



4/27/2022

Task 6

Autopilot -AER 408

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. Design a “Yaw damper” for the Dutch Roll Mode	2
Design No. One	2
Design No. Two.....	4
B. Test The “Yaw damper” controllers on the full state space model	6
Design No. One	6
Design No. Two.....	6
Results:	7
Design No. One	7
Design No. Two.....	7
Appendix: Code.....	8
AirPlane.m.....	8
AirPlaneDerivatives.m	11
RigidBodySolver.m	13
SCT.m.....	15
Main.m.....	15

A. Design a “Yaw damper” for the Dutch Roll Mode

Design No. One

Then the open loop transfer function is

OL_r_rcom =

$$\frac{-126 s^3 - 599.3 s^2 - 168.9 s - 67.62}{s^5 + 15.41 s^4 + 69.2 s^3 + 200.4 s^2 + 492.9 s + 2.303}$$

Continuous-time transfer function.

Design control loop:

yaw_C1_tf	yaw_C2_tf	Architecture
0.065926	$\frac{0.21623 s}{(s+3.281)}$	

Closed Loop transfer function:

CL_r_rcom_tf =

From input "r" to output "y":

$$\frac{-8.307 s^4 - 66.76 s^3 - 140.7 s^2 - 40.98 s - 14.62}{s^6 + 18.69 s^5 + 155.3 s^4 + 623.8 s^3 + 1328 s^2 + 1675 s + 22.18}$$

Control action transfer function:

yaw_C_action_tf =

From input "r" to output "u":

$$0.06593 s^6 + 1.232 s^5 + 7.895 s^4 + 28.18 s^3 + 75.84 s^2 + 106.7 s + 0.4981$$

$$s^6 + 18.69 s^5 + 155.3 s^4 + 623.8 s^3 + 1328 s^2 + 1675 s + 22.18$$

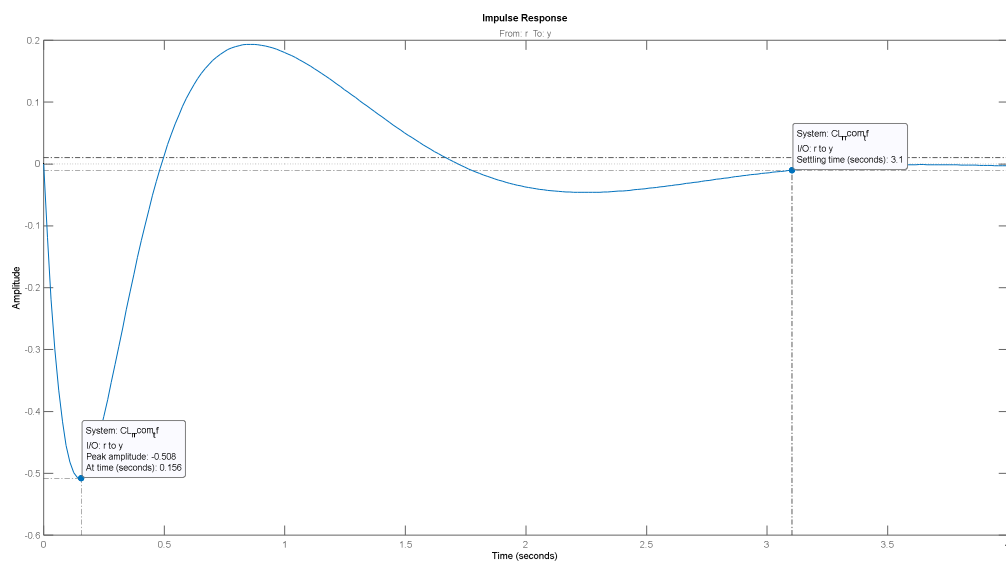


Figure 1. Design 1 - Response

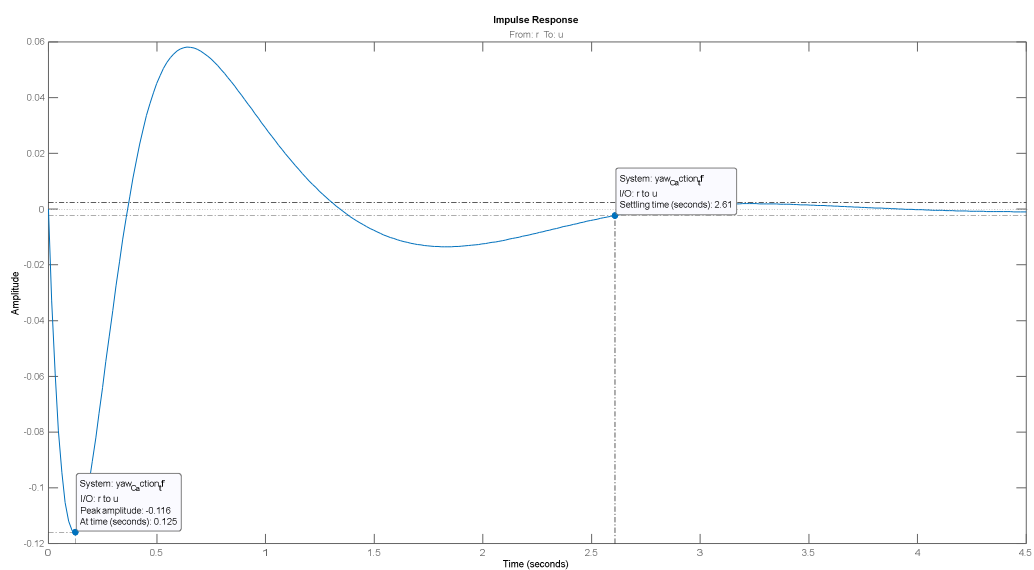


Figure 2. Design 1 - Control Action

Design No. Two

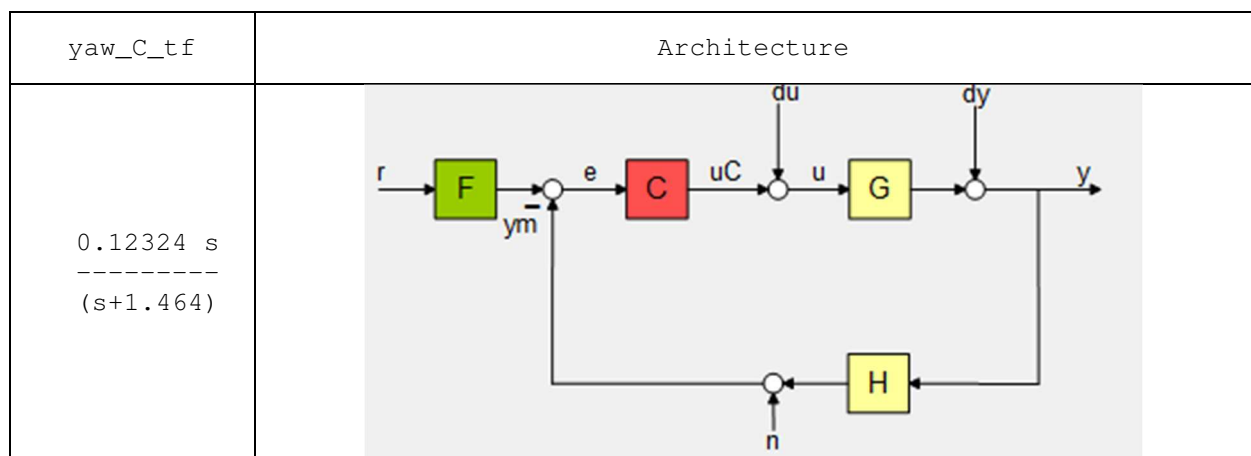
Then the open loop transfer function is

OL_r_rcom =

$$\frac{-126 s^3 - 599.3 s^2 - 168.9 s - 67.62}{s^5 + 15.41 s^4 + 69.2 s^3 + 200.4 s^2 + 492.9 s + 2.303}$$

Continuous-time transfer function.

Design control loop:



Closed Loop transfer function:

CL_r_rcom_tf =

From input "r" to output "y":

$$\frac{-126 s^4 - 783.7 s^3 - 1046 s^2 - 314.8 s - 98.98}{s^6 + 16.87 s^5 + 107.3 s^4 + 375.6 s^3 + 807.1 s^2 + 732.1 s + 3.371}$$

Control action transfer function:

yaw_C_action_tf =

From input "r" to output "u":

$$s^6 + 16.87 s^5 + 91.76 s^4 + 301.7 s^3 + 786.3 s^2 + 723.7 s + 3.371$$

$$s^6 + 16.87 s^5 + 107.3 s^4 + 375.6 s^3 + 807.1 s^2 + 732.1 s + 3.371$$

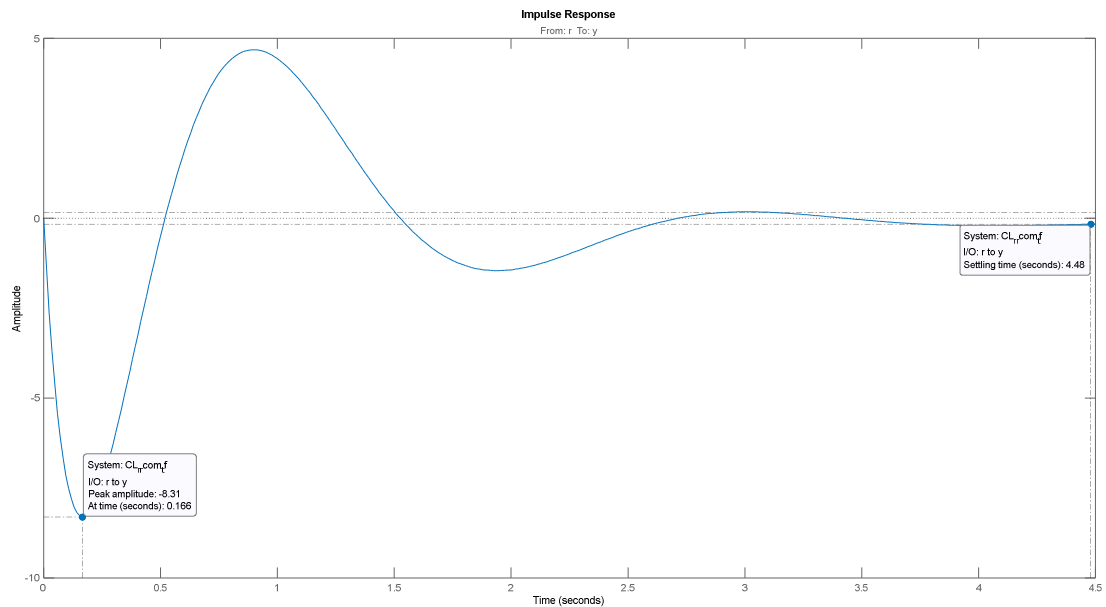


Figure 3. Design 2 - Response

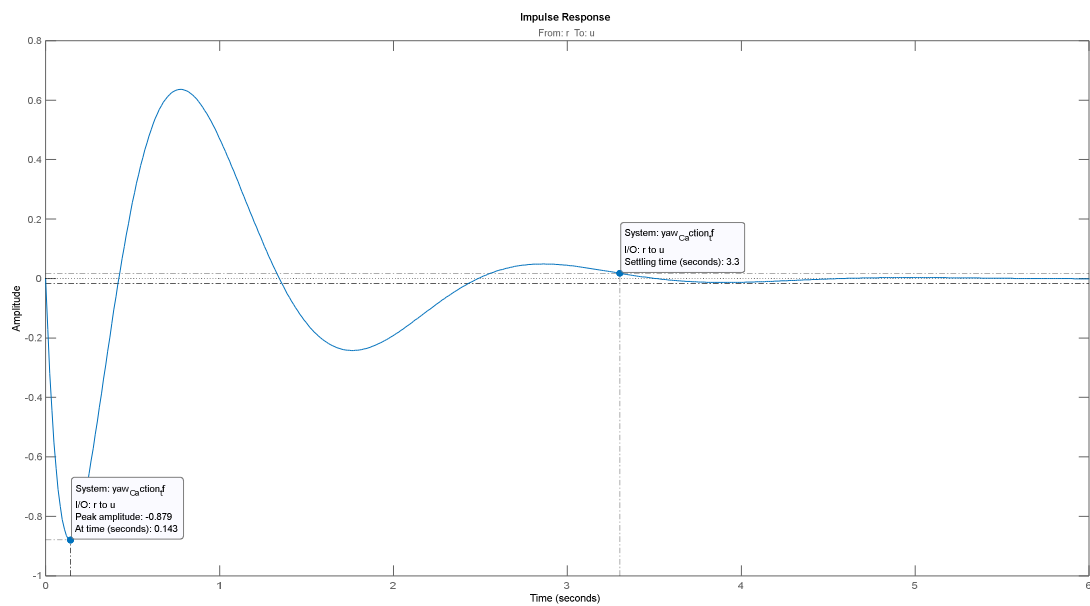
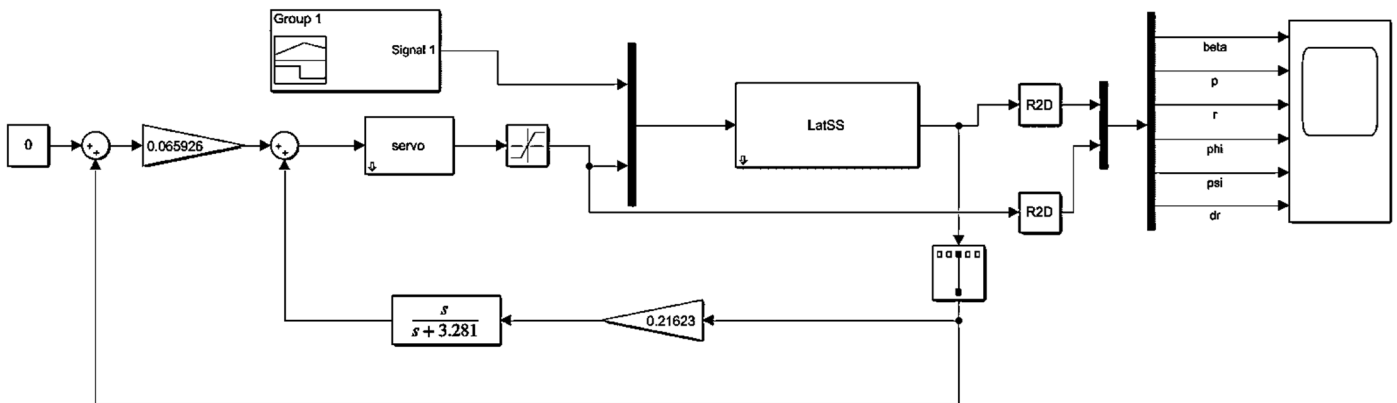


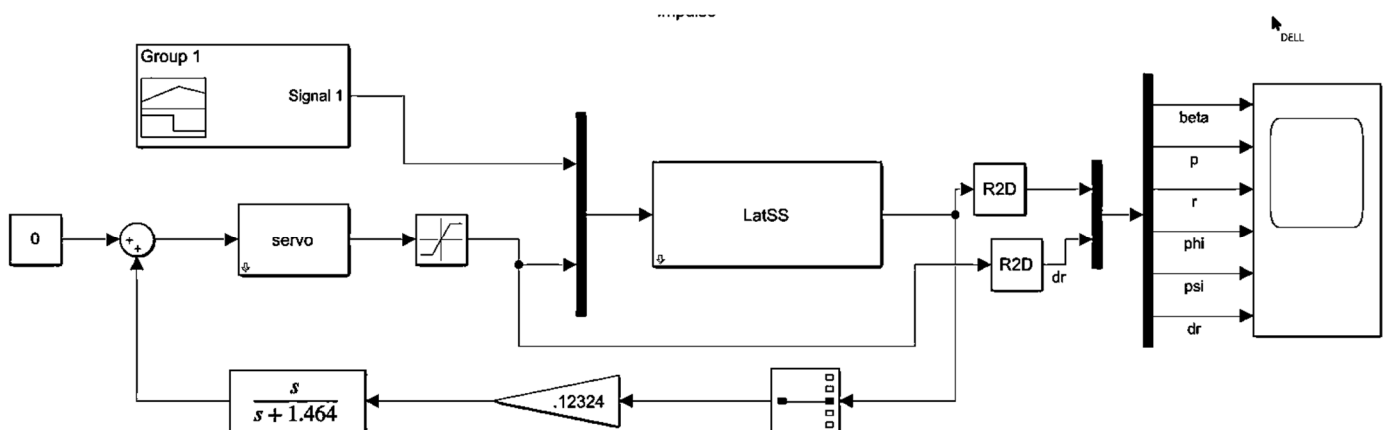
Figure 4. Design 2 - Control Action

B. Test The “Yaw damper” controllers on the full state space model

Design No. One

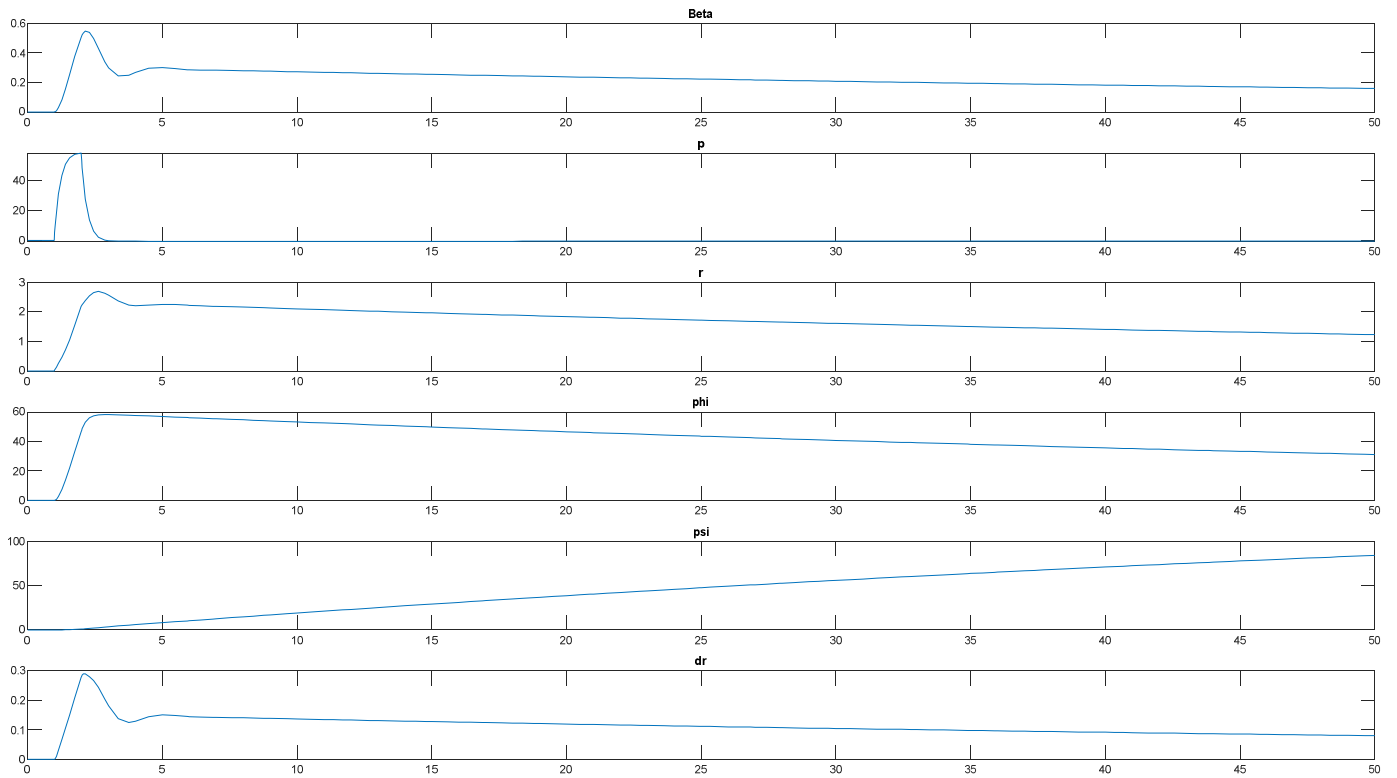


Design No. Two

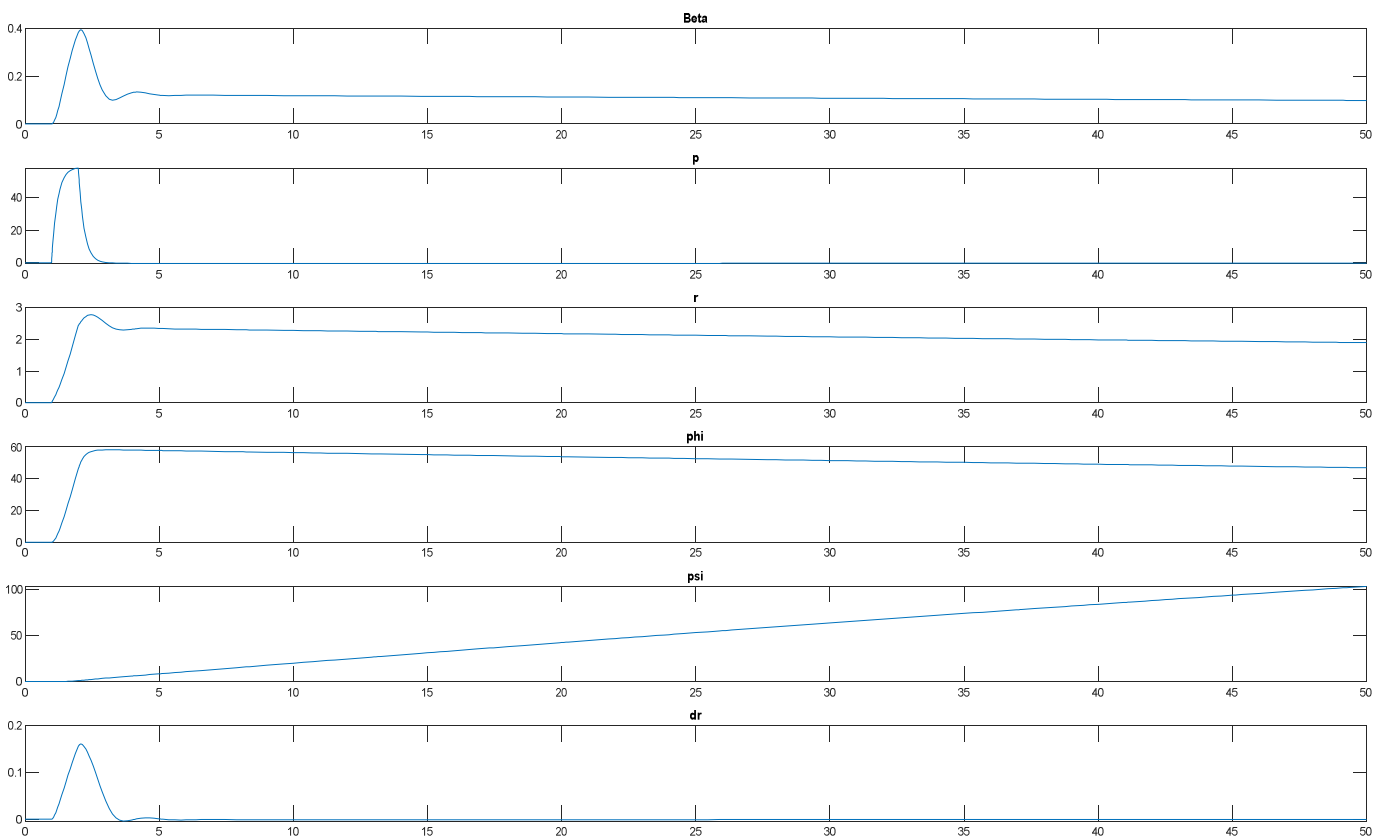


Results:

Design No. One



Design No. Two



Appendix: Code

AirPlane.m

```

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

    properties
        Mass
        g
        I          % Inertia
        invI        % Inverse of Inertia
        timeSpan
        dt
        ICs
        ICs_dot0
        Vt0
        dControl
        SD_Long
        SD_Lat
        SD_Lat_dash
        initialGravity
        airPlaneDerivatives    % Class
        rigidBodySolver        % Class

        u0, v0, w0, theta0, z0

    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);
Iyy = aircraft_data(54);
Izz = aircraft_data(55);
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);

end
function [dForce, dMoment] = airFrame1(obj, state, forces,
moments, dControl)

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

Ixx = obj.I(1,1);
Iyy = obj.I(2,2);
Izz = obj.I(3,3);

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

dM = 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 = 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
        % Longitudinal
        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, NDA, 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.NDA, 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 [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.NDA;

            [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_epsilon = S.epsi;
    c_epsilon = 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_epsilon_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_epsilon, (s_phi*s_theta*c_epsilon - c_phi*s_epsilon),
(c_phi*s_theta*c_epsilon + s_phi*s_epsilon);
        c_theta*s_epsilon, (s_phi*s_theta*s_epsilon + c_phi*c_epsilon),
(c_phi*s_theta*s_epsilon - s_phi*c_epsilon);
        -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_epsilon_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
% Forces, Moments and Inertia
plane = AirPlane("NT-33A_4.xlsx");

steps = (plane.timeSpan(2) - plane.timeSpan(1))/plane.dt;
Result = NaN(12, steps);
Result(:,1) = plane.ICs;
time_V = linspace(0, plane.timeSpan(2), steps+1);

%% Servo 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]);

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Lateral Full Linear Model %%%%%%%%%
[A_lat, B_lat, C_lat, D_lat] = plane.lateralFullLinearModel();
LatSS = ss(A_lat, B_lat, C_lat, D_lat);
LatTF = tf(LatSS);

%% Design a "Yaw damper" for the Dutch roll mode
R_DR_L = LatTF(3, 2);
OL_r_rcom=servo*R_DR_L;

yawDamperControldesignValues =
matfile("DesignValues/yawDamperControlDesignValues-Design1.mat");

yaw_C2_tf = yawDamperControldesignValues.C2;
yaw_C1_tf = yawDamperControldesignValues.C1;
CL_r_rcom_tf = tf(yawDamperControldesignValues.IOTransfer_r2y);
yaw_C_action_tf = tf(yawDamperControldesignValues.IOTransfer_r2u);

```



```

figure;
impulse(CL_r_rcom_tf)
figure;
impulse(yaw_C_action_tf)

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Longitudenal Full Linear Model %%%%%%%%%%
% Two Inputs - Four Output Each
[A_long, B_long, C_long, D_long] = plane.fullLinearModel();
LongSS = ss(A_long, B_long, C_long, D_long);
LongTF = tf(LongSS);

%% pitch control theta/theta_com
theta_dE = LongTF(4,1);
OL_theta_thetacom = -servo * theta_dE;
pitchControldesignValues =
matfile("DesignValues/pitchControldesignValues.mat");

pitch_PD_tf = pitchControldesignValues.C2;
pitch_PI_tf = pitchControldesignValues.C1;
CL_theta_thetacom_tf = tf(pitchControldesignValues.IOTransfer_r2y);
pitch_C_action_tf = tf(pitchControldesignValues.IOTransfer_r2u);

% figure;
% step(CL_theta_thetacom_tf)
% figure;
% step(pitch_C_action_tf)

%% Velocity Controller u/u_com
u_dTh = LongTF(1, 2);
OL_u_ucom = u_dTh * servo * engine_timelag;
velocityControldesignValues =
matfile("DesignValues/velocityControldesignValues.mat");

velocity_C2_tf = velocityControldesignValues.C2;
velocity_C1_tf = velocityControldesignValues.C1;
CL_u_ucom_tf = tf(velocityControldesignValues.IOTransfer_r2y);
velocity_C_action_tf = tf(velocityControldesignValues.IOTransfer_r2u);

% figure;
% step(CL_u_ucom_tf)
% figure;
% step(velocity_C_action_tf)

%% Altitude Controller h/thetacom
w_de = LongTF(2,1);
theta_de = LongTF(4, 1);
w_theta = minreal(w_de/theta_de);
h_theta = -1 * integrator * (w_theta - plane.u0);
OL_h_thetacom = minreal(CL_theta_thetacom_tf * h_theta);

altitudeControldesignValues =
matfile("DesignValues/altitudeControldesignValues.mat");

altitude_C2_tf = altitudeControldesignValues.C2;
altitude_C1_tf = altitudeControldesignValues.C1;
CL_h_thetacom_tf = tf(altitudeControldesignValues.IOTransfer_r2y);
altitude_C_action_tf = tf(altitudeControldesignValues.IOTransfer_r2u);

```

```
% figure;  
% step(CL_h_thetacom_tf)  
% figure;  
% step(altitude_C_action_tf)
```