Cairo University Faculty of Engineering Aerospace Department

AEROSPACE ENGINEERING

5/18/2022

Task 7

Autopilot

Submitted To: Prof. Dr. osama

name	sec	bn
Mohammed Ahmed Hassan Ahmed	2	37
Ibrahim Thabet Allam	1	1
Mohammed Hatem Mohammed Saeed	2	39
Mohamed Hassan Gad Ali	2	41
Mohammed Abd El Mawgoud Ghoneam	2	43

Contents

A. Develop The testing loop containing the "Controller + Simulator"

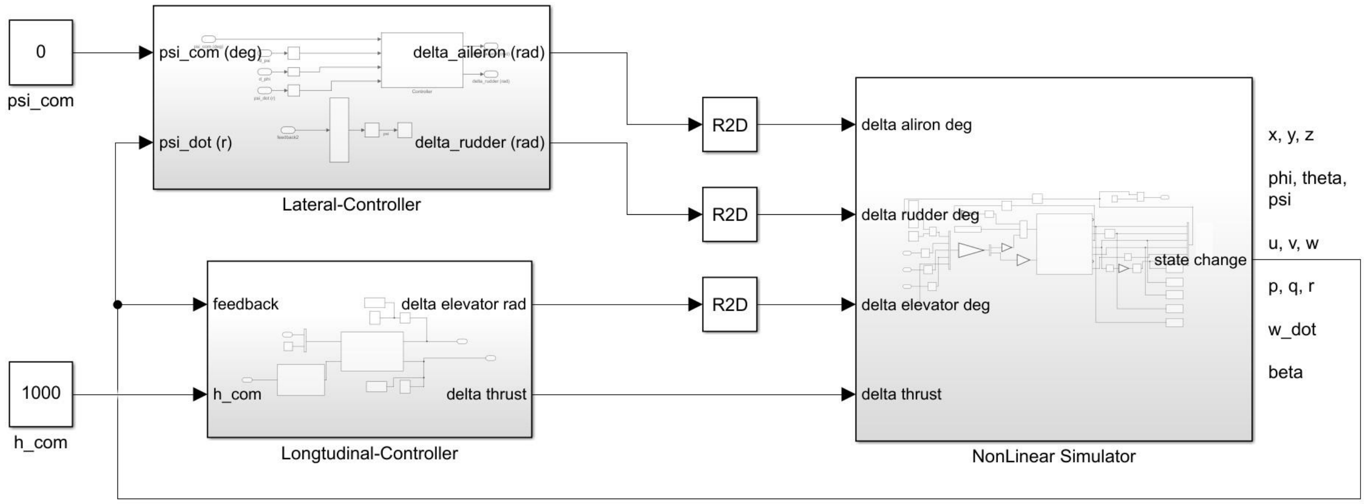


Figure 1. Controllers + Non Linear Simulator

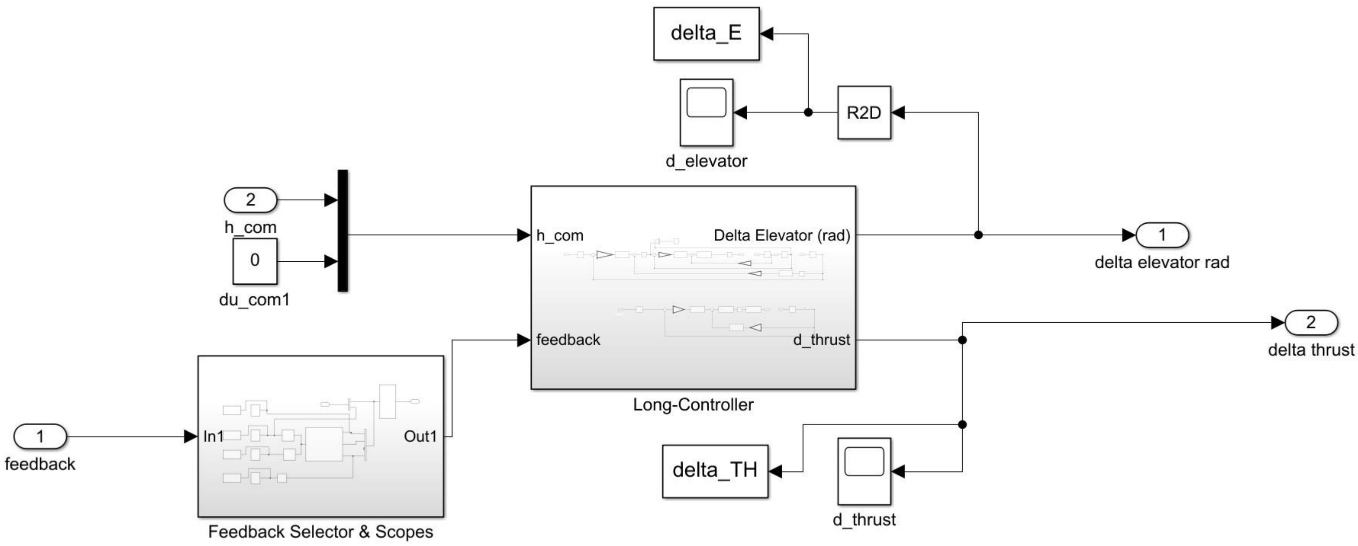


Figure 2. Longitudinal Controller

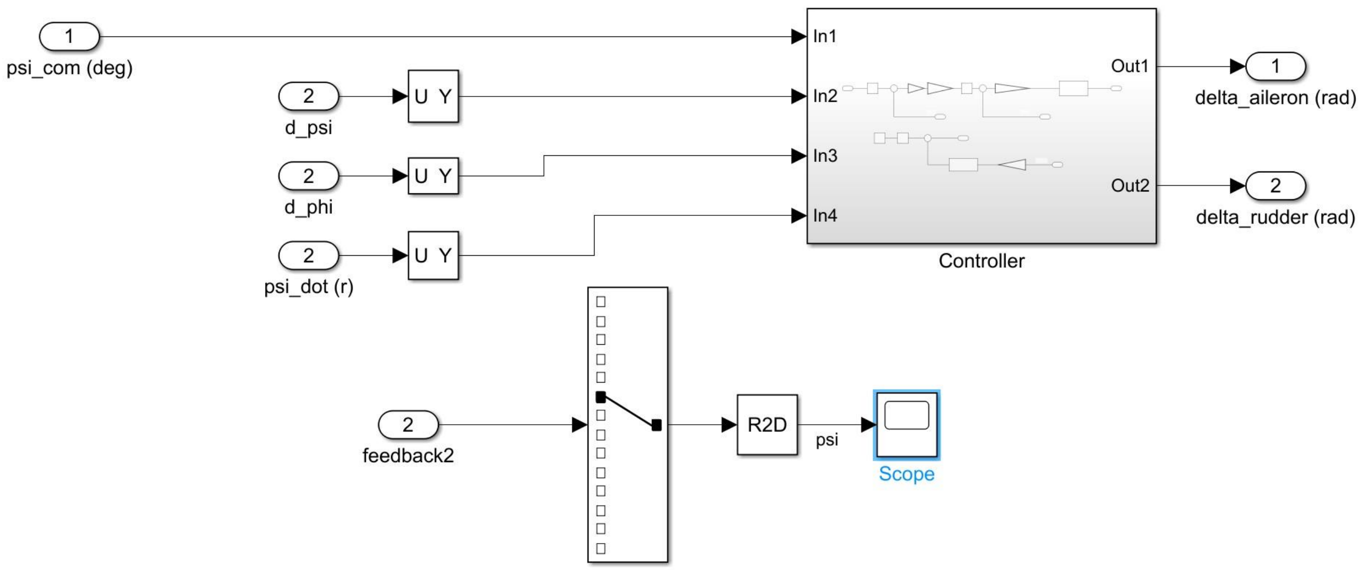


Figure 3. Lateral Controller

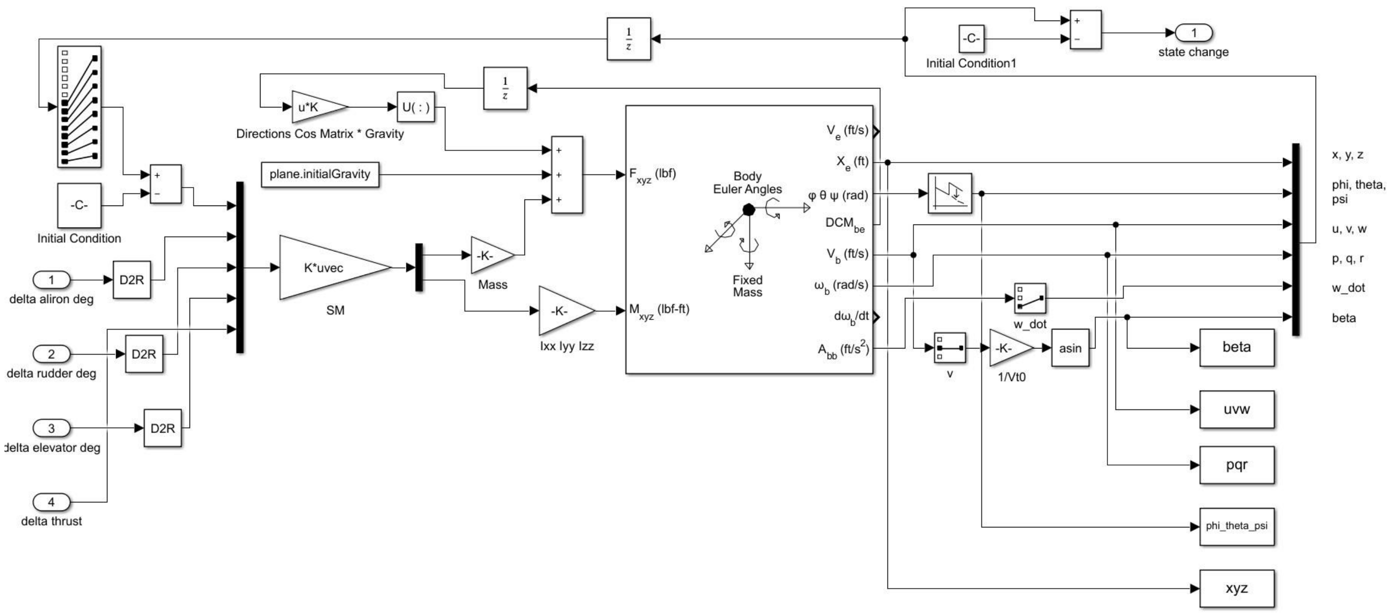
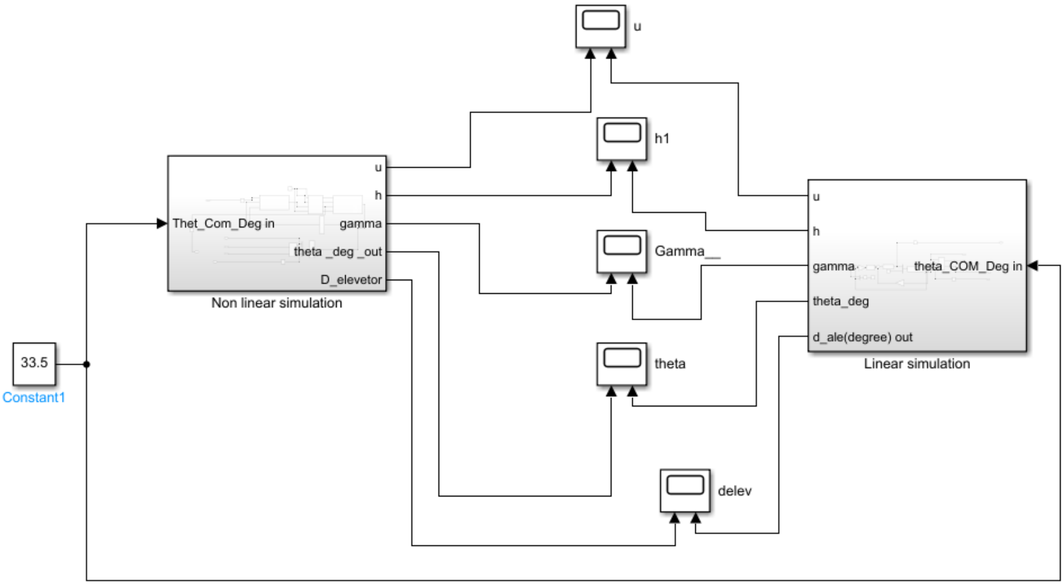
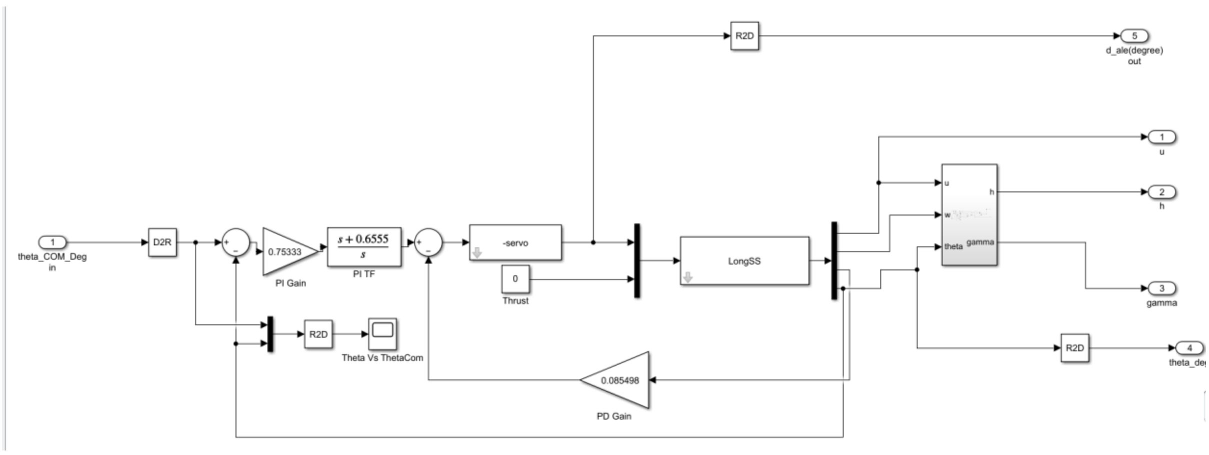


Figure 4. Non Linear Simulator

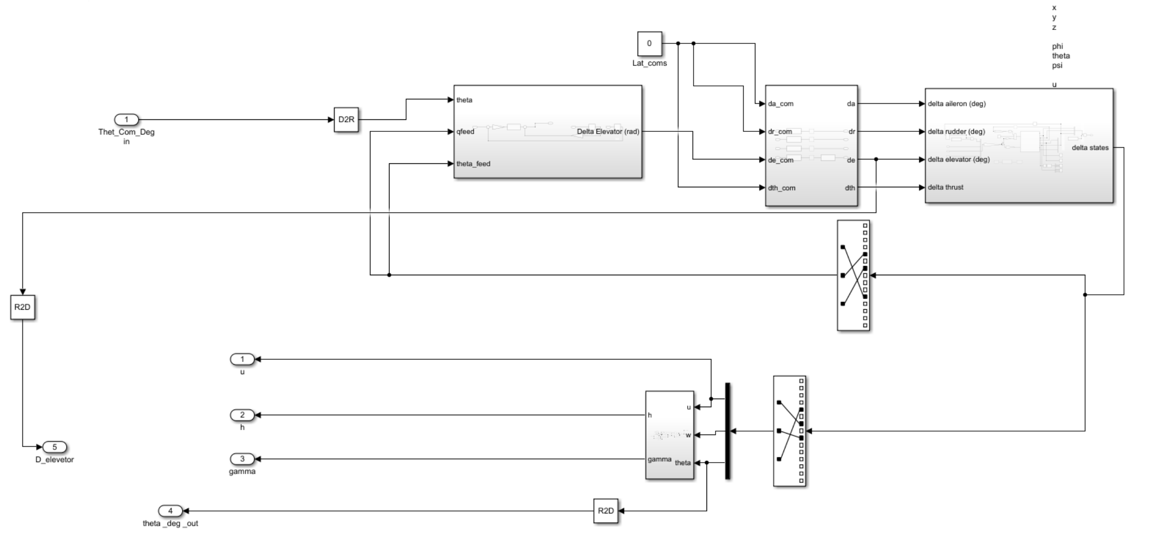
B. Test the “Pitch controller” and compare the response with the same test on the State space model



Linear block

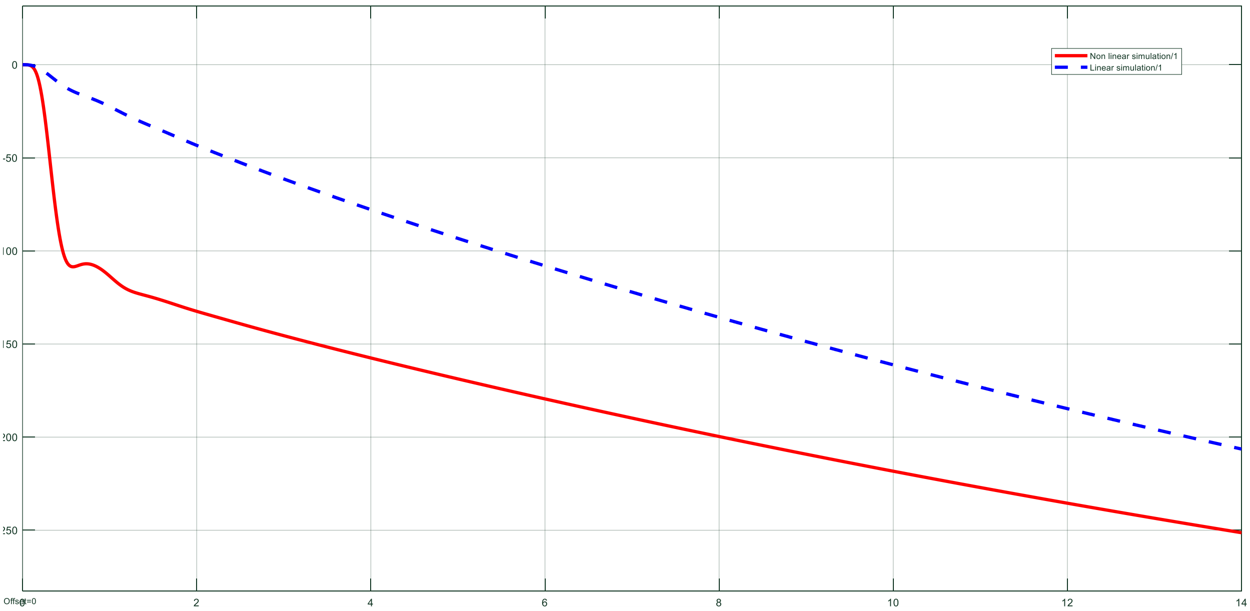


Nonlinear block

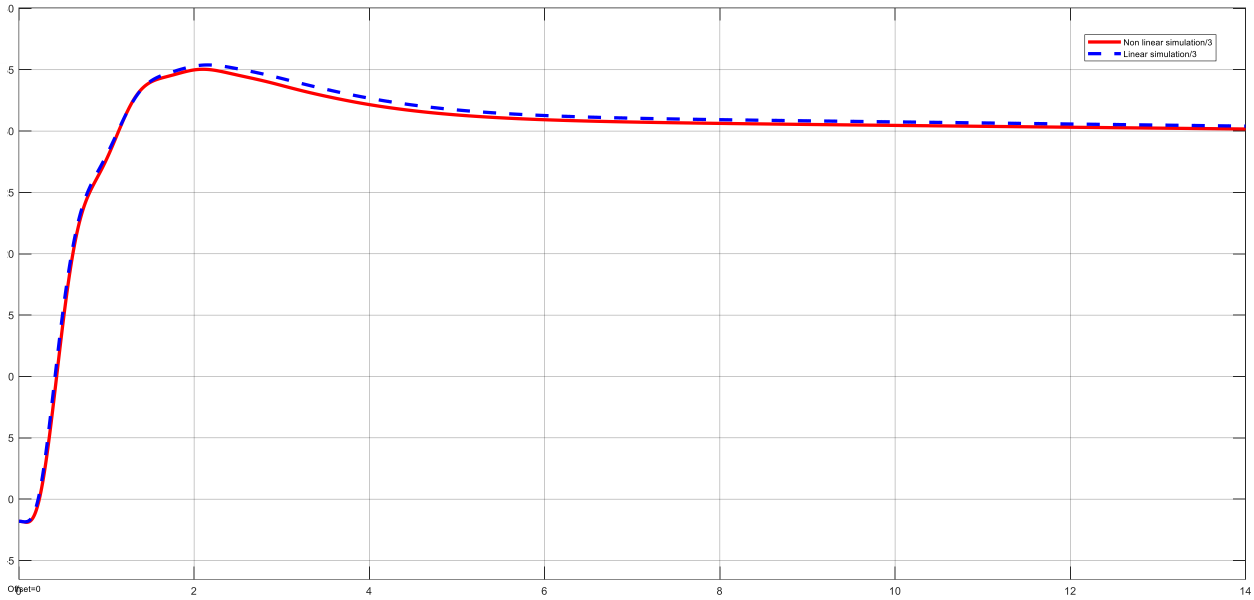


Results for 33.5 deg pitch angle

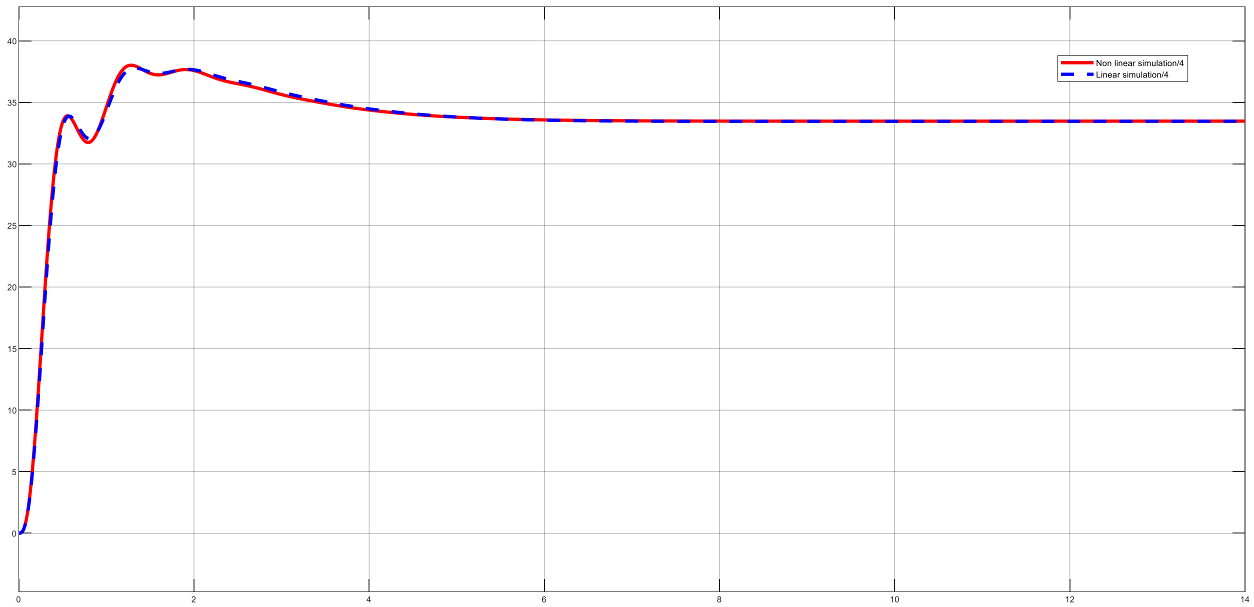
1-u



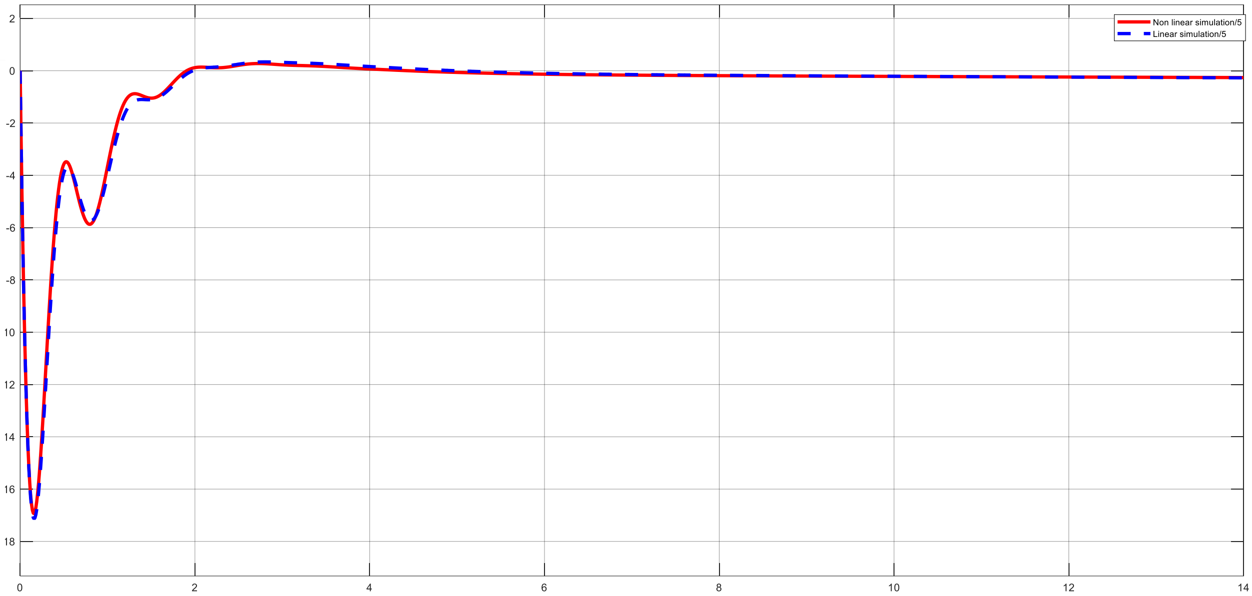
$2\text{-}\gamma$



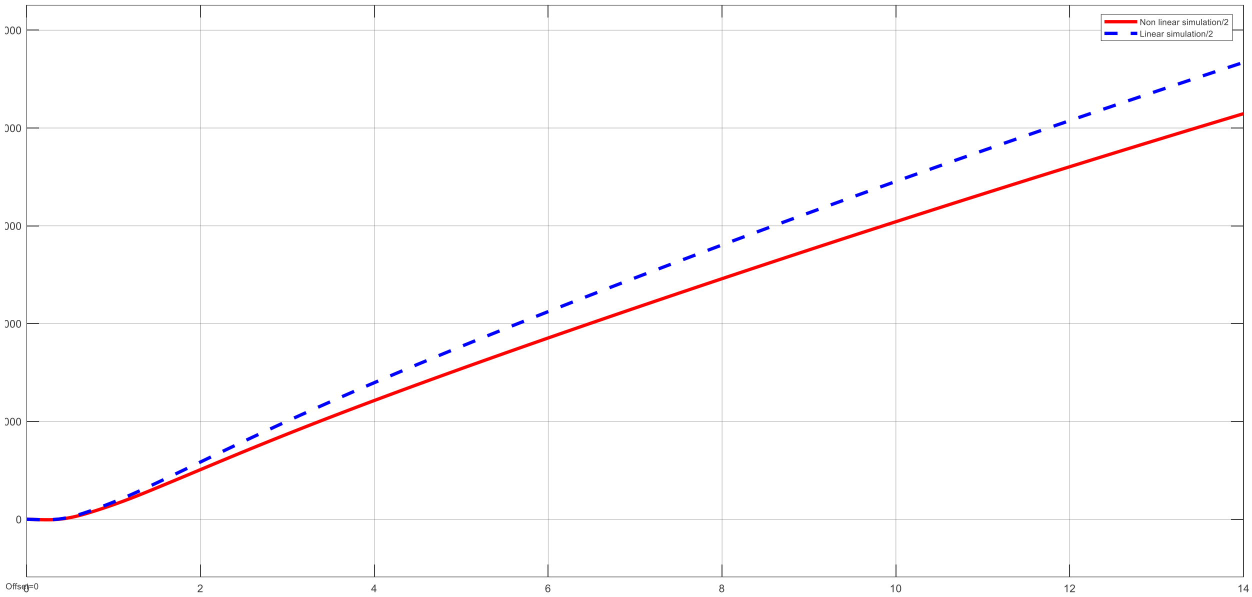
$3\text{-}\theta$



$4\text{-}\delta_{elevator}$



5-h

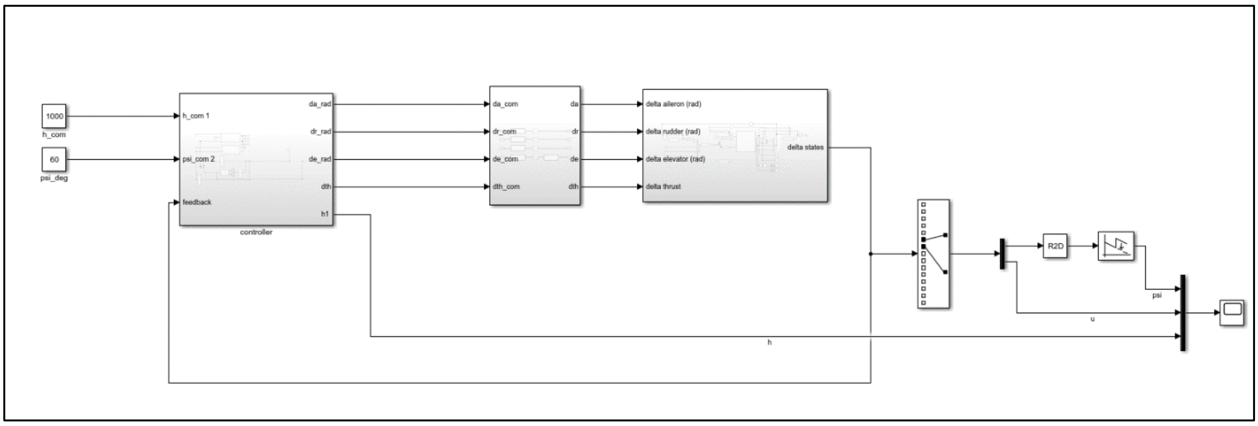


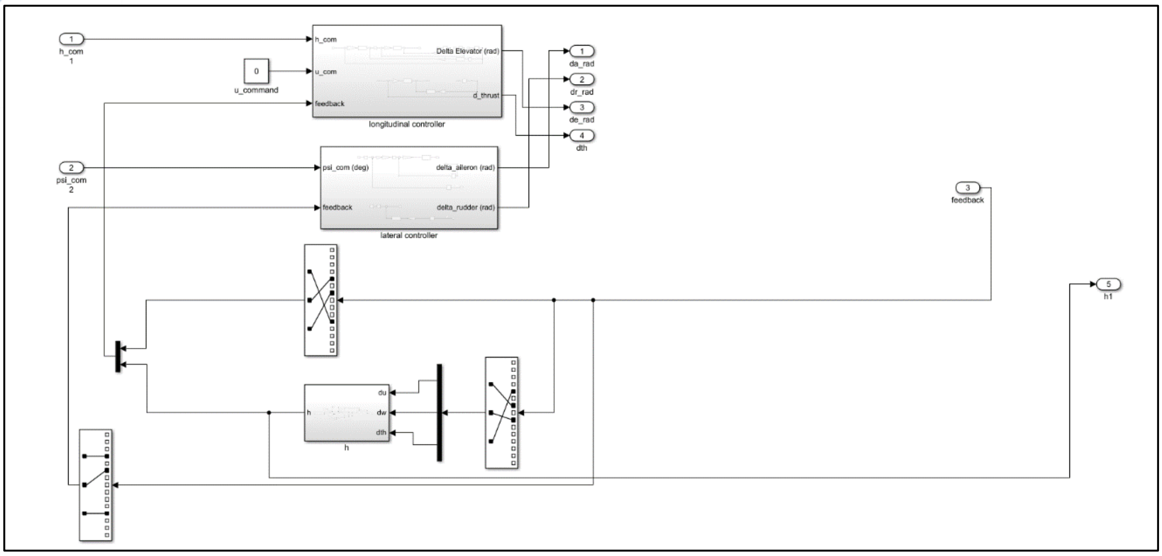
C. Test the “Pitch controller + Velocity controller” and compare the

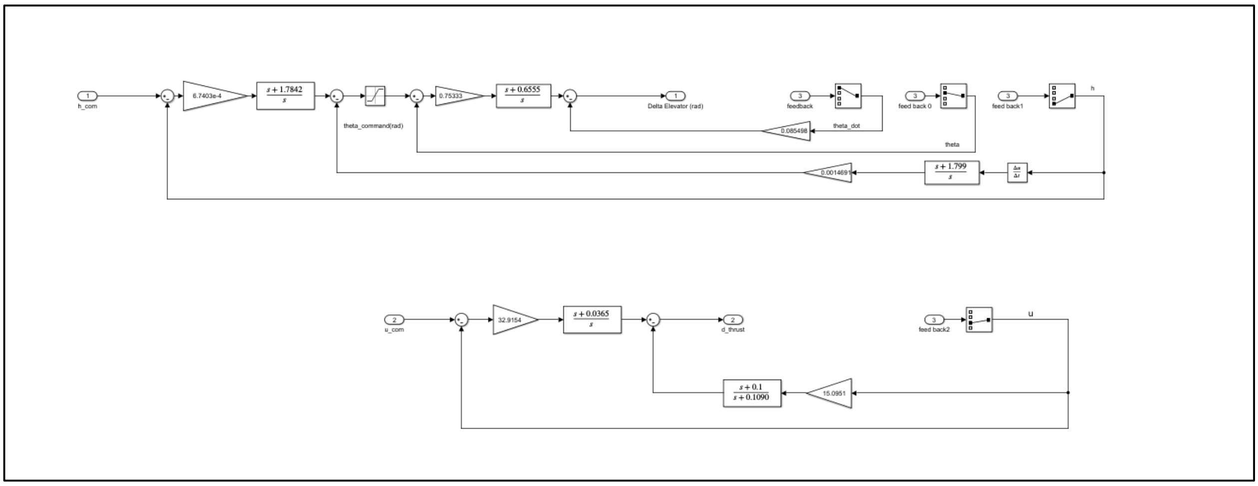
response with the same test on the State space model

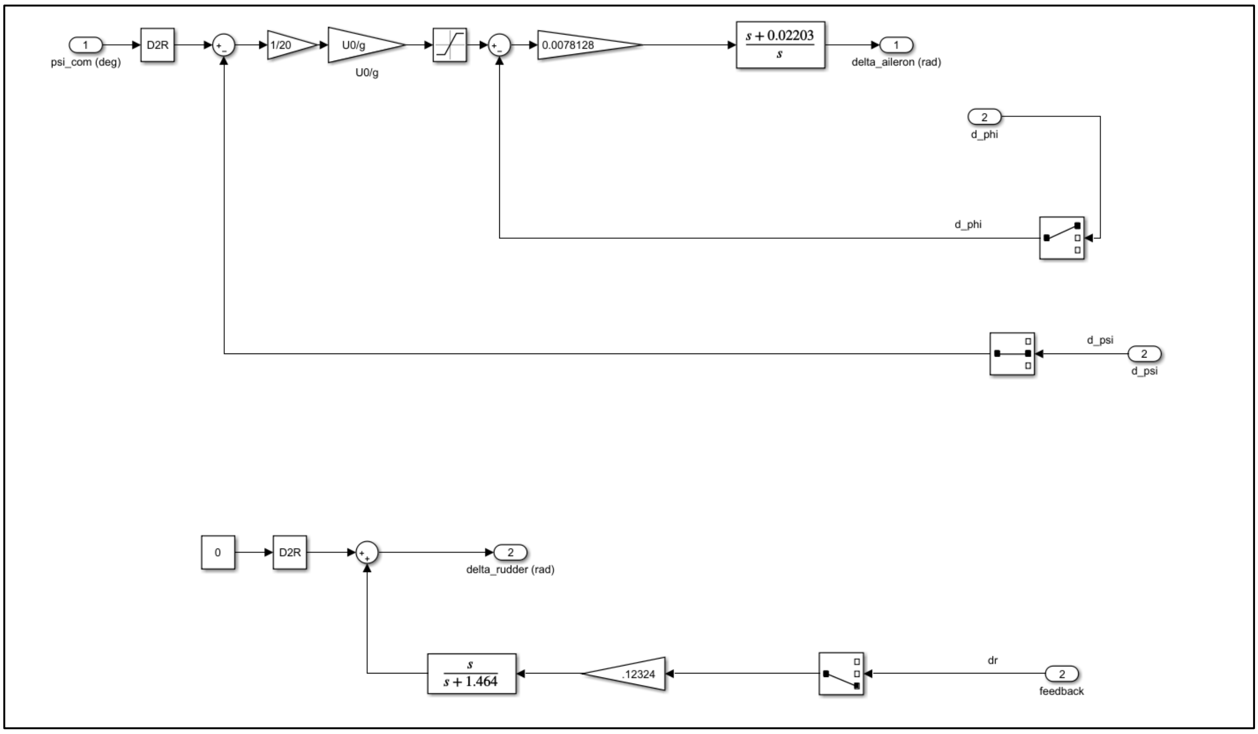
- Give a commanded input signal of pitch angle and observe the response and control action $(\theta, u, \gamma, h, \delta_e . \delta_{th})$.
- Perform the same test on the Linear Longitudinal state space model and observe the response and control action $(\theta, u, \gamma, h, \delta_e . \delta_{th})$
- Plot the results against each other.
- Note: in this test all the control actions are set to zero except the $(\delta_e\ , \delta_{th})$ which are.
- calculated by the Autopilot, this test shows the effect of the velocity controller and validate.
- the operation of both the pitch and velocity controllers.

Simulink Non-linear simulator:

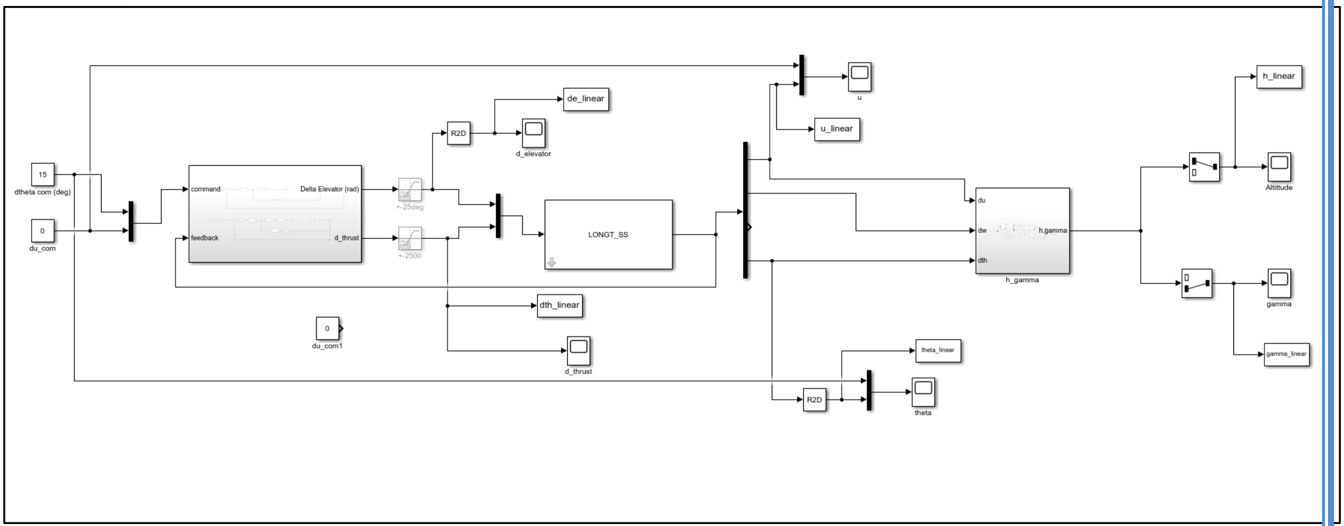


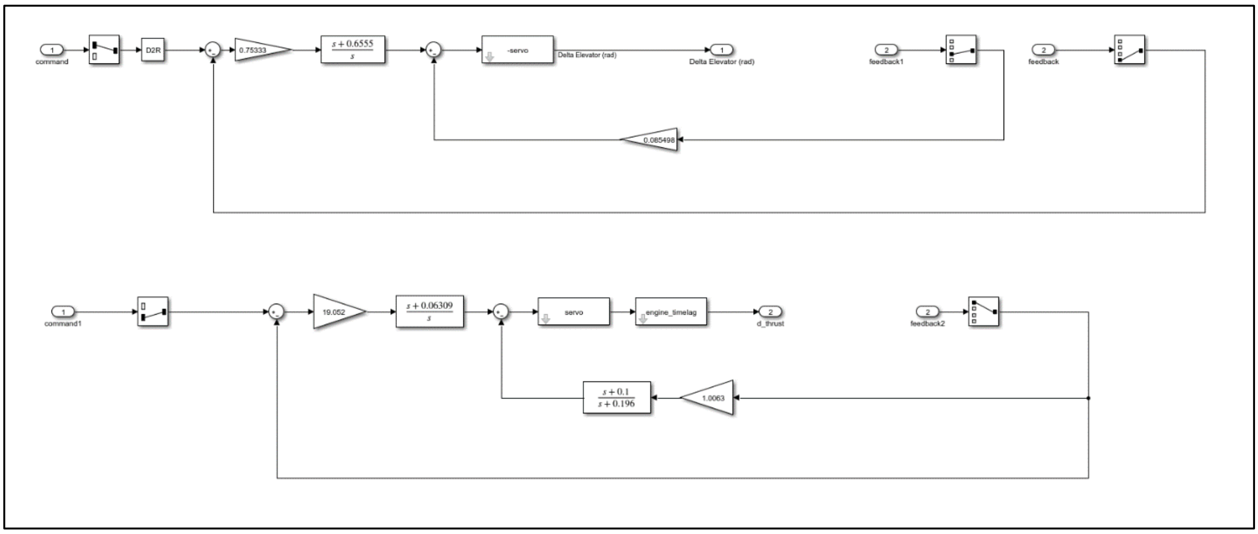

h_com
u_command
longitudinal controller
Delta Elevator (rad)
d_thrust
psi_com
lateral controller
delta_aileron (rad)
delta_rudder (rad)
feedback
da_rad
dr_rad
de_rad
dth
h1

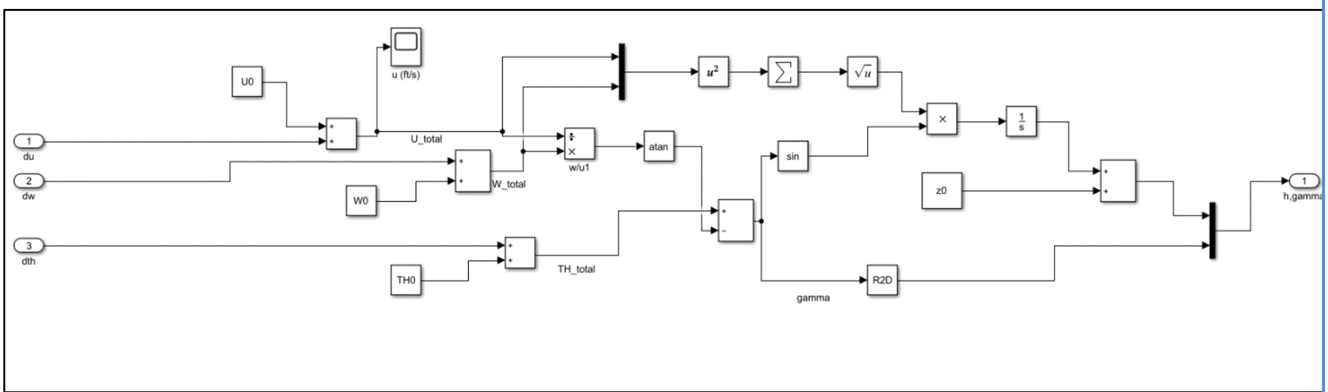

h_com
6.7403e-4
$\frac{s+1.7842}{s}$
theta_command(rad)
0.75333
$\frac{s+0.6555}{s}$
Delta Elevator (rad)
feedback
feed back 0
feed back1
0.085498
theta_dot
theta
0.001469
$\frac{s+1.799}{s}$
u_com
32.9154
$\frac{s+0.0365}{s}$
d_thrust
feed back2
u
$\frac{s+0.1}{s+0.1090}$
15.0951


psi_com (deg)
D2R
1/20
U0/g
U0/g
0.0078128
$\frac{s+0.02203}{s}$
delta_aileron (rad)
d_phi
d_phi
d_psi
d_psi
0
D2R
delta_rudder (rad)
$\frac{s}{s+1.464}$
.12324
dr
feedback

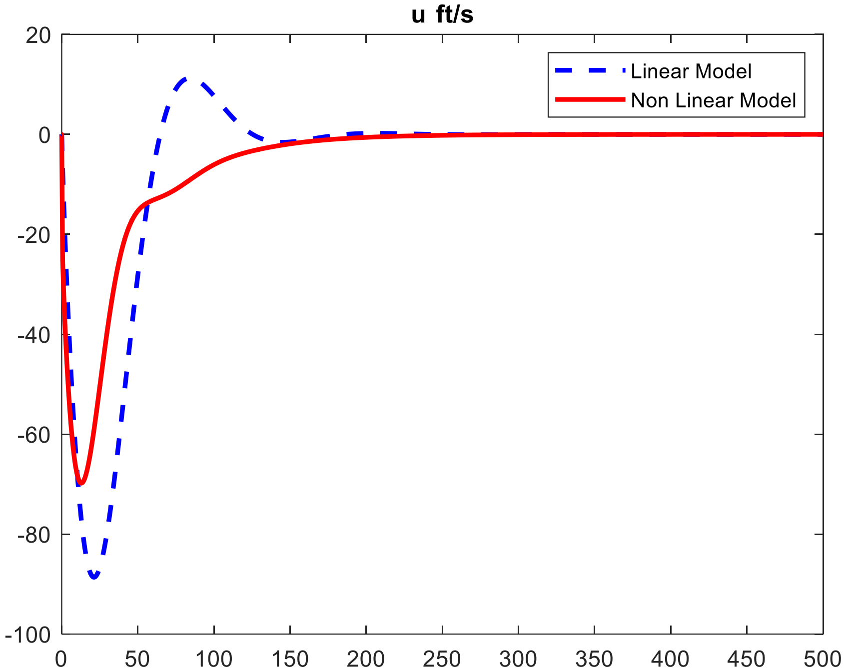
Simulink linear simulator:

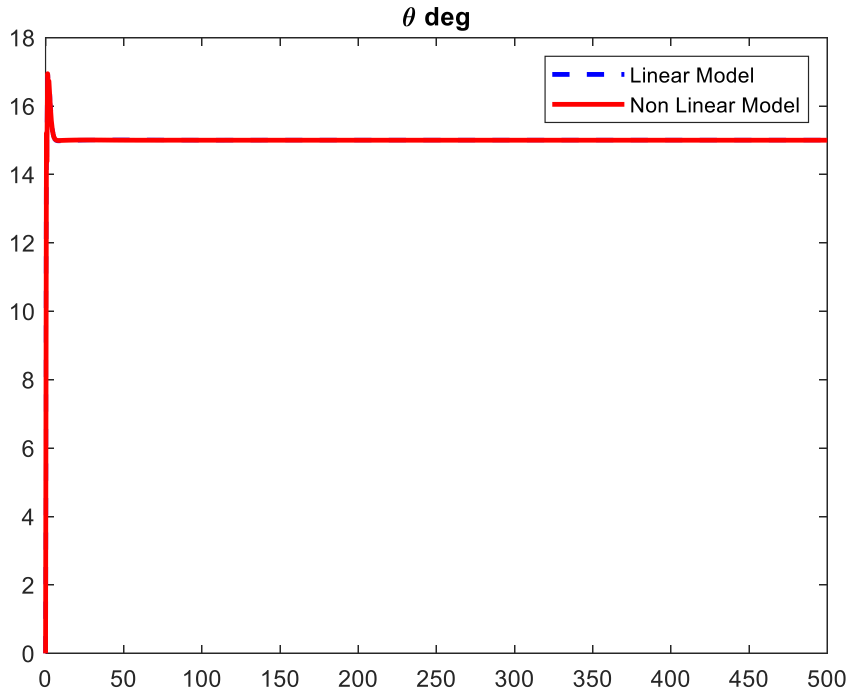


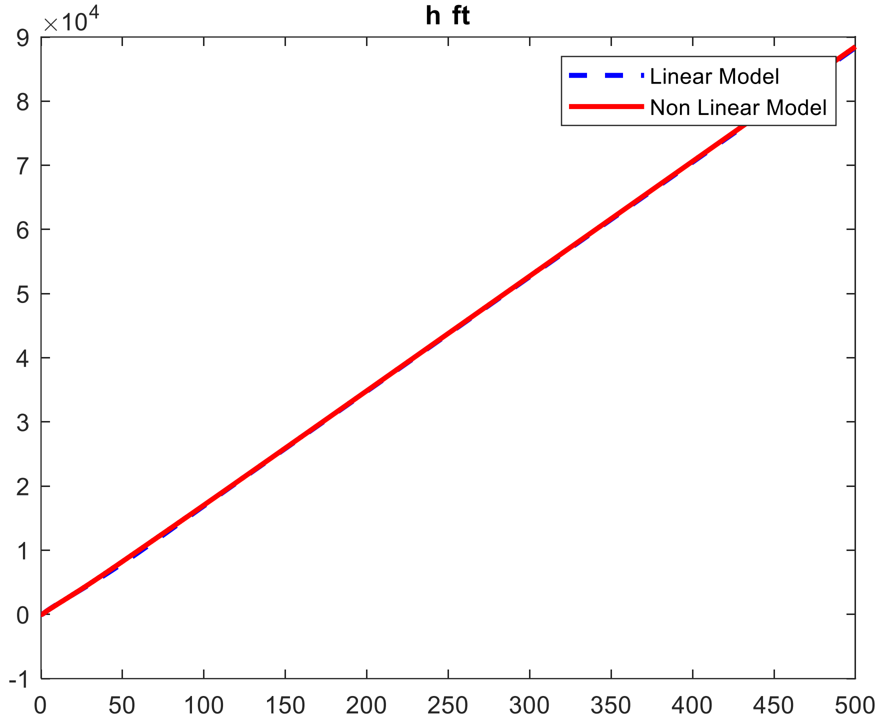


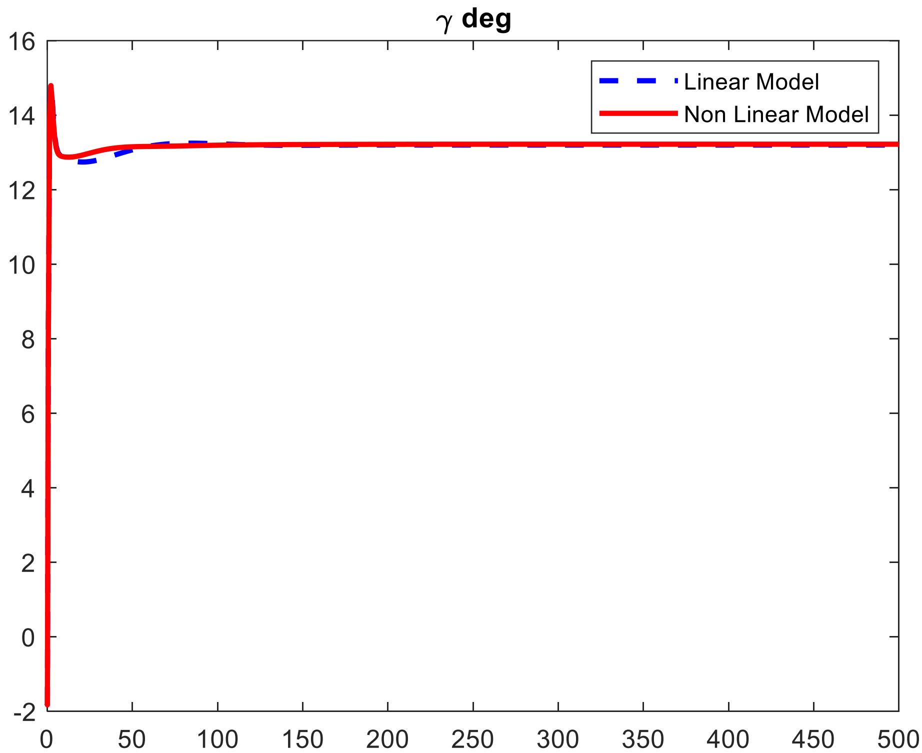


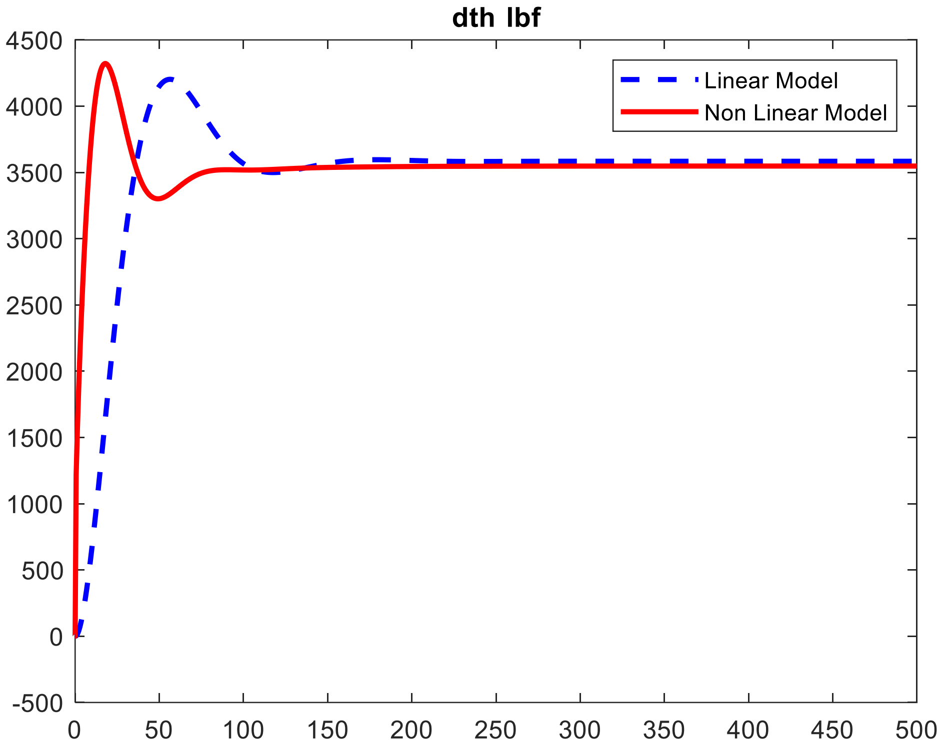
Simulation Results for 15° pitch command:

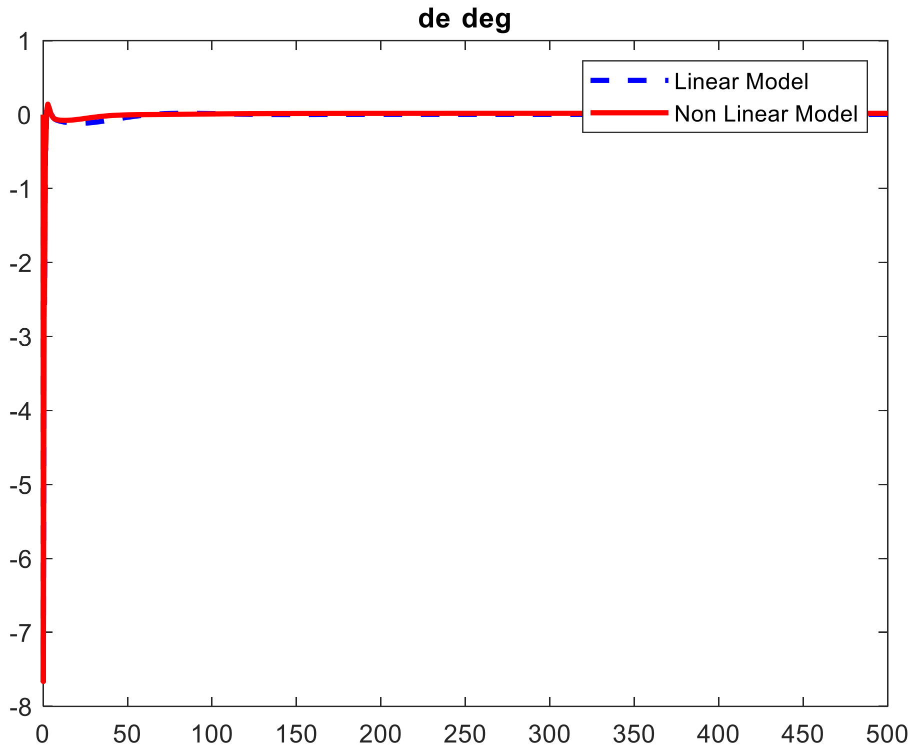





h ft
$\times 10^4$
9
8
7
6
5
4
3
2
1
0
-1
0
50
100
150
200
250
300
350
400
450
500
Linear Model
Non Linear Model

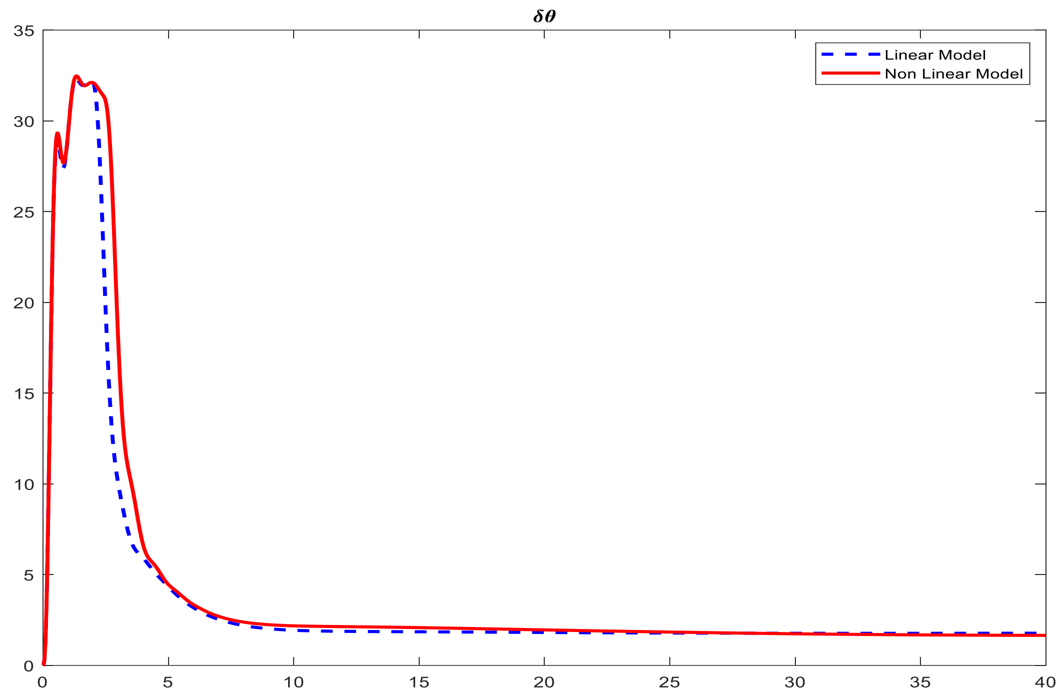

γ deg
16
14
12
10
8
6
4
2
0
-2
0
50
100
150
200
250
300
350
400
450
500
Linear Model
Non Linear Model

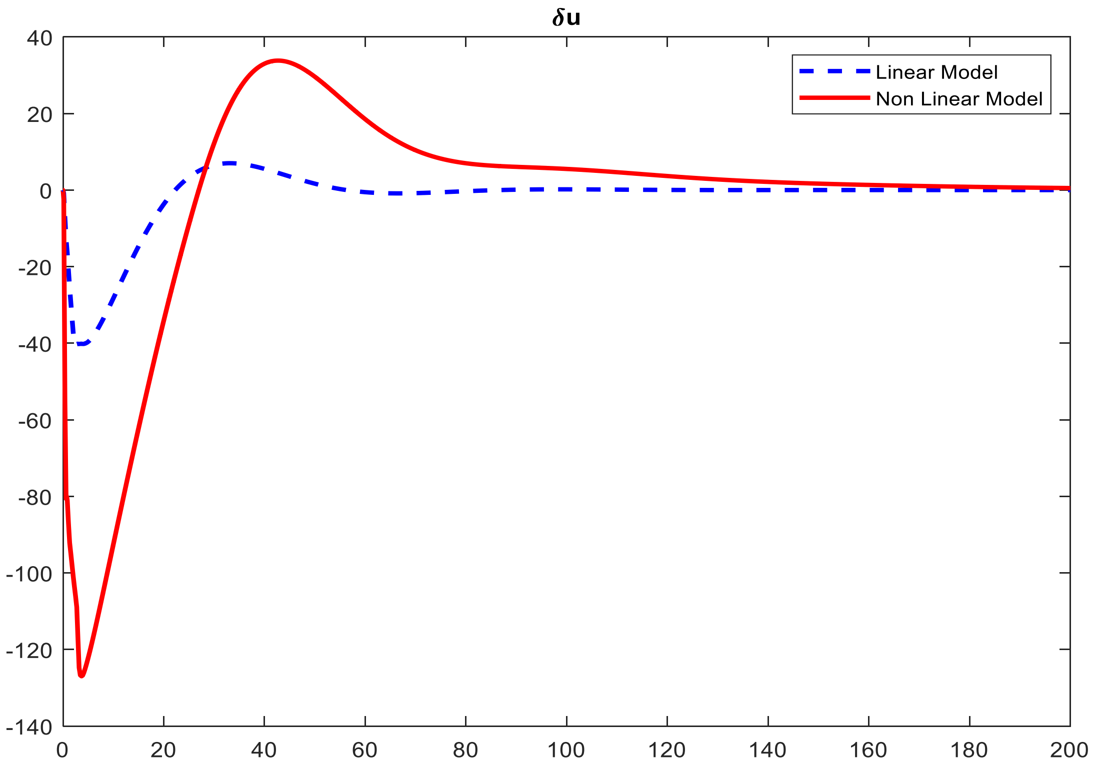

dth lbf
Linear Model
Non Linear Model
4500
4000
3500
3000
2500
2000
1500
1000
500
0
-500
0 50 100 150 200 250 300 350 400 450 500

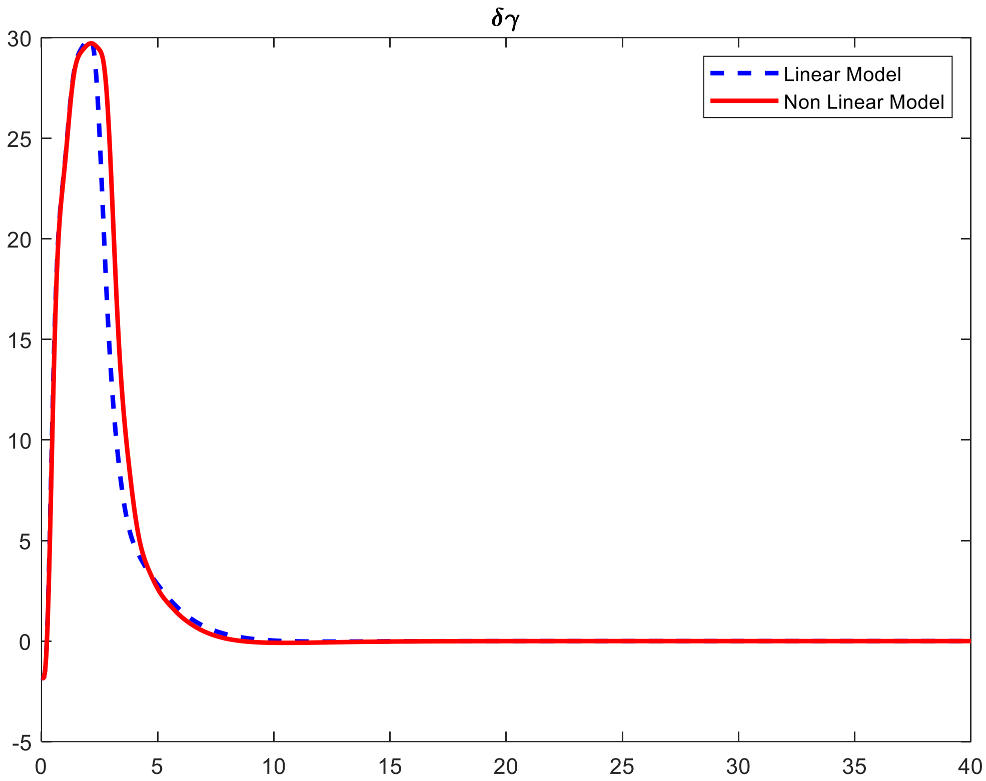

de deg
Linear Model
Non Linear Model
1
0
-1
-2
-3
-4
-5
-6
-7
-8
0 50 100 150 200 250 300 350 400 450 500

D. Test The "Altitude Hold" Controller and Compare the response with the same test on the state space model

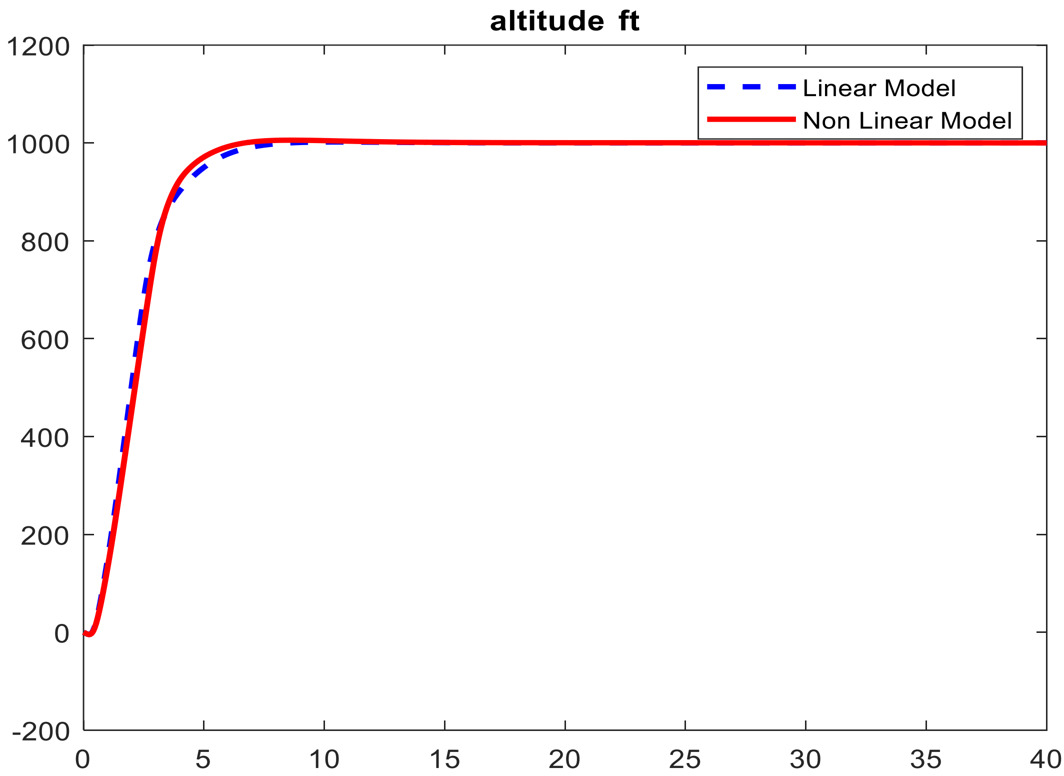
Results for input command of 1000ft.



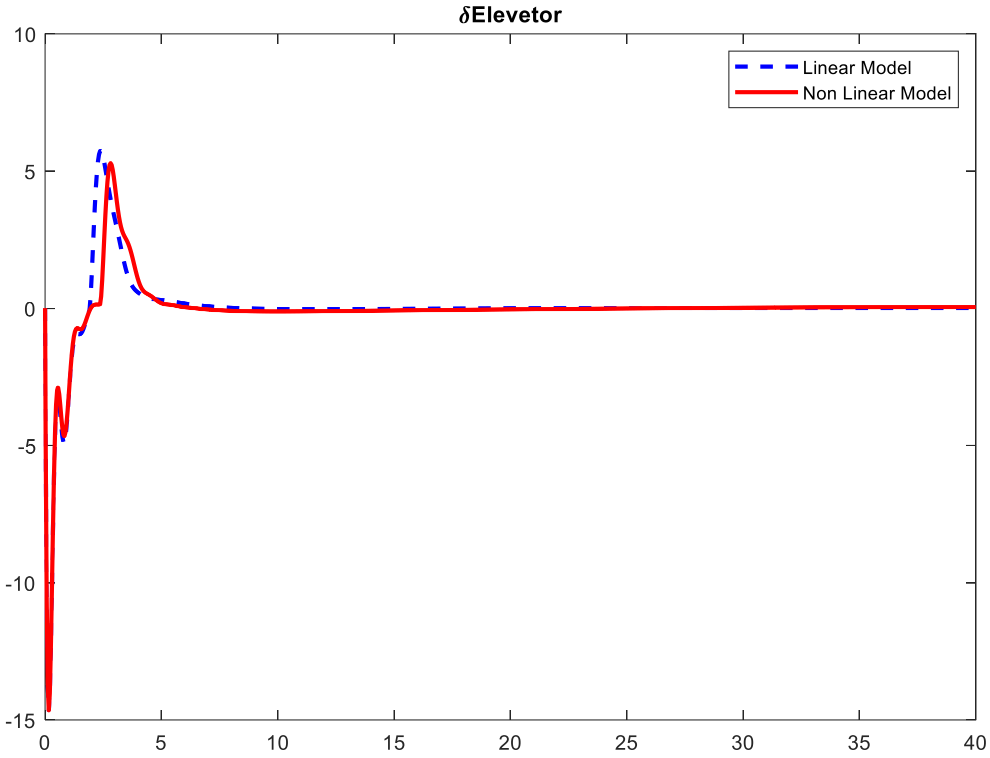


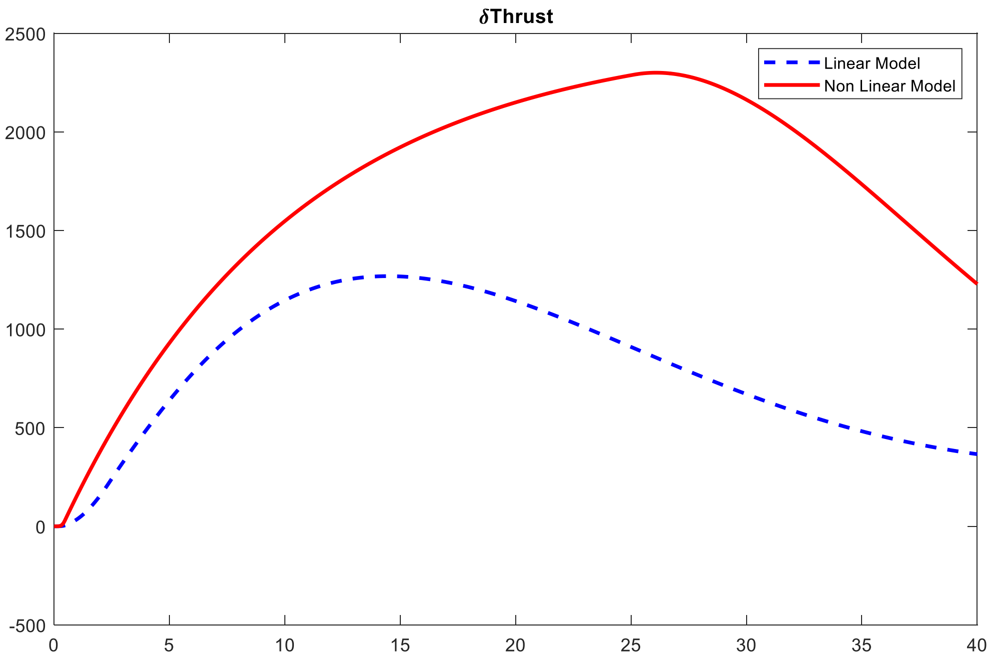


$\delta\gamma$
Linear Model
Non Linear Model
30
25
20
15
10
5
0
-5
0
5
10
15
20
25
30
35
40



altitude ft
Linear Model
Non Linear Model
1200
1000
800
600
400
200
0
-200
0
5
10
15
20
25
30
35
40


δElevetor
Linear Model
Non Linear Model
10
5
0
-5
-10
-15
0
5
10
15
20
25
30
35
40


δThrust
Linear Model
Non Linear Model
2500
2000
1500
1000
500
0
-500
0
5
10
15
20
25
30
35
40

Appendix: Code

AirPlane.m

```
classdef AirPlane < handle
    %UNTITLED Summary of this class goes here
    %   Detailed explanation goes here

    properties
       Mass
       g
       I             % Inirtia
       invI          % Inverse of Inirtia
       Ixx, Iyy, Izz,
       timeSpan
       dt
       ICs
       ICs_dot0
       Vt0
       dControl
       SD_Long
       SD_Lat
       SD_Lat_dash
       initialGravity
       airPlaneDerivatives      % Class
       rigidBodySolver          % Class

       u0, v0, w0, theta0, z0,

       SM % Stability Matrix
    end

    methods
        function airPlane = AirPlane(inputsFilePath)
            % Inputs
            % here B2:B61 means read the excel sheet from cell B2 to cell
B61
            aircraft_data = xlsread(inputsFilePath,'B2:B61');
            % Integration time span & Step
            airPlane.dt = aircraft_data(1);
            tfinal = aircraft_data(2);
            airPlane.timeSpan = [0 tfinal];

            % Initial Conditions
            % [u; v; w; p; q; r; phi; theta; epsi; xe0; ye0; ze0]
            % ICs = [10; 2; 0; 2*pi/180; pi/180; 0; 20*pi/180; 15*pi/180;
30*pi/180; 2; 4; 7];
            airPlane.ICs = aircraft_data(4:15);
            airPlane.ICs_dot0 = zeros(12,1);
            airPlane.Vt0 = sqrt(airPlane.ICs(1)^2 + airPlane.ICs(2)^2 +
airPlane.ICs(3)^2);    % Vto


            % D_a, D_r, D_e, D_th
            airPlane.dControl = [ aircraft_data(57:59) * pi/180 ;
aircraft_data(60)];

            % gravity, mass % inertia
            airPlane.Mass = aircraft_data(51);
            airPlane.g = aircraft_data(52);
```

```
            Ixx = aircraft_data(53); airPlane.Ixx = Ixx;
            Iyy = aircraft_data(54); airPlane.Iyy = Iyy;
            Izz = aircraft_data(55); airPlane.Izz = Izz;
            Ixz = aircraft_data(56);
            Ixy=0;  Iyz=0;
            airPlane.I = [Ixx , -Ixy , -Ixz ;...
                -Ixy , Iyy , -Iyz ;...
                -Ixz , -Iyz , Izz];
            airPlane.invI = inv(airPlane.I);

            % Stability Derivatives Longitudinal motion
            airPlane.SD_Long = aircraft_data(21:36);

            % Stability Derivatives Lateral motion
            airPlane.SD_Lat_dash = aircraft_data(37:50);
            airPlane.SD_Lat_dash(9) =
airPlane.SD_Lat_dash(9)*airPlane.Vt0;    % From dimension-less to
dimensional
            airPlane.SD_Lat_dash(10) =
airPlane.SD_Lat_dash(10)*airPlane.Vt0;  % Form dimension-less to
dimensional



            airPlane.airPlaneDerivatives = AirPlaneDerivatives(...
                airPlane.SD_Lat_dash , airPlane.SD_Long, airPlane.I);

            airPlane.rigidBodySolver = RigidBodySolver(airPlane.Mass,
airPlane.I, airPlane.invI, airPlane.dt, airPlane.g);

            [S, C, ~] = SCT(airPlane.ICs(7:9));
            airPlane.initialGravity = airPlane.Mass*airPlane.g*[
                S.theta;
                -S.phi*C.theta;
                -C.phi*C.theta;
            ];

            airPlane.u0 = airPlane.ICs(1);
            airPlane.v0 = airPlane.ICs(2);
            airPlane.w0 = airPlane.ICs(3);
            airPlane.theta0 = airPlane.ICs(8);
            airPlane.z0 = airPlane.ICs(12);

            airPlane.SM = airPlane.airPlaneDerivatives.stabilityMatrix();
        end

        function [dForce, dMoment] = airFrame(obj, state, forces,
moments, dControl)

            [Da, Dr, De, Dth] = feval(@(x) x{:}, num2cell(dControl));

            state_dot = obj.rigidBodySolver.DOF6(state, forces, moments);

            ds = state - obj.ICs;
            ds_dot = state_dot - obj.ICs_dot0;

            beta0 = asin(obj.ICs(2)/obj.Vt0);
            beta = asin(state(2)/obj.Vt0);
```

```
            dbeta = beta-beta0;

            dX = obj.Mass*(obj.airPlaneDerivatives.XU*ds(1)+ ...
                    obj.airPlaneDerivatives.XW*ds(3)+ ...
                    obj.airPlaneDerivatives.XDE*De+ ...
                    obj.airPlaneDerivatives.XD_TH*Dth);

            dY = obj.Mass*(obj.airPlaneDerivatives.YV*ds(2)+ ...
                obj.airPlaneDerivatives.YB*dbeta + ...
                obj.airPlaneDerivatives.YDA*Da + ...
                obj.airPlaneDerivatives.YDR*Dr);

            dZ = obj.Mass*(obj.airPlaneDerivatives.ZU*ds(1) + ...
                obj.airPlaneDerivatives.ZW*ds(3) + ...
                obj.airPlaneDerivatives.ZWD*ds_dot(3) + ...
                obj.airPlaneDerivatives.ZQ*ds(5) + ...
                obj.airPlaneDerivatives.ZDE*De + ...
                obj.airPlaneDerivatives.ZD_TH*Dth);

            dL = obj.Ixx*(obj.airPlaneDerivatives.LB*dbeta + ...
                obj.airPlaneDerivatives.LP*ds(4) + ...
                obj.airPlaneDerivatives.LR*ds(6) + ...
                obj.airPlaneDerivatives.LDR*Dr + ...
                obj.airPlaneDerivatives.LDA*Da);

            dM = obj.Iyy*(obj.airPlaneDerivatives.MU*ds(1) + ...
                obj.airPlaneDerivatives.MW*ds(3) + ...
                obj.airPlaneDerivatives.MWD*ds_dot(3) + ...
                obj.airPlaneDerivatives.MQ*ds(5) + ...
                obj.airPlaneDerivatives.MDE*De+ ...
                obj.airPlaneDerivatives.MD_TH*Dth);

            dN = obj.Izz*(obj.airPlaneDerivatives.NB*dbeta + ...
                obj.airPlaneDerivatives.NP*ds(4) + ...
                obj.airPlaneDerivatives.NR*ds(6) + ...
                obj.airPlaneDerivatives.NDR*Dr + ...
                obj.airPlaneDerivatives.NDA*Da);

            dForce = [dX dY dZ];
            dMoment = [dL dM dN];

        end

        function [A_long, B_long, C_long, D_long] = fullLinearModel(obj)
            [A_long, B_long, C_long, D_long] =
obj.airPlaneDerivatives.fullLinearModel(obj.ICs, obj.g);
        end

        function [A_long, B_long, C_long, D_long] =
lateralFullLinearModel(obj)
            [A_long, B_long, C_long, D_long] =
obj.airPlaneDerivatives.lateralFullLinearModel(obj.ICs, obj.g);
        end

        function [A_phug, B_phug, C_phug, D_phug] = longPeriodModel(obj)
            [A_phug, B_phug, C_phug, D_phug] =
obj.airPlaneDerivatives.longPeriodModel(obj.ICs, obj.g);
        end
```

```
    end
end
```

AirPlaneDerivatives.m

```
classdef AirPlaneDerivatives < handle
    %UNTITLED2 Summary of this class goes here
    %   Detailed explanation goes here

    properties
        % Longtudinal
        XU, ZU, MU, XW, ZW, MW, ZWD, ZQ, MWD, MQ, XDE, ZDE, MDE, XD_TH,
ZD_TH, MD_TH
        % Lateral
        YV
        YB
        LBd, NBd, LPd, NPd, LRd, NRd, LDAd, LDRd,NDAd, NDRd
        LB, NB, LP, NP, LR, NR, YDA, YDR, LDA, NDA, LDR, NDR
    end

    methods
        function obj = AirPlaneDerivatives(SD_Lat_dash , SD_Long,
Inertia, ICs, g)

            [obj.YV, obj.YB, obj.LBd, obj.NBd, obj.LPd, obj.NPd, ...
                obj.LRd, obj.NRd, obj.YDA, obj.YDR, obj.LDAd, ...
                obj.NDAd, obj.LDRd, obj.NDRd] = feval(@(x) x{:},
num2cell(SD_Lat_dash));

            [obj.XU, obj.ZU, obj.MU, obj.XW, obj.ZW, obj.MW, obj.ZWD,...
                obj.ZQ, obj.MWD, obj.MQ, obj.XDE, obj.ZDE, obj.MDE,
obj.XD_TH,...
                obj.ZD_TH, obj.MD_TH] = feval(@(x) x{:},
num2cell(SD_Long));

            LateralSD2BodyAxes(obj, Inertia);
        end

        function [SM] = stabilityMatrix(obj)
            % u v w p q r -- w_dot -- beta -- Da Dr De Dth
            SM = double(vpa([ obj.XU 0 obj.XW 0 0 0 0 0 0 0 obj.XDE
obj.XD_TH; ...
                0 obj.YV 0 0 0 0 0 obj.YB obj.YDA obj.YDR 0 0; ...
                obj.ZU 0 obj.ZW 0 obj.ZQ 0 obj.ZWD 0 0 0 obj.ZDE
obj.ZD_TH; ...
                0 0 0 obj.LP 0 obj.LR 0 obj.LB obj.LDA obj.LDR 0 0; ...
                obj.MU 0 obj.MW 0 obj.MQ 0 obj.MWD 0 0 0 obj.MDE
obj.MD_TH; ...
                0 0 0 obj.NP 0 obj.NR 0 obj.NB obj.NDA obj.NDR 0 0]));
        end

        function [obj] = LateralSD2BodyAxes(obj, Inertia)
            Ixx = Inertia(1);
            Izz = Inertia(9);
            Ixz = -Inertia(3);
            G = 1/(1 - Ixz^2 / Ixx / Izz);
            syms LB_ LP_ LR_ LDR_ LDA_ NB_ NP_ NR_ NDR_ NDA_
            eq1 = (LB_+Ixz*NB_/Ixx)*G == obj.LBd;
```

```
            eq2 = (NB_+Ixz*LB_/Izz)*G == obj.NBd;
            eq3 = (LP_+Ixz*NP_/Ixx)*G == obj.LPd;
            eq4 = (NP_+Ixz*LP_/Izz)*G == obj.NPd;
            eq5 = (LR_+Ixz*NR_/Ixx)*G == obj.LRd;
            eq6 = (NR_+Ixz*LR_/Izz)*G == obj.NRd;
            eq7 = (LDR_+Ixz*NDR_/Ixx)*G == obj.LDRd;
            eq8 = (NDR_+Ixz*LDR_/Izz)*G == obj.NDRd;
            eq9 = (LDA_+Ixz*NDA_/Ixx)*G == obj.LDAd;
            eq10 = (NDA_+Ixz*LDA_/Izz)*G == obj.NDAd;

            [A,B] = equationsToMatrix(...
            [eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8, eq9, eq10],...
            [LB_ LP_ LR_ LDR_ LDA_ NB_ NP_ NR_ NDR_ NDA_]);

            X = A\B;
            X = vpa(X);

            obj.LB = X(1);
            obj.LP = X(2) ;
            obj.LR = X(3) ;
            obj.LDR = X(4);
            obj.LDA = X(5);
            obj.NB = X(6);
            obj.NP = X(7);
            obj.NR = X(8);
            obj.NDR = X(9);
            obj.NDA = X(10);

        end

        function [A, B, C, D] = fullLinearModel(obj, ICs, g)

            u0 = ICs(1);
            w0 = ICs(3);
            theta0 = ICs(8);

            A =[obj.XU obj.XW -w0 -g*cos(theta0)
                obj.ZU/(1-obj.ZWD) obj.ZW/(1-obj.ZWD) (obj.ZQ+u0)/(1-
obj.ZWD) -g*sin(theta0)/(1-obj.ZWD)
                obj.MU+obj.MWD*obj.ZU/(1-obj.ZWD)
obj.MW+obj.MWD*obj.ZW/(1-obj.ZWD) obj.MQ+obj.MWD*(obj.ZQ+u0)/(1-obj.ZWD)
-obj.MWD*g*sin(theta0)/(1-obj.ZWD)
                0 0 1 0];
            B = [obj.XDE obj.XD_TH;
                obj.ZDE/(1-obj.ZWD) obj.ZD_TH/(1-obj.ZWD);
                obj.MDE+obj.MWD*obj.ZDE/(1-obj.ZWD)
obj.MD_TH+obj.MWD*obj.ZD_TH/(1-obj.ZWD);
                0 0];
            C = eye(4);
            D = zeros(4,2);

        end

        function [A, B, C, D] = lateralFullLinearModel(obj, ICs, g)

            u0 = ICs(1);
            v0 = ICs(2);
```

```
            w0 = ICs(3);
            theta0 = ICs(8);

            Vto = sqrt(u0^2 + v0^2 + w0^2);
            YDA_star = obj.YDA/Vto;
            YDR_star = obj.YDR/Vto;
            Yp = 0;
            Yr = 0;

            A = [obj.YB/Vto (Yp+w0)/Vto (Yr-u0)/Vto g*cos(theta0)/Vto
0;...
                   obj.LBd obj.LPd obj.LRd 0 0;...
                   obj.NBd obj.NPd obj.NRd 0 0;...
                   0 1 tan(theta0) 0 0;...
                   0 0 1/cos(theta0) 0 0];
            B = [YDA_star YDR_star;...
               obj.LDAd obj.LDRd;...
               obj.NDAd obj.NDRd;...
               0 0;0 0];
            C = eye(5); D = zeros(5,2);

        end

        function [A, B, C, D] = longPeriodModel(obj,ICs, g)
            u0 = ICs(1);

            A =[obj.XU -g
                -obj.ZU/(u0+obj.ZQ) 0];
            B =[obj.XDE obj.XD_TH
                -obj.ZDE/(obj.ZQ+u0) -obj.ZD_TH/(obj.ZQ+u0)];
            C = eye(2);
            D = zeros(2,2);

        end

    end
end
```

RigidBodySolver.m

```
classdef RigidBodySolver < handle
    %UNTITLED3 Summary of this class goes here
    %   Detailed explanation goes here

    properties
       Mass, Inertia, invInertia, dt, g
    end

    methods
        function obj = RigidBodySolver(Mass, Inertia, invInertia, dt,g)
            obj.Mass = Mass;
            obj.Inertia = Inertia;
            obj.invInertia = invInertia;
            obj.dt = dt;
            obj.g = g;
        end

        function state = nextStep(RBS, currentState, Force, Moments)
            K = zeros(12, 4);

            K(:, 1) = RBS.dt*DOF6(RBS, currentState ,Force, Moments);
            K(:, 2) = RBS.dt*DOF6(RBS, currentState+0.5*K(:, 1) ,Force,
Moments);
            K(:, 3) = RBS.dt*DOF6(RBS, currentState+0.5*K(:, 2) ,Force,
Moments);
            K(:, 4) = RBS.dt*DOF6(RBS, currentState+K(:, 3) ,Force,
Moments);

            state = currentState + (...
                K(:, 1)+...
                2*K(:, 2)+...
                2*K(:, 3)+...
                K(:, 4))/6;
        end

        function F = DOF6(RBS, currentState, forces, Moments)

            % (Sin, Cos, Tan) of (phi, theta, epsi)
            [S, C, T] = SCT(currentState(7:9));
            s_theta = S.theta;
            c_theta = C.theta;
            t_theta = T.theta;
            s_epsi = S.epsi;
            c_epsi = C.epsi;
            s_phi = S.phi;
            c_phi = C.phi;

            Forces = forces + RBS.Mass*RBS.g*[
                -s_theta;
                s_phi*c_theta;
                c_phi*c_theta;
            ];

            % (u, v, w) dot
            u_v_w_dot = (1/RBS.Mass)*Forces - cross(...
                currentState(4:6, 1), currentState(1:3, 1)...
            );
```

```
            % (p, q, r) dot
            p_q_r_dot = RBS.invInertia *(Moments - cross(...
                currentState(4:6, 1), RBS.Inertia * currentState(4:6,
1)...
            ));

            % (phi, theta, epsi) dot
            phi_theta_epsi_dot = [
                1, s_phi*t_theta, c_phi*t_theta;
                0, c_phi, -s_phi;
                0, s_phi/c_theta, c_phi/c_theta;
            ] * currentState(4:6, 1);

            % (x, y, z) dot
            x_y_z_dot = [
                c_theta*c_epsi, (s_phi*s_theta*c_epsi - c_phi*s_epsi),
(c_phi*s_theta*c_epsi + s_phi*s_epsi);
                c_theta*s_epsi, (s_phi*s_theta*s_epsi + c_phi*c_epsi),
(c_phi*s_theta*s_epsi - s_phi*c_epsi);
                -s_theta, s_phi*c_theta, c_phi*c_theta
            ] * currentState(1:3, 1);

            F = [u_v_w_dot; p_q_r_dot; phi_theta_epsi_dot; x_y_z_dot];

        end


    end
end
```

SCT.m

```
% Calculate Sin, Cos ,Tan for any set of three angles
% and return results in struct form for easy access in code.
function [S, C, T] = SCT(ICs)
    S = struct(...
        'phi', sin(ICs(1)),...
        'theta', sin(ICs(2)),...
        'epsi', sin(ICs(3))...
    );
    C = struct(...
        'phi', cos(ICs(1)),...
        'theta', cos(ICs(2)),...
        'epsi', cos(ICs(3))...
    );
    T = struct(...
        'phi', tan(ICs(1)),...
        'theta', tan(ICs(2)),...
        'epsi', tan(ICs(3))...
    );
end
```

Main.m

```
clc; clear; close all;

%% Inputs
% Initial AirPlane
plane = AirPlane("NT-33A_4.xlsx");
stability_matrix = plane.SM;

%% Initial Important Transfer Function
servo = tf(10,[1 10]);
integrator = tf(1,[1 0]);
differentiator = tf([1 0],1);
engine_timelag = tf(0.1 , [1 0.1]);

%% D. Test the "Altitude Hold" controller and compare the response with
the same test on the State space model
D_linear = load('./Results/linear_simulation_1000ft_altitude_hold.mat');
D_non_linear =
load('./Results/nonlinear_simulation_1000ft_altitude_hold.mat');

figure
plot(D_linear.delta_theta.Time, D_linear.delta_theta.Data, '--b',
'LineWidth', 2);
hold on
plot(D_non_linear.delta_theta.Time, D_non_linear.delta_theta.Data, '-r',
'LineWidth', 2);
xlim([0 40]);
legend('Linear Model', 'Non Linear Model');
title('{\delta}{\theta}');

figure
plot(D_linear.delta_u.Time, D_linear.delta_u.Data, '--b', 'LineWidth',
2);
hold on
plot(D_non_linear.delta_u.Time, D_non_linear.delta_u.Data, '-r',
'LineWidth', 2);
legend('Linear Model', 'Non Linear Model');
title('{\delta}{u}');

figure
plot(D_linear.gamma.Time, D_linear.gamma.Data, '--b', 'LineWidth', 2);
hold on
plot(D_non_linear.gamma.Time, D_non_linear.gamma.Data, '-r', 'LineWidth',
2);
xlim([0 40]);
legend('Linear Model', 'Non Linear Model');
title('{\delta}{\gamma}');

figure
plot(D_linear.altitude.Time, D_linear.altitude.Data, '--b', 'LineWidth',
2);
hold on
plot(D_non_linear.altitude.Time, D_non_linear.altitude.Data, '-r',
'LineWidth', 2);
xlim([0 40]);
legend('Linear Model', 'Non Linear Model');
```

```
title('{altitude} {ft}');

figure
plot(D_linear.delta_E.Time, D_linear.delta_E.Data, '--b', 'LineWidth',
2);
hold on
plot(D_non_linear.delta_E.Time, D_non_linear.delta_E.Data, '-r',
'LineWidth', 2);
xlim([0 40]);
legend('Linear Model', 'Non Linear Model');
title('{\delta}{Elevetor}');

figure
plot(D_linear.delta_TH.Time, D_linear.delta_TH.Data, '--b', 'LineWidth',
2);
hold on
plot(D_non_linear.delta_TH.Time, D_non_linear.delta_TH.Data, '-r',
'LineWidth', 2);
xlim([0 40]);
legend('Linear Model', 'Non Linear Model');
title('{\delta}{Thrust}');


%% Plotter
u = uvw.Data(:,1);
v = uvw.Data(:,2);
w = uvw.Data(:,3);
p = pqr.Data(:,1);

q = pqr.Data(:,2);
r = pqr.Data(:,3);
phi = phi_theta_psi.Data(:,1);
theta = phi_theta_psi.Data(:,2);
psi = phi_theta_psi.Data(:,3);
x = xyz.Data(:, 1);
y = xyz.Data(:, 2);
z = xyz.Data(:, 3);

beta_deg= beta.Data(:,1) * 180/pi;
alpha_deg = atan(w./u)*180/pi;
p_deg = p*180/pi;
q_deg = q*180/pi;
r_deg = r*180/pi;
phi_deg = phi*180/pi;
theta_deg = theta*180/pi;
psi_deg = psi*180/pi;

figure
plot3(x,-y,-z);
title('Trajectory')
figure
subplot(4,3,1)
plot(uvw.Time,u)
title('u (ft/sec)')
xlabel('time (sec)')
subplot(4,3,2)
plot(uvw.Time,beta_deg)
```

```
title('\beta (deg)')
xlabel('time (sec)')
subplot(4,3,3)
plot(uvw.Time,alpha_deg)
title('\alpha (deg)')
xlabel('time (sec)')
subplot(4,3,4)
plot(uvw.Time,p_deg)
title('p (deg/sec)')
xlabel('time (sec)')
subplot(4,3,5)
plot(uvw.Time,q_deg)
title('q (deg/sec)')
xlabel('time (sec)')
subplot(4,3,6)
plot(uvw.Time,r_deg)
title('r (deg/sec)')
xlabel('time (sec)')
subplot(4,3,7)
plot(uvw.Time,phi_deg)
title('\phi (deg)')
xlabel('time (sec)')
subplot(4,3,8)
plot(uvw.Time,theta_deg)
title('\theta (deg)')
xlabel('time (sec)')
subplot(4,3,9)
plot(uvw.Time,psi_deg)
title('\psi (deg)')
xlabel('time (sec)')
subplot(4,3,10)
plot(uvw.Time,x)
title('x (ft)')
xlabel('time (sec)')
subplot(4,3,11)
plot(uvw.Time,y)
title('y (ft)')
xlabel('time (sec)')
subplot(4,3,12)
plot(uvw.Time,z)
title('z (ft)')
xlabel('time (sec)')
```