Contents

[A. Design a “Yaw damper” for the Dutch Roll Mode 2](#_Toc101913655)

[Design No. One 2](#_Toc101913656)

[Design No. Two 4](#_Toc101913657)

[B. Test The “Yaw damper” controllers on the full state space model 6](#_Toc101913658)

[Design No. One 6](#_Toc101913659)

[Design No. Two 6](#_Toc101913660)

[Results: 7](#_Toc101913661)

[Design No. One 7](#_Toc101913662)

[Design No. Two 7](#_Toc101913663)

[Appendix: Code 8](#_Toc101913664)

[AirPlane.m 8](#_Toc101913665)

[AirPlaneDerivatives.m 11](#_Toc101913666)

[RigidBodySolver.m 13](#_Toc101913667)

[SCT.m 15](#_Toc101913668)

[Main.m 15](#_Toc101913669)

# Design a “Yaw damper” for the Dutch Roll Mode

## Design No. One

Then the open loop transfer function is

OL\_r\_rcom =

-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 | 0.21623 s  ---------  (s+3.281) |  |

Closed Loop transfer function:

CL\_r\_rcom\_tf =

From input "r" to output "y":

-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 =

-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\_C\_tf | Architecture |
| 0.12324 s  ---------  (s+1.464) |  |

Closed Loop transfer function:

CL\_r\_rcom\_tf =

From input "r" to output "y":

-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

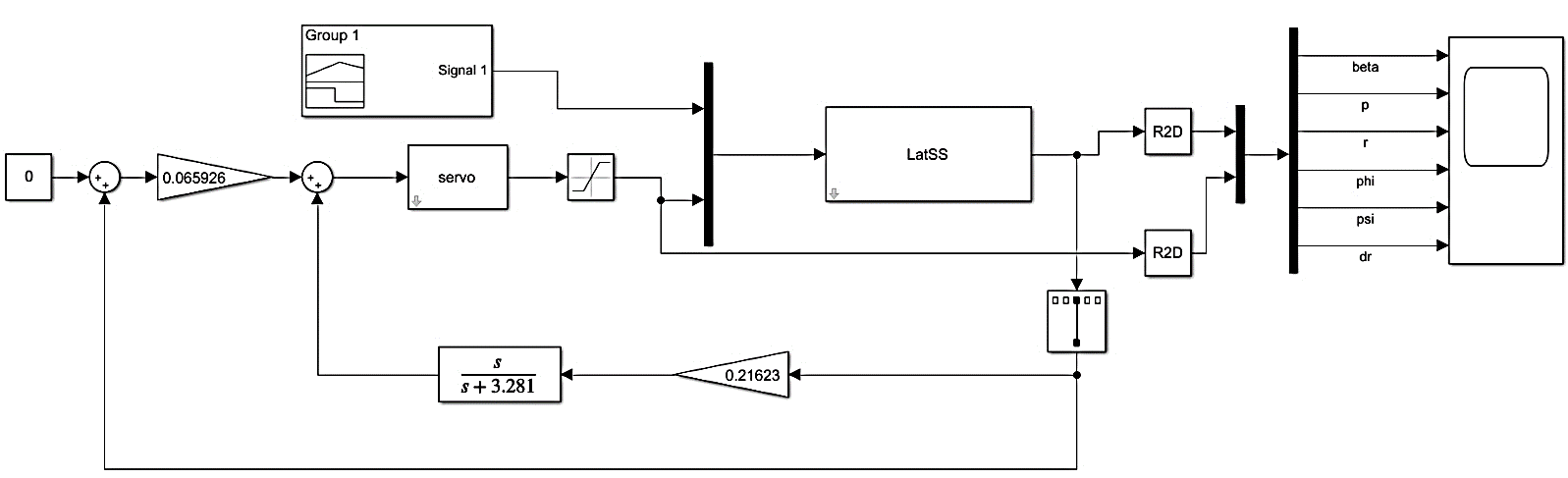
---------------------------------------------------------------------

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

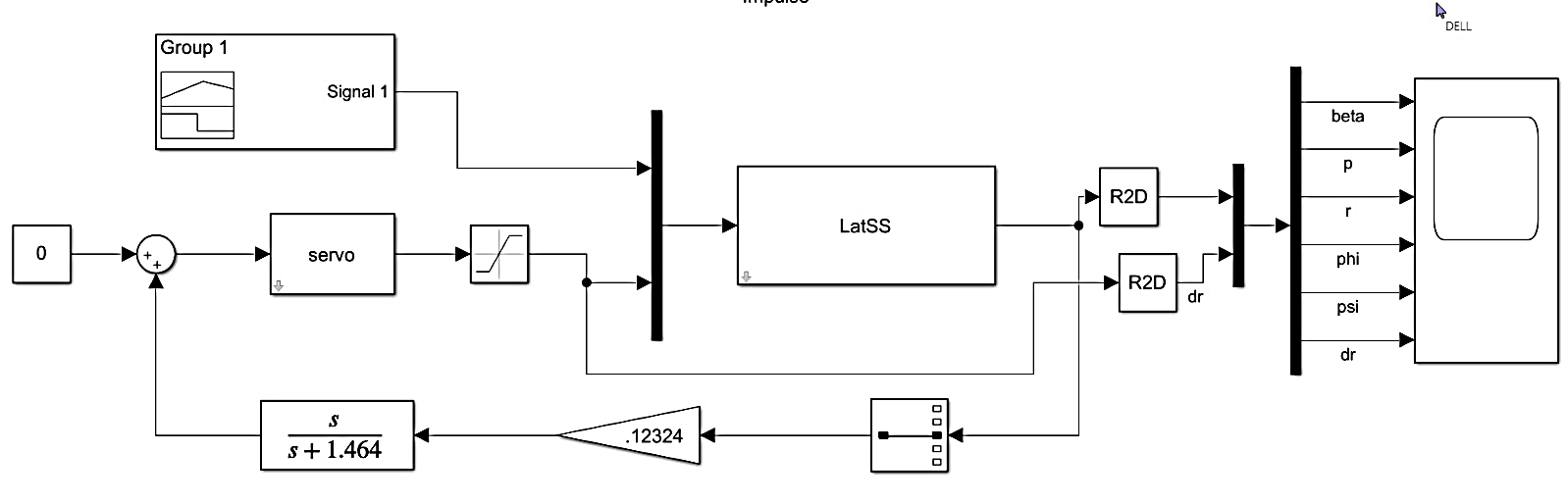
|  |
| --- |
| Figure 3. Design 2 - Response |
| Figure 4. Design 2 - Control Action |

# 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 % Inirtia  invI % Inverse of Inirtia  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  % 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 [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  % 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) |