Contents

[A. Develop The testing loop containing the “Controller + Simulator” 2](#_Toc103738431)

[B. Test the “Pitch controller” and compare the response with the same test on the State space model 4](#_Toc103738432)

[Linear block 4](#_Toc103738433)

[Results for 33.5 deg pitch angle 5](#_Toc103738434)

[1-u 5](#_Toc103738435)

[2- 6](#_Toc103738436)

[3- 6](#_Toc103738437)

[4- 7](#_Toc103738438)

[5-h 7](#_Toc103738439)

[C. Test the “Pitch controller + Velocity controller” and compare the 8](#_Toc103738440)

[response with the same test on the State space model 8](#_Toc103738441)

[Simulink Non-linear simulator: 8](#_Toc103738442)

[Simulink linear simulator: 10](#_Toc103738443)

[Simulation Results for 15° pitch command: 11](#_Toc103738444)

[D. Test The “Altitude Hold” Controller and Compare the response with the same test on the state space model 14](#_Toc103738445)

[appendix: Code 17](#_Toc103738446)

[AirPlane.m 17](#_Toc103738447)

[AirPlaneDerivatives.m 20](#_Toc103738448)

[RigidBodySolver.m 23](#_Toc103738449)

[SCT.m 24](#_Toc103738450)

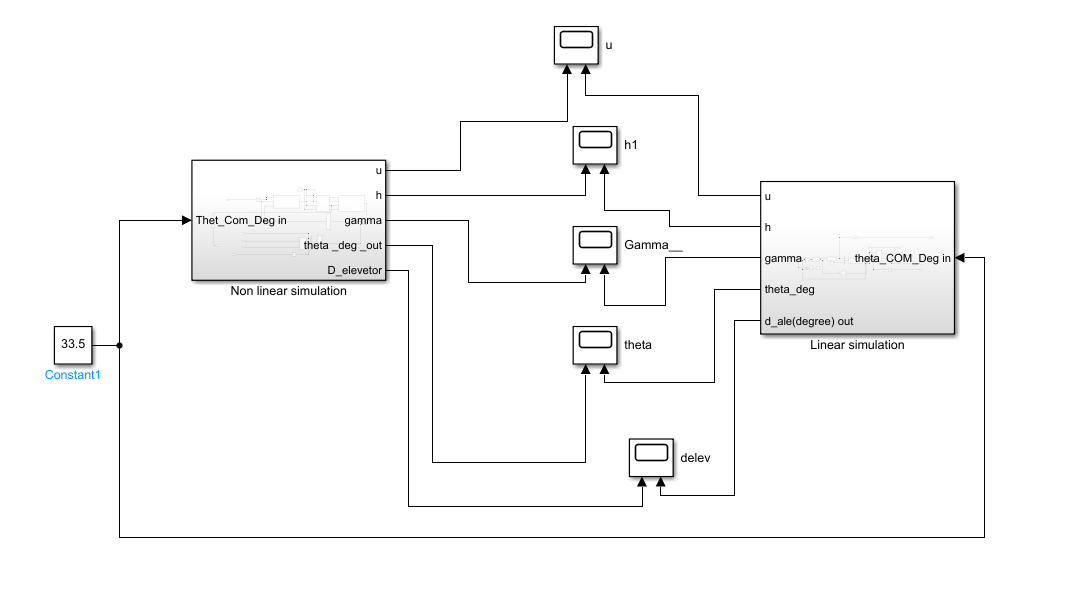
[Main.m 25](#_Toc103738451)

# Develop The testing loop containing the “Controller + Simulator”

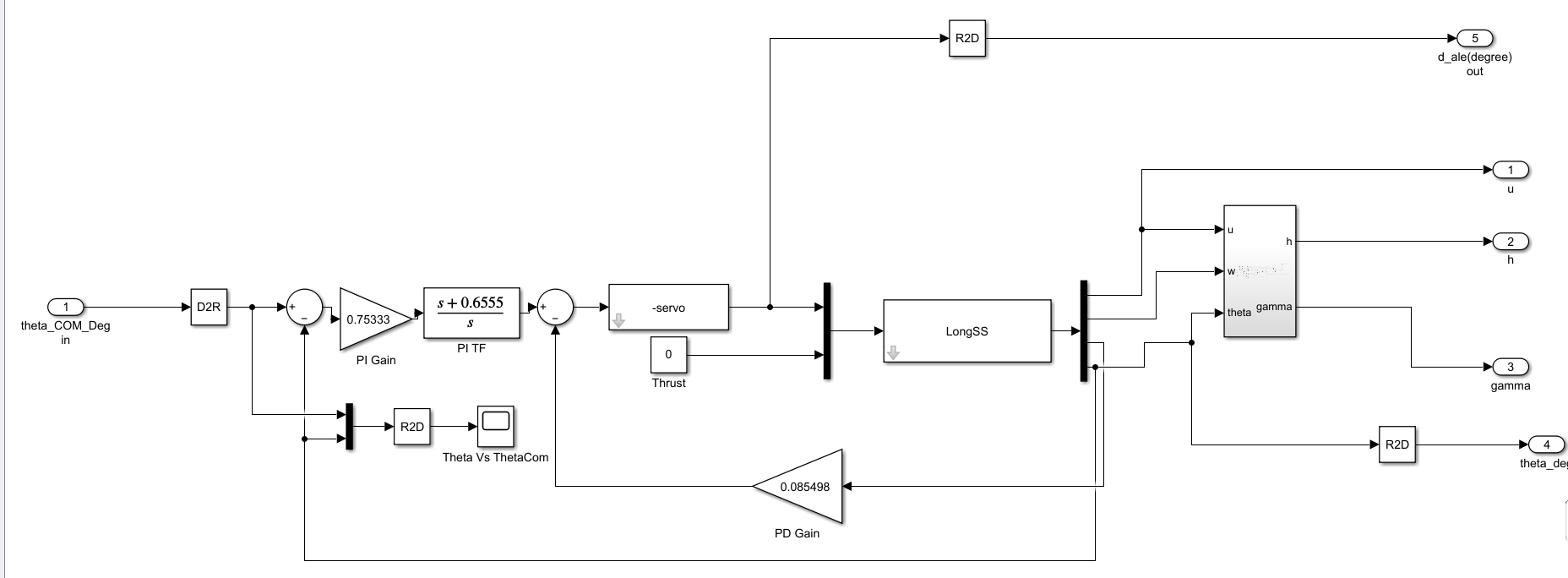
|  |
| --- |
| C:\Users\OWNER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sim1.jpg  Figure 1. Controllers + Non Linear Simulator |
| C:\Users\OWNER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sim2.jpg  Figure 2. Longitudinal Controller |

|  |
| --- |
| C:\Users\OWNER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\lateral controller.jpg  Figure 3. Lateral Controller |
| C:\Users\OWNER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\non linear simulator.jpg  Figure 4. Non Linear Simulator |

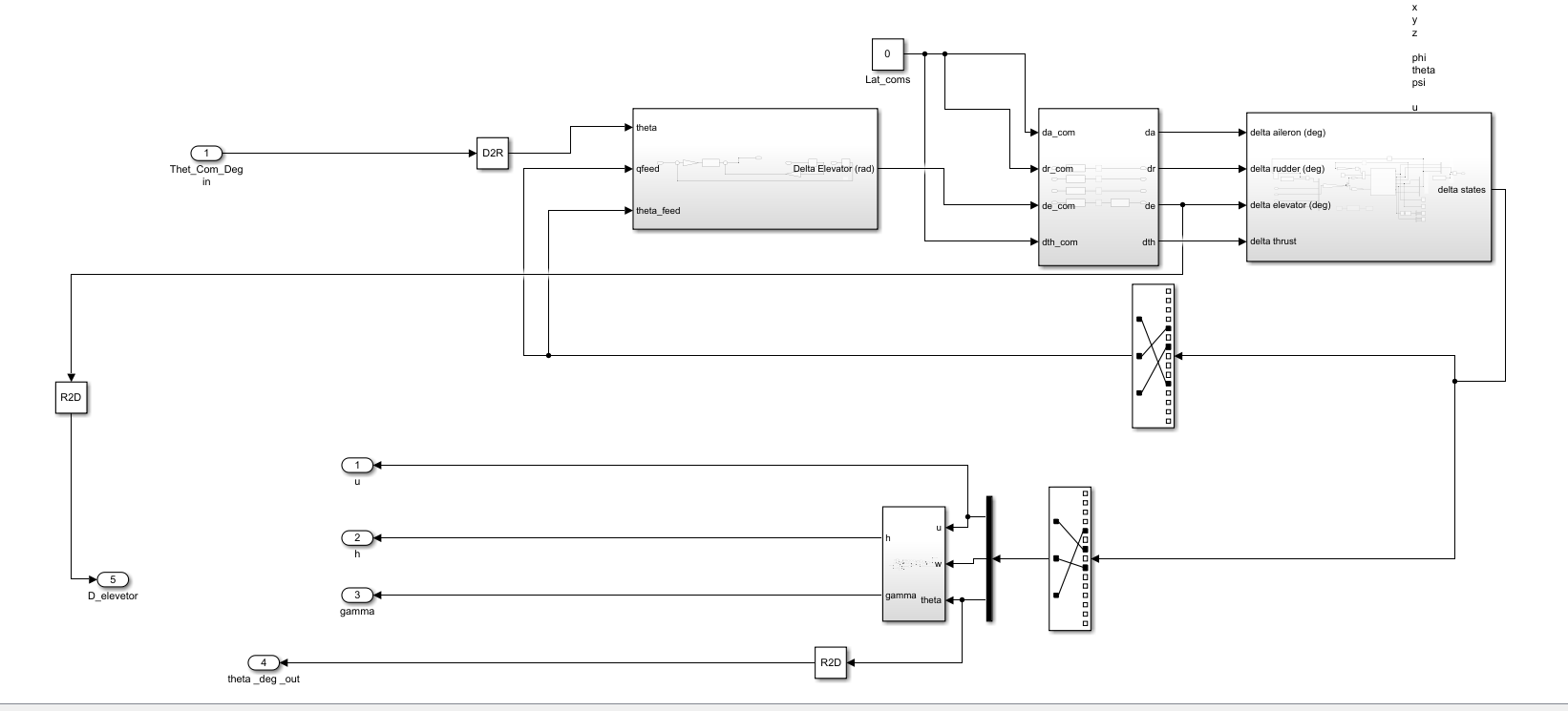
# Test the “Pitch controller” and compare the response with the same test on the State space model



## Linear block



Nonlinear block



## Results for 33.5 deg pitch angle

### 1-u



### 2-



## 3-



### 4-



### 5-h

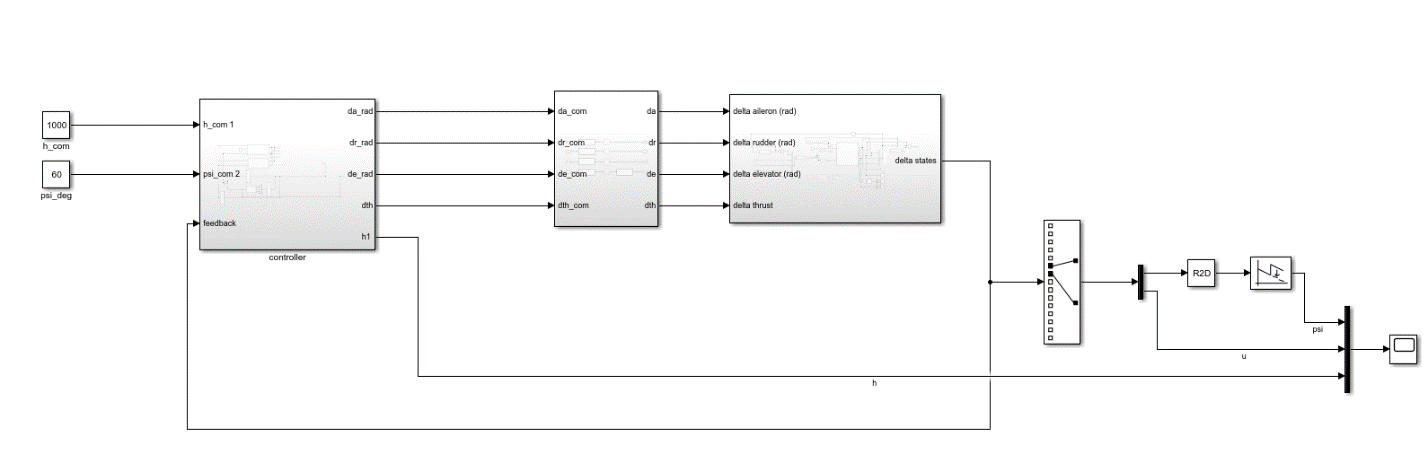


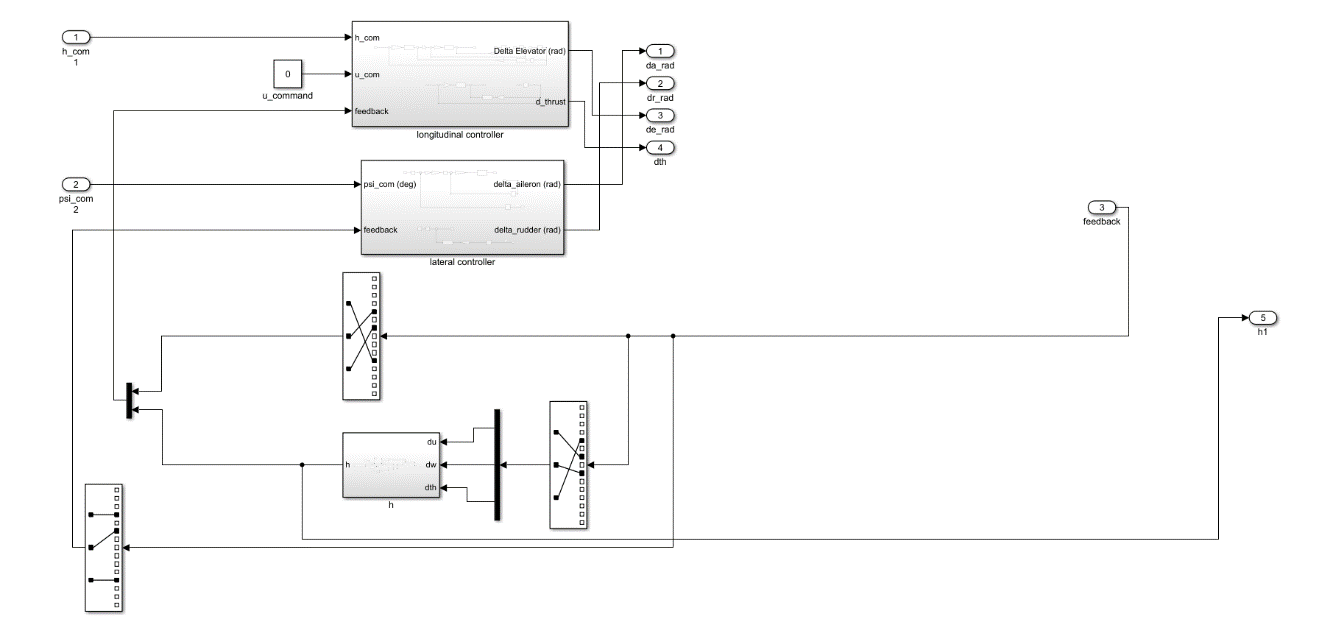
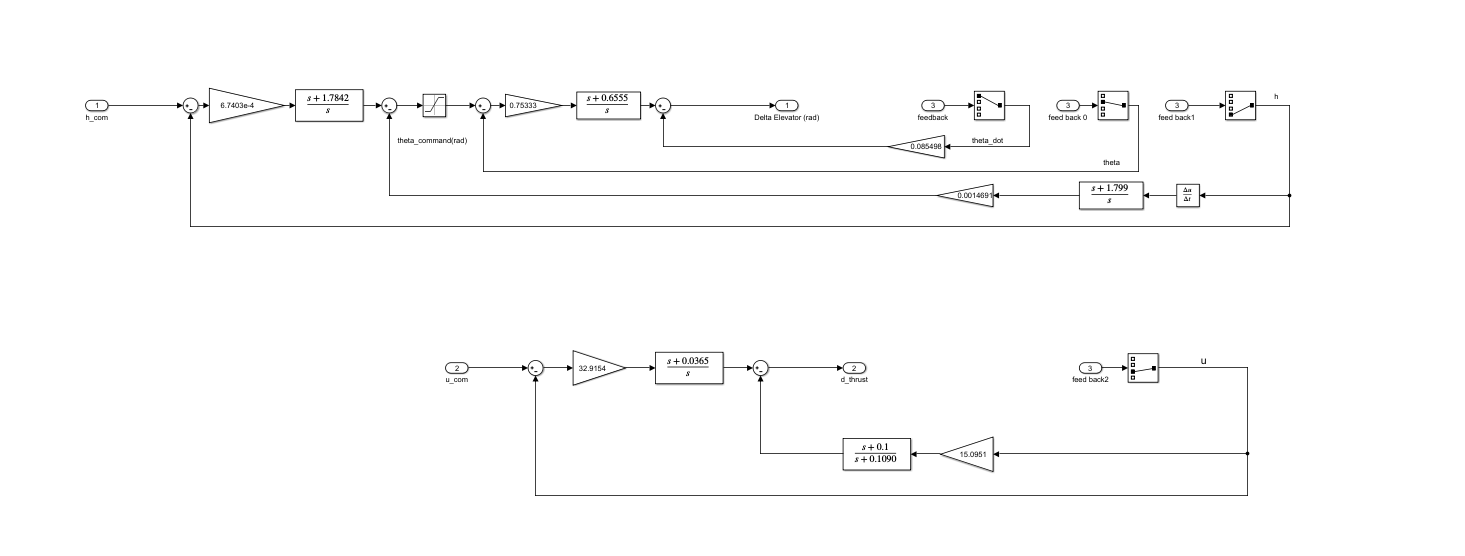
# Test the “Pitch controller + Velocity controller” and compare the

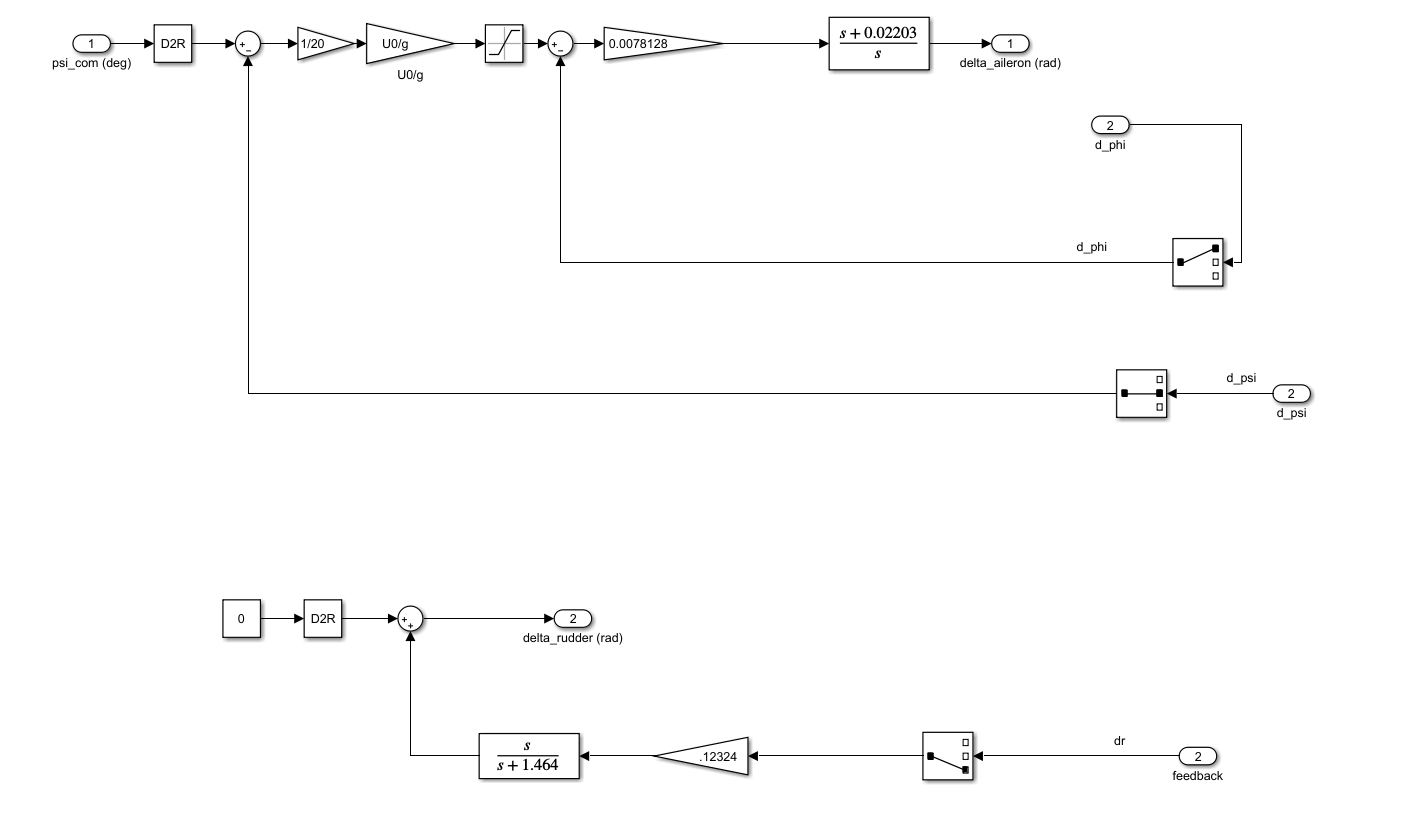
# response with the same test on the State space model

* Give a commanded input signal of pitch angle and observe the response and control action .
* Perform the same test on the Linear Longitudinal state space model and observe the response and control action
* Plot the results against each other.
* Note: in this test all the control actions are set to zero except the 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.

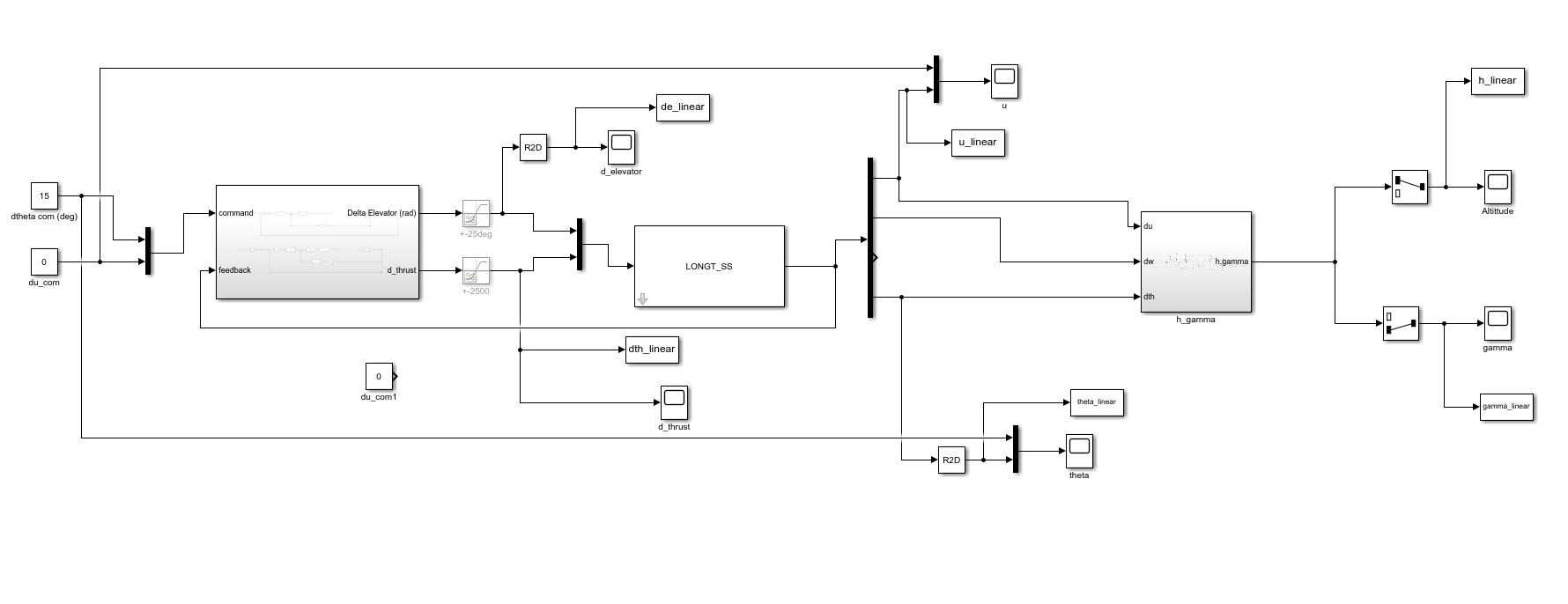
## Simulink Non-linear simulator:

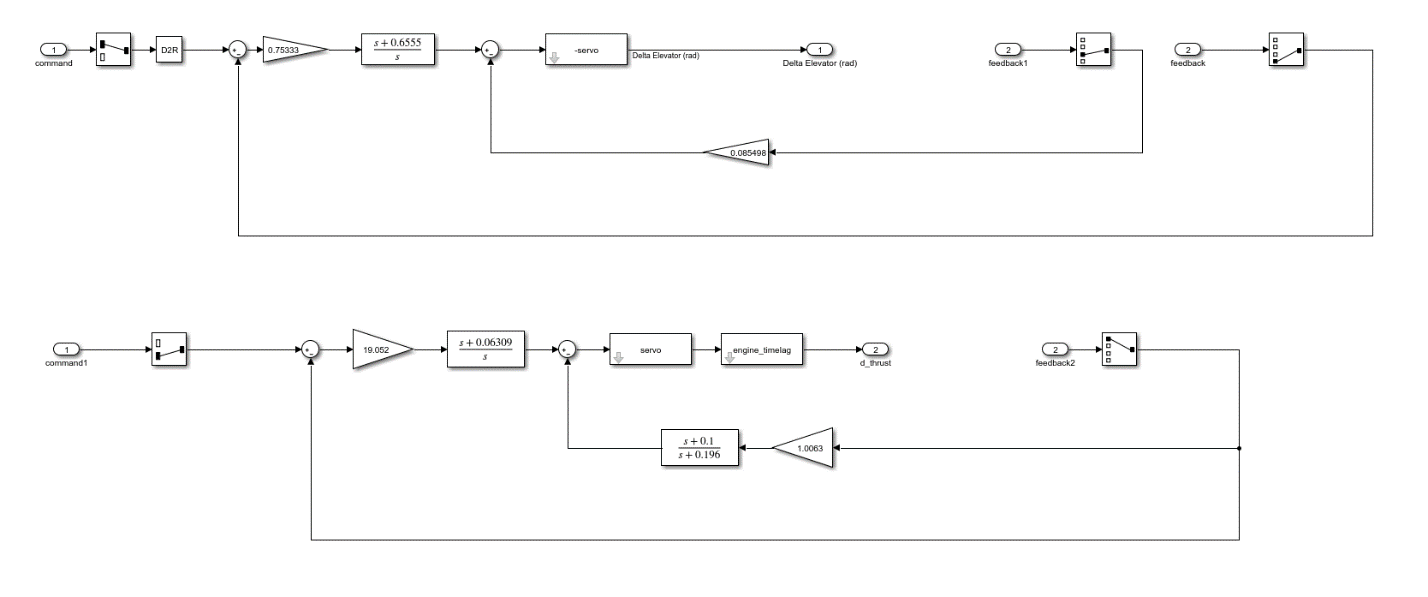


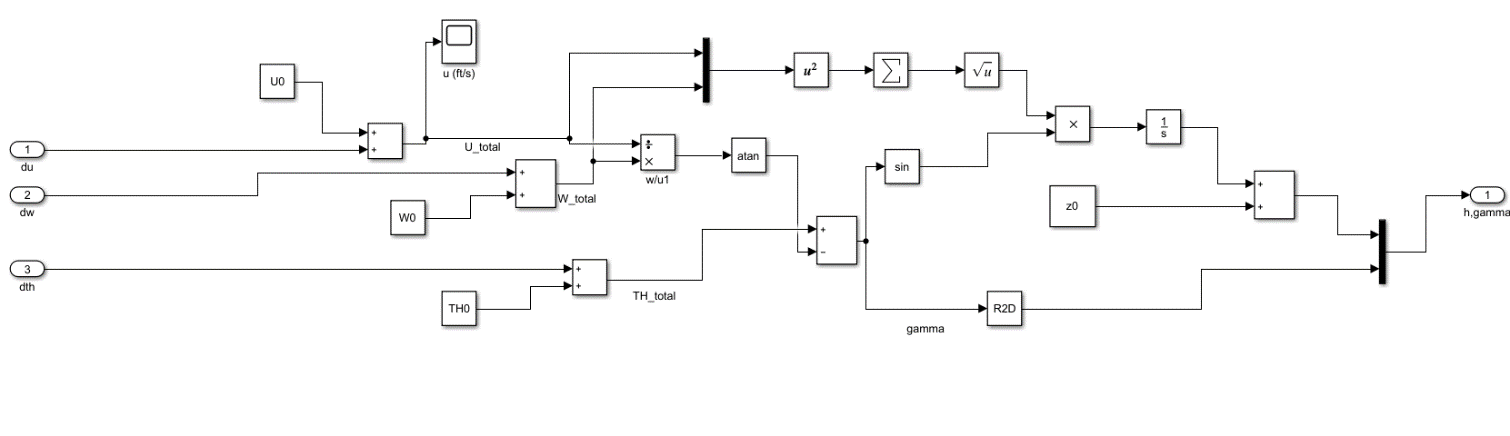
 



## Simulink linear simulator:







## Simulation Results for 15° pitch command:











# Test The “Altitude Hold” Controller and Compare the response with the same test on the state space model

Results for input command of 1000ft.

|  |
| --- |
| C:\Users\OWNER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\delta_theta.emf |
| C:\Users\OWNER\AppData\Local\Microsoft\Windows\INetCache\Content.Word\delta_u.emf |
| Y:\Aerospace\AeroSpace 4\Term 2\AutoPilot (Dr. Osama Saaid)\Tasks\Task 7\code\Results\AltitudeHold\delta_gamma.emf |
| Y:\Aerospace\AeroSpace 4\Term 2\AutoPilot (Dr. Osama Saaid)\Tasks\Task 7\code\Results\AltitudeHold\altitude.emf |
| sY:\Aerospace\AeroSpace 4\Term 2\AutoPilot (Dr. Osama Saaid)\Tasks\Task 7\code\Results\AltitudeHold\delta_elevetor.emf |
| Y:\Aerospace\AeroSpace 4\Term 2\AutoPilot (Dr. Osama Saaid)\Tasks\Task 7\code\Results\AltitudeHold\delta_thrust.emf |

# 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)') |