set(gcf, 'Visible', 'off')
54 if exist(['./ Ass3figs_fbc/', titleCell{i}, '_pqrFBC.png'], 'file') ~=2
55     saveas(gcf, ['./ Ass3figs_fbc/', titleCell{i}, '_pqrFBC.png']);
56 end
57
58
59 %% uvw Plots
60 figure('pos', [sSize(3)*0.25, sSize(4)*0.15, ...
61              sSize(3)*0.40, sSize(4)*0.70]);
62 suplabel(titleCell{i}, 't');
63 subplot(3, 1, 1)
64 hold on; grid on;
65 plot(t.FBC, F.FBC(:, 1), 'b-', 'linewidth', 1.2)
66 xlabel('Time, [s]')
67 ylabel('u, [m/s]')
68
69 subplot(3, 1, 2)
70 hold on; grid on;
71 plot(t.FBC, F.FBC(:, 2), 'b-', 'linewidth', 1.2)
72 xlabel('Time, [s]')
73 ylabel('v, [rad/s]')
74
75 subplot(3, 1, 3)
76 hold on; grid on;
77 plot(t.FBC, F.FBC(:, 3), 'b-', 'linewidth', 1.2)
78 xlabel('Time, [s]')
79 ylabel('w, [rad/s]')
80
81 [~, t] = suplabel([titleCell{i}, ' - With Feedback Control'], ...
82                  't', [.1 .1 .84 .84]);
83 set(t, 'fontsize', 15)
84 set(gcf, 'Visible', 'off')
85 if exist(['./ Ass3figs_fbc/', titleCell{i}, '_uvwFBC.png'], 'file') ~=2
86     saveas(gcf, ['./ Ass3figs_fbc/', titleCell{i}, '_uvwFBC.png']);
87 end
88 end

```

```

1 function dfdt = quadCopterODE_FBC(t, F, trim)
2 % Defines dynamics of quad copter

```



```

3
4 %% Physical properties and constants
5 alpha = 2e-6;      % [N/(m/s) ^ 2]
6 beta = 1e-6;      % [N/(m/s) ^ 2]
7 heta = 1e-3;      % [N/(rad/s) ^ 2]
8 xsi = 3e-3;      % [N/(rad/s) ^ 2]
9
10 I_x = 6.8e-5;      % [kg m^ 2]
11 I_y = 9.2e-5;      % [ ' ' ]
12 I_z = 1.35e-4;      % [ ' ' ]
13
14 m = 0.068;        % [kg]
15 d = 0.06;         % [m]
16 k = 0.0024;       % [ ~ ]
17 rad = d/sqrt(2);  % [m]
18 g = 9.81;         % [m/s ^ 2]
19
20 % Gains
21 k_p = 0.003;      % [Nm/(rad/s) ]
22 k_q = 0.003;      % [Nm/(rad/s) ]
23 k_r = 0.0012;     % [Nm/(rad/s) ]
24
25 %% Pulling from input, F and trim
26 % Inertial velocity in body coords
27 u = F(1);
28 v = F(2);
29 w = F(3);
30
31 % Inertial angular velocity in body coords
32 p = F(4);
33 q = F(5);
34 r = F(6);
35
36 % Euler angles
37 phi = F(7);
38 theta = F(8);
39 psi = F(9);
40
41 % Position vector from origin to COM in inertial coords
42 x_E = F(10);
43 y_E = F(11);
44 z_e = F(12);
45
46 f1 = trim(1); % motor forces
47 f2 = trim(2);
48 f3 = trim(3);
49 f4 = trim(4);
50
51 %% Aerodynamic and Control Moments and Forces
52
53 % Moments
54 L_a = - alpha^2 * p^2 * sign(p);
55 M_a = - alpha^2 * q^2 * sign(q);
56 N_a = - beta^2 * r^2 * sign(r);
57

```

```

58 L_c = -k_p * p;
59 M_c = -k_q * q;
60 N_c = -k_r * r;
61
62 L = L_a + L_c;
63 M = M_a + M_c;
64 N = N_a + N_c;
65
66 % Forces
67 X_a = - heta^2 * u^2 * sign(u);
68 Y_a = - heta^2 * v^2 * sign(v);
69 Z_a = - xsi^2 * w^2 * sign(w);
70
71 X_c = 0;
72 Y_c = 0;
73 Z_c = -sum(trim);
74
75 X = X_a + X_c;
76 Y = Y_a + Y_c;
77 Z = Z_a + Z_c;
78
79 % Determining Roll, Pitch, and Yaw rates of change
80 p_dot = (I_y - I_z)/I_x * q * r + 1/I_x * L;
81 q_dot = (I_z - I_x)/I_y * p * r + 1/I_y * M;
82 r_dot = (I_x - I_y)/I_z * p * q + 1/I_z * N;
83 dOmega_bdt = [p_dot, q_dot, r_dot]';
84
85 % Determining translation rates
86 u_dot = r*v - q*w - g*sin(theta) + 1/m * X;
87 v_dot = p*w - r*u + g*sin(phi)*cos(theta) + 1/m * Y;
88 w_dot = q*u - p*v + g*cos(phi)*cos(theta) + 1/m * Z;
89 dV_bdt = [u_dot, v_dot, w_dot]';
90
91 % Tranlating body to inertial coordinates
92 x_dot = u * cos(theta)*cos(psi) + ...
93         v * (sin(phi)*sin(theta)*cos(psi) - cos(phi)*sin(psi)) + ...
94         w * (cos(phi)*sin(theta)*cos(psi) + sin(phi)*sin(psi));
95 y_dot = u * cos(theta)*sin(psi) + ...
96         v * (sin(phi)*sin(theta)*sin(psi) + cos(phi)*cos(psi)) + ...
97         w * (cos(phi)*sin(theta)*sin(psi) - sin(phi)*cos(psi));
98 z_dot = -u * sin(theta) + ...
99         v * sin(phi)*cos(theta) + ...
100        w * cos(phi)*cos(theta);
101 dV_Edt = [x_dot, y_dot, z_dot]';
102
103 % Translating Angular Momentum to Euler angles
104 phi_dot = p + (q*sin(phi) + r*cos(phi))*tan(theta);
105 theta_dot = q*cos(phi) - r*sin(phi);
106 psi_dot = q*sin(phi)*sec(theta) + r*cos(phi)*sec(theta);
107 dEuldt = [phi_dot, theta_dot, psi_dot]';
108
109 % Concatenating for output
110 dfdt = [dV_bdt; dOmega_bdt; dEuldt; dV_Edt];
111
112 end

```

```

1 function [value, isTerm, direction] = termEvents(t, F)
2 % Define events for quad copter ODE
3 value = [F(12), F(12) + 100]';
4 isTerm = [1,1]';
5 direction = [];
6 end

```