Cairo University Faculty of
Engineering Aerospace
Department

AEROSPACE ENGINEERING

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

Task (1)

Autopilot literature review

Research questions

a) What is an autopilot?

An autopilot is a system that allows an airplane, marine vehicle, or spaceship to navigate without the need for constant manual control by a human operator. Human operators are not replaced by autopilots. Instead, the autopilot supports the driver in maintaining vehicle control.

b) What are the inputs & outputs of Autopilot system onboard an airplane?

Inputs:

Control Inputs:

1. Radio Control Receivers
2. MAVLink Data Streams, i.e., ground control stations or companion computers

Sensor inputs:

GPS, Compass, Airspeed, Rangefinders, IMU

Power Management Unit Inputs:

1. Received Signal Strength Input (RSSI)
2. Analog Airspeed Sensors

Outputs:

1. ESCs (electronic circuit that controls and regulates the speed of an electric motor) for motors
2. Servos for control surfaces
3. Telemetry data
4. Actuators and General Purpose I/O like LEDs, buzzers etc

c) What would be the role of the pilot in an airplane equipped with an autopilot?

The pilot sets the flight plan and turn on the autopilot sometimes the pilot reprograms the autopilot in case if worked incorrectly. autopilot is not smart enough to fly a plane by It self

d) What is the difference between Autopilot & SAS?

SAS (Stability augmentation system) generally used during low and slow maneuvering where the pilot may be making constant attitude changes in preparation for landing.

Autopilot do same functions as SAS in addition it provides more functions, Autopilot Is more sophisticated than SAS

e) What is the role of the onboard sensors like (GPS, gyroscopes, ...etc.)? give example.

Sensors provide the autopilot computers with data like speed, coordinates, position so the computers can estimate the states and give the correct control actions to the actuators for example

Gyroscope an IMU supplies the autopilot with position data so if the airplane in incorrect position it gives the actuator signal so that it can adjust position

Flight Mechanics review:

a) State the general rigid body dynamics (RBD) equations in 3D space

Assume a mass element at a rigid body, the velocity of this mass w.r.t fixed axes:

$$\vec{V} = \vec{V_c} + \frac{d\vec{r}}{dt} \qquad (1)$$

By using the calculations of linear momentum:

The force at this point is given by:

$$\delta\vec{F} = \delta m \frac{d\vec{V}}{dt} \qquad (2)$$

Sub. By (1) in (2) and take summation for all point masses is in the rigid body:

$$\vec{F} = \sum \delta\vec{F} = \frac{d}{dt}\sum(\vec{V_c} + \frac{d\vec{r}}{dt})\delta m$$

$$\vec{F} = m\frac{d\vec{V_c}}{dt} + \frac{d}{dt}\sum\frac{d\vec{r}}{dt}\,\delta m = m\frac{d\vec{V_c}}{dt} + \frac{d^2}{dt^2}\sum r\,\delta m$$

Since: $r\delta m = 0$

$$\therefore \vec{F} = m\frac{d\vec{V_c}}{dt}$$

From general definition:

$$\frac{d\vec{A}}{dt}_I = \frac{d\vec{A}}{dt}_B + \omega \times \vec{A}$$

So:

$$\vec{F} = m\frac{d\vec{V_c}}{dt} + m(\omega \times \vec{V_c})$$

Where:

$$\vec{F} = F_x\vec{\imath} + F_y\vec{\jmath} + F_z\vec{k}, \quad \vec{\omega} = P\vec{\imath} + q\vec{\jmath} + r\vec{k}, \quad \vec{V_c} = u\boldsymbol{i} + v\boldsymbol{j} + w\boldsymbol{k}$$

So:

$$F_x = m(\dot{u} + qw - rv)$$

$$F_y = m(\dot{v} + ru - pw)$$

$$F_z = m(\dot{w} + pv - qu)$$

By using the calculations of angular momentum:

$$\delta\vec{M} = \frac{d}{dt}\delta\vec{H} = \frac{d}{dt}(\vec{r} \times \vec{V})\delta m$$

By using the upper general definition:

$$\frac{d\vec{r}}{dt}_I = \frac{d\vec{r}}{dt}_B \overset{0}{} + \omega \times \vec{r}$$

$$Then: \qquad \vec{H} = \sum \delta\vec{H} = \sum(\vec{r} \times \vec{V_c})\delta m + \sum[r \times (\omega \times \vec{r})]\,\delta m$$

Where: $\vec{r} = x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}$

Then:

$$H_x = p\sum(y^2 + z^2)\,\delta m - q\sum xy\,\delta m - r\sum xz\,\delta m$$

$$H_y = -p\sum xy\,\delta m + q\sum(x^2 + z^{)2}\,\delta m - r\sum yz\,\delta m$$

$$H_z = -p\sum xz\,\delta m + q\sum yz\delta m - r\sum(x^2 + y^2)\,\delta m$$

By expressing the above equations in terms of moment of inertia about body axes:

$$H_x = pI_x - qI_{xy} - rI_{xz}$$

$$H_y = -pI_{xy} + qI_y - rI_{yz}$$

$$H_z = -pI_{xz} - qI_{yz} + rI_z$$

Since:

$$\vec{M} = \frac{d}{dt}\vec{H} = \frac{d\vec{H}}{dt}_B + \vec{\omega} \times \vec{H}$$

$$L = \dot{H}_x + qH_z - rH_y$$

$$M = \dot{H}_y + rH_x - pH_z$$

$$N = \dot{H}_z + pH_y - qH_x$$

we are dealing with an aircraft which means $I_{xy} = I_{yz} = 0$ $(symmetry\ cond.)$

Then: The airplane translational and rotational EOM are:

$$F_x = m(\dot{u} + qw - rv)$$

$$F_y = m(\dot{v} + ru - pw)$$

$$F_z = m(\dot{w} + pv - qu)$$

$$L = I_x\dot{p} - I_{xz}\dot{r} + qr(I_z - I_y) - I_{xz}pq$$

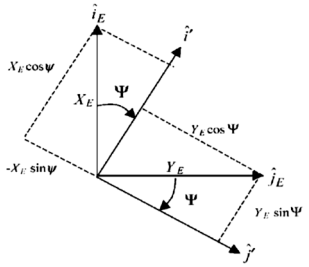
$$M = y\dot{q} + rp(I_x - I_z) + I_{xz}(p^2 - r^2)$$

$$N = -I_{xz}\dot{p} + I_z\dot{r} + pq(I_y - I_x) + I_{xz}qr$$

By adding the calculations of earth axis to body axis transformation:

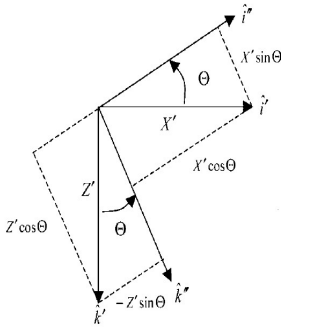
$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_E \\ Y_E \\ Z_E \end{bmatrix}$$

$$\boldsymbol{F_1} = R_3(\psi)\boldsymbol{F_E}$$



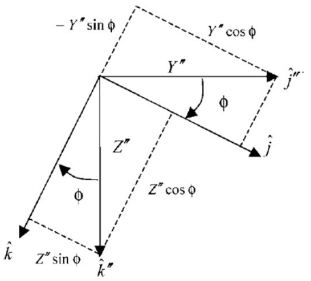
$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix}$$

$$\boldsymbol{F_2} = R_2(\theta)\boldsymbol{F_1} = R_2(\theta)R_3(\psi)\boldsymbol{F_E}$$



$$\begin{bmatrix} X_B \\ Y_B \\ Z_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix}$$

$$\boldsymbol{F_B} = R_1(\phi)\boldsymbol{F_2} = R_1(\phi)R_2(\theta)R_3(\psi)\boldsymbol{F_E}$$



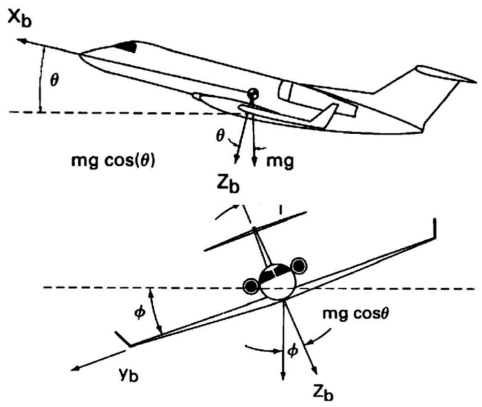
Linear Velocities transformation:

$$\begin{Bmatrix}\dot{X}\\ \dot{Y}\\ \dot{Z}\end{Bmatrix}_E = \begin{bmatrix}C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi\\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi\\ -S_\theta & S_\theta C_\phi & C_\phi C_\theta\end{bmatrix}\begin{Bmatrix}u\\ v\\ w\end{Bmatrix}_B$$

Let's apply the above transformation on the gravitational force of the airplane as following:

$$\vec{F}_{Gravity_B} = \begin{bmatrix}1 & 0 & 0\\ 0 & cos\phi & sin\phi\\ 0 & -sin\phi & cos\phi\end{bmatrix}\begin{bmatrix}cos\theta & 0 & -sin\theta\\ 0 & 1 & 0\\ sin\theta & 0 & cos\theta\end{bmatrix}\begin{bmatrix}cos\psi & sin\psi & 0\\ -sin\psi & cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0\\ 0\\ mg\end{bmatrix}_E$$

$$\vec{F}_{Gravity_B} = \begin{bmatrix}-mgsin\theta\\ mgsin\phi cos\theta\\ mgsin\theta cos\phi\end{bmatrix}_B$$



Then:

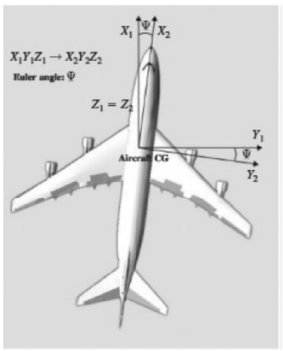
$$-mgsin\theta + F_{T_X} = m(\dot{u} + qw - rv)$$

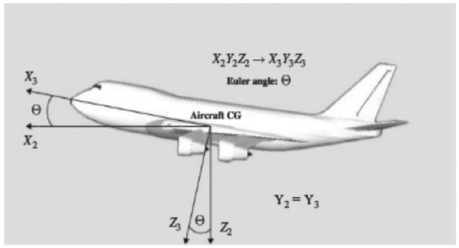
$$mgsin\phi cos\theta + F_{T_Y} = m(\dot{v} + ru - pw)$$

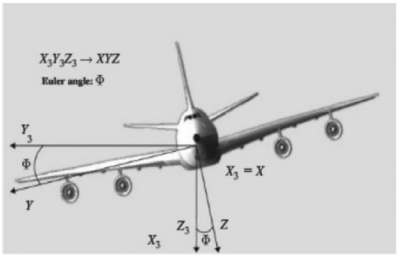
$$mgsin\theta cos\phi + F_{T_Z} = m(\dot{w} + pv - qu)$$

Angular velocities transformation:

$$\bar{\dot{\Psi}} = \dot{\Psi}\bar{K}_1 = \dot{\Psi}\bar{K}_2 \qquad \bar{\dot{\theta}} = \dot{\theta}\bar{j}_2 = \dot{\theta}\bar{j}_3 \qquad \bar{\dot{\Phi}} = \dot{\Phi}\bar{i}_3 = \dot{\Phi}\bar{i}$$







$$\bar{\omega} = \bar{\dot{\Psi}} + \bar{\dot{\theta}} + \bar{\dot{\Phi}} = P\bar{i} + Q\bar{j} + R\bar{k} = \dot{\Psi}\bar{K}_2 + \dot{\theta}\bar{j}_3 + \dot{\Phi}\bar{i}$$

$$\begin{Bmatrix}i_3\\ j_3\\ k_3\end{Bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\Phi & -\sin\Phi\\ 0 & \sin\Phi & \cos\Phi\end{bmatrix}\begin{Bmatrix}i\\ j\\ k\end{Bmatrix}$$

$$j_3 = \cos(\Phi)\,\bar{j} - \sin(\Phi)\,\bar{k}$$

$$k_3 = \sin(\Phi)\,\bar{j} + \cos(\Phi)\,\bar{k}$$

$$k_2 = -\sin(\theta)\,i + \cos(\theta)\,(\sin(\Phi)\,j + \cos(\Phi)\,k)$$

$$= -sin(\theta)i + \cos(\theta)\sin(\Phi)\,j + \cos(\theta)\cos(\phi)\,k$$

$$\bar{\omega} = Pi + Qj + R\bar{k} = \bar{\dot{\Psi}} + \bar{\dot{\theta}} + \bar{\dot{\Phi}} = \dot{\Psi}k_2 + \dot{\theta}j_3 + \dot{\Phi}i$$
$$= \dot{\Psi}(-sin(\theta)i + cos(\theta)\,sin(\Phi)\,j + cos(\theta)\,cos(\phi)\,k)$$
$$+ \dot{\theta}\left(\cos(\Phi)\,\bar{j} - \sin(\Phi)\,\bar{k}\right) + \dot{\Phi}i$$

$$\therefore \begin{Bmatrix} P \\ Q \\ R \end{Bmatrix} = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\phi & cos\theta sin\phi \\ 0 & -sin\phi & cos\phi cos\theta \end{bmatrix} \begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix}$$

b) Classify the upper equations into (Kinetics & Kinematics) equations

Kinetics equations	Kinematics equations
$-mgsin\theta + F_{T_X} = m(\dot{u} + qw - rv)$ $mgsin\phi cos\theta + F_{T_Y} = m(\dot{v} + ru - pw)$ $mgsin\theta cos\phi + F_{T_Z} = m(\dot{w} + pv - qu)$ $L = I_x\dot{p} - I_{xz}\dot{r} + qr(I_z - I_y) - I_{xz}pq$ $M = y\dot{q} + rp(I_x - I_z) + I_{xz}(p^2 - r^2)$ $N = -I_{xz}\dot{p} + I_z\dot{r} + pq(I_y - I_x) + I_{xz}qr$	$\begin{Bmatrix} P \\ Q \\ R \end{Bmatrix} = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\phi & cos\theta sin\phi \\ 0 & -sin\phi & cos\phi cos\theta \end{bmatrix} \begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix}$ $\begin{Bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{Bmatrix}_E = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\theta C_\theta & C_\phi C_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}_B$

c) What are the assumptions introduced while deriving those equations?

The assumptions are:

I. The body is rigid which means the mass is constant and there is no change in geometry (Stress calculations are neglected).
II. Fix a body axis at the center of mass of the rigid body.

d) State the set of equations added to the (RBD) equations to form the Fixed wing Airplanes (EOM)

The additional EOM of airplanes:

→ The effect of the control surfaces: δ_e , δ_a , δ_r , δ_T

I. The change in forces:

$$\Delta X = m\left(\frac{\partial X}{\partial u}\,\Delta u + \frac{\partial X}{\partial w}\,\Delta w + \frac{\partial X}{\partial \delta_e}\,\Delta\delta_e + \frac{\partial X}{\partial \delta_T}\,\Delta\delta_T\right)$$

$$\Delta Y = m\left(\frac{\partial Y}{\partial v}\,\Delta v + \frac{\partial Y}{\partial \beta}\,\Delta\beta + \frac{\partial Y}{\partial \delta_a}\,\Delta\delta_a + \frac{\partial Y}{\partial \delta_r}\,\Delta\delta_r\right)$$

$$\Delta Z = m\left(\frac{\partial Z}{\partial u}\,\Delta u + \frac{\partial Z}{\partial w}\,\Delta w + \frac{\partial Z}{\partial \dot{w}}\,\Delta\dot{w} + \frac{\partial Z}{\partial q}\,\Delta q + \frac{\partial Z}{\partial \delta_e}\,\Delta\delta_e + \frac{\partial Z}{\partial \delta_T}\,\Delta\delta_T\right)$$

II. The change in moments:

$$\Delta L = I_{xx}\left(\frac{\partial L}{\partial \beta}\Delta\beta + \frac{\partial L}{\partial p}\Delta p + \frac{\partial L}{\partial r}\Delta r + \frac{\partial L}{\partial \delta_r}\Delta\delta_r + \frac{\partial L}{\partial \delta_a}\Delta\delta_a\right)$$

$$\Delta M = I_{yy}\left(\frac{\partial M}{\partial u}\Delta u + \frac{\partial M}{\partial w}\Delta w + \frac{\partial M}{\partial \dot{w}}\Delta\dot{w} + \frac{\partial M}{\partial q}\Delta q + \frac{\partial M}{\partial \delta_e}\Delta\delta_e + \frac{\partial M}{\partial \delta_T}\Delta\delta_T\right)$$

$$\Delta N = I_{zz}\left(\frac{\partial N}{\partial \beta}\Delta\beta + \frac{\partial N}{\partial p}\Delta p + \frac{\partial N}{\partial r}\Delta r + \frac{\partial N}{\partial \delta_r}\Delta\delta_r + \frac{\partial N}{\partial \delta_a}\Delta\delta_a\right)$$

e) Classify the airplanes EOM equations mathematically.

The EOMs are coupled, nonlinear and first order DE.

f) What is the difference between the (Body axes) and the (earth or inertial axes)?

I. Body axes: they are a set of axes which are fixed at the body in its translational and rotations.
II. Inertial axes: they are a set of axes which are fixed at a specified position on the ground.

g) What is the difference between the pitch angle (θ) and the angle of attack (α), and between the sideslip angle (β) and the heading angle (ψ)?

I. **Pitch angle:** it's formed due to rotation about y-axis.
II. **AOA:** it's formed due to a difference between the direction of flow and the plane wing.
III. **Sideslip angle:** it's formed due to due to lateral deviation between the plane and the direction of the flow.
IV. **Yaw angle (heading):** it's formed due to due to the rotation about z-axis.

Numerical solutions of ODEs

Some numerical solving algorithms for ODE

one step methods, are methods that involve only y_i and intermediate quantities to compute the next value y_{i+1}. In contrast, multi-step methods use more than the previous point

One-step methods, like:

- Euler-Cauchy method
- Improved Euler method
- Raunge-Kutta method
- Backward Euler method

Multistep methods, like:

- Adams-Bash forth method
- Adams-Moulton method

Other methods

- Predictor-Corrector Methods
- Exponential integrator methods

The chosen method for solving the aircraft EOM.

We choose to use Raunge-Kutta method since its one of the most accurate one step methods used in the numerical solutions, in addition it's the method used in the MATLAB ODE45 algorithm.

The order of the Raunge-Kutta indicates the number of slopes used in the approximation; so higher orders mean higher accuracy of the solution; the wights of the slopes is determined comparing the coefficients with Taylor expansion of the function.[1] anyway, the fourth order Raunge-Kutta is commonly as it gives good accuracy and acceptable computational power.

Initial conditions:

$u_0, v_0, w_0, x_0, y_0, z_0, \theta_0, \varphi_0, \psi_0, p_0, q_0, r_0$

Inputs:

$\delta_a, \delta_e, \delta_r, \delta_T$

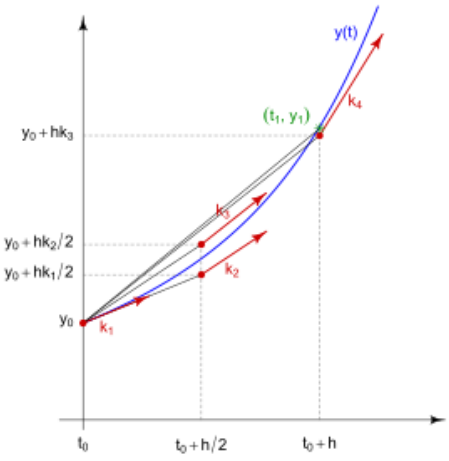
Outputs:

Components of air craft velocity: u , v , w

Position: x , y , z

Euler angles: θ , φ , ψ

Angular velocity components: p , q , r



[1] for more details regarding the derivation of Raunge-Kutta method use the following link https://www.math.hkust.edu.hk/~machas/numerical-methods-for-engineers.pdf?fbclid=IwAR1mOjc35AS3QV_ZuL0Z_nV8Ph3nCop4FlsaS_ueBMz31whmOBPh9LJ2GlM

The general solution of 4th order Runge-Kutta method

$$k_1 = \Delta t f(t_n, x_n)$$

$$k_2 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1\right)$$

$$k_3 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_2\right)$$

$$k_4 = \Delta t f(t_n + \Delta t, x_n + k_3)$$

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

For the following system of ODEs

$$f = \frac{dy_1}{dt} = \sin(t) + \cos(y_1) + \sin(y_2)$$
$$g = \frac{dy_2}{dt} = \cos(t) + \sin(y_2)$$

IC @$t = 0$, $y_1 = 0, y_2 = 0$ $t_f = 20\ sec$ $n = 100$ $h = \frac{\Delta t}{n}$

$$K_1 = hf(t_n, y_n^1, y_n^2)$$

$$K_2 = hf\left(t_n + \frac{1}{2}h, y_n^1 + \frac{1}{2}K_1, y_n^2 + \frac{1}{2}q_1\right)$$

$$K_3 = hf\left(t_n + \frac{1}{2}h, y_n^1 + \frac{1}{2}K_2, y_n^2 + \frac{1}{2}q_2\right)$$

$$K_4 = hf(t_n + h, y_n^1 + K_3, y_n^2 + q_3)$$

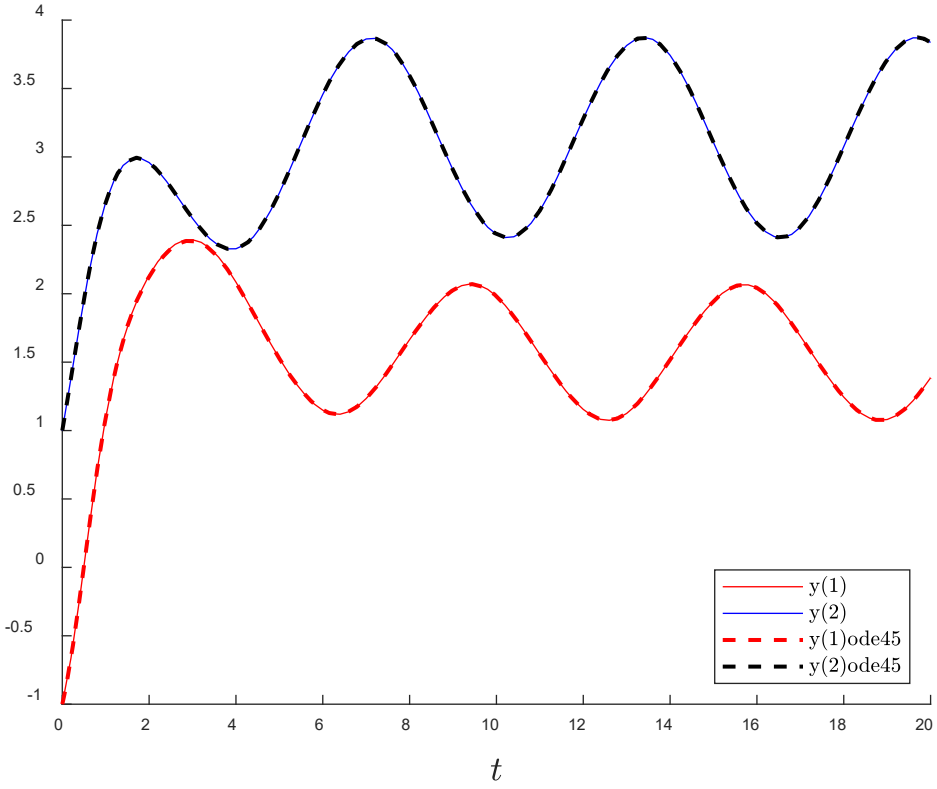
$$q_1 = hg(t_n, y_n^1, y_n^2)$$

$$q_2 = hg\left(t_n + \frac{1}{2}h, y_n^1 + \frac{1}{2}K_1, y_n^2 + \frac{1}{2}q_1\right)$$

$$q_3 = hg\left(t_n + \frac{1}{2}h, y_n^1 + \frac{1}{2}K_2, y_n^2 + \frac{1}{2}q_2\right)$$

$$q_4 = hg(t_n + h, y_n^1 + K_3, y_n^2 + q_3)$$

$$y_{n+1}^1 = y_n^1 + \frac{1}{6}(q_1 + 2q_2 + 2q_3 + q_4)$$
$$y_{n+1}^2 = y_n^2 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$



Task (2)

RBD Equations Solver

In this task we aim to solve the 12 rigid body equations of dynamics equations that we were derived in task1, the solver is built basically to compare the results of solution from 3 sources (our code, ODE45 and Simulink).

We will start by reviewing the 12 equations and the data given in this task, then we will show the figures and the code.

RBD Equations in a vector form

Kinetics

$\begin{aligned} F_x &= m(\dot{u} + qw - rv) \\ F_y &= m(\dot{v} + ru - pw) \\ F_z &= m(\dot{w} + pv - qu) \end{aligned}$	$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m\left(\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix}\right) = \begin{bmatrix} X - mg\,S_\theta \\ Y + mg\,C_\theta\,S_\varphi \\ Z + mg\,C_\theta\,C_\varphi \end{bmatrix}$
$\begin{aligned} L &= \dot{H}_x + qH_z - rH_y \\ M &= \dot{H}_y + rH_x - pH_z \\ N &= \dot{H}_z + pH_y - qH_x \end{aligned}$	$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}$

Kinematics

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & S\varphi\,T\theta & C\varphi\,T\theta \\ 0 & C\varphi & -S\varphi \\ 0 & S\varphi/C\theta & C\varphi/C\theta \end{bmatrix} \begin{Bmatrix} P \\ Q \\ R \end{Bmatrix}$$

$$\begin{Bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\theta C_\theta & C_\phi C_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}$$

Givens:

$$\mathbf{t_{final} = 15\ sec}\,,\quad \mathbf{Forces} = [10\ 5\ 9]\ \mathbf{N}\,, \mathbf{Moments} = [10\ 20\ 5]\mathbf{N.m}$$

$$\mathbf{Mass = 15\ Kg}$$

$$\mathbf{I} = \begin{bmatrix} I_x & I_{xy} & I_{xz} \\ I_{yx} & I_y & I_{yz} \\ I_{zx} & I_{zy} & I_z \end{bmatrix} = \begin{bmatrix} 1 & -2 & -1 \\ -2 & 5 & -3 \\ -1 & -3 & 0.1 \end{bmatrix}$$

IC's:

$$[\mathbf{u\ v\ w, p\ q\ r, \phi\ \theta\ \psi, x\ y\ z}] = [\mathbf{10, 2, 0}; \frac{\mathbf{2\pi}}{\mathbf{180}}, \frac{\mathbf{\pi}}{\mathbf{180}}\ \mathbf{0}; \frac{\mathbf{20\pi}}{\mathbf{180}}, \frac{\mathbf{15\pi}}{\mathbf{180}}, \frac{\mathbf{30\pi}}{\mathbf{180}}; \mathbf{2, 4, 7}]$$

Our solver code has main function called RBDSolver which takes the initial condition given and returns the 12 rates of change of each state, by integrating each one we get the states of the rigid body.

Raunge-Kutta 4^{th} *order* (our code)

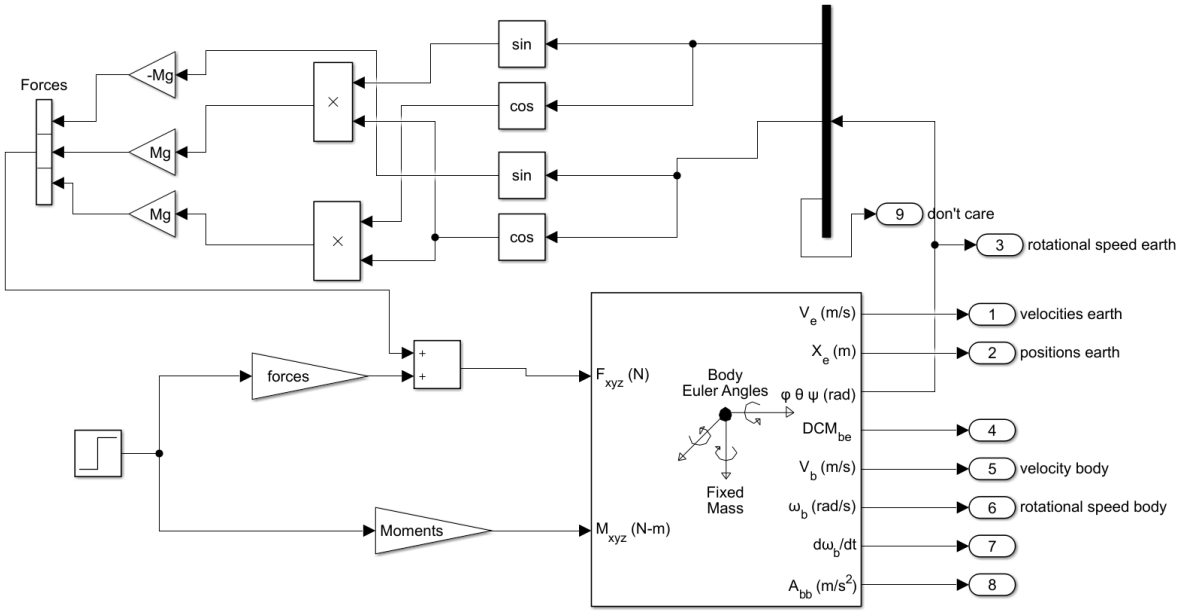
First, we need to put our 12 equations in a form suitable for Raunge-kutta method like, $\dot{y} = function(t, x, y, z, \dots)$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \mathrm{X} - \mathrm{mg\ S_\theta} \\ \mathrm{Y} + \mathrm{mg\ C_\theta\ S_\varphi} \\ \mathrm{Z} + \mathrm{mg\ C_\theta\ C_\varphi} \end{bmatrix} / m - \left(\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \right)$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \left(\begin{bmatrix} L \\ M \\ N \end{bmatrix} - \left(\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \right) \right)$$

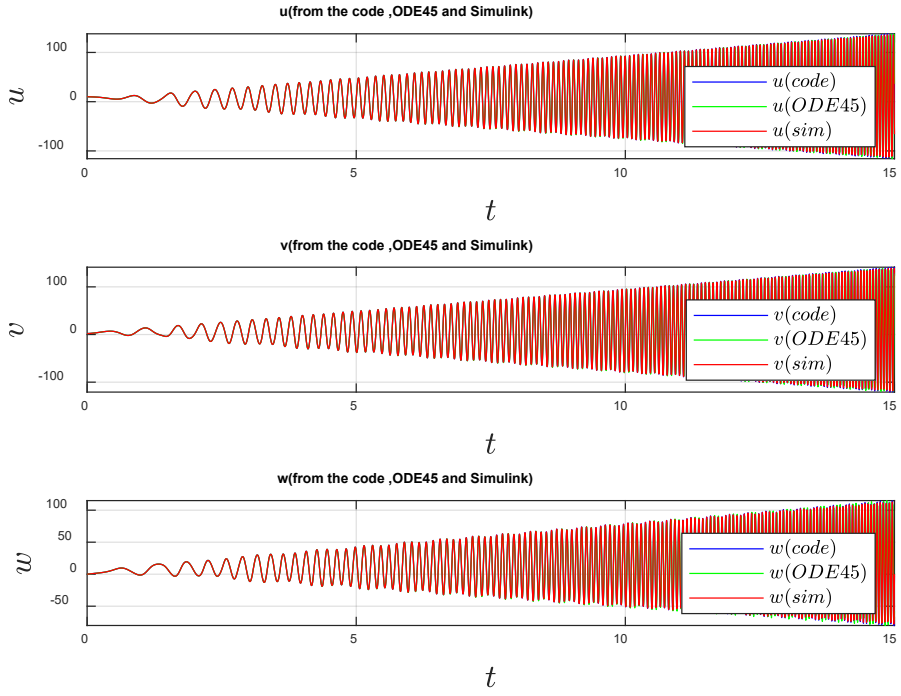
Validation with Simulink and ODE45

we validated our results by using Simulink (here) and ODE45 (appendix (task 2))

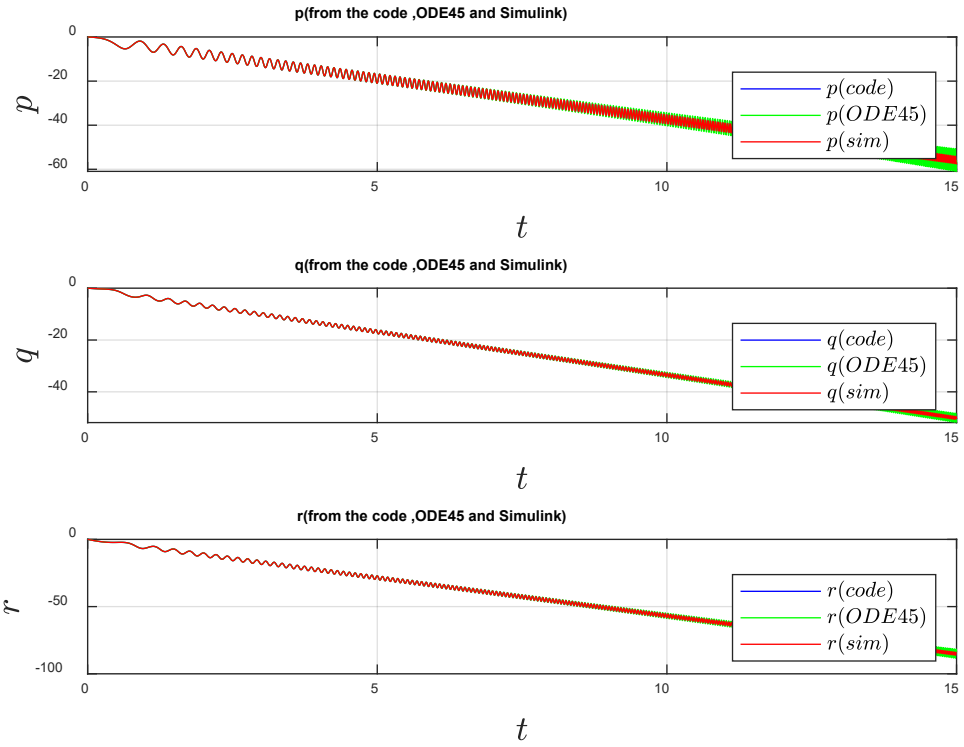


Results

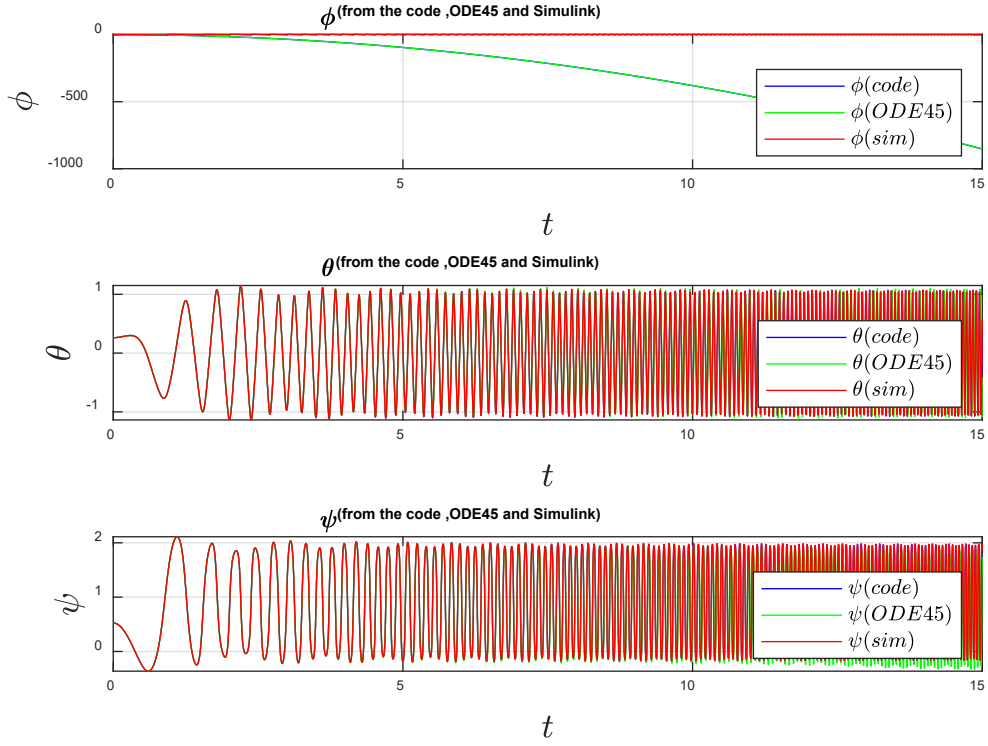
velocity with respect to body Axes:



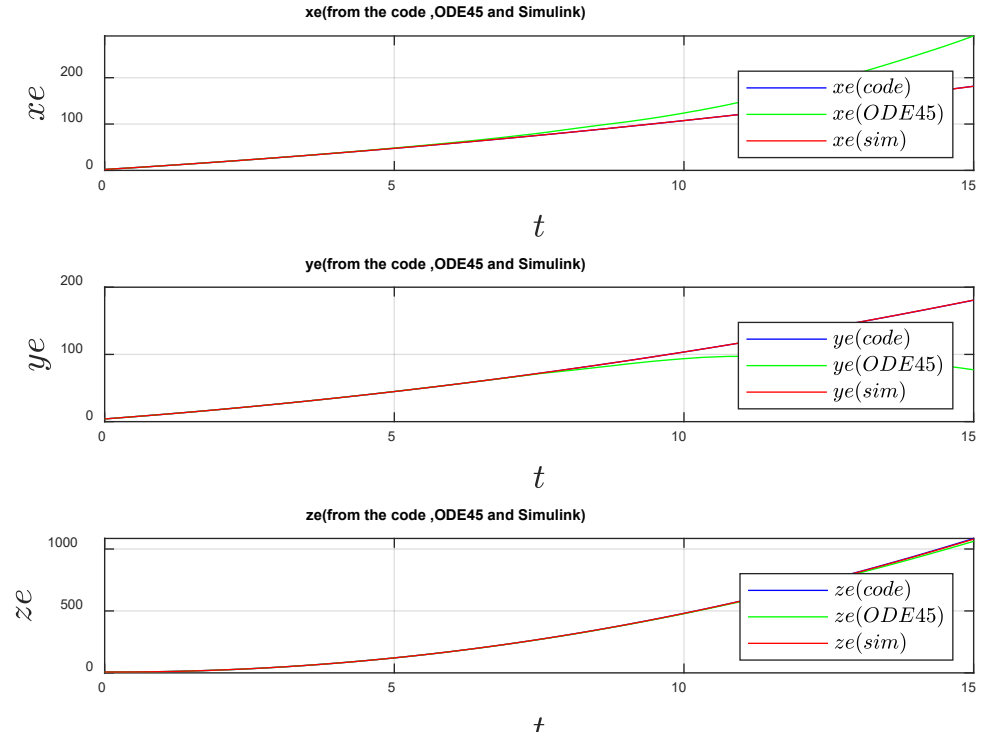
Rotational speed W.r.t body axes



Rotational speed w.r.t Earth Axes



Position w.r.t Earth Axes



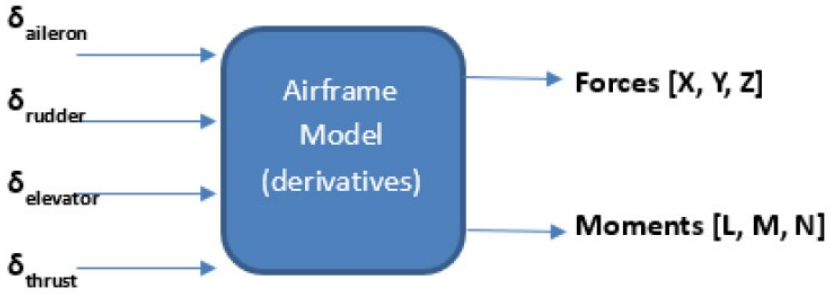
Task (3)

Introduction

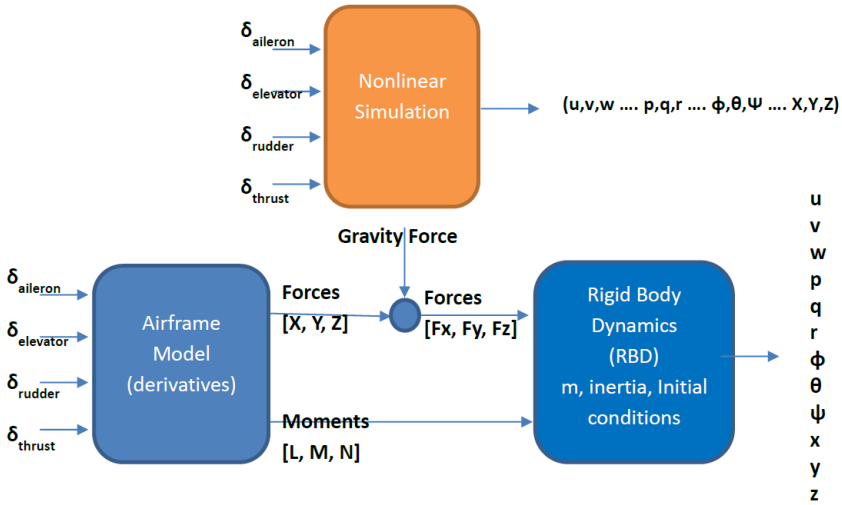
In this task we are concerned about calculating the change in states due to change in control surfaces at certain aircraft data.

We will build an airframe model code that takes the control surfaces deflection values

$$\delta_{aileron}, \delta_{thrust}, \delta_{rudder}, \delta_{elevator}$$



We will combine the airframe model code with the RBD solver code that we made to make a nonlinear simulator



Equations representing the change in the Aerodynamic & thrust forces & moments:

$$\Delta X = m\left(\frac{\partial X}{\partial u}\,\Delta u + \frac{\partial X}{\partial w}\,\Delta w + \frac{\partial X}{\partial \delta_e}\,\Delta\delta_e + \frac{\partial X}{\partial \delta_T}\,\Delta\delta_T\right)$$

$$\Delta Y = m\left(\frac{\partial Y}{\partial v}\,\Delta v + \frac{\partial Y}{\partial \beta}\,\Delta\beta + \frac{\partial Y}{\partial \delta_a}\,\Delta\delta_a + \frac{\partial Y}{\partial \delta_r}\,\Delta\delta_r\right)$$

$$\Delta Z = m\left(\frac{\partial Z}{\partial u}\,\Delta u + \frac{\partial Z}{\partial w}\,\Delta w + \frac{\partial Z}{\partial \dot{w}}\,\Delta\dot{w} + \frac{\partial Z}{\partial q}\,\Delta q + \frac{\partial Z}{\partial \delta_e}\,\Delta\delta_e + \frac{\partial Z}{\partial \delta_T}\,\Delta\delta_T\right)$$

$$\Delta L = I_{xx}\left(\frac{\partial L}{\partial \beta}\,\Delta\beta + \frac{\partial L}{\partial p}\,\Delta p + \frac{\partial L}{\partial r}\,\Delta r + \frac{\partial L}{\partial \delta_r}\,\Delta\delta_r + \frac{\partial L}{\partial \delta_a}\,\Delta\delta_a\right)$$

$$\Delta M = I_{yy}\left(\frac{\partial M}{\partial u}\,\Delta u + \frac{\partial M}{\partial w}\,\Delta w + \frac{\partial M}{\partial \dot{w}}\,\Delta\dot{w} + \frac{\partial M}{\partial q}\,\Delta q + \frac{\partial M}{\partial \delta_e}\,\Delta\delta_e + \frac{\partial M}{\partial \delta_T}\,\Delta\delta_T\right)$$

$$\Delta N = I_{zz}\left(\frac{\partial N}{\partial \beta}\,\Delta\beta + \frac{\partial N}{\partial p}\,\Delta p + \frac{\partial N}{\partial r}\,\Delta r + \frac{\partial N}{\partial \delta_r}\,\Delta\delta_r + \frac{\partial N}{\partial \delta_a}\,\Delta\delta_a\right)$$

Axes systems

There are three types of body axes

- Principle axes ($I_{xy} = 0$)
- Stability Axes ($w_0 = 0$), X_b *axis* concides with V_T
- Body axes (fuselage reference axis)

we will transform from stability axes derivatives to body axes

$$L'_\beta = (L_\beta + I_{xz}N_\beta/I_x)G \qquad 1/sec^2$$

$$L'_p = (L_p + I_{xz}N_p/I_x)G \qquad 1/sec$$

$$L'_r = (L_r + I_{xz}N_r/I_x)G \qquad 1/sec$$

$$L'_{\delta_r} = (L_{\delta_r} + I_{xz}N_{\delta_r}/I_x)G \qquad 1/sec^2$$

$$L'_{\delta_a} = (L_{\delta_a} + I_{xz}N_{\delta_a}/I_x)G \qquad 1/sec^2$$

$$N'_\beta = (N_\beta + I_{xz}L_\beta/I_z)G \qquad 1/sec^2$$

$$N'_p = (N_p + I_{xz}L_p/I_z)G \qquad 1/sec$$

$$N'_r = (N_r + I_{xz}L_r/I_z)G \qquad 1/sec$$

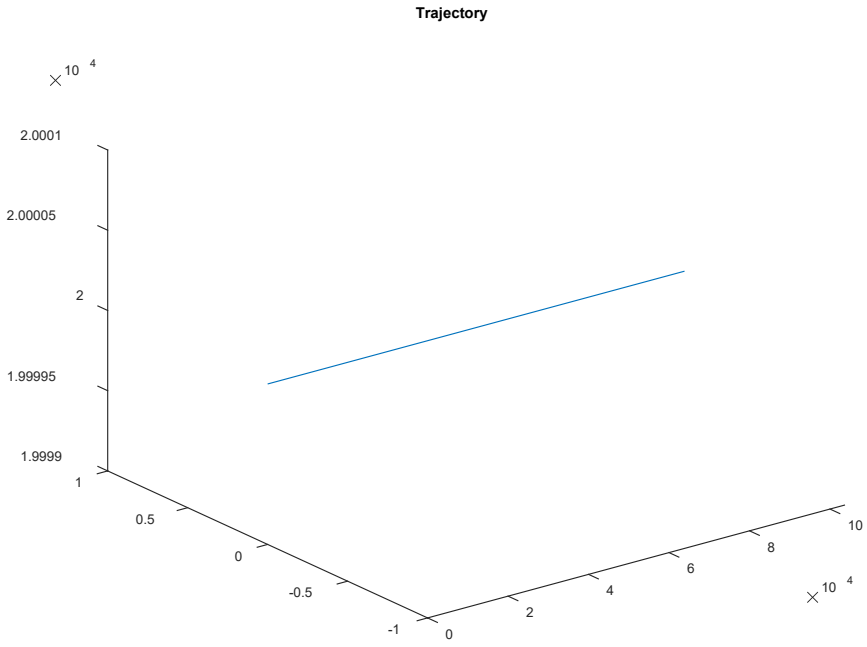
$$N'_{\delta_r} = (N_{\delta_r} + I_{xz}L_{\delta_r}/I_z)G \qquad 1/sec^2$$

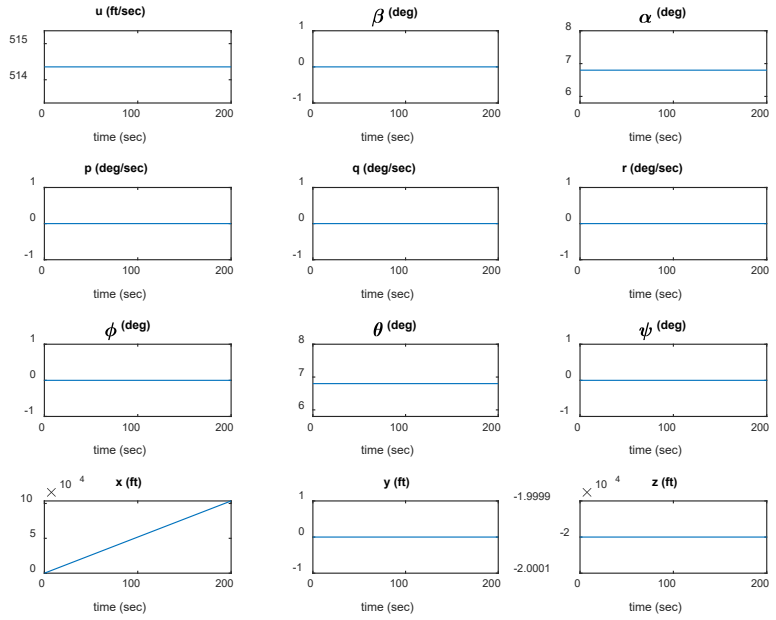
$$N'_{\delta_a} = (N_{\delta_a} + I_{xz}L_{\delta_a}/I_z)G \qquad 1/sec^2$$

$$G = \frac{1}{1 - \frac{I_{xz}^2}{I_x I_z}}$$

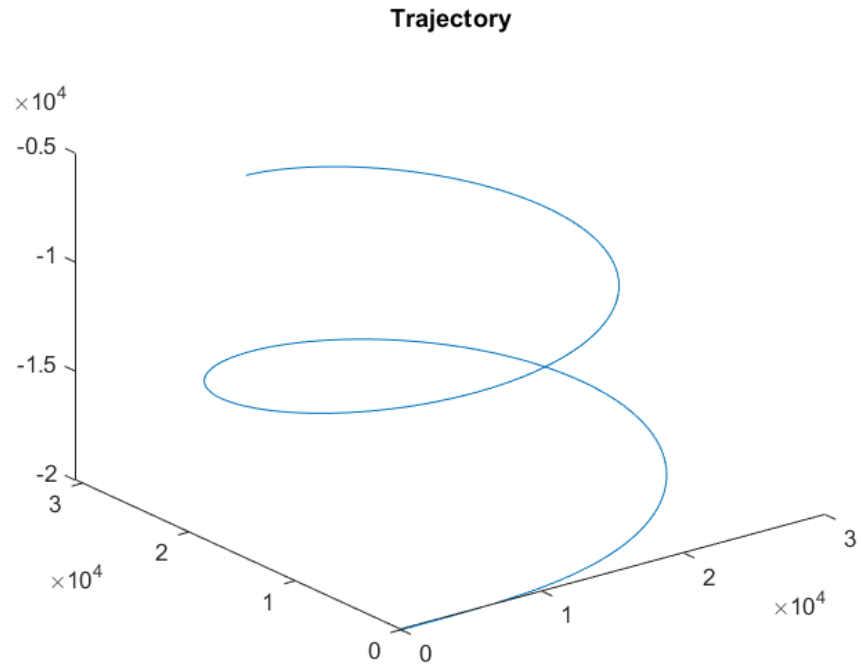
Benchmark test plots B-747 airplane

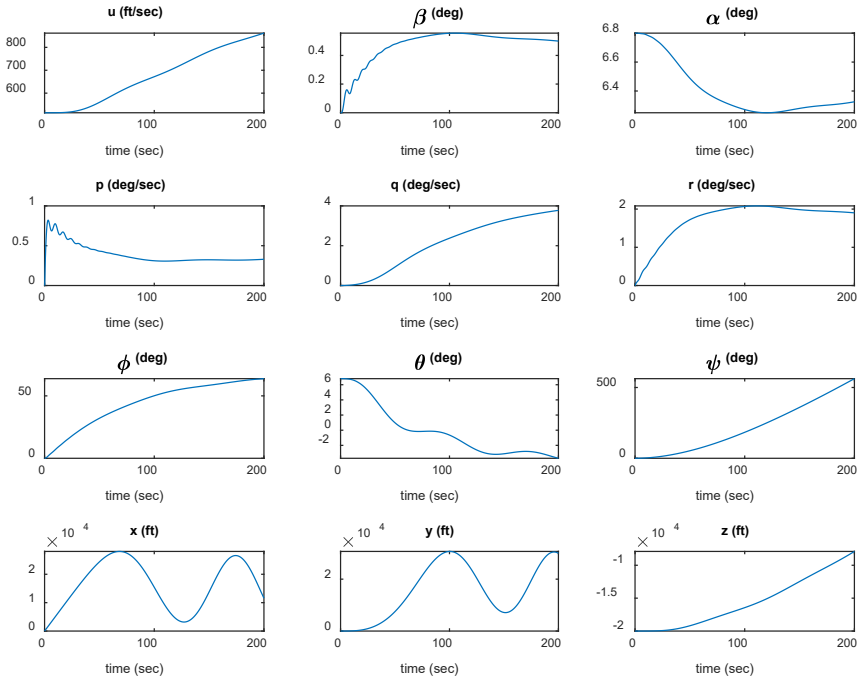
For No input



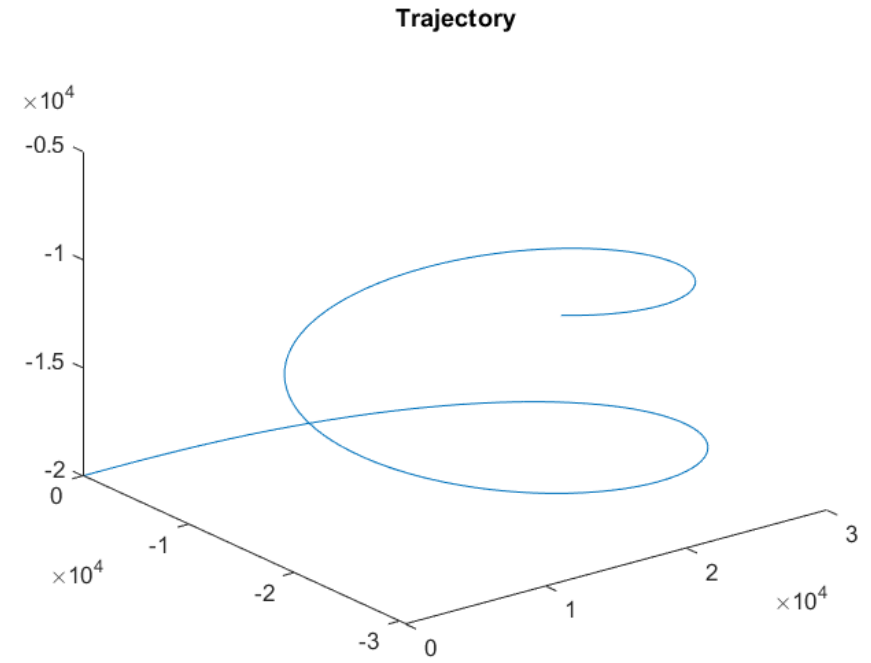


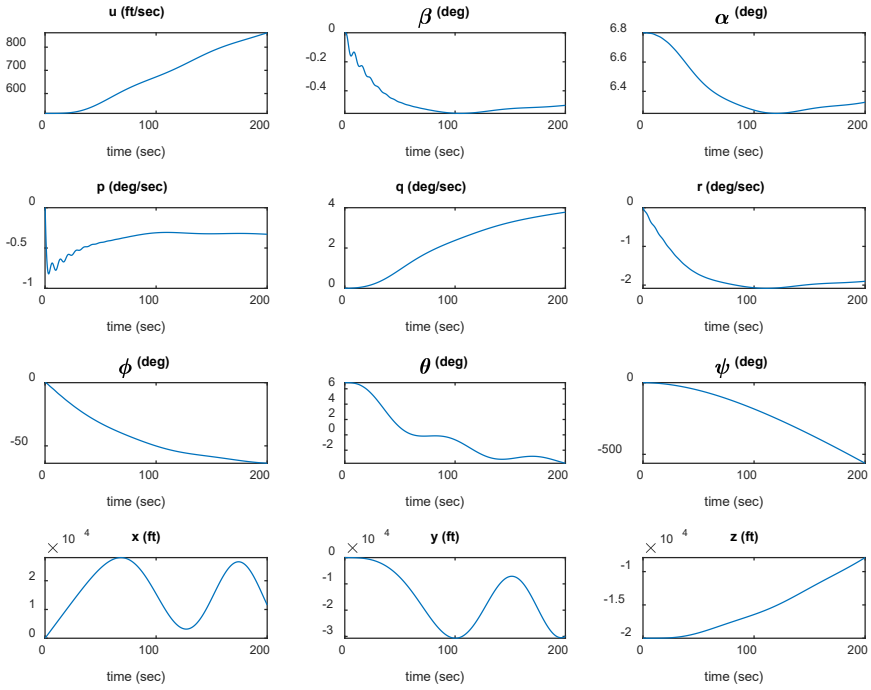
For +ve 5-degree aileron



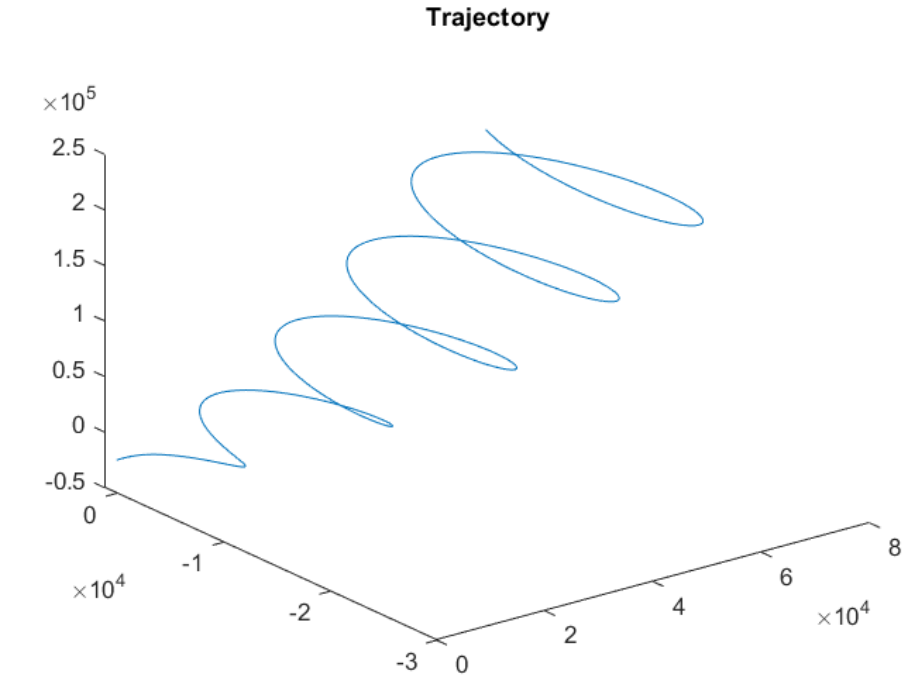


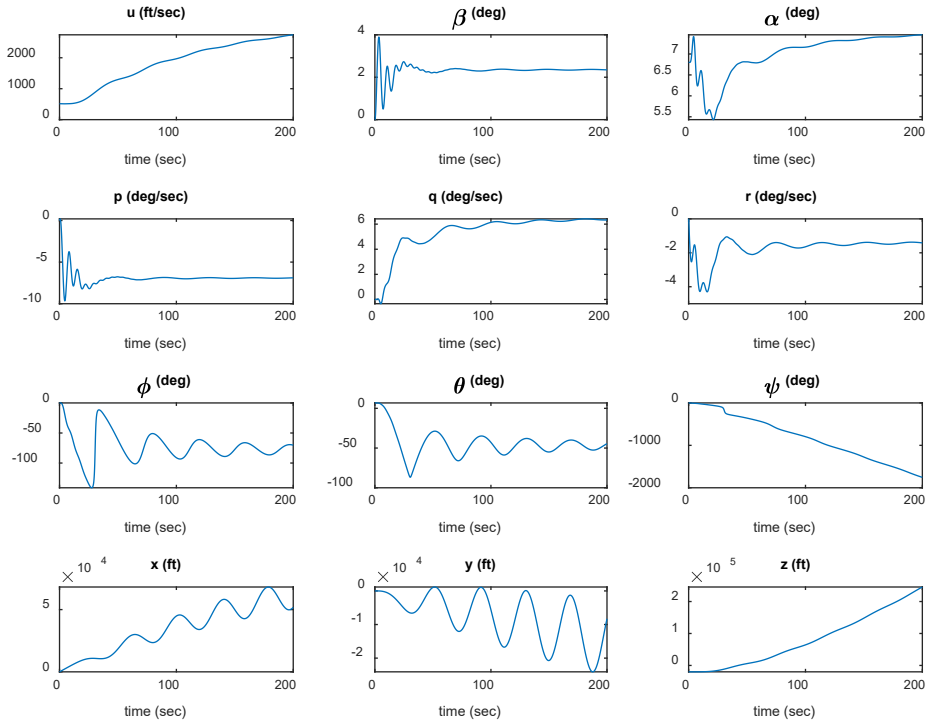
For -ve 5-degree aileron



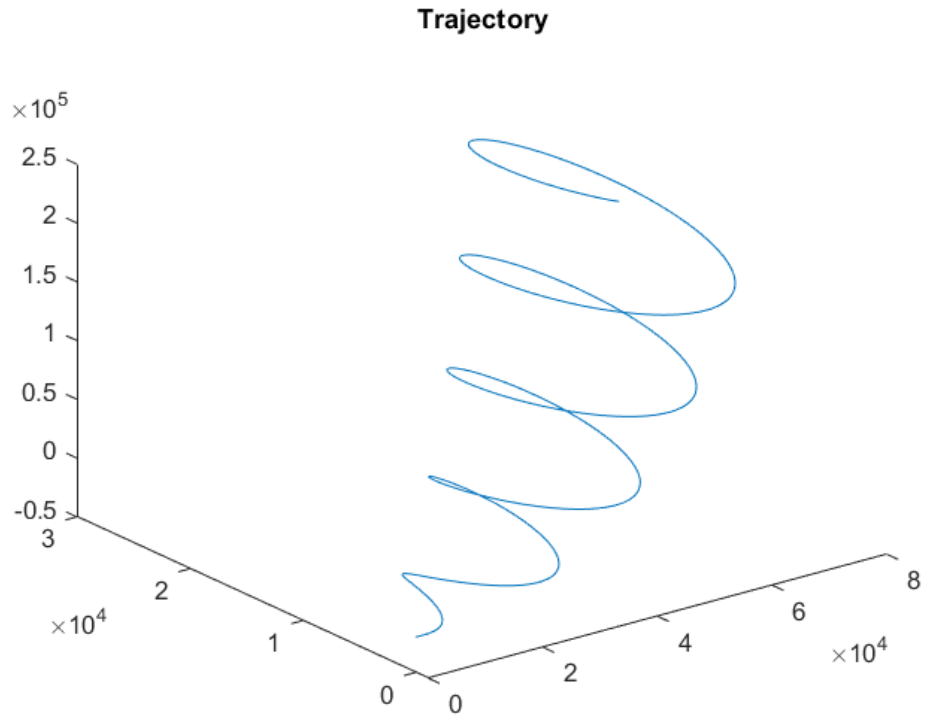


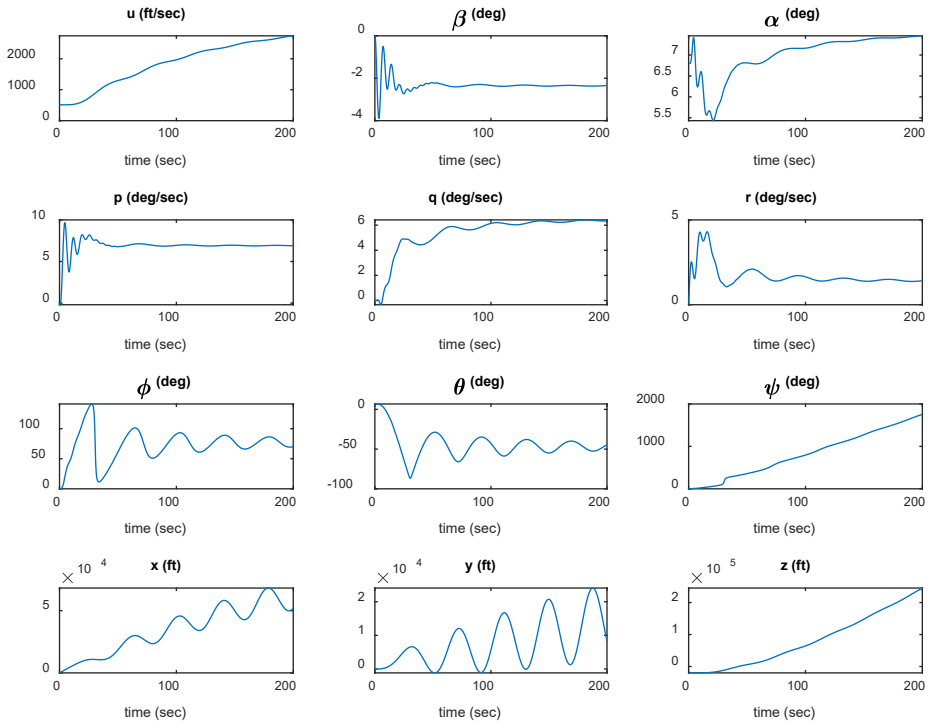
For +ve 5-degree rudder



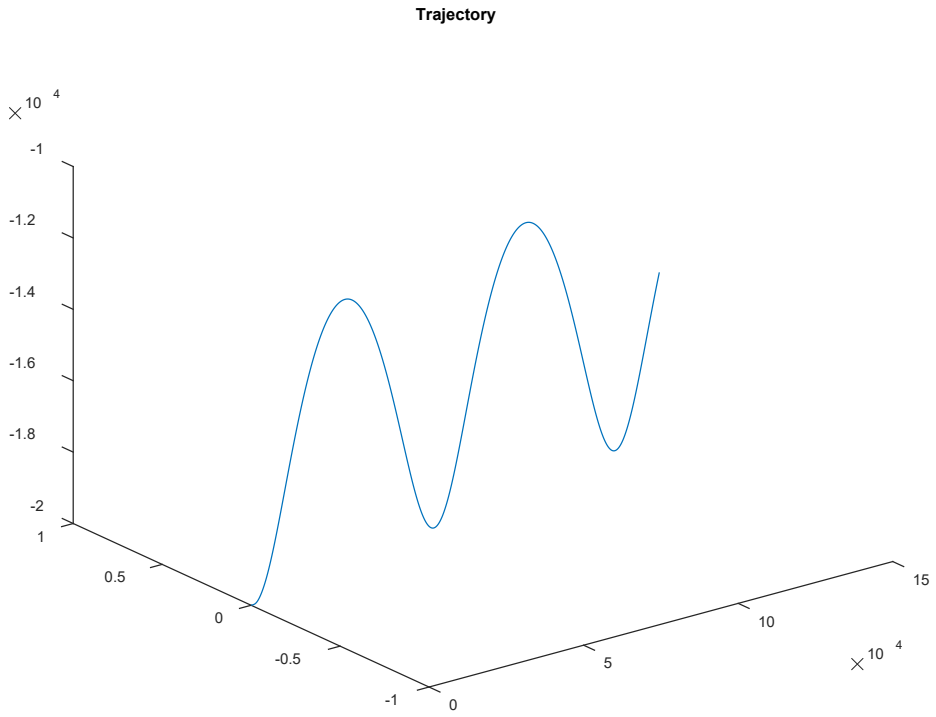


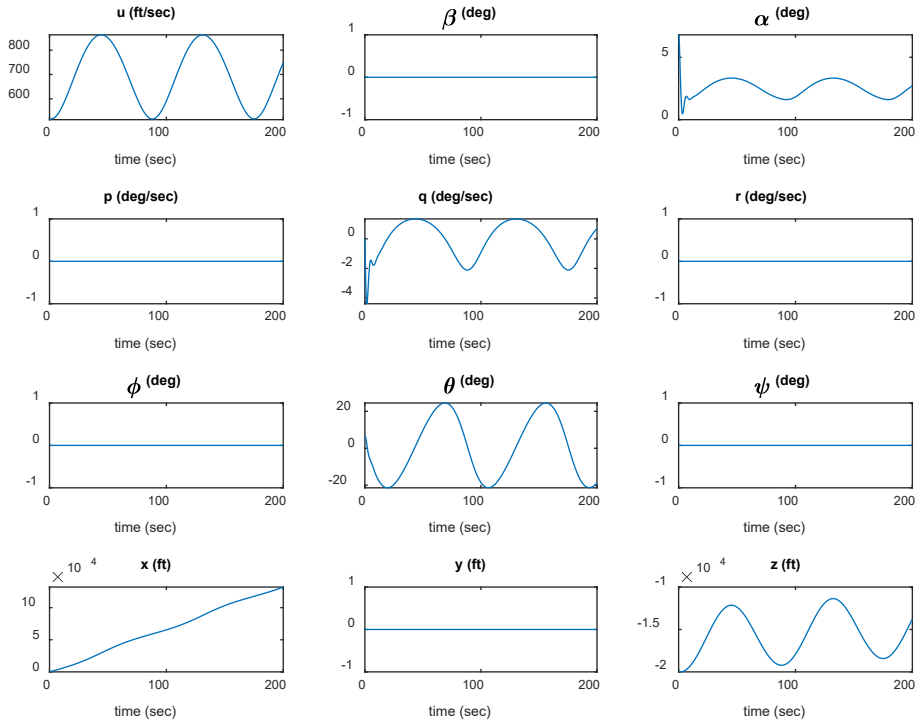
For -ve 5-degree rudder



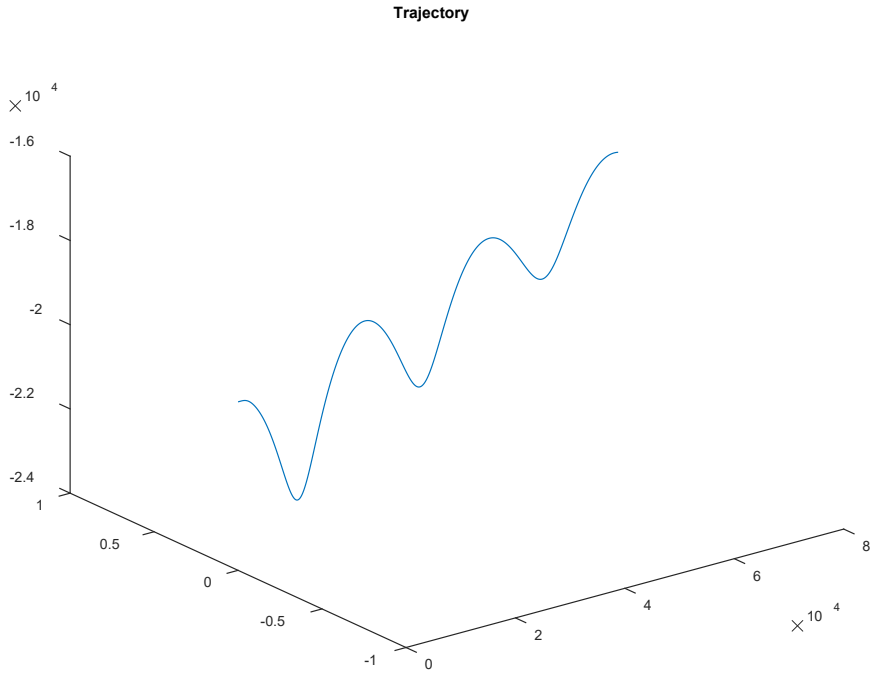


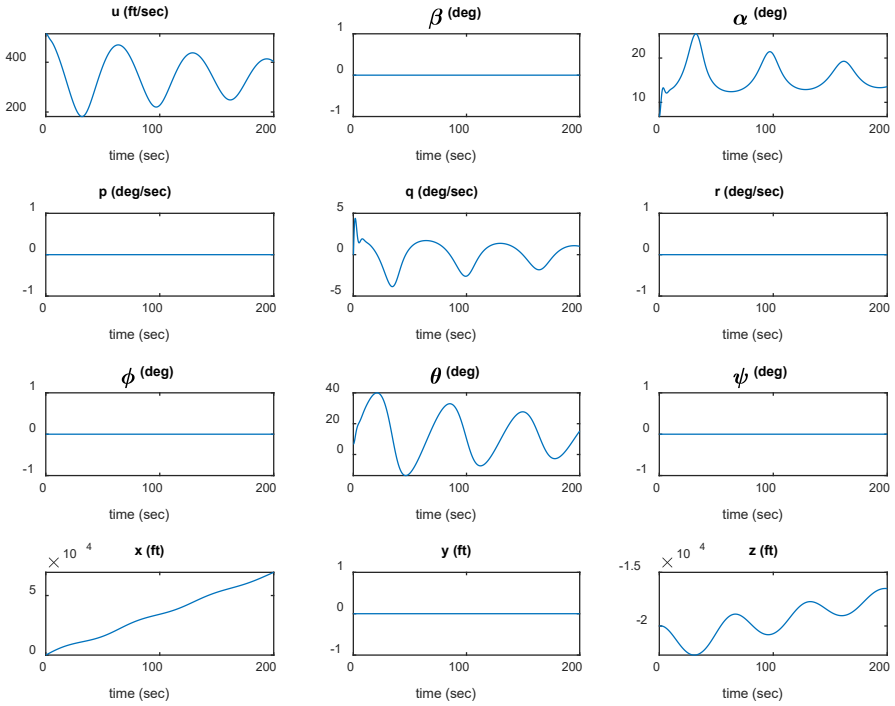
For +ve 5-degree elevator



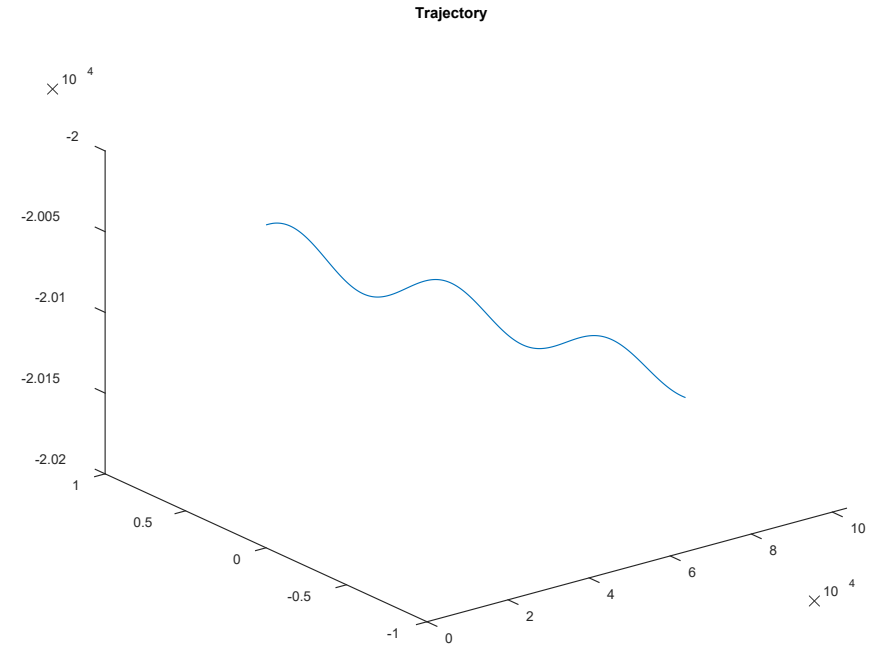


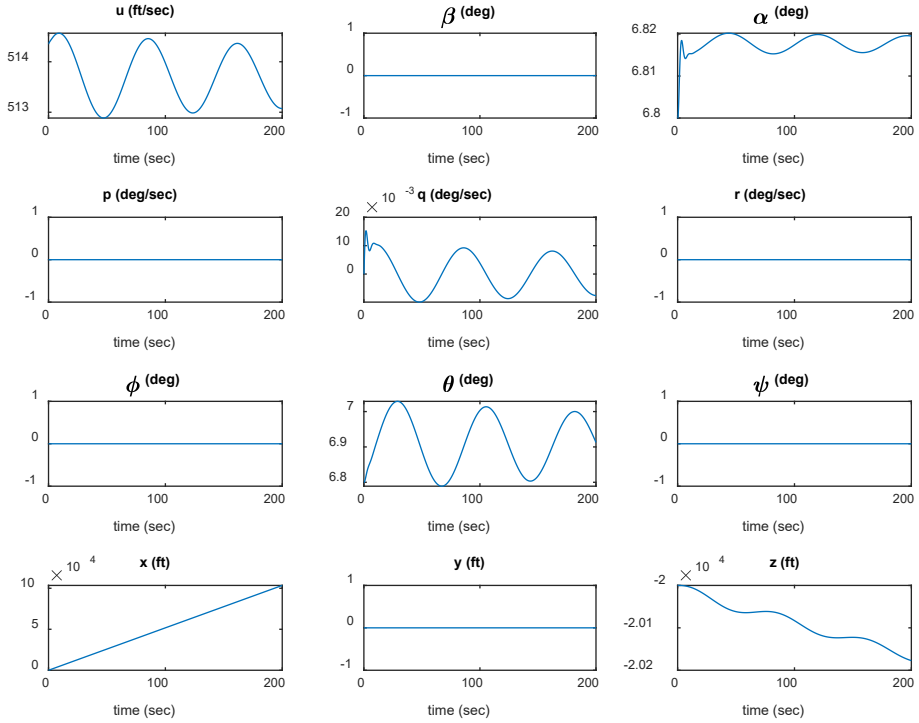
For -ve 5-degree elevator



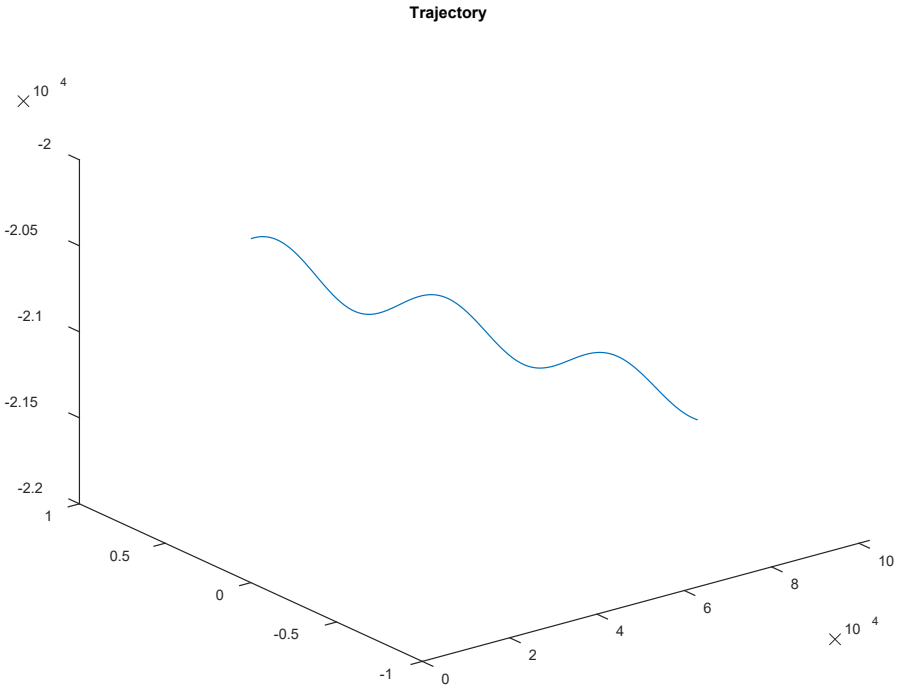


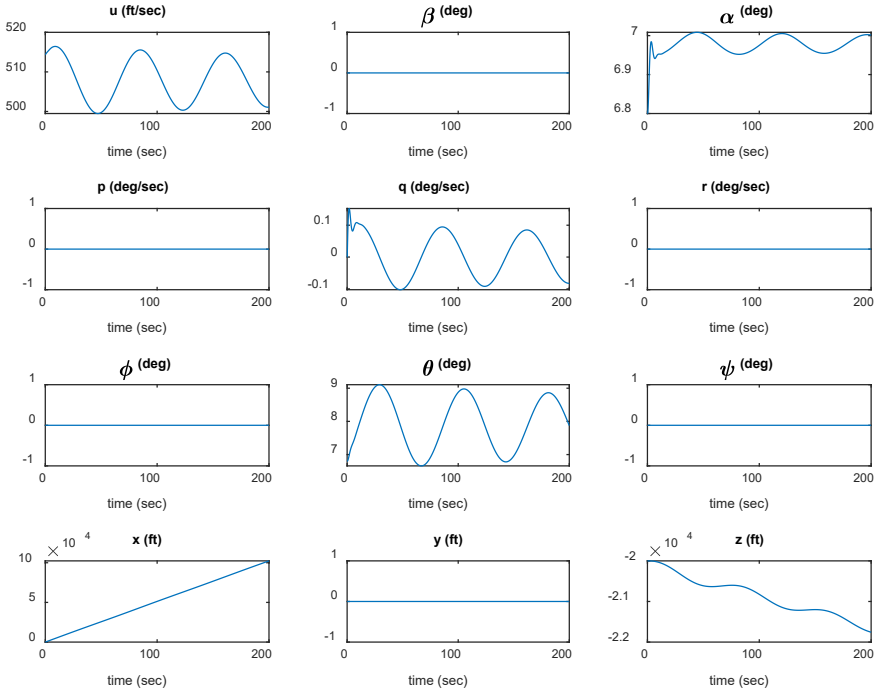
For 1000 in thrust





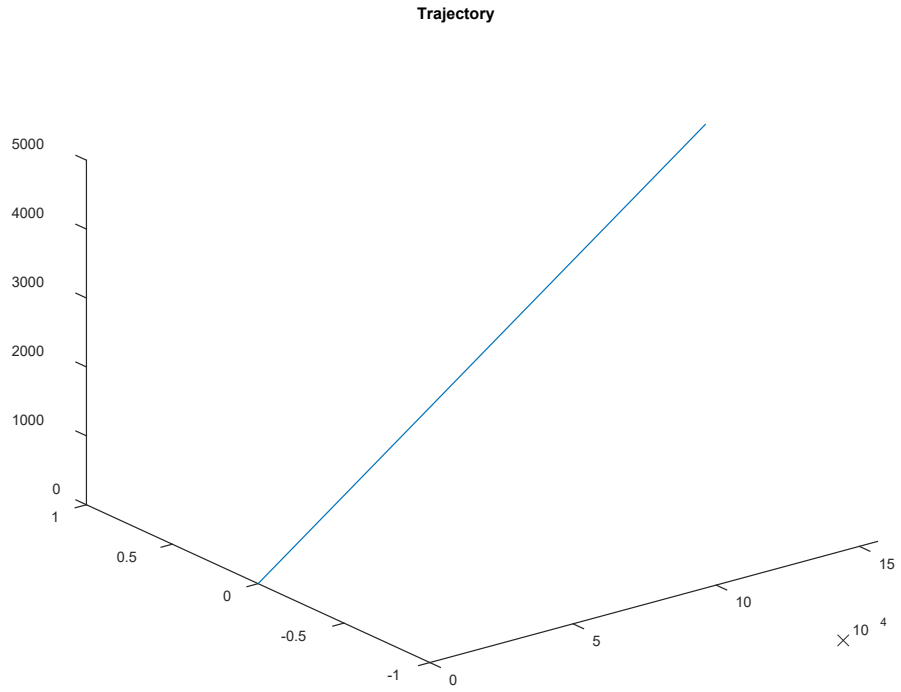
For 10000 in thrust

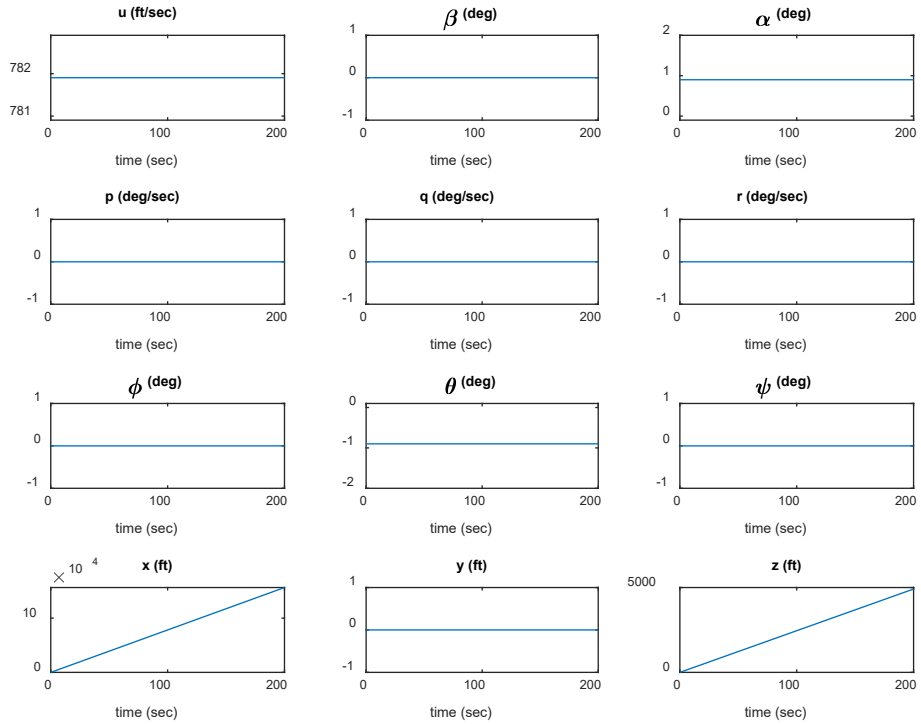




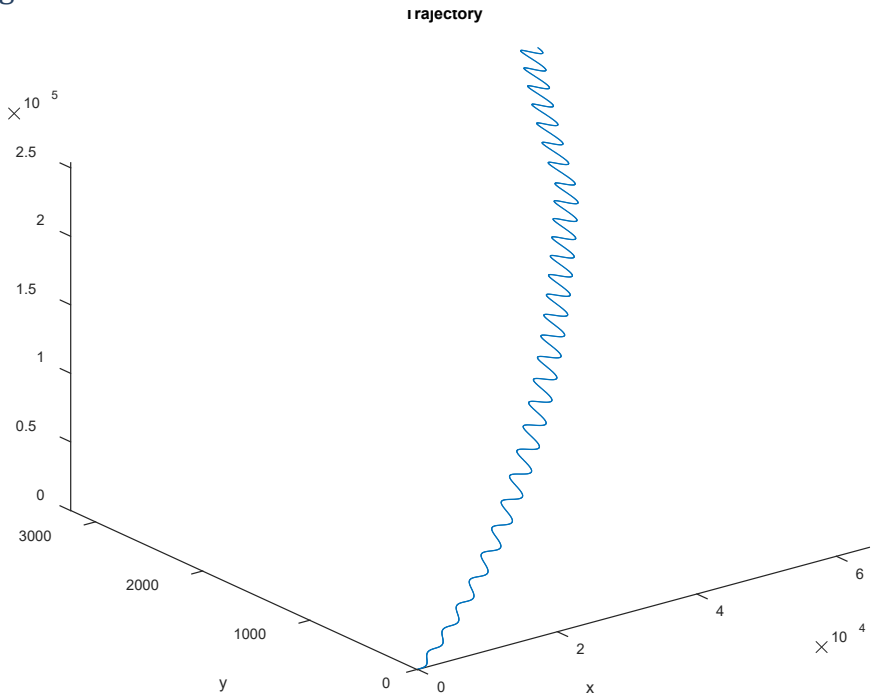
Our airplane Plots (NT-33A)

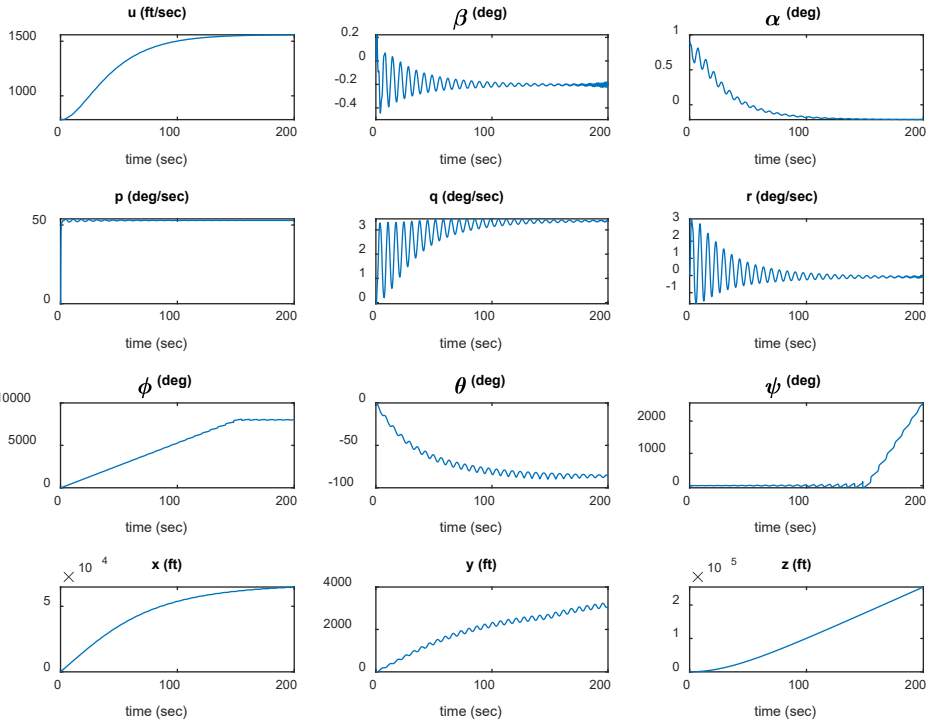
For No input



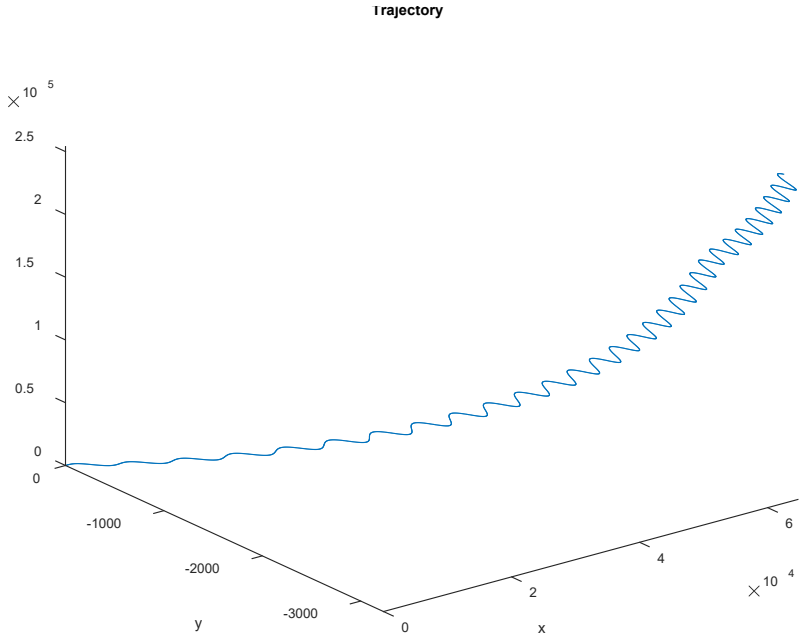


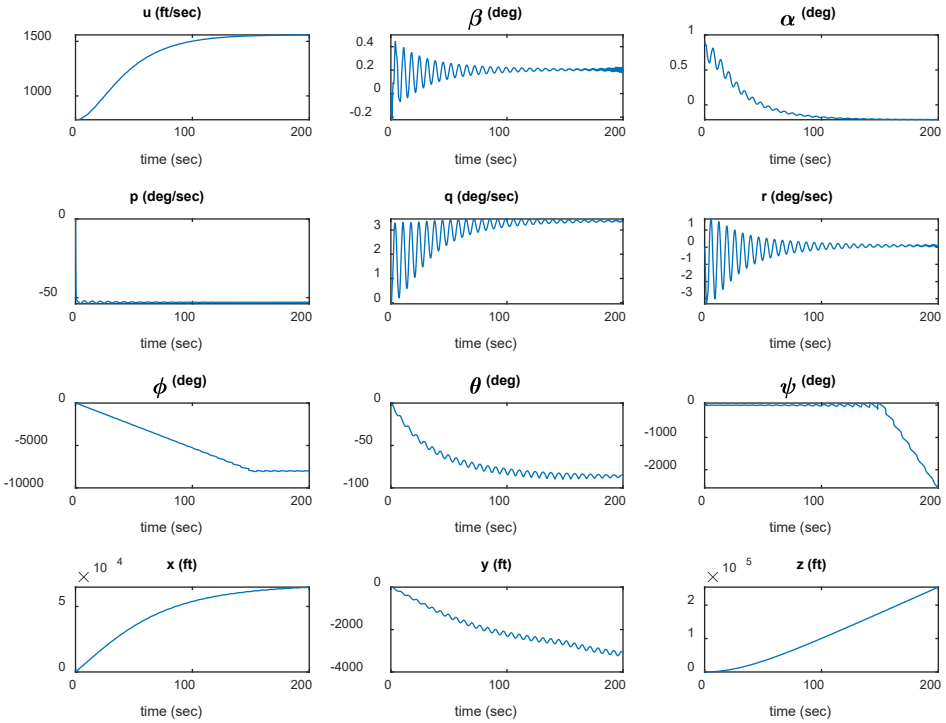
For +ve 5-degree aileron



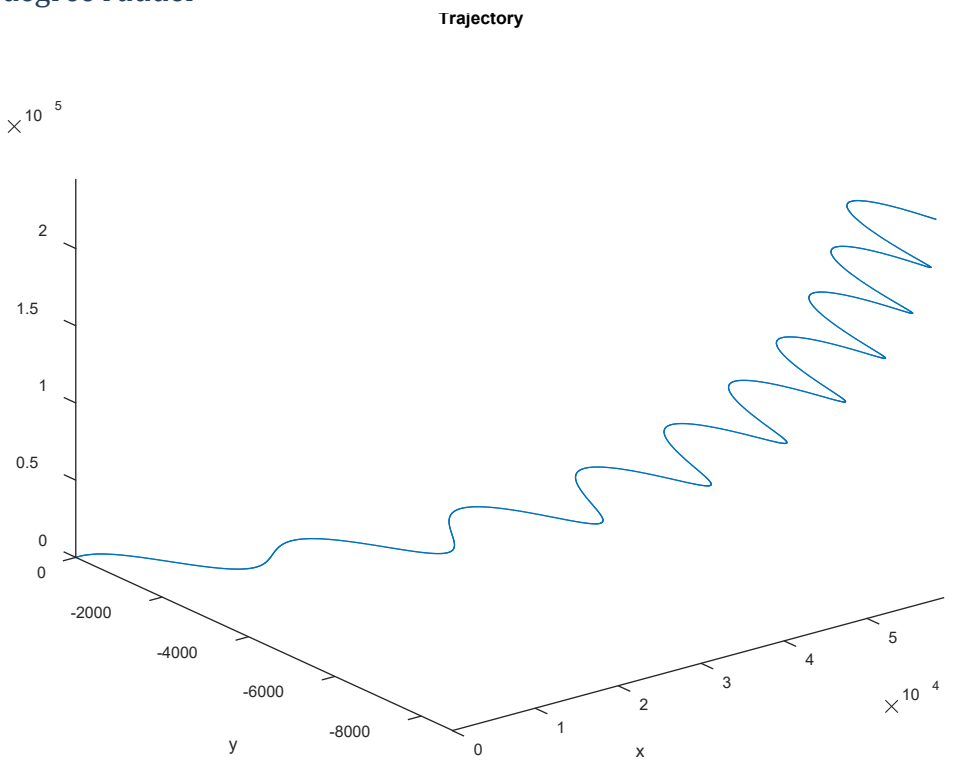


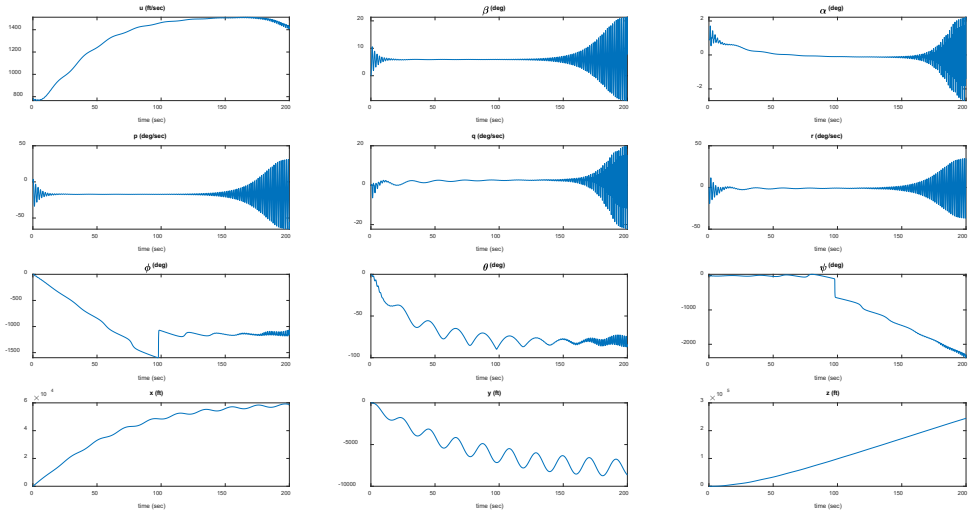
For -ve 5-degree aileron



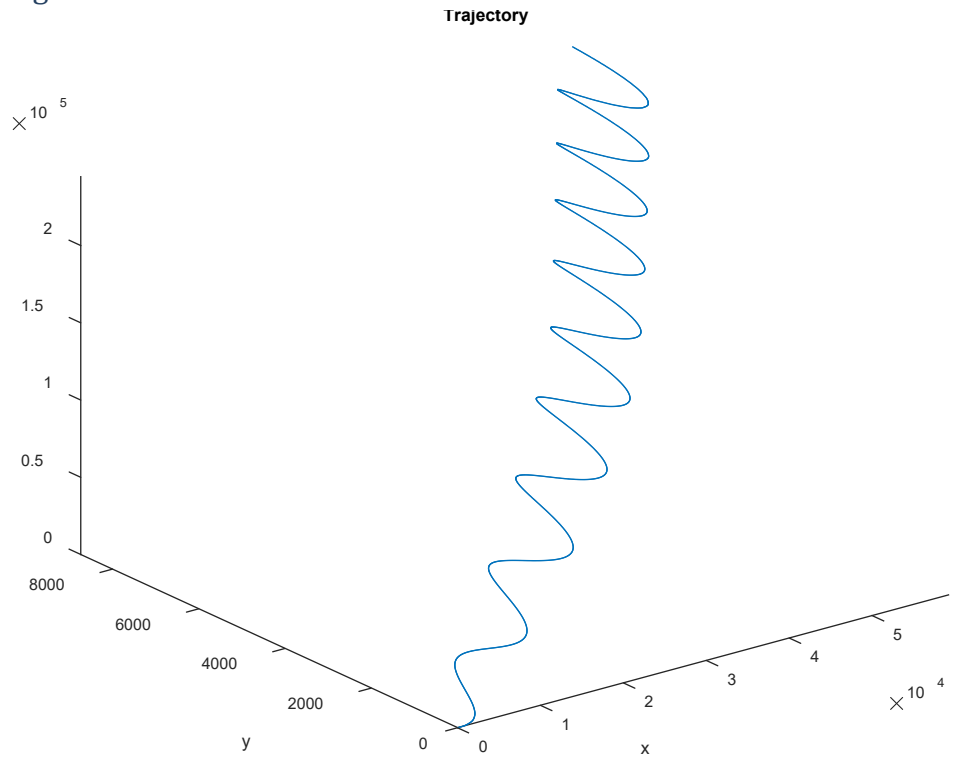


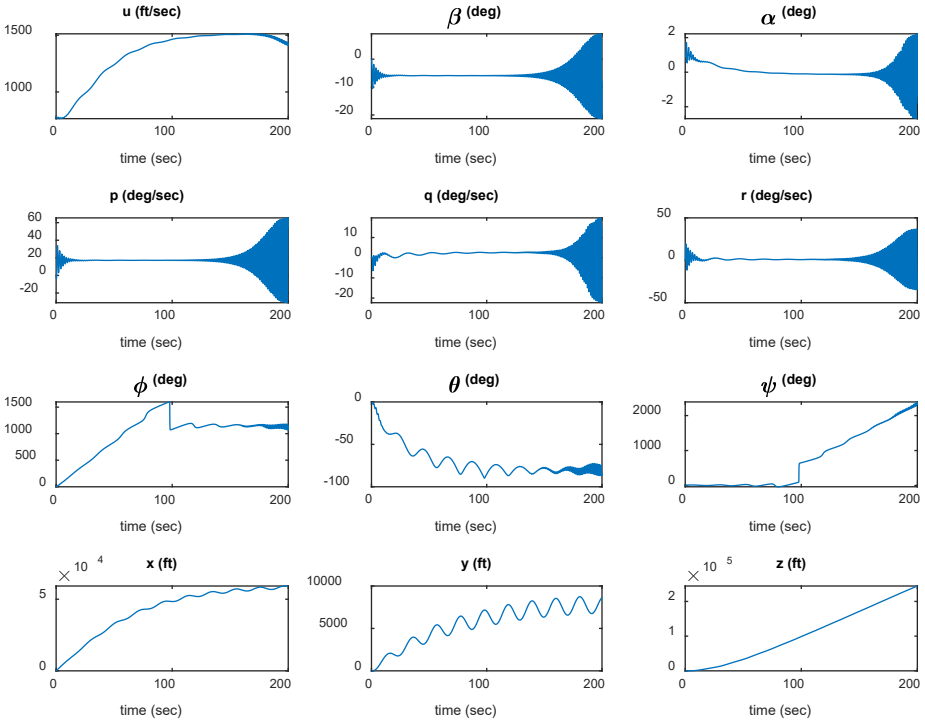
For +ve 5-degree rudder



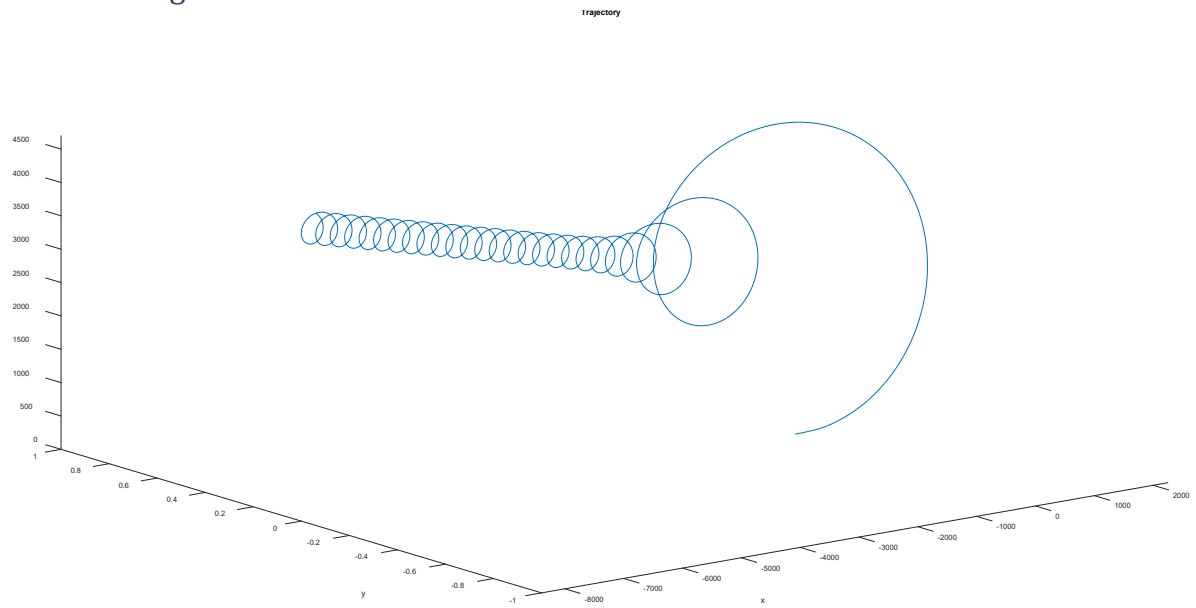


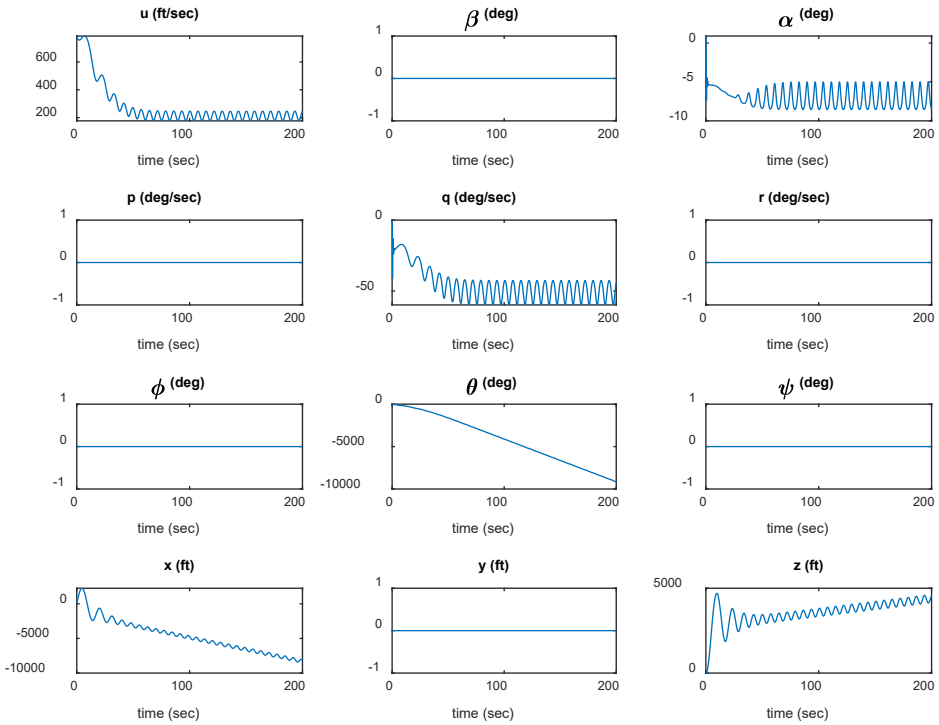
For -ve 5-degree rudder



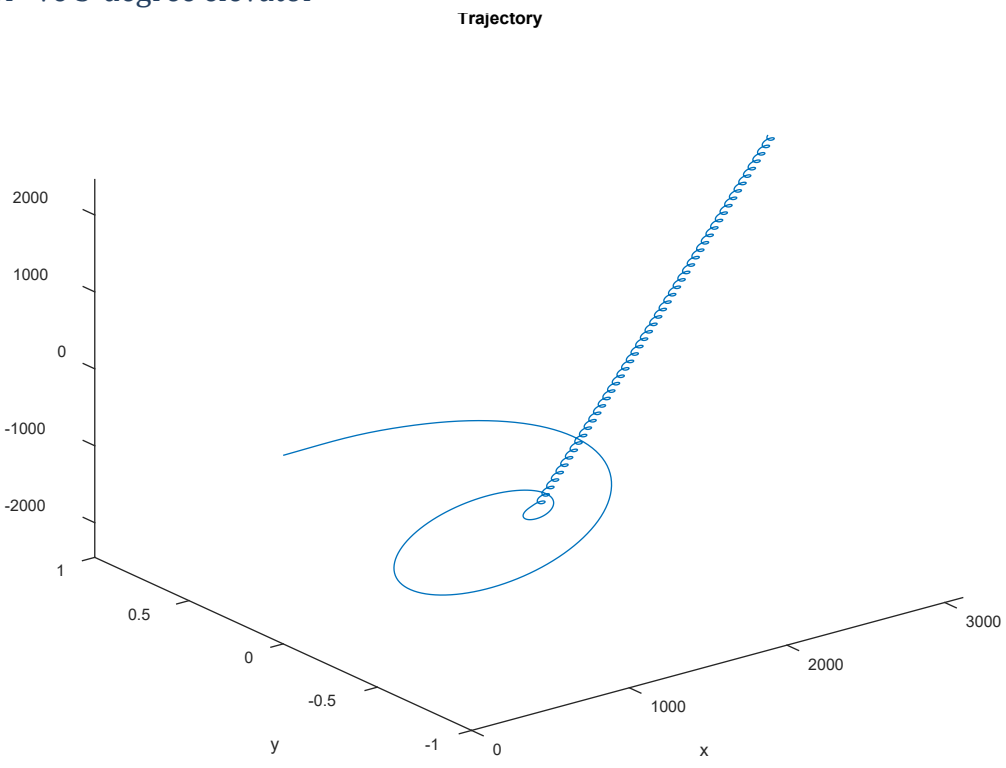


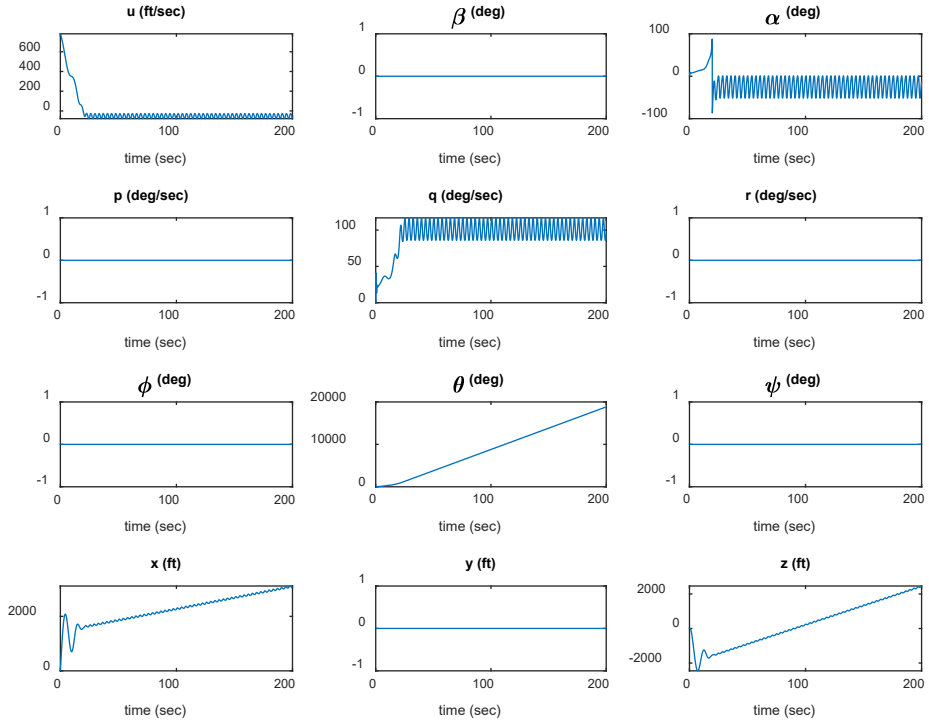
For +ve 5-degree elevator



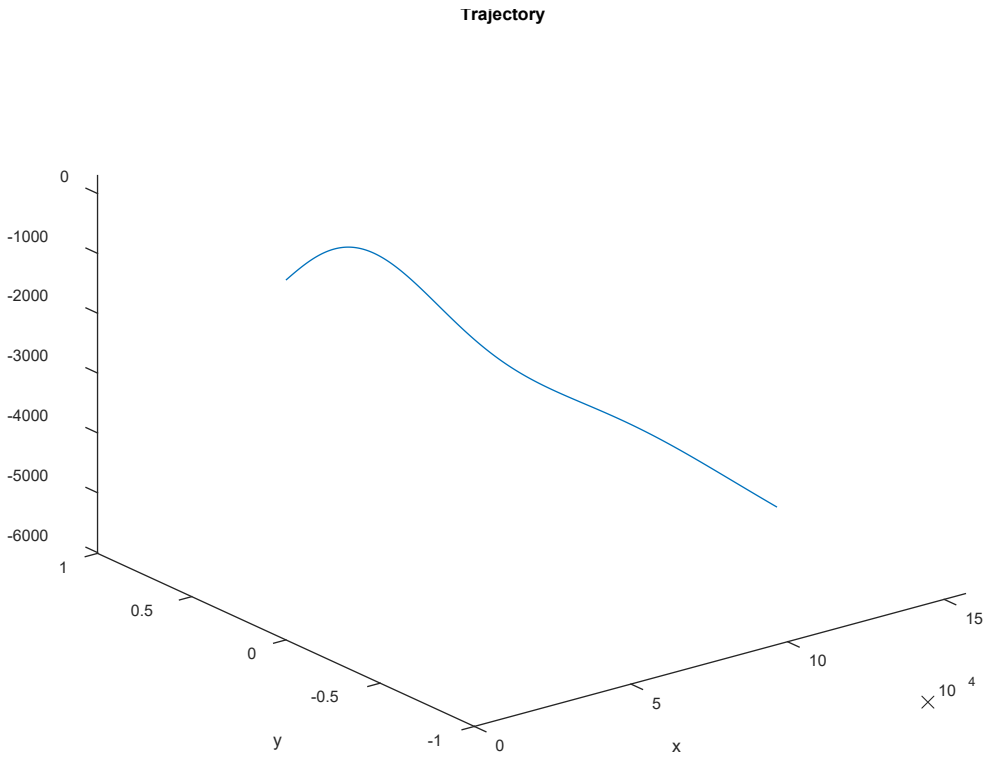


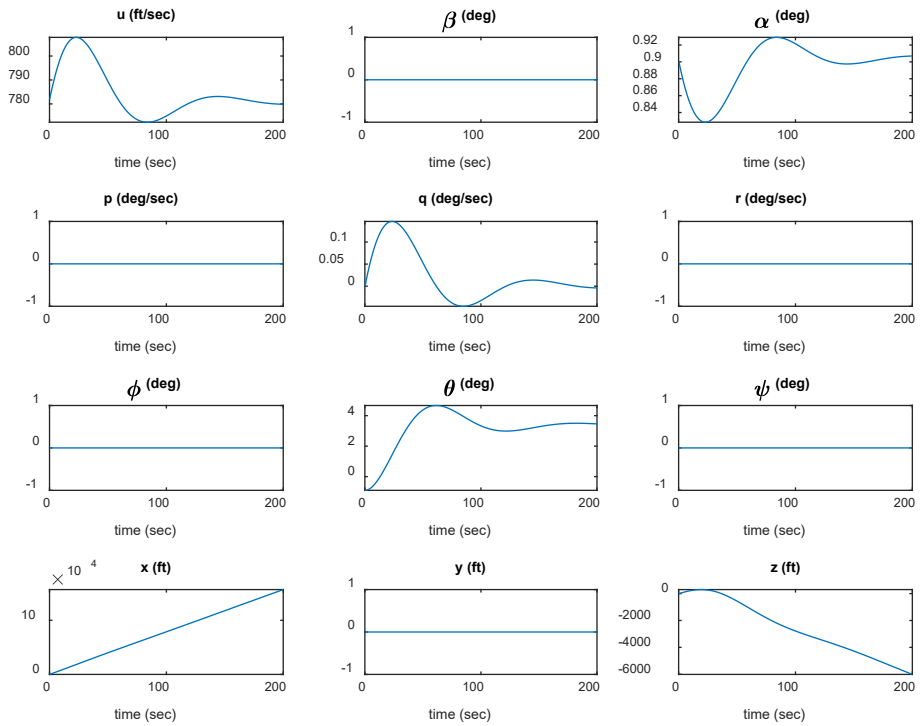
For -ve 5-degree elevator



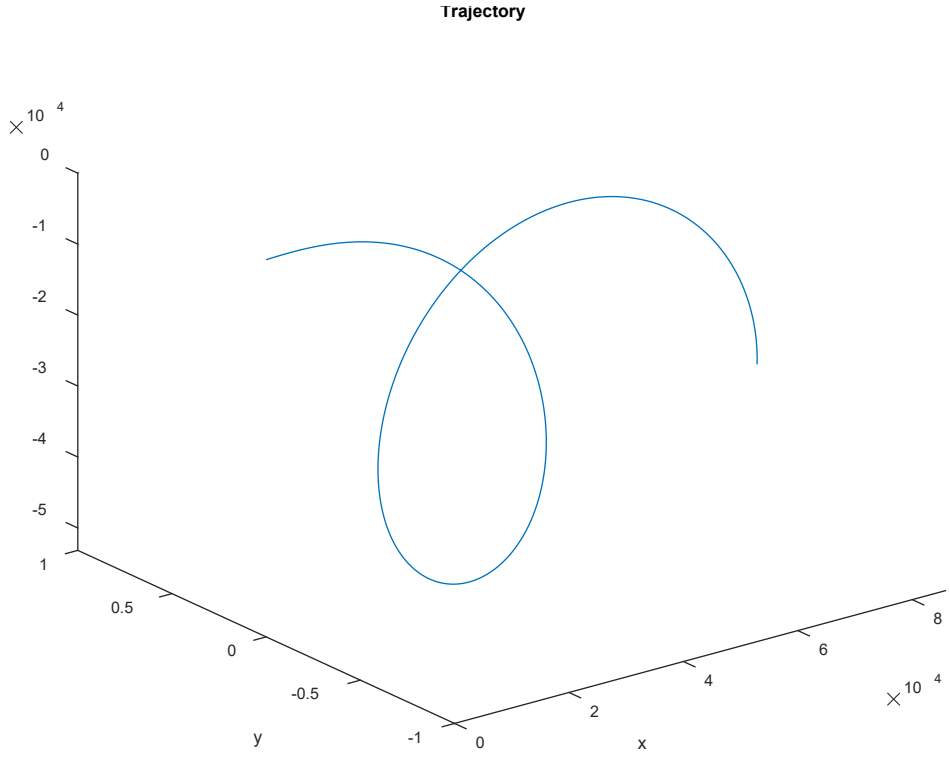


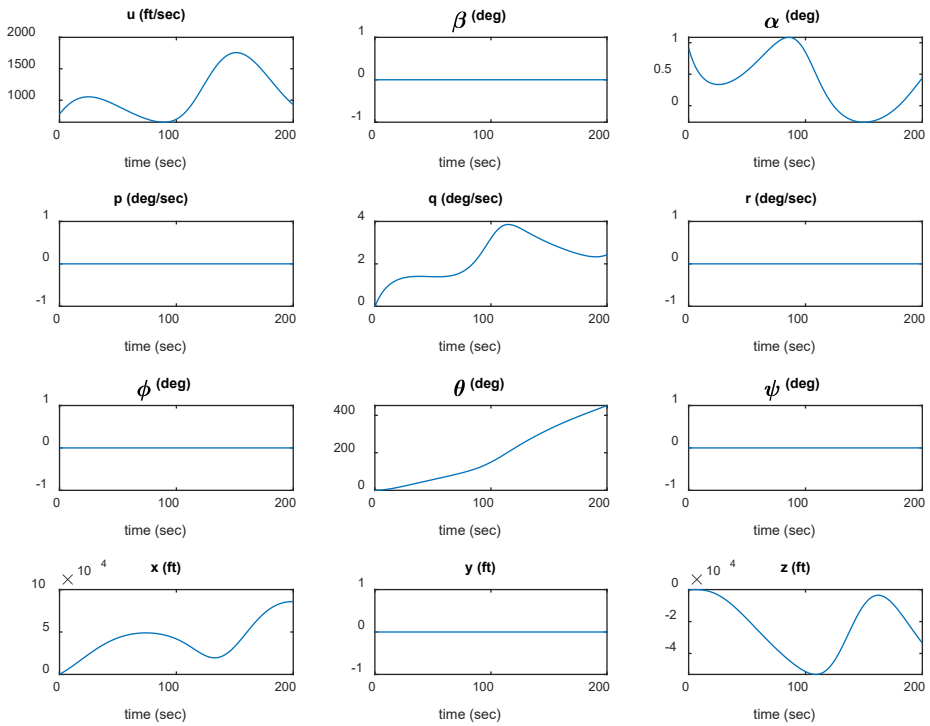
For 1000 in thrust





For 10000 in thrust





Task (4)

a) Linearization of 12 EOM for a fixed wing A/C

EOM linearization:

EOM

EOM	Airframe derivatives
$\boldsymbol{X - mg\,S_\theta = m(\dot{u} + qw - rv)}$	$\Delta X/_m = X_u\,\Delta u + X_w\,\Delta w + X_{\delta_e}\,\Delta\delta_e + X_{\delta_{th}}\,\Delta\delta_{th}$
$\boldsymbol{Y + mg\,C_\theta\,S_\varphi = m(\dot{v} + ru - pw)}$	$\Delta Y/_m = Y_\beta\,\Delta\beta + Y_r\,\Delta r + Y_p\,\Delta p + Y_{\delta a}\,\Delta\delta_a + Y_{\delta_r}\,\Delta\delta_r$
$\boldsymbol{Z + mg\,C_\theta\,C_\varphi = m(\dot{w} + pv - qu)}$	$\Delta Z/_m = Z_u\,\Delta u + Z_w\,\Delta w + Z_{\dot{w}}\,\Delta\dot{w} + Z_q\,\Delta q + Z_{\delta_e}\,\Delta\delta_e + Z_{\delta_{th}}\,\Delta\delta_{th}$

EOM	Airframe derivatives
$\boldsymbol{L = I_x\dot{p} - I_{xz}\dot{r} + qr(I_z - I_y) - I_{xz}pq}$	$\Delta L/_{I_{xx}} = L_\beta\,\Delta\beta + L_p\,\Delta p + L_r\,\Delta r + L_{\delta_r}\,\Delta\delta_r + L_{\delta_a}\,\Delta\delta_a$
$\boldsymbol{M = I_y\dot{q} + rp(I_x - I_z) - I_{xz}(p^2 - r^2)}$	$\Delta M/_{I_{yy}} = M_u\,\Delta u + M_w\,\Delta w + M_{\dot{w}}\,\Delta\dot{w} + M_q\,\Delta q + M_{\delta_e}\,\Delta\delta_e + M_{\delta_{th}}\,\Delta\delta_{th}$
$\boldsymbol{N = -I_{xz}\dot{p} + I_z\dot{r} + pq(I_y - I_x) + I_{xz}qr}$	$\Delta N/_{I_{zz}} = N_\beta\,\Delta\beta + N_p\,\Delta p + N_r\,\Delta r + N_{\delta_r}\,\Delta\delta_r + N_{\delta_a}\,\Delta\delta_a$

$$\dot{\varphi} = p + q\,S_\varphi\,T_\theta + r\,C_\varphi\,T_\theta$$
$$\dot{\theta} = q\,C_\varphi - r\,S_\varphi$$
$$\dot{\psi} = (q\,S_\varphi + r\,C_\varphi)\,sec(\theta)$$

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}C_\theta C_\psi & S_\varphi S_\theta C_\psi - C_\varphi S_\psi & C_\psi S_\theta C_\psi + S_\varphi S_\psi\\ C_\theta S_\psi & S_\varphi S_\theta S_\psi + C_\varphi C_\psi & C_\varphi S_\theta S_\psi - S_\varphi C_\psi\\ -S_\theta & S_\varphi C_\theta & C_\varphi C_\theta\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix}$$

Let:$(A = A_o + \Delta A)$

$\boldsymbol{u = u_o + \Delta u}$	$\boldsymbol{p = p_o + \Delta p}$	$\boldsymbol{X = X_o + \Delta X}$	$\boldsymbol{L = L_o + \Delta L}$	$\boldsymbol{\varphi = \varphi_o + \Delta\varphi}$
$\boldsymbol{v = v_o + \Delta v}$	$q = q_o + \Delta q$	$Y = Y_o + \Delta Y$	$M = M_o + \Delta M$	$\theta = \theta_o + \Delta\theta$
$\boldsymbol{w = w_o + \Delta w}$	$r = r_o + \Delta r$	$Y = Y_o + \Delta Y$	$N = N_o + \Delta N$	$\psi = \psi_o + \Delta\psi$

And $v_o = p_o = q_o = r_o = \varphi_o = \psi_o = 0$ but $u_o, w_o, \theta_o \neq 0$ for Cruise condition.

Longitudinal calculations:

a) $X - mg\ Sin(\theta) = m(\dot{u} + qw - rv)$

i. Recast each variable in terms of a steady-state value and a perturbed value.

$$X_o + \Delta X - mg\ Sin(\theta_o + \Delta\theta) = m(\dot{u}_o + \Delta\dot{u} + (q_o + \Delta q)(w_o + \Delta w) - (r_o + \Delta r)(v_o + \Delta v))$$

$$\sin(a+b) = \sin(a)\cos(b) + \sin(b)\cos(a)$$

$$X_o + \Delta X - mg\ (\sin(\theta_o)\cos(\Delta\theta) + \sin(\Delta\theta)\cos(\theta_o)) = m\big(\Delta\dot{u} + (\Delta q)(w_o + \Delta w) - (\Delta r)(\Delta v)\big)$$

ii. Apply the small-angle assumption to trig functions of perturbed angles

$$\sin(a \to 0) = a\,, \quad \cos(a \to 0) = 1$$

$$X_o + \Delta X - mg\ \sin(\theta_o) - mg\ \cos(\theta_o)\,\Delta\theta = m(\Delta\dot{u} + \Delta q * w_o + \Delta q * \Delta w - \Delta r * \Delta v)$$

iii. Assume products of small perturbations are negligible.

$$X_o + \Delta X - mg\ \sin(\theta_o) - mg\ \cos(\theta_o)\,\Delta\theta = m(\Delta\dot{u} + \Delta q * w_o)$$

iv. Remove the steady-state equation from the perturbed equation.

$$\therefore \Delta X - mg\ \cos(\theta_o)\,\Delta\theta = m(\Delta\dot{u} + \Delta q * w_o)$$

$$\therefore \Delta\dot{u} = \frac{\Delta X}{m} - g\ \cos(\theta_o)\,\Delta\theta - \Delta q * w_o \ , \ \frac{\Delta X}{m} = X_u\,\Delta u + X_w\,\Delta w + X_{\delta_e}\,\Delta\delta_e + X_{\delta_{th}}\,\Delta\delta_{th}$$

$$\therefore \Delta\dot{u} = X_u\,\Delta u + X_w\,\Delta w + X_{\delta_e}\,\Delta\delta_e + X_{\delta_{th}}\,\Delta\delta_{th} - g\ \cos(\theta_o)\,\Delta\theta - \Delta q * w_o$$

b) $Z + mg\ Cos(\theta) * Cos(\varphi) = m(\dot{w} + pv - qu)$

Same as (a)

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$

$$\therefore (1 - Z_{\dot{w}})\Delta\dot{w} = Z_u\,\Delta u + Z_w\,\Delta w + (Z_q + u_o)\,\Delta q + Z_{\delta_e}\,\Delta\delta_e + Z_{\delta_{th}}\,\Delta\delta_{th} - g\ \sin(\theta_o)\,\Delta\theta$$

$$\therefore \Delta\dot{w} = \frac{Z_u}{(1 - Z_{\dot{w}})}\,\Delta u + \frac{Z_w}{(1 - Z_{\dot{w}})}\,\Delta w + \frac{(Z_q + u_o)}{(1 - Z_{\dot{w}})}\,\Delta q + \frac{Z_{\delta_e}}{(1 - Z_{\dot{w}})}\,\Delta\delta_e + \frac{Z_{\delta_{th}}}{(1 - Z_{\dot{w}})}\,\Delta\delta_{th} - \frac{g\ \sin(\theta_o)}{(1 - Z_{\dot{w}})}\Delta\theta$$

c) $M = I_y\dot{q} + rp(I_x - I_z) - I_{xz}(p^2 - r^2)$

$$\therefore \Delta\dot{q} = M_u\,\Delta u + M_w\,\Delta w + M_{\dot{w}} * \left(\frac{Z_u}{(1-Z_{\dot{w}})}\,\Delta u + \frac{Z_w}{(1-Z_{\dot{w}})}\,\Delta w + \frac{(Z_q+u_o)}{(1-Z_{\dot{w}})}\,\Delta q + \frac{Z_{\delta_e}}{(1-Z_{\dot{w}})}\,\Delta\delta_e + \frac{Z_{\delta_{th}}}{(1-Z_{\dot{w}})}\,\Delta\delta_{th} - \frac{g\,\sin(\theta_o)}{(1-Z_{\dot{w}})}\Delta\theta\right) + M_q\,\Delta q + M_{\delta_e}\,\Delta\delta_e + M_{\delta_{th}}\,\Delta\delta_{th}$$

$$\therefore \Delta\dot{q} = \left(M_u + M_{\dot{w}}\frac{Z_u}{(1-Z_{\dot{w}})}\right)\Delta u + \left(M_w + M_{\dot{w}}\frac{Z_w}{(1-Z_{\dot{w}})}\right)\Delta w + \left(M_q + M_{\dot{w}}\frac{(Z_q+u_o)}{(1-Z_{\dot{w}})}\right)\Delta q + \left(M_{\delta_e} + M_{\dot{w}}\frac{Z_{\delta_e}}{(1-Z_{\dot{w}})}\right)\Delta\delta_e + \left(M_{\delta_{th}} + M_{\dot{w}}\frac{Z_{\delta_{th}}}{(1-Z_{\dot{w}})}\right)\Delta\delta_{th} - M_{\dot{w}} * \frac{g\,\sin(\theta_o)}{(1-Z_{\dot{w}})}\Delta\theta$$

d) $\dot{\theta} = q\,C_\varphi - r\,S_\varphi$

$$\therefore \Delta\dot{\theta} = \Delta q$$

Lateral calculations:

a) $Y + mg\,C_\theta\,S_\varphi = m(\dot{v} + ru - pw)$

$$\Delta\dot{v} = Y_\beta\,\Delta\beta + Y_r\,\Delta r + Y_p\,\Delta p + Y_{\delta a}\,\Delta\delta_a + Y_{\delta_r}\,\Delta\delta_r + g\,\cos(\theta_o)\,\Delta\emptyset - \Delta r * u_o + w_o * \Delta p$$

$$\Delta\dot{\beta} = \frac{\Delta\dot{v}}{V_{to}}$$

$$\therefore \Delta\dot{\beta} = \frac{Y_\beta}{V_{to}}\,\Delta\beta + \frac{Y_r - u_o}{V_{to}}\,\Delta r + \frac{Y_p + w_o}{V_{to}}\,\Delta p + \frac{Y_{\delta a}}{V_{to}}\,\Delta\delta_a + \frac{Y_{\delta_r}}{V_{to}}\,\Delta\delta_r + \frac{g}{V_{to}}\,\cos(\theta_o)\,\Delta\varphi$$

b) $L = I_x\dot{p} - I_{xz}\dot{r} + qr(I_z - I_y) - I_{xz}pq$

$$eq(1):\quad \Delta\dot{p} - \frac{I_{xz}}{I_x} * \Delta\dot{r} = \frac{\Delta L}{I_x} = L_\beta\,\Delta\beta + L_p\,\Delta p + L_r\,\Delta r + L_{\delta_r}\,\Delta\delta_r + L_{\delta_a}\,\Delta\delta_a$$

c) $N = -I_{xz}\dot{p} + I_z\dot{r} + pq(I_y - I_x) + I_{xz}qr$

$$eq(2):\quad \Delta\dot{r} - \frac{I_{xz}}{I_z} * \Delta\dot{p} = \frac{\Delta N}{I_z} = N_\beta\,\Delta\beta + N_p\,\Delta p + N_r\,\Delta r + N_{\delta_r}\,\Delta\delta_r + N_{\delta_a}\,\Delta\delta_a$$

Solve $eq(1)$ *and* $eq(2)$ *in* $(\Delta\dot{\boldsymbol{r}} \quad \Delta\dot{\boldsymbol{p}})$ *we get*:

$$\therefore \Delta\dot{p} = \left(\frac{\Delta L}{I_x} + \frac{I_{xz}}{I_x} * \frac{\Delta N}{I_z}\right) * G \quad , \quad where \quad G = \frac{I_x I_z}{I_x I_z - (I_{xz})^2}$$

$$\therefore \Delta\dot{r} = \left(\frac{I_{xz}}{I_z} * \frac{\Delta L}{I_x} + \frac{\Delta N}{I_z}\right) * G \quad , \quad where \quad G = \frac{I_x I_z}{I_x I_z - (I_{xz})^2}$$

$$\frac{\Delta L}{I_x} = \sum L_i \, \Delta i \quad , \quad \frac{\Delta N}{I_z} = \sum N_i \, \Delta i$$

$$\therefore \Delta\dot{p} = \sum G * \left(L_i + \frac{I_{xz}}{I_x} * N_i\right) \Delta i = \sum L_i^* \, \Delta i$$

$$= L_\beta^* \, \Delta\beta + L_p^* \, \Delta p + L_r^* \, \Delta r + L_{\delta_r}^* \, \Delta\delta_r + L_{\delta_a}^* \, \Delta\delta_a$$

$$\therefore \Delta\dot{r} = \sum G * \left(N_i + \frac{I_{xz}}{I_z} * L_i\right) \Delta i = \sum N_i^* \, \Delta i$$

$$= N_\beta^* \, \Delta\beta + N_p^* \, \Delta p + N_r^* \, \Delta r + N_{\delta_r}^* \, \Delta\delta_r + N_{\delta_a}^* \, \Delta\delta_a$$

d) $\dot{\varphi} = p + q \, Sin(\varphi) \, Tan(\theta) + r \, Cos(\varphi) \, Tan(\theta)$

$$\dot{\varphi} \cos(\theta) = p \cos(\theta) + q \, Sin(\varphi) \, sin(\theta) + r \, Cos(\varphi) \, sin(\theta)$$

$$\therefore \Delta\dot{\varphi} = \Delta p + \Delta r * \tan(\theta_0)$$

e) $\dot{\psi} = \left(q \, S_\varphi + r \, C_\varphi\right) \, sec(\theta)$

$$\therefore \Delta\dot{\psi} = \Delta r * \sec(\theta_0)$$

State space Model:

Longitudinal

- $\Delta\dot{u} = X_u \, \Delta u + X_w \, \Delta w + X_{\delta_e} \, \Delta\delta_e + X_{\delta_{th}} \, \Delta\delta_{th} - g \, \cos(\theta_o) \, \Delta\theta - \Delta q * w_o$
- $\therefore \Delta\dot{w} = \frac{Z_u}{(1-Z_{\dot{w}})} \, \Delta u + \frac{Z_w}{(1-Z_{\dot{w}})} \, \Delta w + \frac{(Z_q+u_o)}{(1-Z_{\dot{w}})} \, \Delta q + \frac{Z_{\delta_e}}{(1-Z_{\dot{w}})} \, \Delta\delta_e + \frac{Z_{\delta_{th}}}{(1-Z_{\dot{w}})} \, \Delta\delta_{th} - \frac{g \, \sin(\theta_o)}{(1-Z_{\dot{w}})} \Delta\theta$
- $\therefore \Delta\dot{q} = \left(M_u + M_{\dot{w}} \frac{Z_u}{(1-Z_{\dot{w}})}\right) \Delta u + \left(M_w + M_{\dot{w}} \frac{Z_w}{(1-Z_{\dot{w}})}\right) \Delta w + \left(M_q + M_{\dot{w}} \frac{(Z_q+u_o)}{(1-Z_{\dot{w}})}\right) \Delta q + \left(M_{\delta_e} + M_{\dot{w}} \frac{Z_{\delta_e}}{(1-Z_{\dot{w}})}\right) \Delta\delta_e + \left(M_{\delta_{th}} + M_{\dot{w}} \frac{Z_{\delta_{th}}}{(1-Z_{\dot{w}})}\right) \Delta\delta_{th} - M_{\dot{w}} * \frac{g \, \sin(\theta_o)}{(1-Z_{\dot{w}})} \Delta\theta$
- $\therefore \Delta\dot{\theta} = \Delta q$

$$\begin{bmatrix}\Delta\dot{u}\\ \Delta\dot{w}\\ \Delta\dot{q}\\ \Delta\dot{\theta}\end{bmatrix}=\begin{bmatrix} X_u & X_w & -w_o & -g\cos(\theta_o)\\ \dfrac{Z_u}{(1-Z_{\dot{w}})} & \dfrac{Z_w}{(1-Z_{\dot{w}})} & \dfrac{(Z_q+u_o)}{(1-Z_{\dot{w}})} & \dfrac{-g\sin(\theta_o)}{(1-Z_{\dot{w}})}\\ M_u+M_{\dot{w}}\dfrac{Z_u}{(1-Z_{\dot{w}})} & M_w+M_{\dot{w}}\dfrac{Z_w}{(1-Z_{\dot{w}})} & M_q+M_{\dot{w}}\dfrac{(Z_q+u_o)}{(1-Z_{\dot{w}})} & -M_{\dot{w}}*\dfrac{g\sin(\theta_o)}{(1-Z_{\dot{w}})}\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix}$$
$$+\begin{bmatrix} X_{\delta_e} & X_{\delta_{th}}\\ \dfrac{Z_{\delta_e}}{(1-Z_{\dot{w}})} & \dfrac{Z_{\delta_{th}}}{(1-Z_{\dot{w}})}\\ M_{\delta_e}+M_{\dot{w}}\dfrac{Z_{\delta_e}}{(1-Z_{\dot{w}})} & M_{\delta_{th}}+M_{\dot{w}}\dfrac{Z_{\delta_{th}}}{(1-Z_{\dot{w}})}\\ 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\delta_e\\ \Delta\delta_{th}\end{bmatrix}$$

Lateral

- $\therefore \Delta\dot{\beta}=\frac{Y_\beta}{V_{to}}\Delta\beta+\frac{Y_r-u_o}{V_{to}}\Delta r+\frac{Y_p+w_o}{V_{to}}\Delta p+\frac{Y_{\delta a}}{V_{to}}\Delta\delta_a+\frac{Y_{\delta_r}}{V_{to}}\Delta\delta_r+\frac{g}{V_{to}}\cos(\theta_o)\Delta\varphi$
- $\therefore \Delta\dot{p}=\sum G*\left(L_i+\frac{I_{xz}}{I_x}*N_i\right)\Delta i=\sum L_i^*\Delta i=L_\beta^*\Delta\beta+L_p^*\Delta p+L_r^*\Delta r+L_{\delta_r}^*\Delta\delta_r+L_{\delta_a}^*\Delta\delta_a$
- $\therefore \Delta\dot{r}=\sum G*\left(N_i+\frac{I_{xz}}{I_z}*L_i\right)\Delta i=\sum N_i^*\Delta i=N_\beta^*\Delta\beta+N_p^*\Delta p+N_r^*\Delta r+N_{\delta_r}^*\Delta\delta_r+N_{\delta_a}^*\Delta\delta_a$
- $\therefore \Delta\dot{\varphi}=\Delta p+\Delta r*\tan(\theta_0)$
- $\therefore \Delta\dot{\psi}=\Delta r*\sec(\theta_0)$

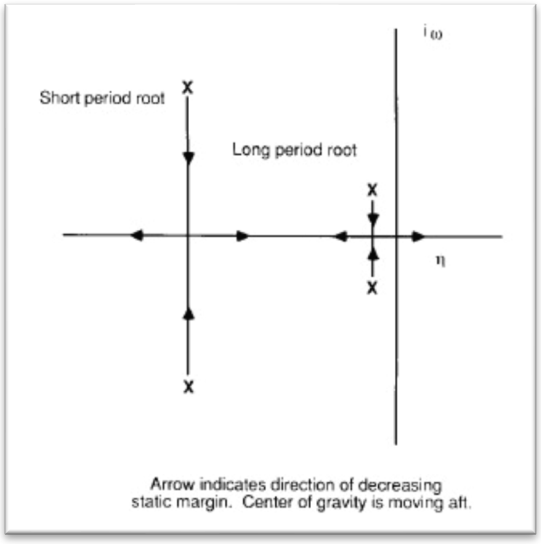
$$\begin{bmatrix}\Delta\dot{\beta}\\ \Delta\dot{p}\\ \Delta\dot{r}\\ \Delta\dot{\varphi}\\ \Delta\dot{\psi}\end{bmatrix}=\begin{bmatrix}\dfrac{Y_\beta}{V_{to}} & \dfrac{Y_p+w_o}{V_{to}} & \dfrac{Y_r-u_o}{V_{to}} & \dfrac{g\cos(\theta_o)}{V_{to}} & 0\\ L_\beta^* & L_p^* & L_r^* & 0 & 0\\ N_\beta^* & N_p^* & N_r^* & 0 & 0\\ 0 & 1 & \tan(\theta_0) & 0 & 0\\ 0 & 0 & \sec(\theta_0) & 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\beta\\ \Delta p\\ \Delta r\\ \Delta\varphi\\ \Delta\psi\end{bmatrix}+\begin{bmatrix}\dfrac{Y_{\delta a}}{V_{to}} & \dfrac{Y_{\delta_r}}{V_{to}}\\ L_{\delta_a}^* & L_{\delta_r}^*\\ \Delta\delta_a & N_{\delta_r}^*\\ 0 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\delta_a\\ \Delta\delta_r\end{bmatrix}$$

b) (1) Longitudinal dynamics approximations

The longitudinal dynamics of the aircraft refer to dynamics in the x-z plane in the longitudinal axis

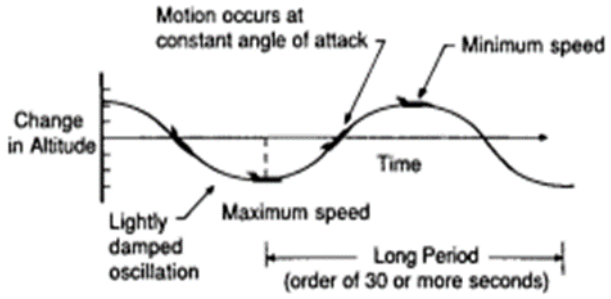
there are two types of approximations for that system of equations

1. Long period approximation (mode)
2. Short period approximation (mode)



By solving the A matrix from part, a we get 2 pairs of conjugate roots the pair which has small real part has big settling time according to $T_s = \frac{4}{\zeta \omega_n}$
on the other hand with the pair which has bigger real part

Long period approximation (mode)

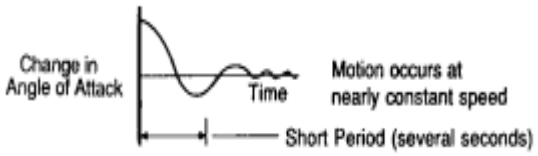


When the aircraft is exposed to some disturbances in the longitudinal direction a change in the altitude is formed due to this disturbance assuming that the angle of attack is constant the approximation to the long period mode is obtained by neglecting the pitching moment equation and assuming that the changes in angle of attack is zero.

$$\triangle \alpha = 0 \quad \Delta \dot{w} = 0$$
$$\triangle \alpha = \frac{\Delta w}{U_o} \quad \Delta \dot{q} = 0$$

$$\begin{bmatrix}\dot{u}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix} X_u+\frac{w_o Z_u}{Z_q+u_o} & -g\cos(\theta_o)-\frac{w_o g\sin(\theta_o)}{Z_q+u_o}\\ \frac{-Z_u}{Z_q+u_o} & \frac{g\sin(\theta_o)}{Z_q+u_o}\end{bmatrix}\begin{bmatrix}u\\ \theta\end{bmatrix}$$
$$+\begin{bmatrix} X_{\delta_e}+\frac{w_o Z_{\delta_e}}{Z_q+u_o} & X_{\delta_{th}}+\frac{w_o Z_{\delta_{th}}}{Z_q+u_o}\\ \frac{-Z_{\delta_e}}{Z_q+u_o} & \frac{-Z_{\delta_{th}}}{Z_q+u_o}\end{bmatrix}\begin{bmatrix}\Delta\delta_e\\ \Delta\delta_{th}\end{bmatrix}$$

short period approximation (mode)



The same definition which we applied in the previous mode we will use here but with taking in consideration that the disturbance will damp quickly and but there will be a change in angle of attack and neglecting the change in velocity.

$$\Delta u = 0$$

$$\Delta\dot{\theta} = 0$$

$$M_\alpha = \frac{1}{I_y} * \frac{\partial M}{\partial \alpha} = \frac{1}{I_y} * \frac{\partial M}{\partial\left(\frac{\Delta w}{u_o}\right)} = \frac{u_o}{I_y} * \frac{\partial M}{\partial w} = u_o M_w$$

$$Z_\propto = u_o Z_w$$

$$M_{\dot{\alpha}} = u_o M_{\dot{w}}$$

$$\begin{bmatrix}\dot{\alpha}\\ \dot{q}\end{bmatrix}=\begin{bmatrix} \frac{Z_\alpha}{u_o} & 1\\ M_\alpha+\frac{M_{\dot{\alpha}} Z_\alpha}{u_o} & M_q+M_{\dot{\alpha}}\end{bmatrix}\begin{bmatrix}\alpha\\ q\end{bmatrix}+\begin{bmatrix} Z_{\delta_e} & Z_{\delta_{th}}\\ M_{\delta_e}+\frac{M_\alpha Z_{\delta_e}}{u_o} & M_{\dot{\alpha}}+\frac{M_\alpha Z_{\delta_{th}}}{u_o}\end{bmatrix}\begin{bmatrix}\Delta\delta_e\\ \Delta\delta_{th}\end{bmatrix}$$

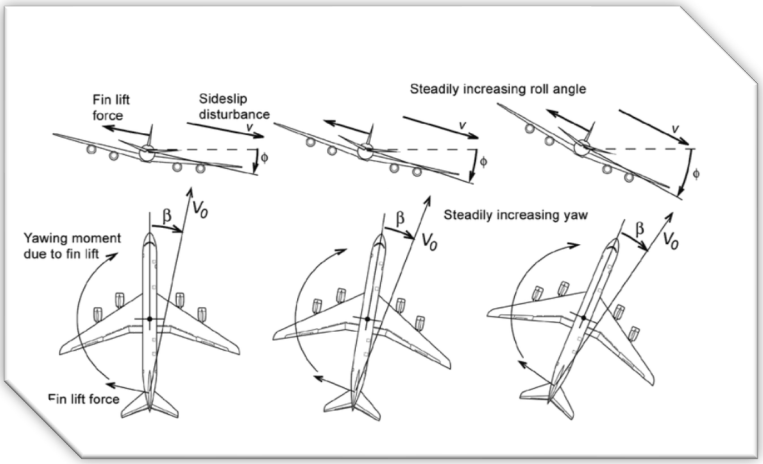
and by solving the characteristic equation of the system and getting its Eigen values we finally reach that;

$$w_n = \sqrt{(M_q \frac{Z_\alpha}{u_o} - M_\alpha)}$$

$$\zeta = \frac{\frac{Z_\alpha}{u_o} + M_q + M_{\dot{\alpha}}}{-2w_n}$$

b) (2) Lateral dynamics approximations

3-DOF spiral mode



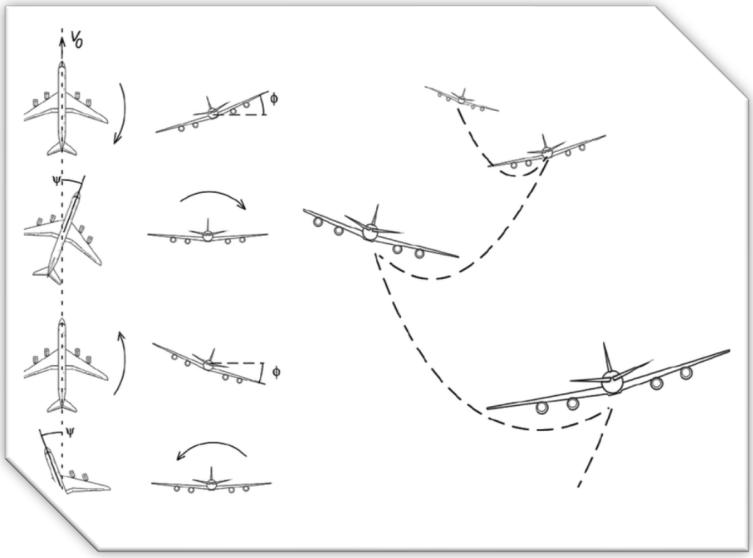
As it is clear in the attached figure how does the airplane behave when it is in the spiral mode.

The side force equation of motion will be neglected and will put$\Delta\beta = 0$.

Therefore, the 3 DOF spiral approximation will be as shown in the following:

$$\begin{bmatrix} \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\varphi} \end{bmatrix} = \begin{bmatrix} \grave{L_p} & \grave{L_r} & 0 \\ \grave{N_p} & \grave{N_r} & 0 \\ 1 & \tan(\theta_o) & 0 \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta r \\ \Delta\varphi \end{bmatrix} + \begin{bmatrix} \acute{L}_{\delta_r} \\ \acute{N}_{\delta_r} \\ 0 \end{bmatrix} [\Delta\delta_r]$$

2-DOF Dutch-Roll mode



The combination of the roll and yaw approximations is called the Dutch roll Motion.

In this equation the rolling moment equation is neglected and we put$\Delta\varphi = 0$.

Therefore, the 2 DOF spiral approximation will be as shown in the following:

$$\begin{bmatrix}\Delta\dot{\beta}\\ \Delta\dot{r}\end{bmatrix} = \begin{bmatrix} Y_v & \dfrac{Y_r - 1}{V_{to}} \\ \grave{N}_\beta & \grave{N}_r \end{bmatrix}\begin{bmatrix}\Delta\beta\\ \Delta r\end{bmatrix} + \begin{bmatrix} {Y_{\delta_a}}^* & {Y_{\delta_r}}^* \\ \acute{N}_{\delta_a} & \acute{N}_{\delta_r}\end{bmatrix}\begin{bmatrix}\Delta\delta_a\\ \Delta\delta_r\end{bmatrix}$$

1-Roll mode

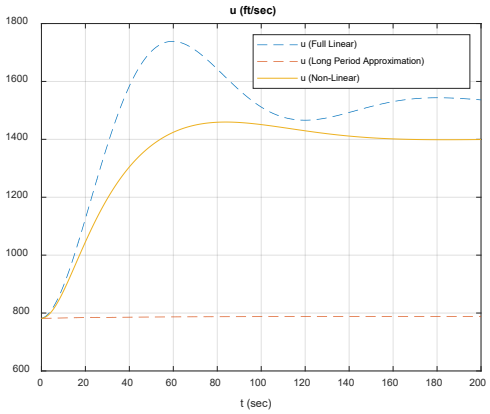
It has been seen that the rolling mode almost corresponds to the pure rolling. Thus it is reasonable to neglect all equations except the rolling moment equation and all perturbations except p. So we will reach the following relation;

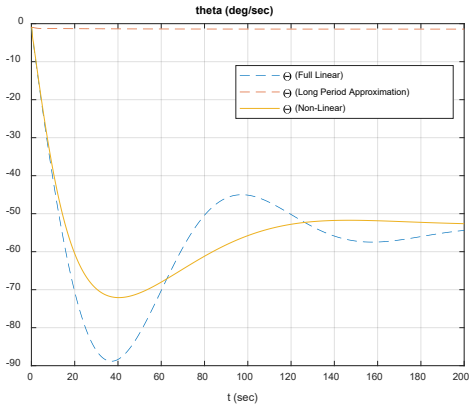
$$\dot{P} - L_p{}'p = L_{\delta a}{}'\delta_a$$

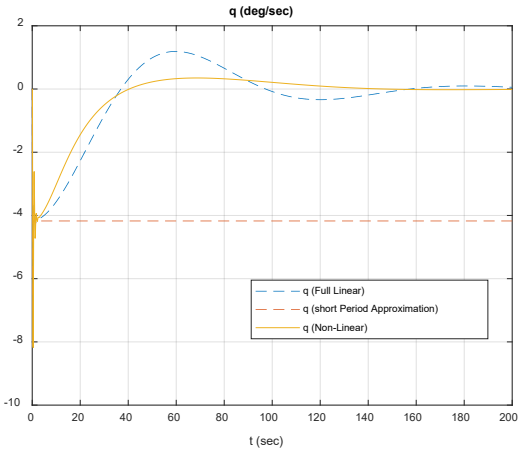
c) Results of the longitudinal dynamics

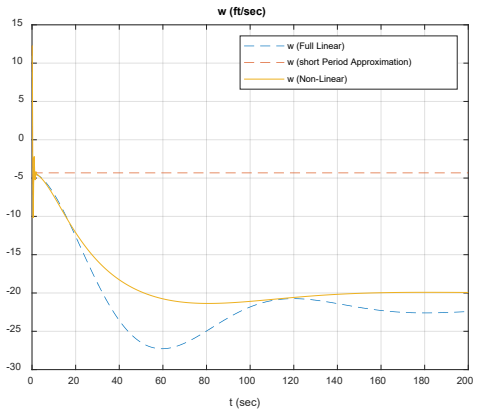
For states u , θ, w , q

$$\delta_{elevator} = 1$$

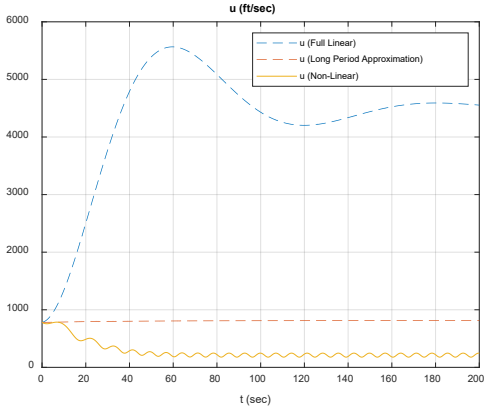


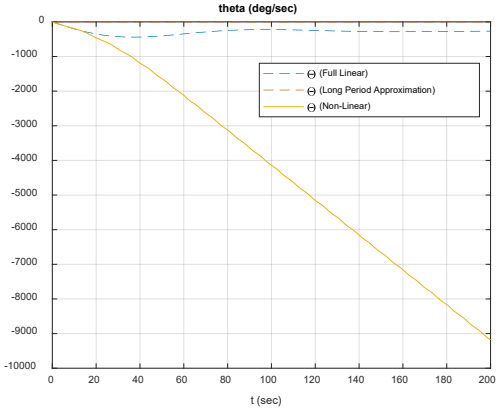


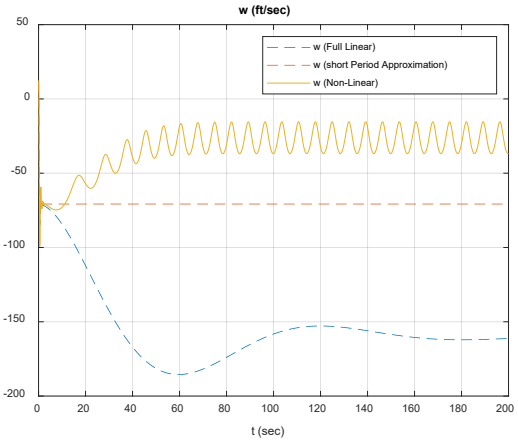


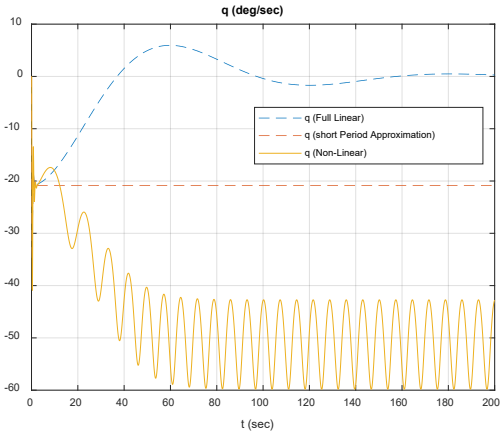


$$\delta_{elevator} = 5$$

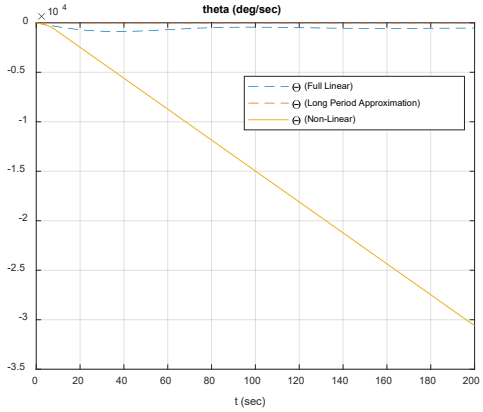


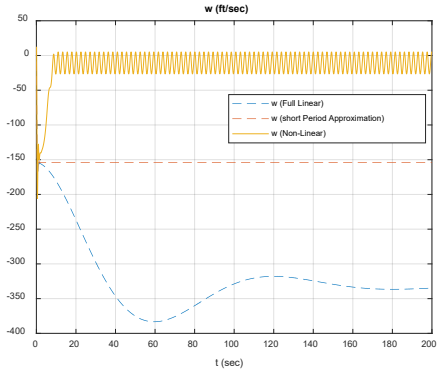


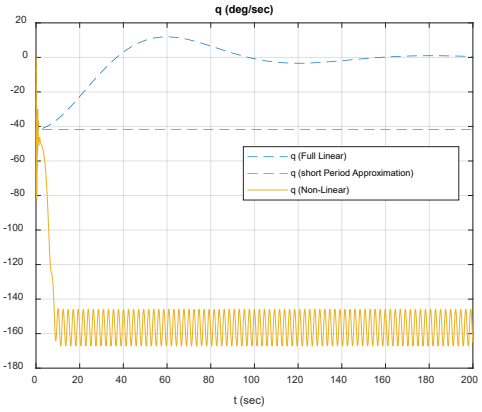


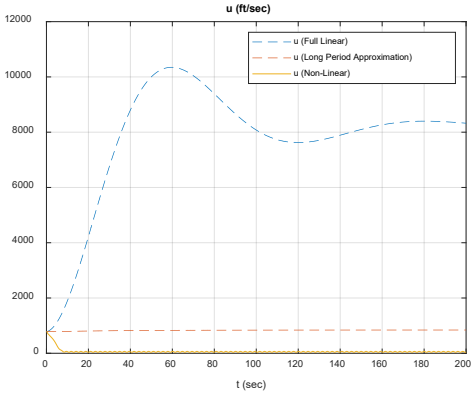


$$\delta_{elevator} = 10$$

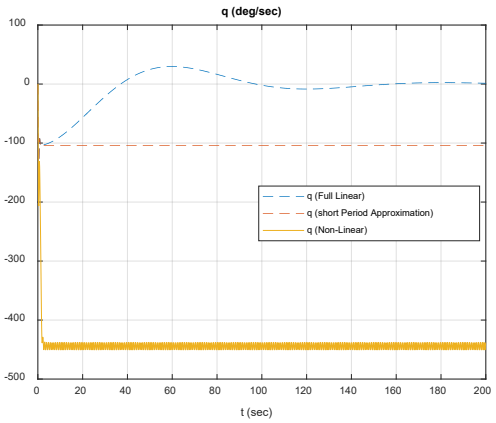


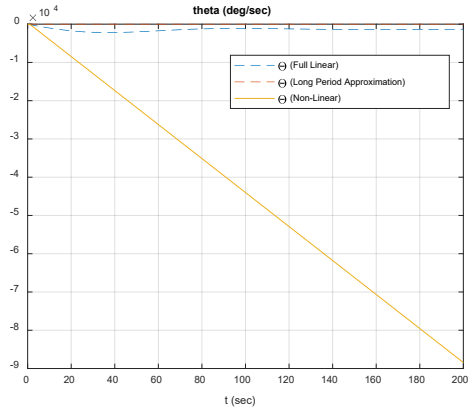


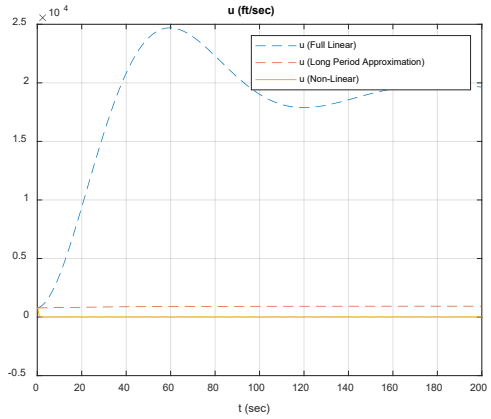


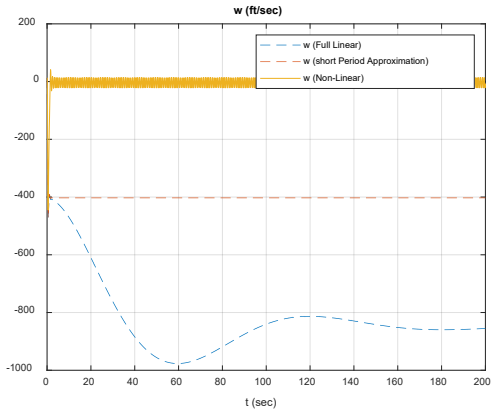


$$\delta_{elevator} = 25$$

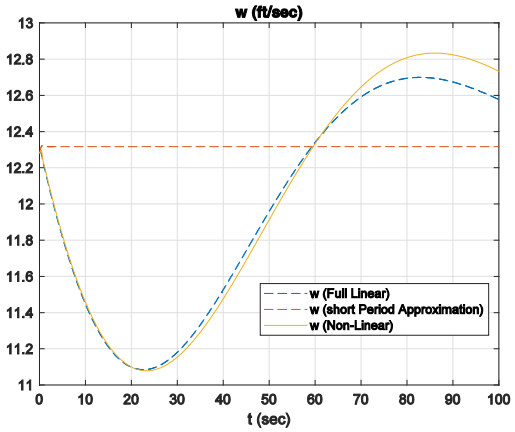


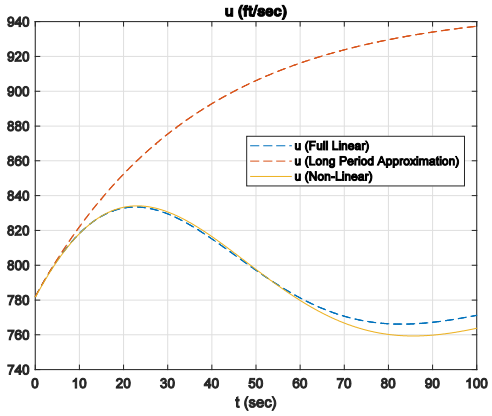


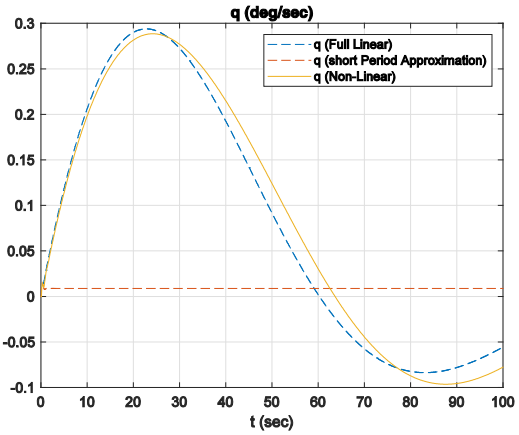


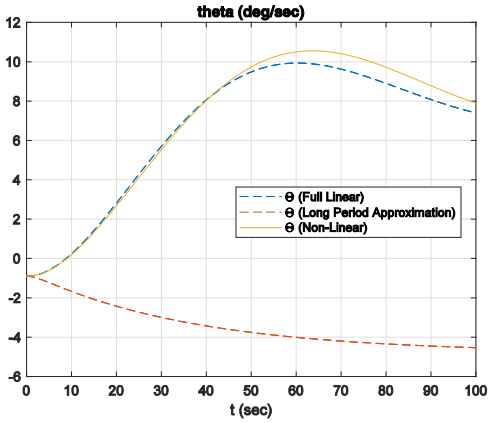


$$\delta_{thrust} = 2000$$

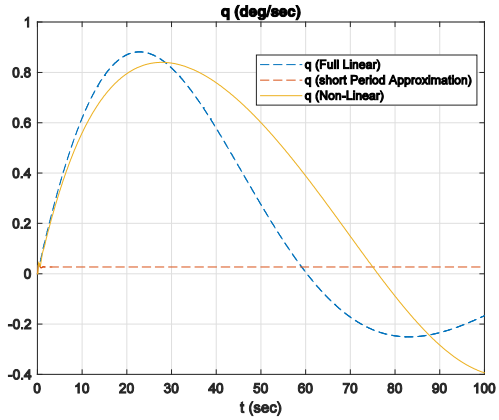


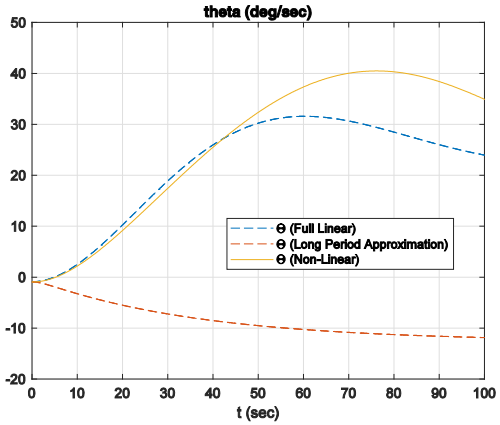


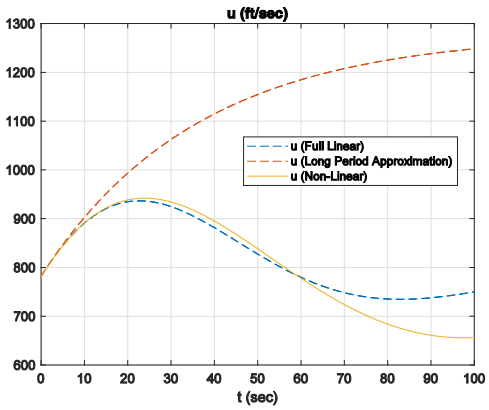


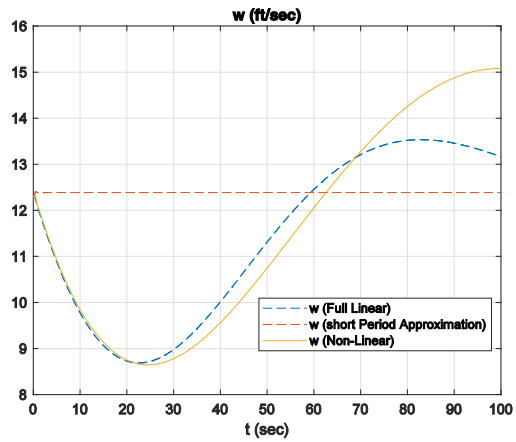


$$\delta_{thrust} = 6000$$

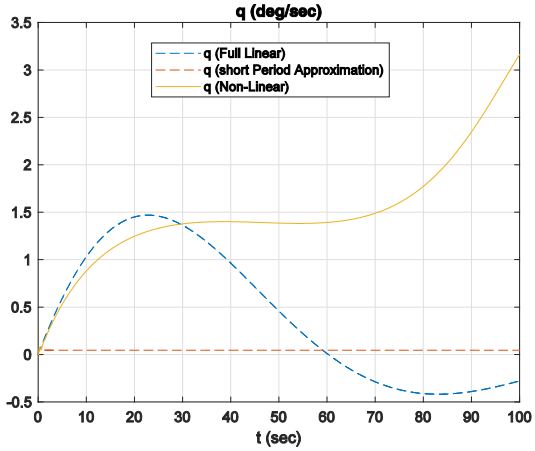


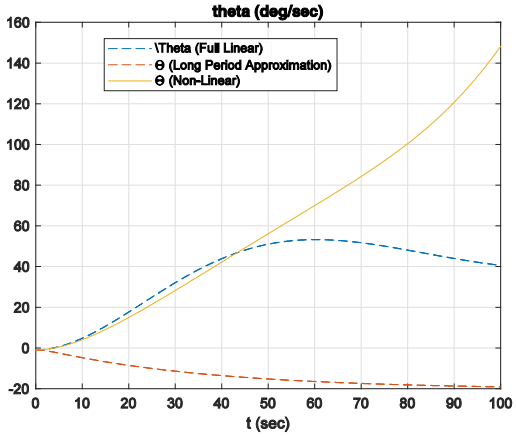


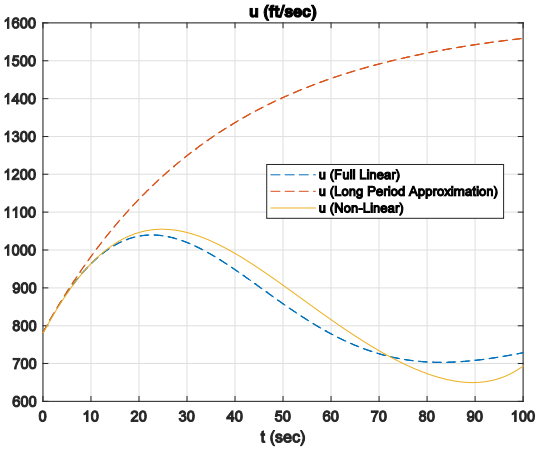


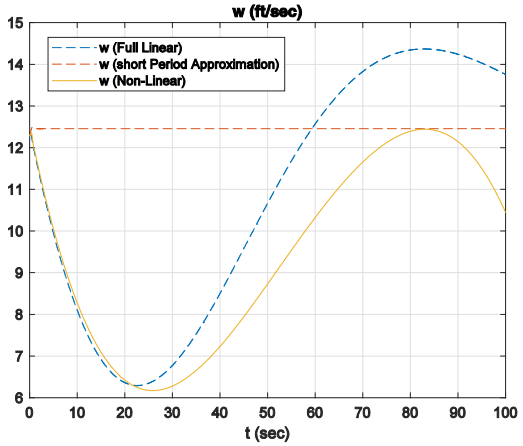


$$\delta_{thrust} = 10000$$



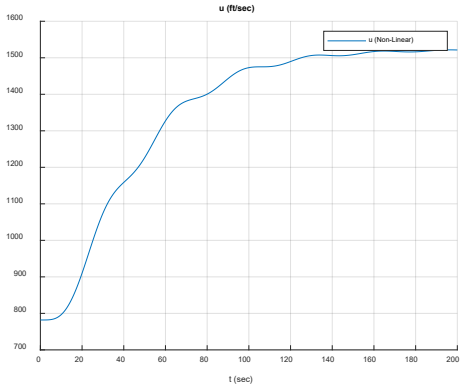


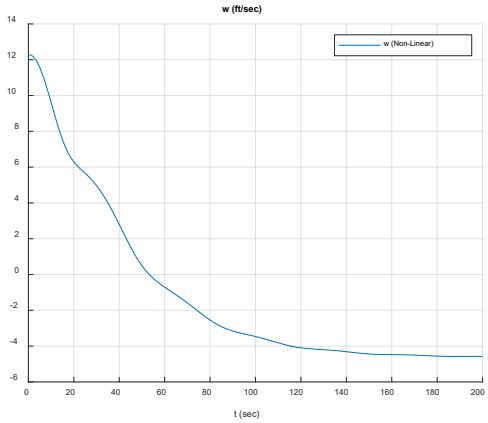


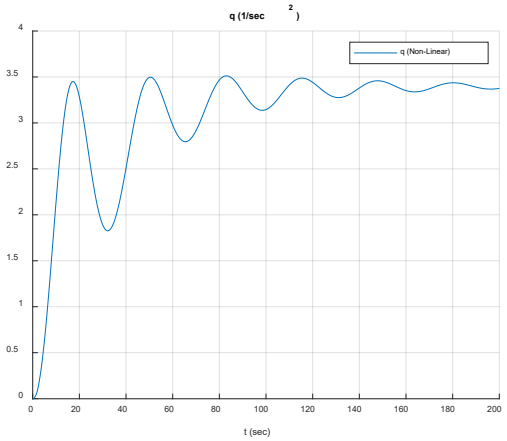


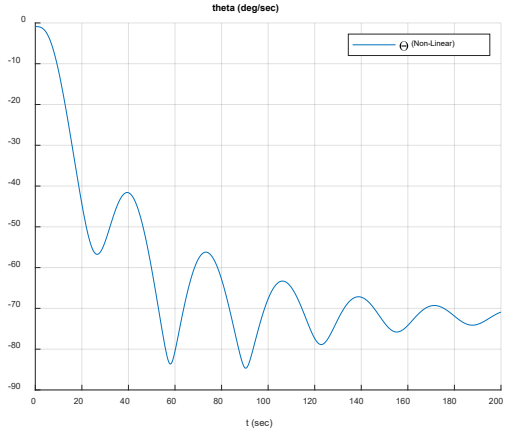
d) Results of Step response (lateral Dynamics)

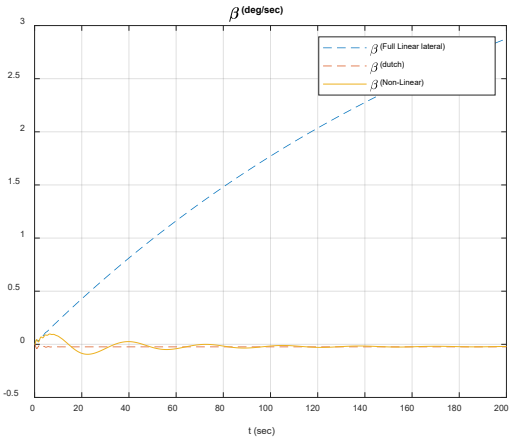
Response due to Aileron ($\delta_a = 1$)

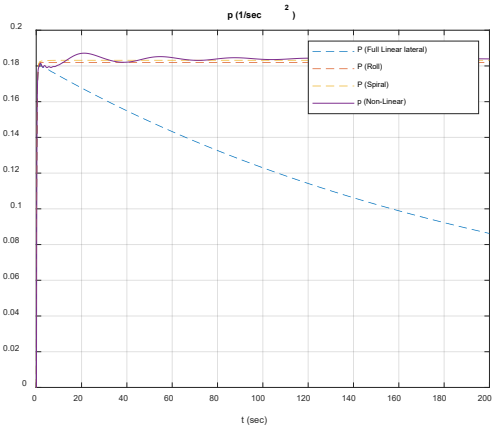


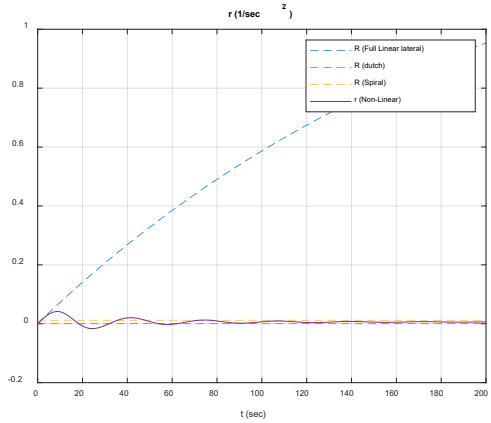


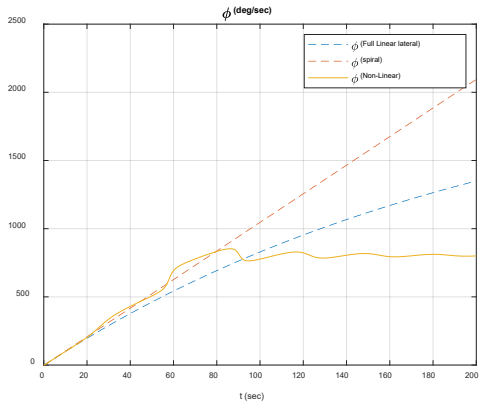


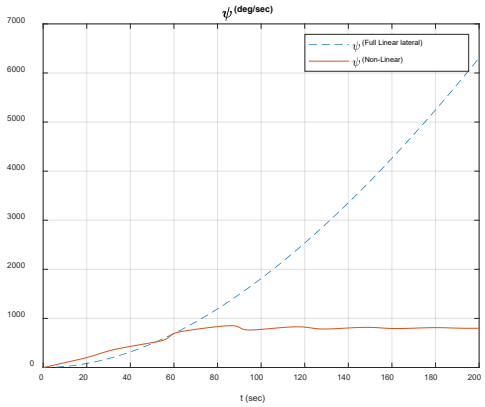




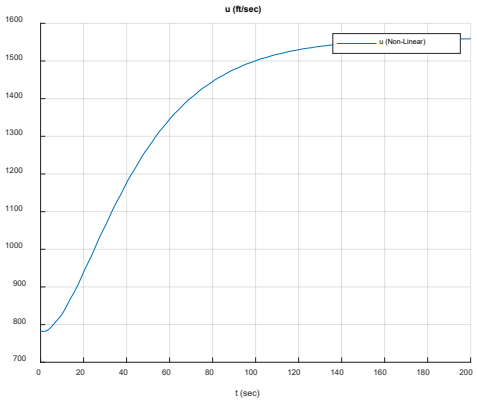


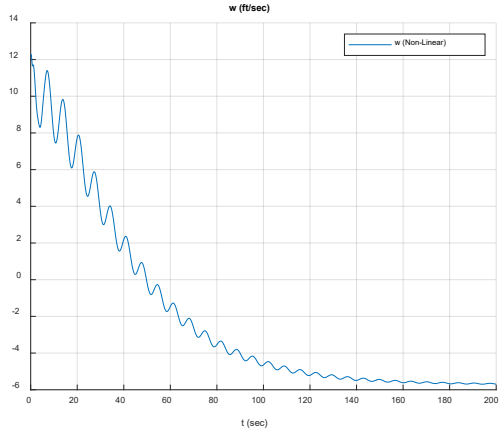


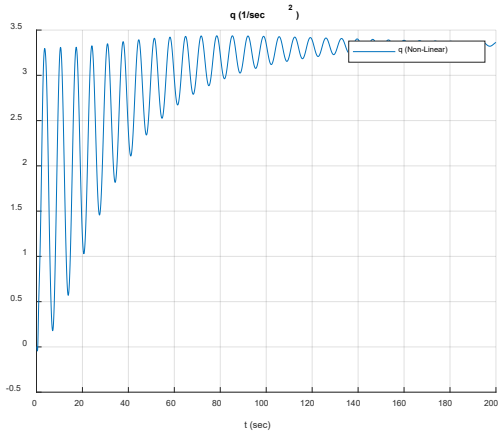


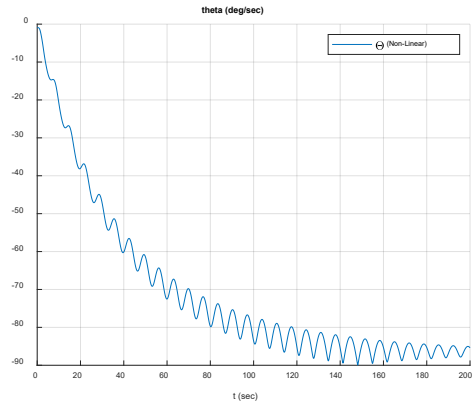


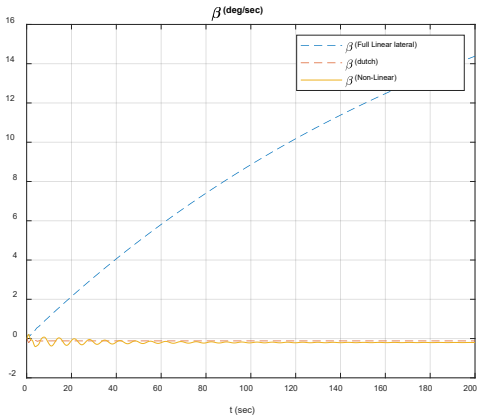
Response due to Aileron ($\delta_a = 5$)

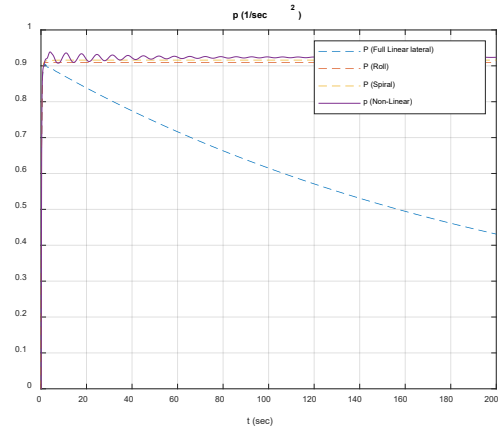




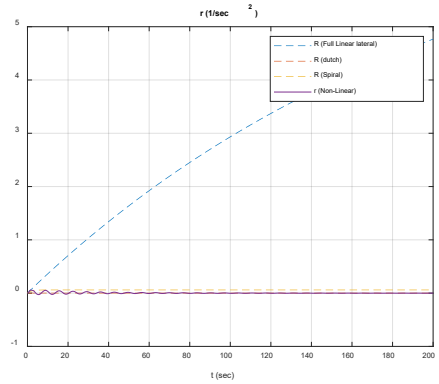

q (1/sec 2)
q (Non-Linear)
t (sec)

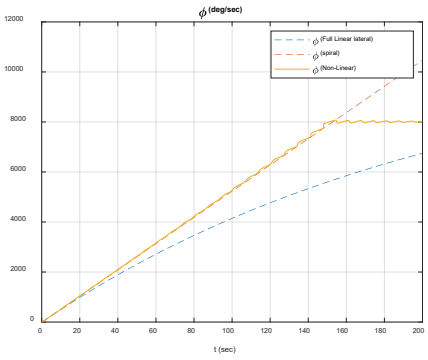

theta (deg/sec)
Θ (Non-Linear)
t (sec)

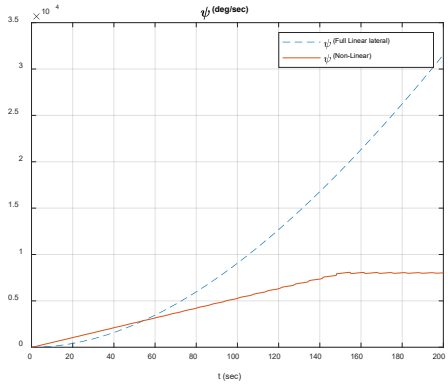

β (deg/sec)
β (Full Linear lateral)
β (dutch)
β (Non-Linear)
t (sec)


p (1/sec 2)
P (Full Linear lateral)
P (Roll)
P (Spiral)
p (Non-Linear)
t (sec)

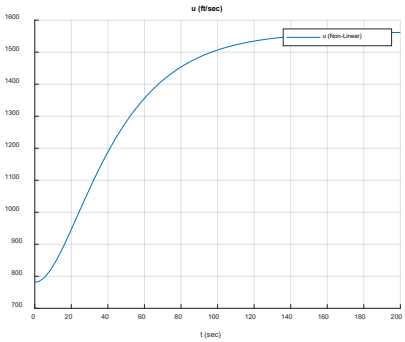
‘

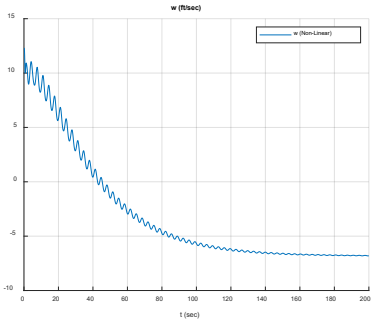

r (1/sec 2)
R (Full Linear lateral)
R (dutch)
R (Spiral)
r (Non-Linear)
t (sec)

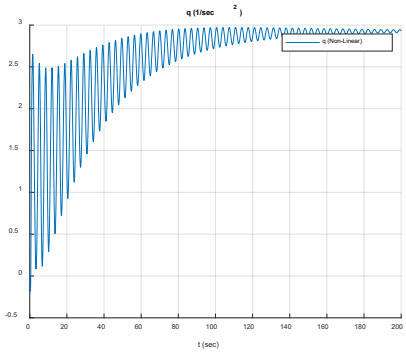

ϕ (deg/sec)
ϕ (Full Linear lateral)
ϕ (spiral)
ϕ (Non-Linear)
t (sec)

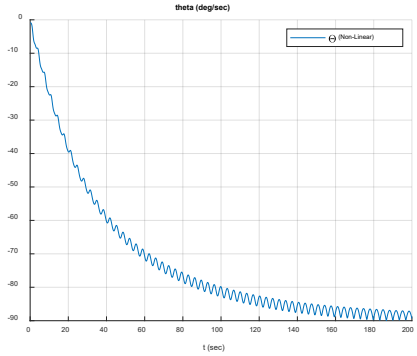


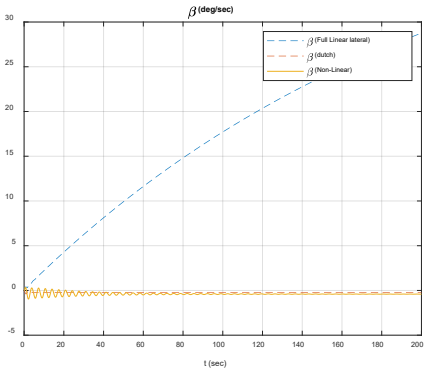
Response due to Rudder ($\delta_a = 10$)

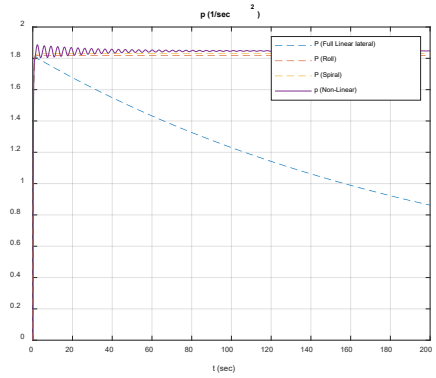


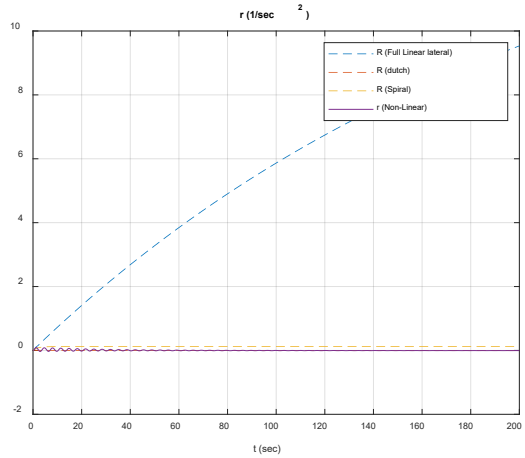


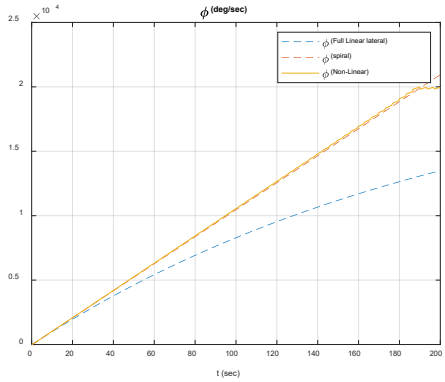


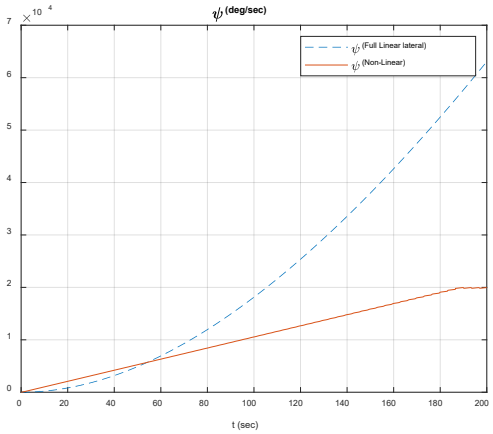




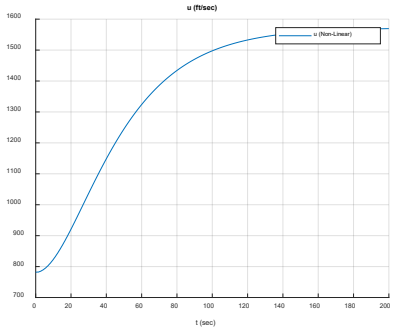


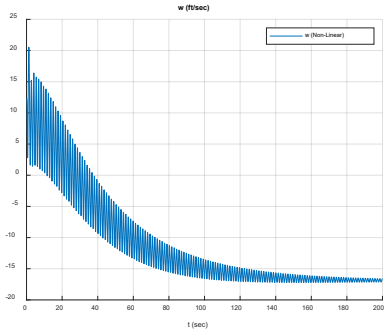


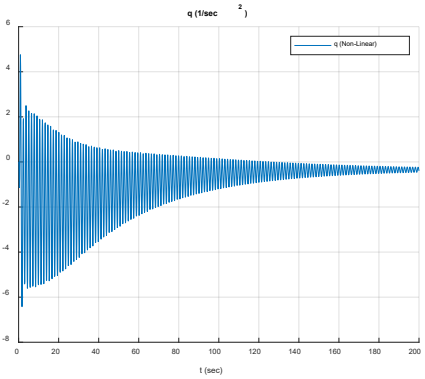


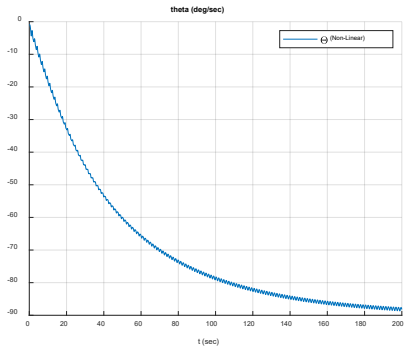


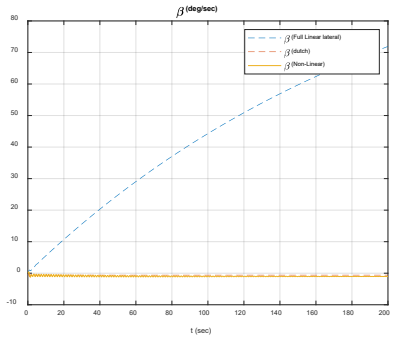
Response due to Rudder ($\delta_a = 25$)

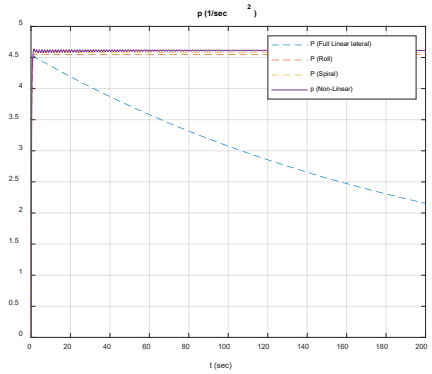


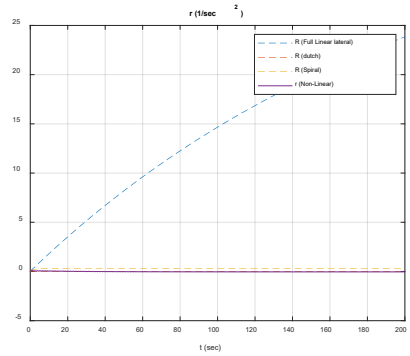


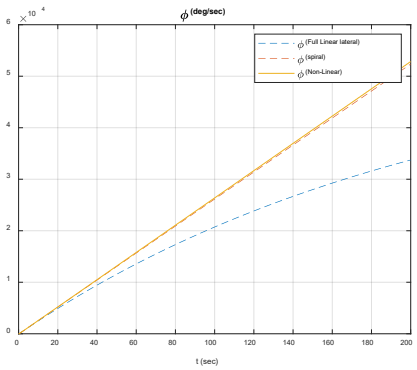


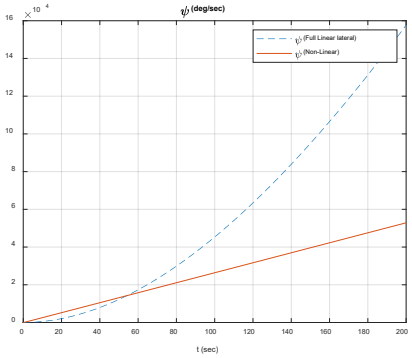




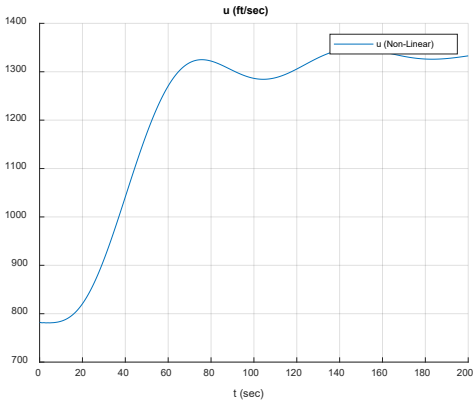


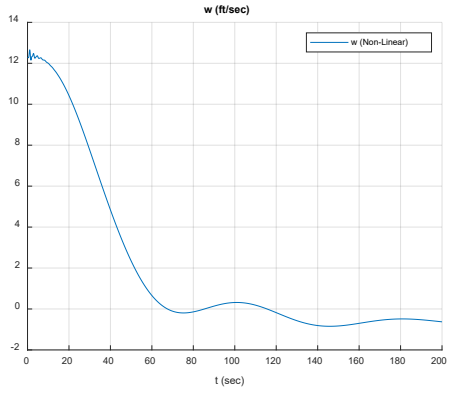


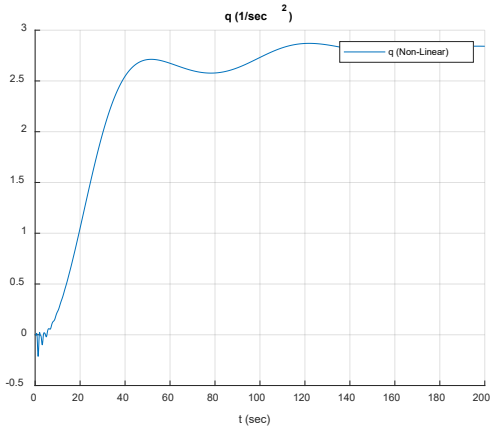


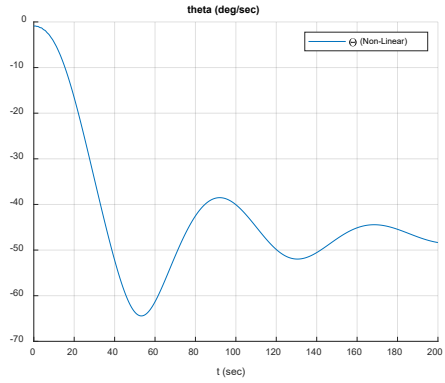


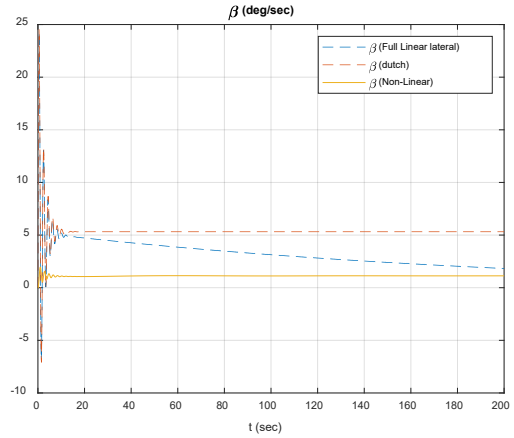
Response due to Rudder ($\delta_r = 1$)

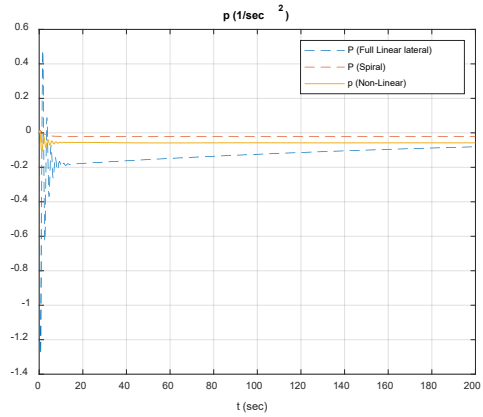


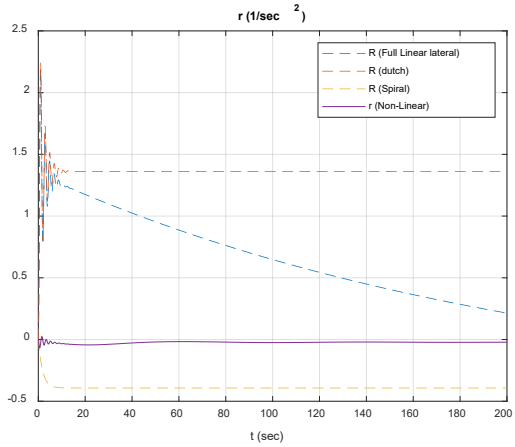


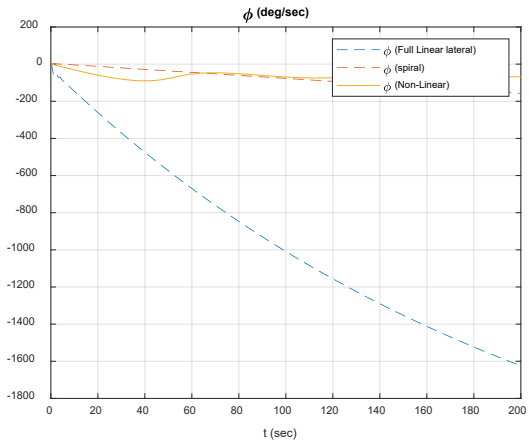

q (1/sec 2)
q (Non-Linear)
t (sec)

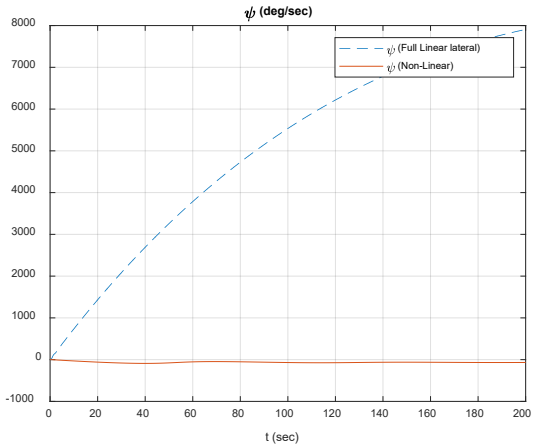

theta (deg/sec)
Θ (Non-Linear)
t (sec)


β (deg/sec)
β (Full Linear lateral)
β (dutch)
β (Non-Linear)
t (sec)

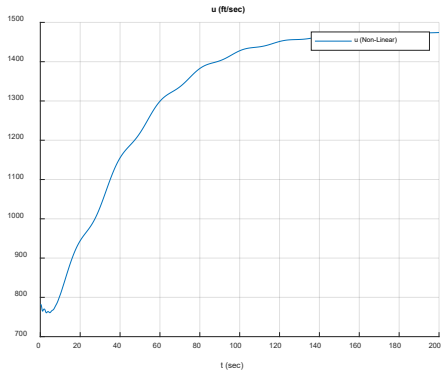

p (1/sec 2)
P (Full Linear lateral)
P (Spiral)
p (Non-Linear)
t (sec)

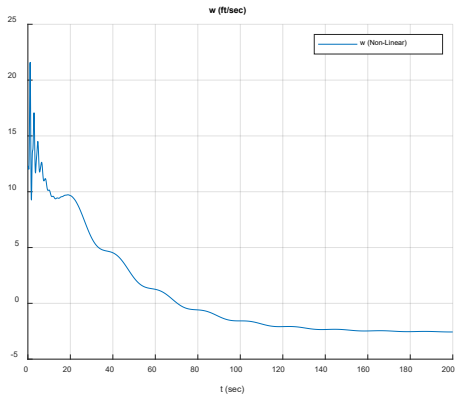

r (1/sec 2)
R (Full Linear lateral)
R (dutch)
R (Spiral)
r (Non-Linear)
t (sec)

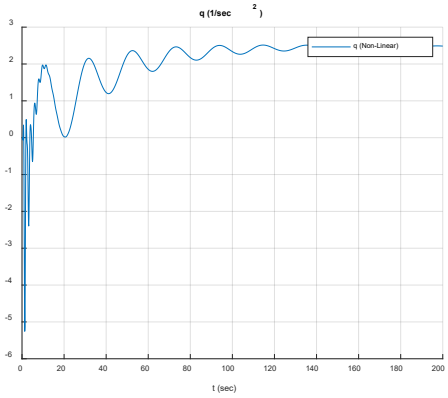

ϕ (deg/sec)
ϕ (Full Linear lateral)
ϕ (spiral)
ϕ (Non-Linear)
t (sec)

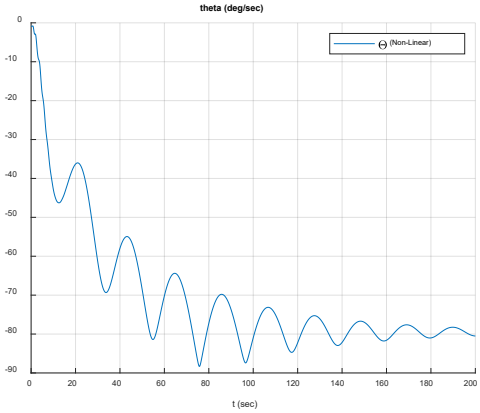

ψ (deg/sec)
ψ (Full Linear lateral)
ψ (Non-Linear)
t (sec)

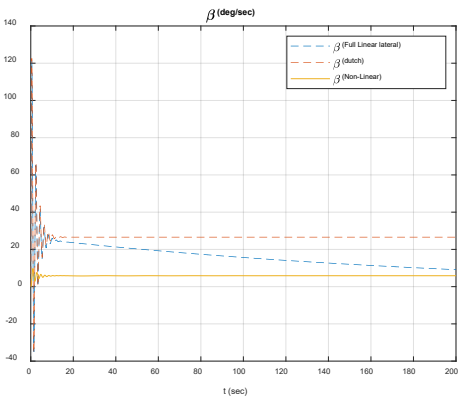
Response due to Rudder ($\delta_r = 5$)

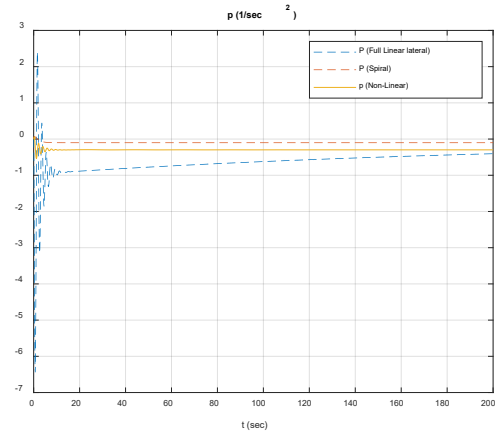


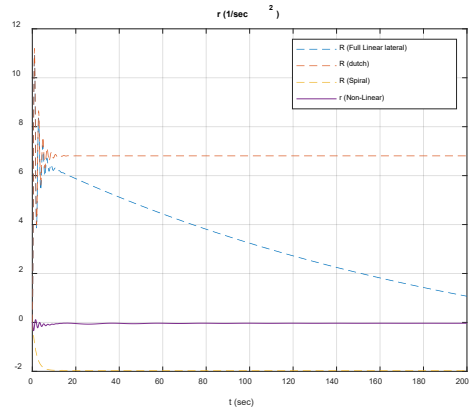


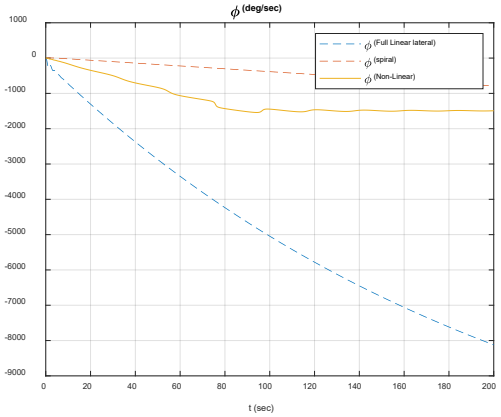


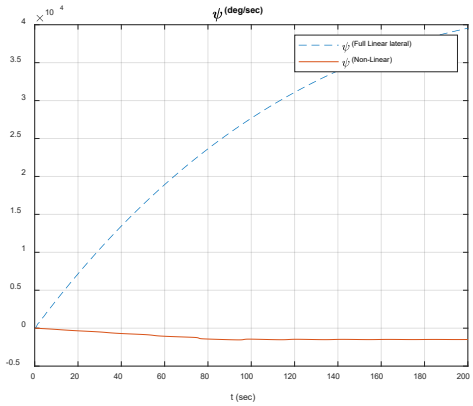




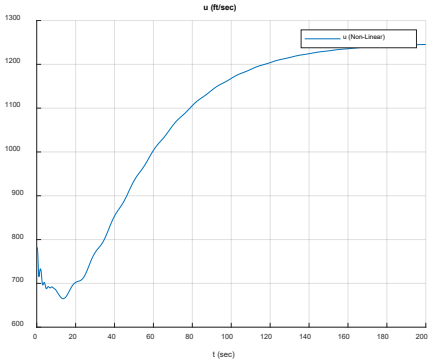


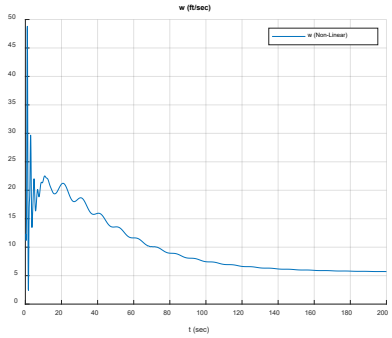


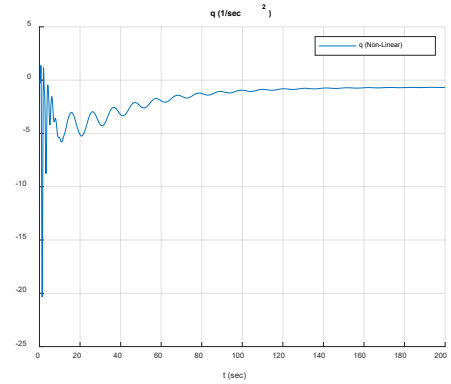


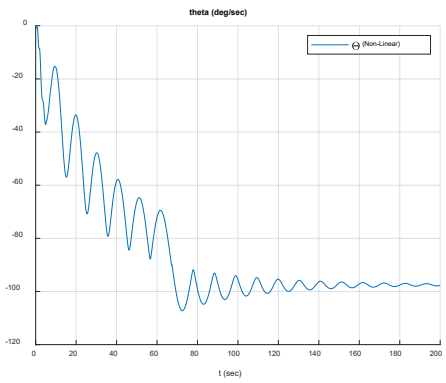


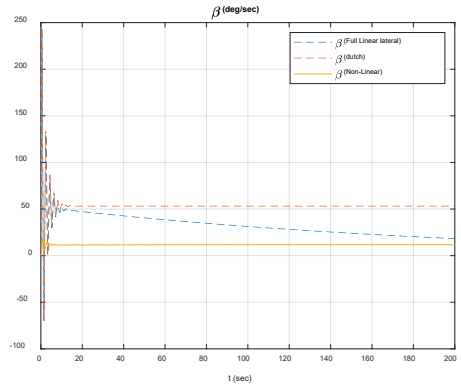
Response due to Rudder ($\delta_r = 10$)

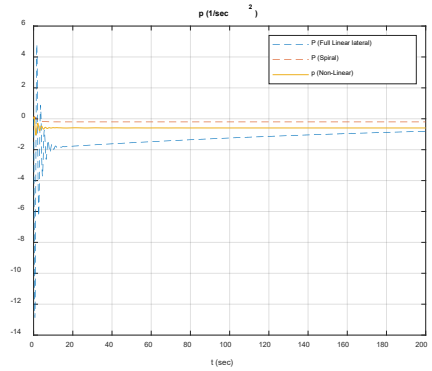


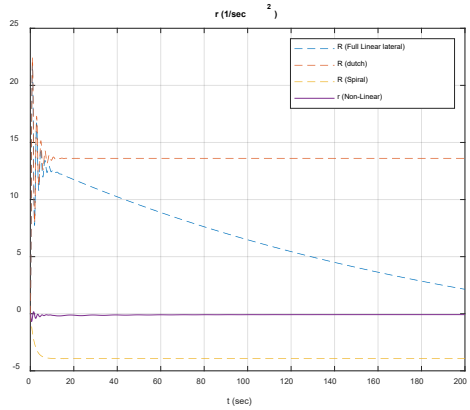


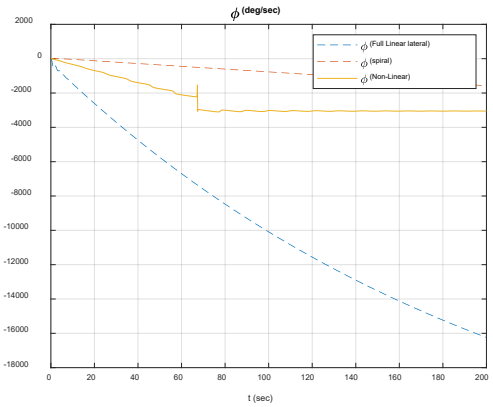


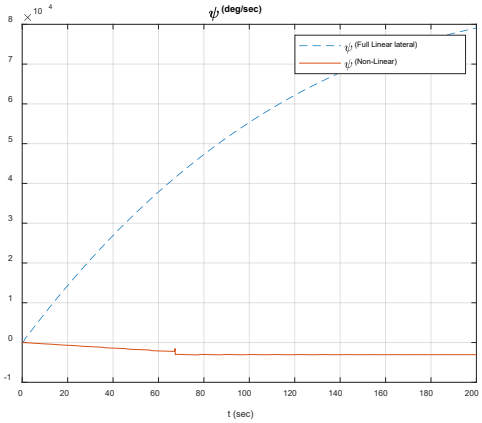




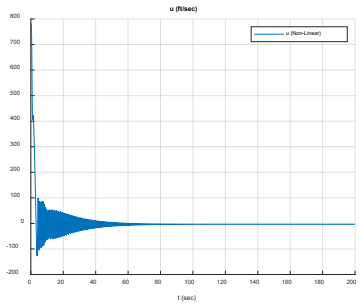


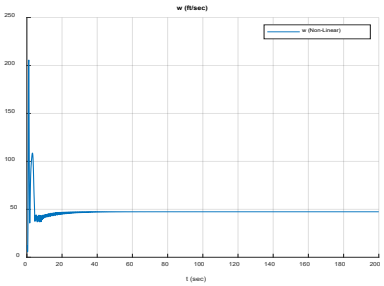


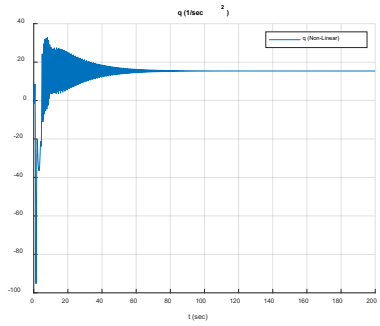


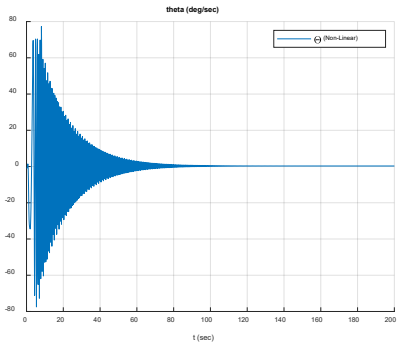


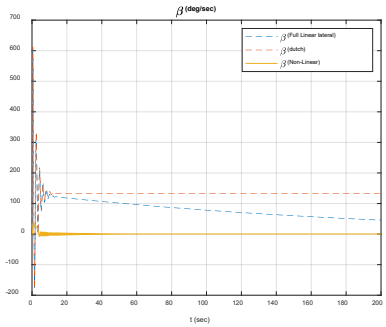
Response due to Rudder ($\delta_r = 25$)

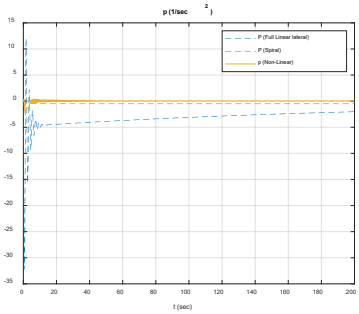


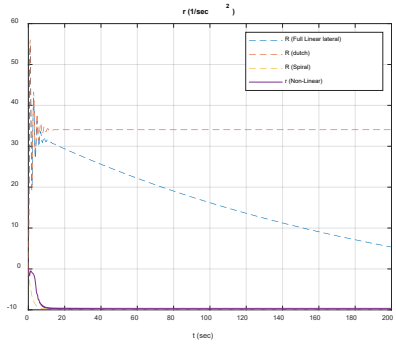


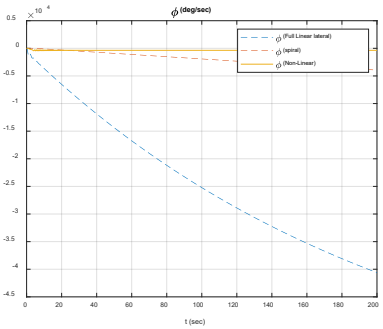


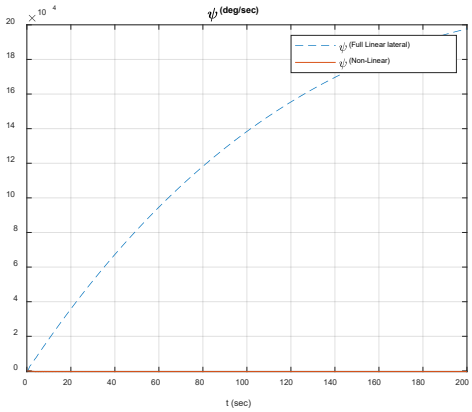



β (deg/sec)
β (Full Linear lateral)
β (dutch)
β (Non-Linear)
t (sec)

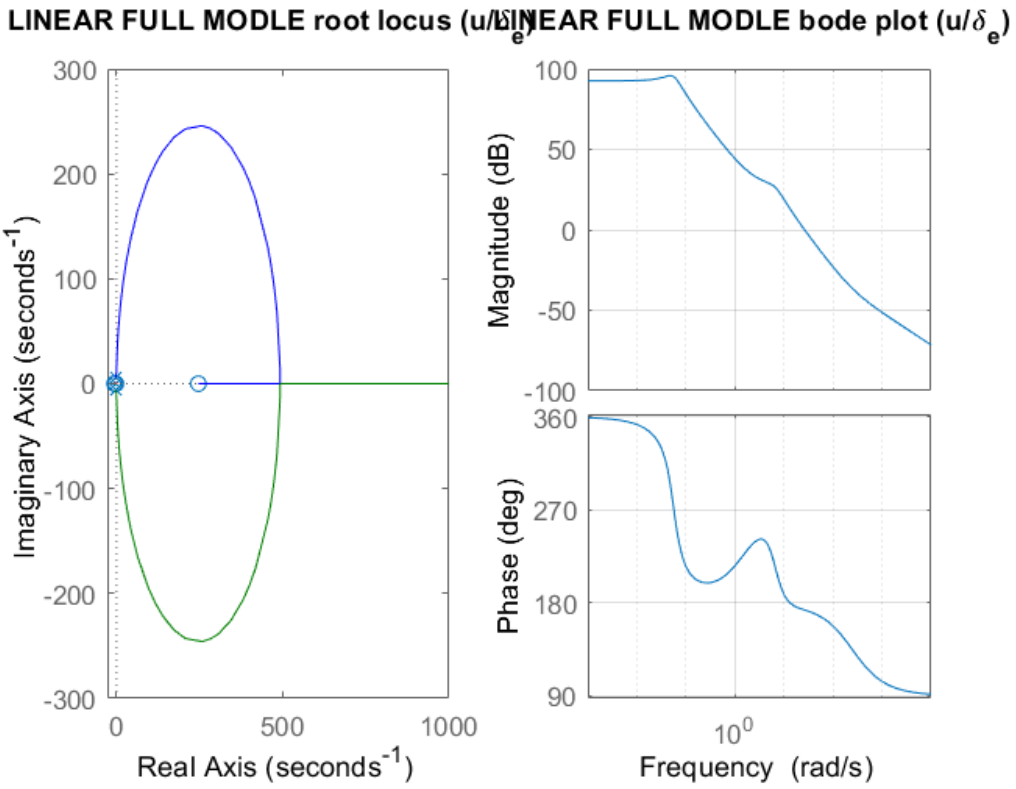

p (1/sec 2)
P (Full Linear lateral)
P (Spiral)
p (Non-Linear)
t (sec)

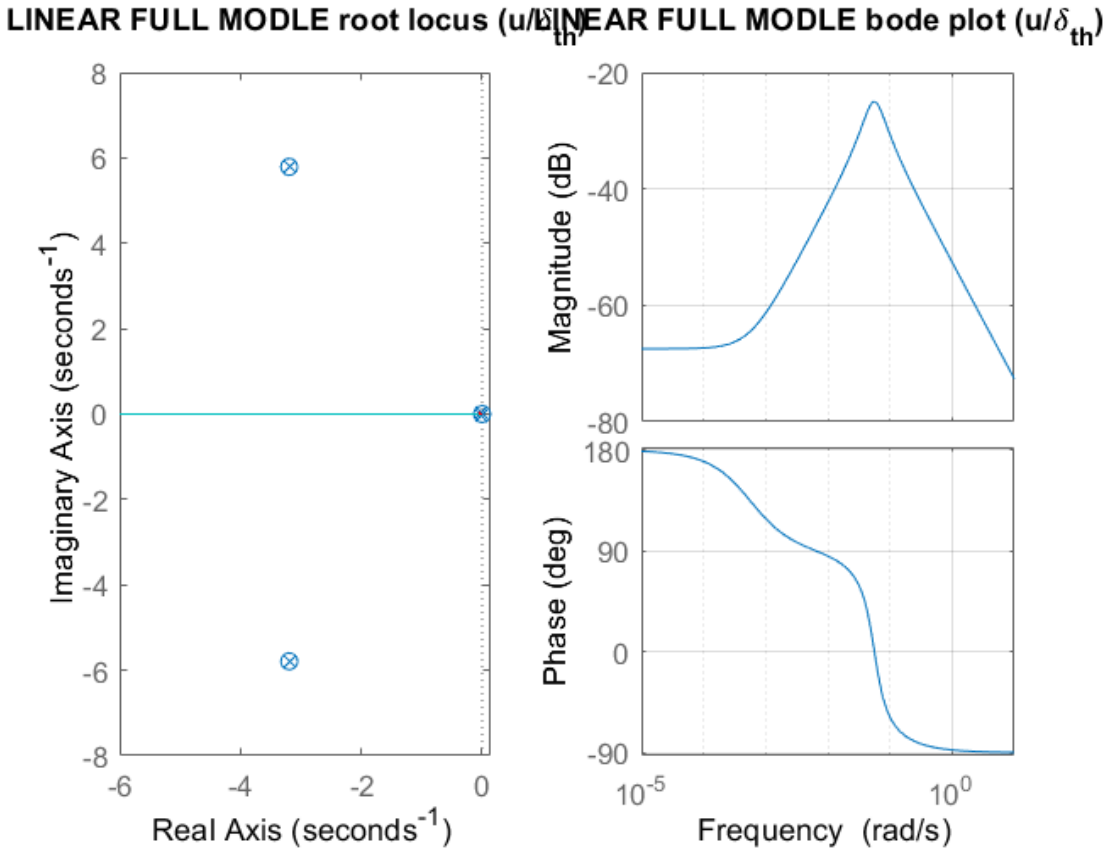

r (1/sec 2)
R (Full Linear lateral)
R (dutch)
R (Spiral)
r (Non-Linear)
t (sec)


ϕ (deg/sec)
ϕ (Full Linear lateral)
ϕ (spiral)
ϕ (Non-Linear)
t (sec)

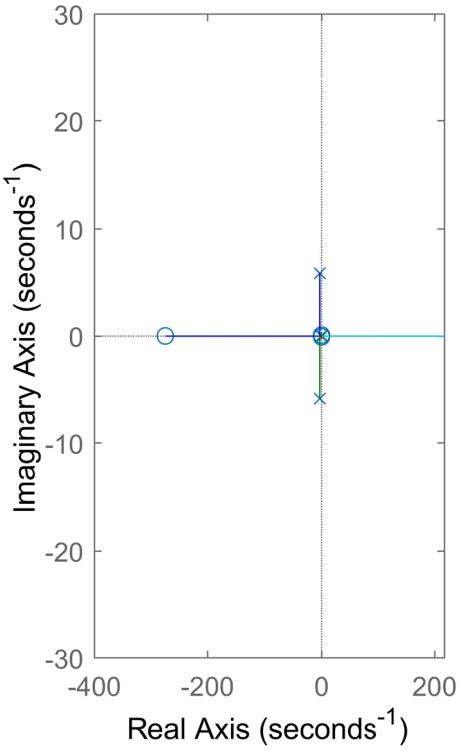

ψ (deg/sec)
ψ (Full Linear lateral)
ψ (Non-Linear)
t (sec)

f) Transfer Function graphs (Longitudinal Dynamics)

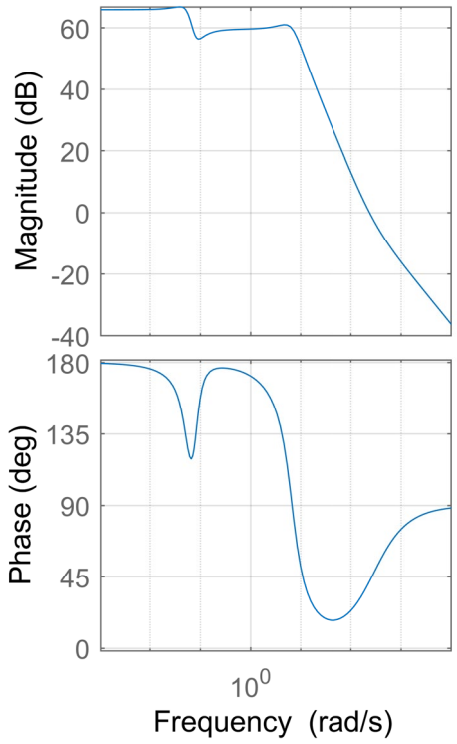




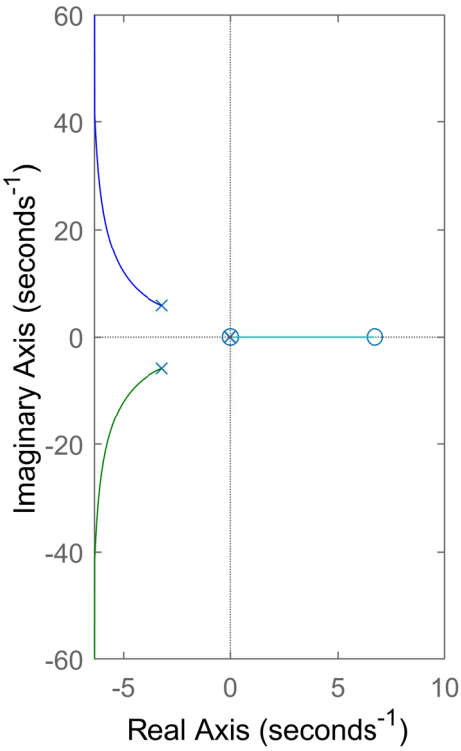
LINEAR FULL MODLE root locus (w/δ_e)



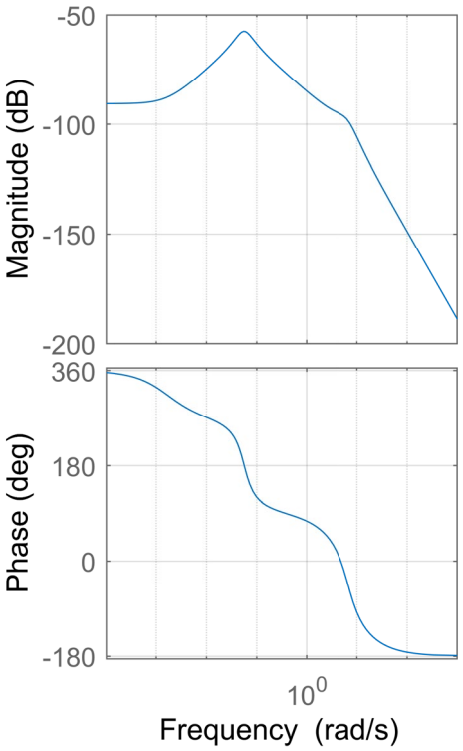
LINEAR FULL MODLE bode plot (w/δ_e)



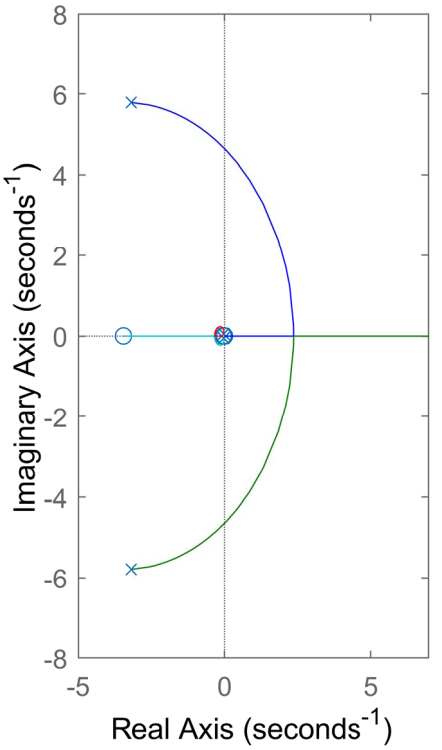
LINEAR FULL MODLE root locus (w/δ_{th})



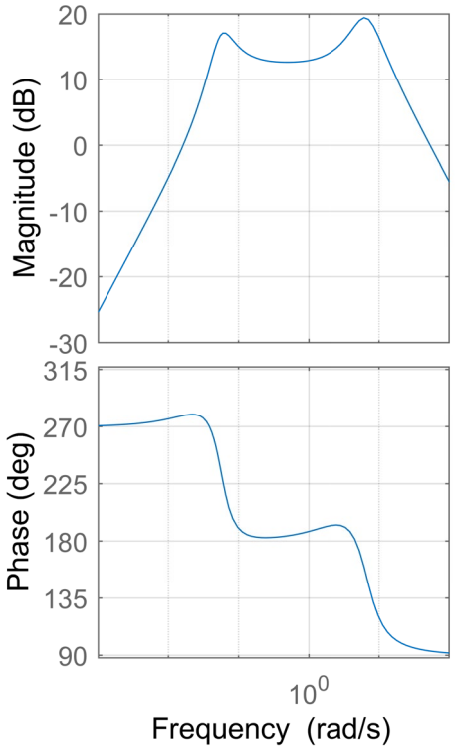
LINEAR FULL MODLE bode plot (w/δ_{th})



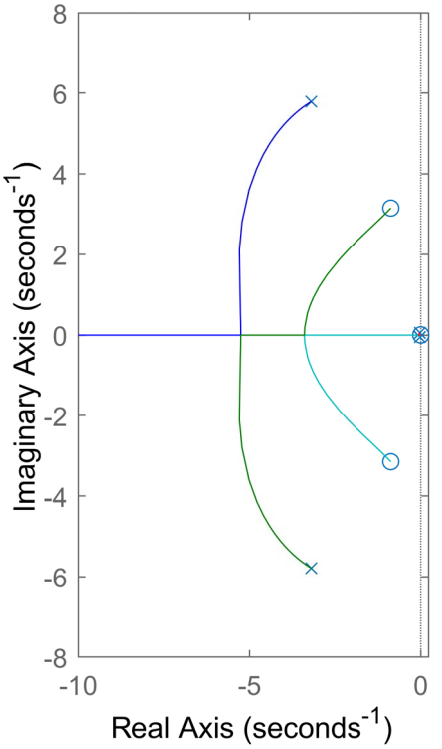
LINEAR FULL MODLE root locus (q/δ_e)



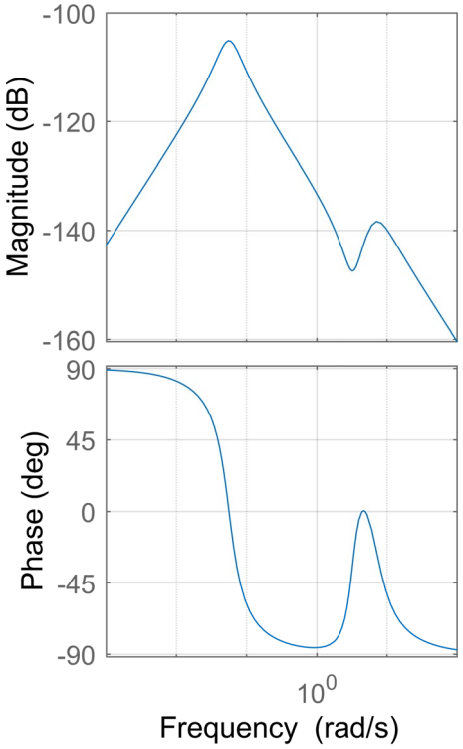
LINEAR FULL MODLE bode plot (q/δ_e)



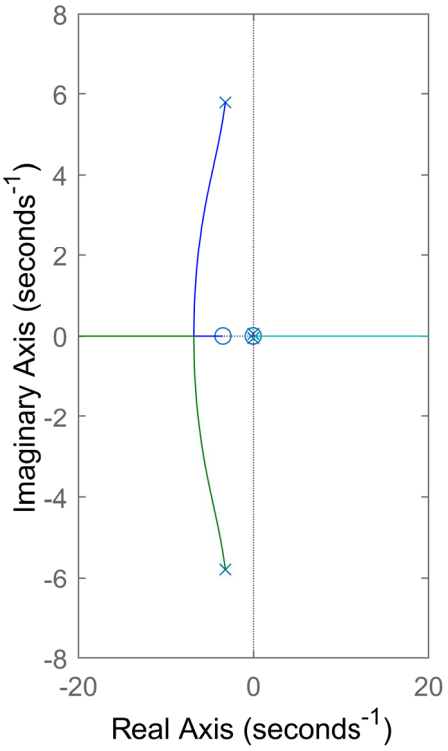
LINEAR FULL MODLE root locus (q/δ_{th})



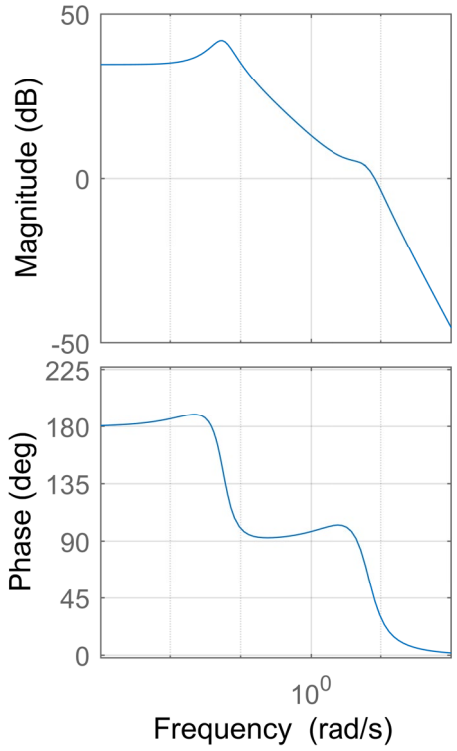
LINEAR FULL MODLE bode plot (q/δ_{th})



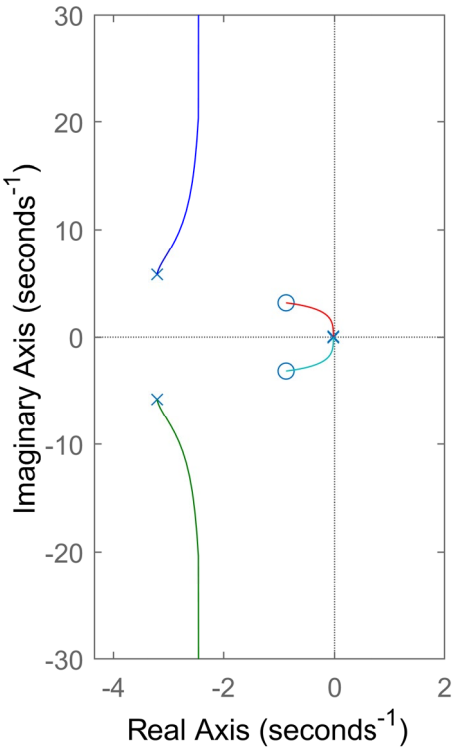
LINEAR FULL MODLE root locus (θ/δ_e)



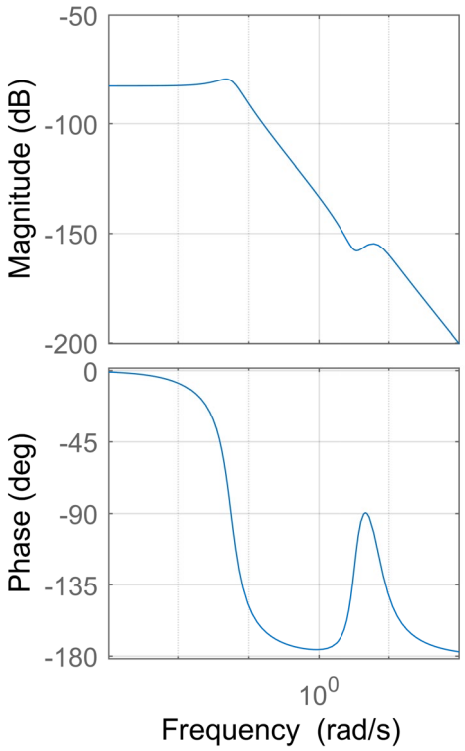
LINEAR FULL MODLE bode plot (θ/δ_e)



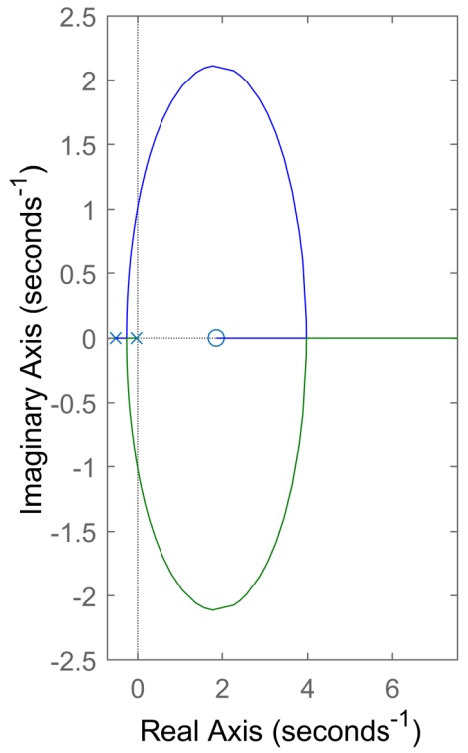
LINEAR FULL MODLE root locus (θ/δ_{th})



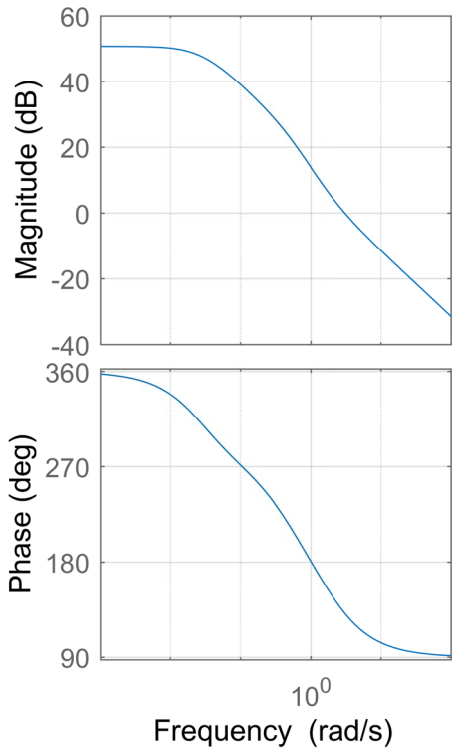
LINEAR FULL MODLE bode plot (θ/δ_{th})



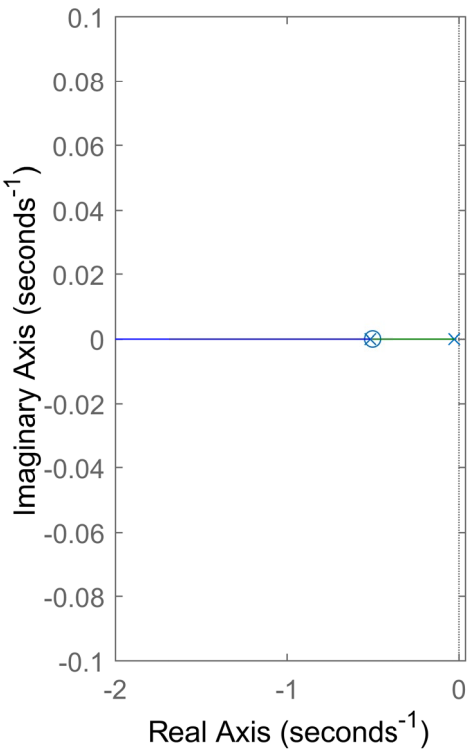
LONG PERIOD MODE root locus (u/δ_e)



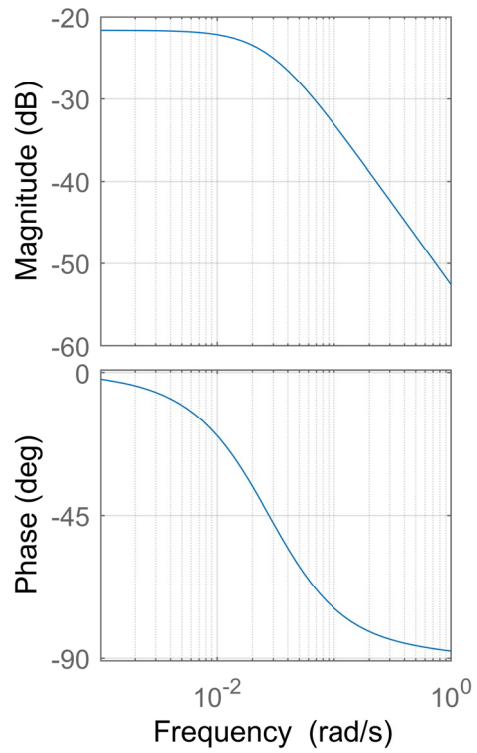
LONG PERIOD MODE bode plot (u/δ_e)



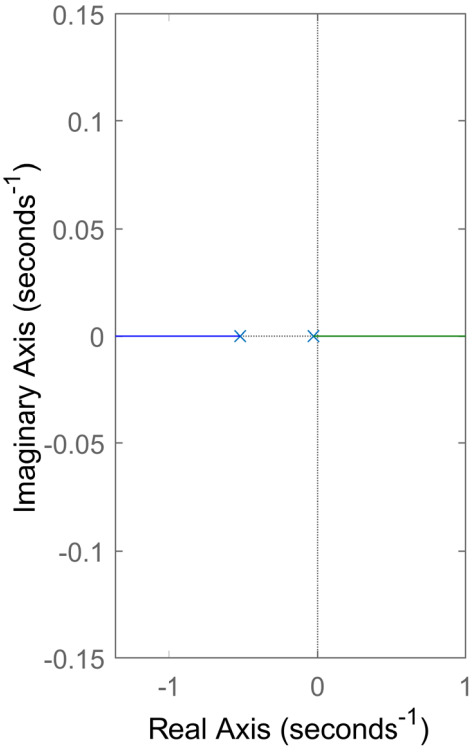
LONG PERIOD MODE root locus (u/δ_{th})



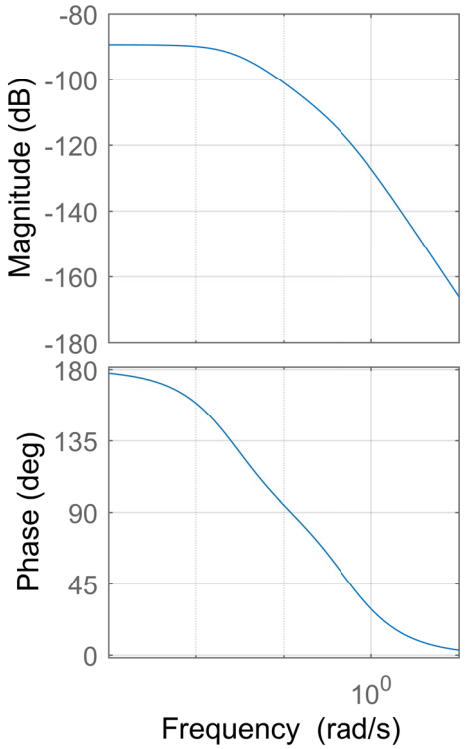
LONG PERIOD MODE bode plot (u/δ_{th})



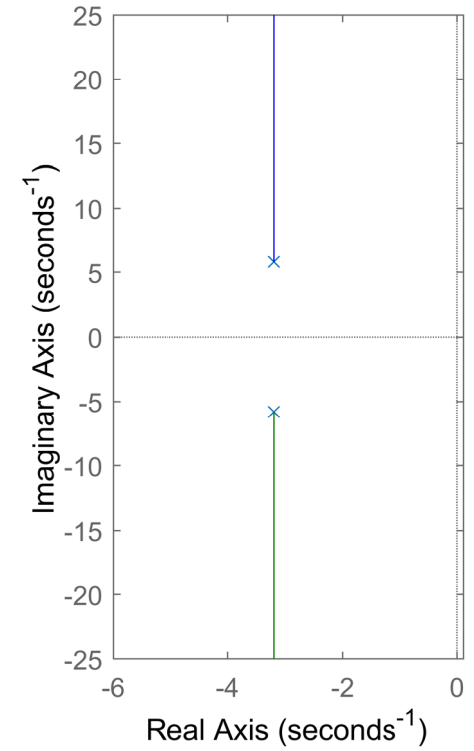
LONG PERIOD MODE root locus (θ/δ_{th})



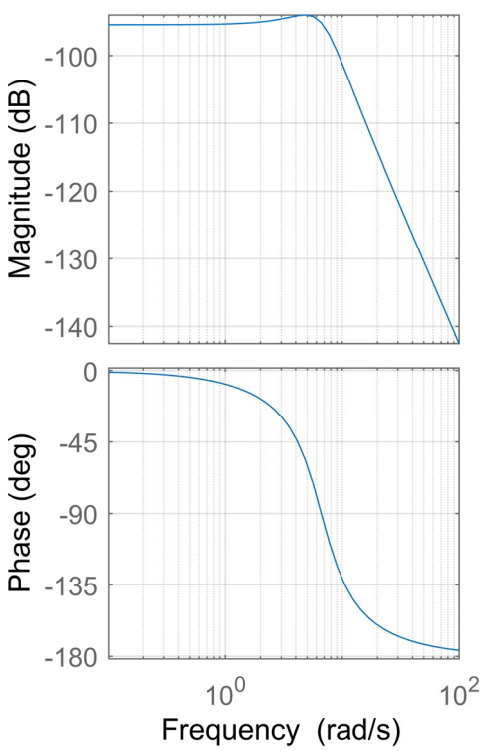
LONG PERIOD MODE bode plot (θ/δ_{th})



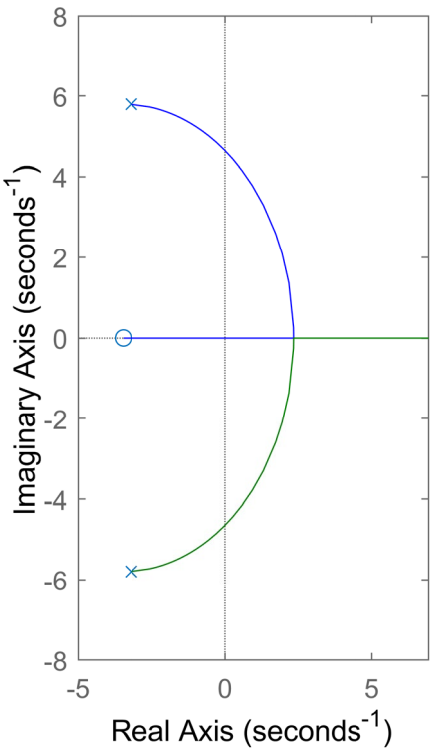
SHORT PERIOD MODE root locus (w/δ_{th})



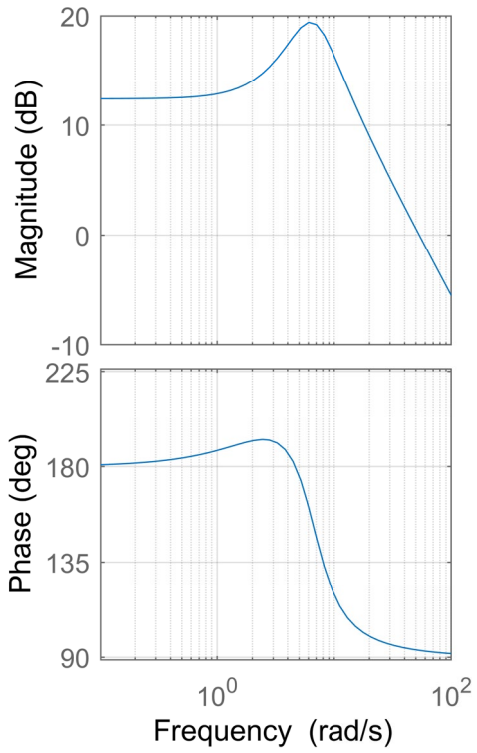
SHORT PERIOD MODE bode plot (w/δ_{th})



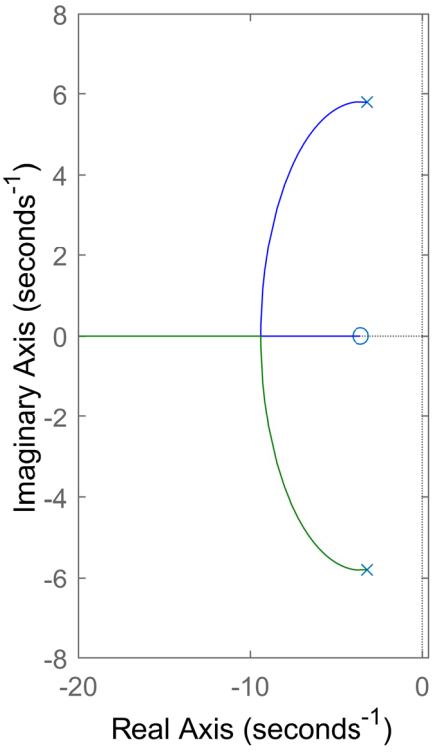
SHORT PERIOD MODE root locus (q/δ_e)



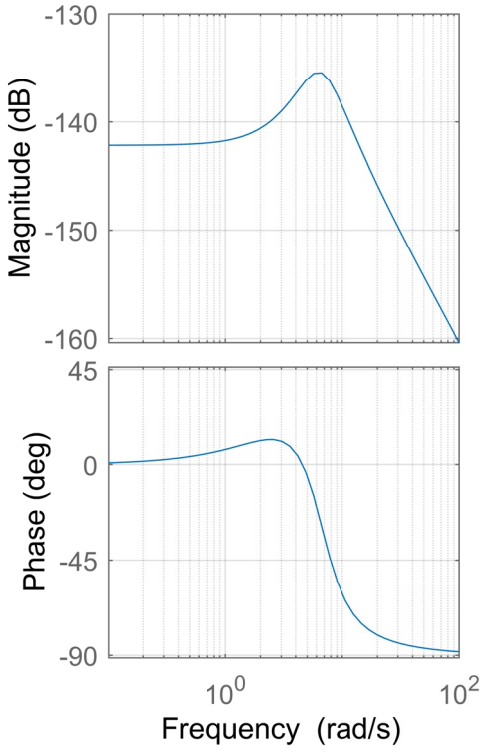
SHORT PERIOD MODE bode plot (q/δ_e)



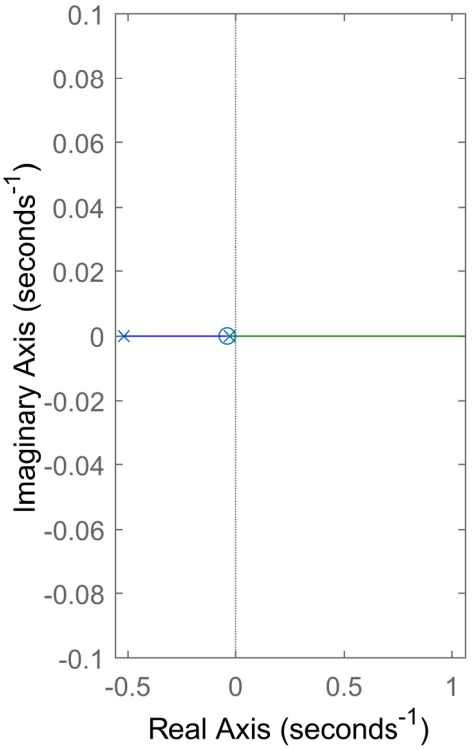
SHORT PERIOD MODE root locus (q/δ_{th})



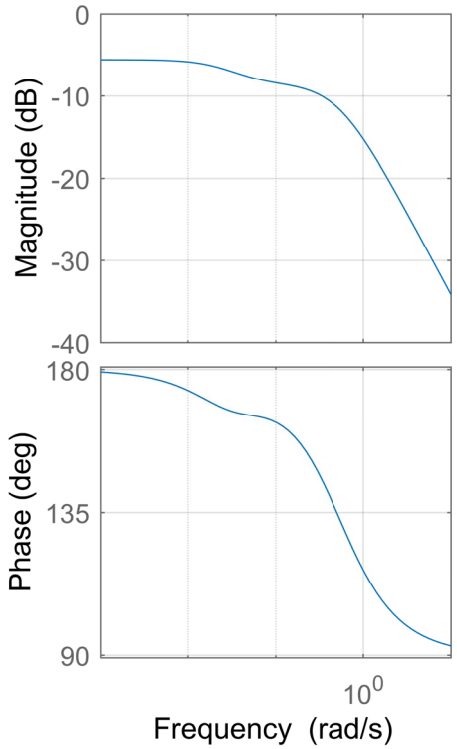
SHORT PERIOD MODE bode plot (q/δ_{th})



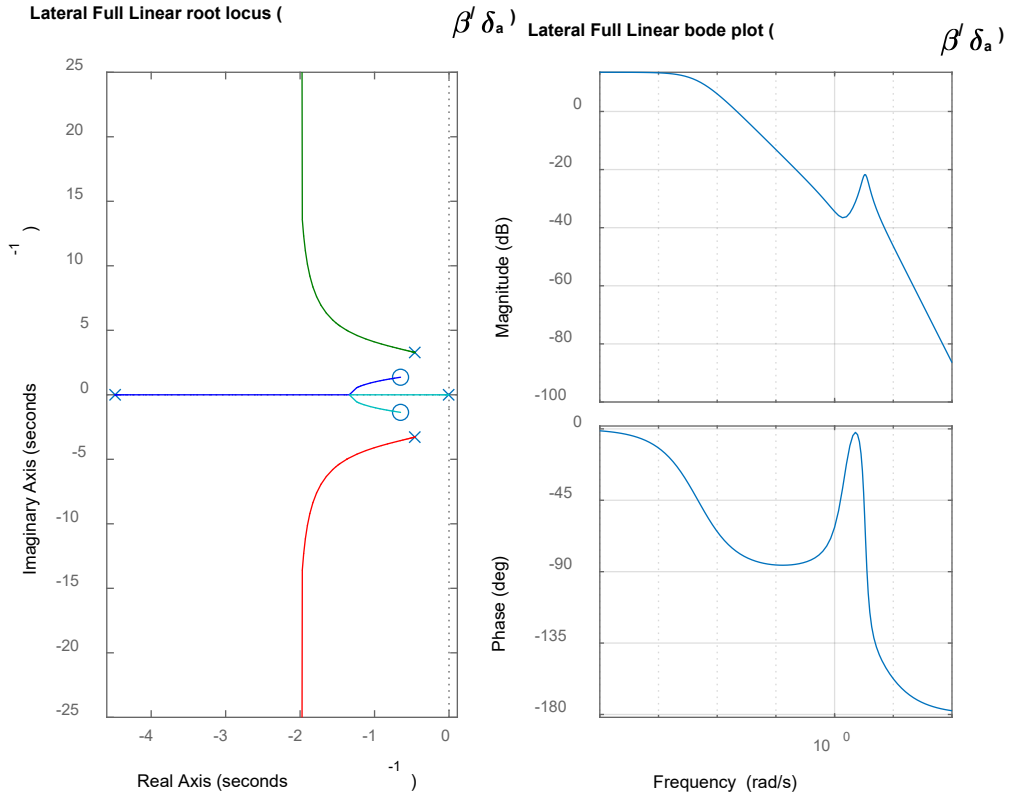
LONG PERIOD MODE root locus (θ/δ_e)

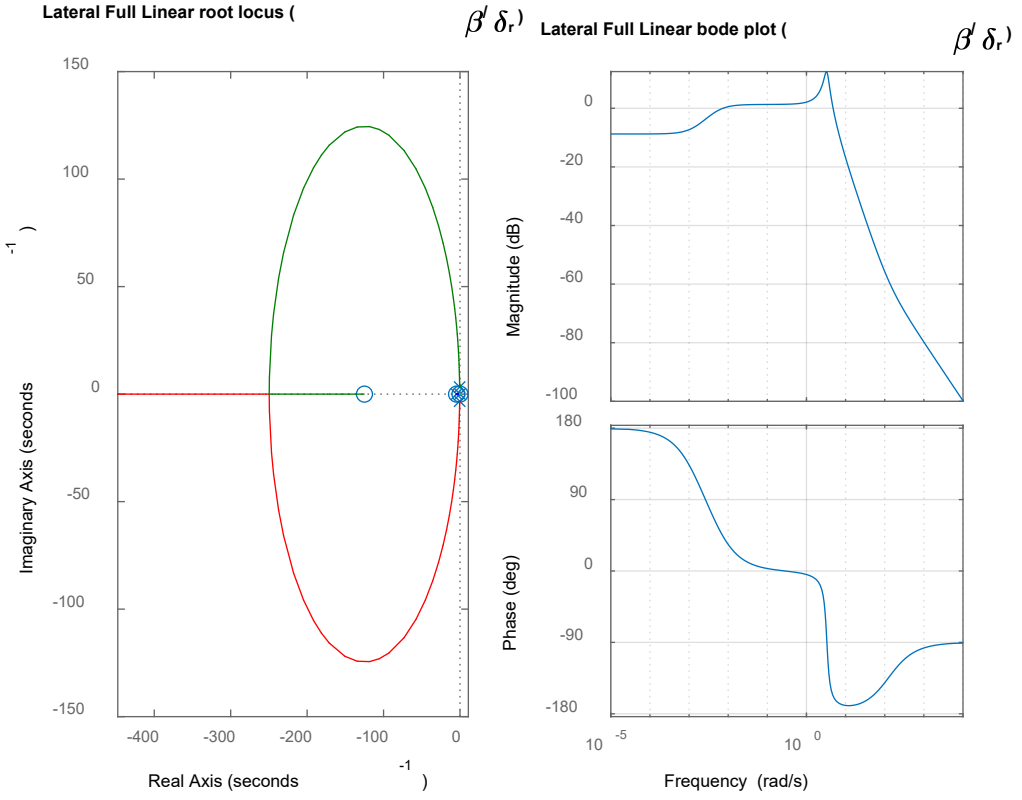


LONG PERIOD MODE bode plot (θ/δ_e)

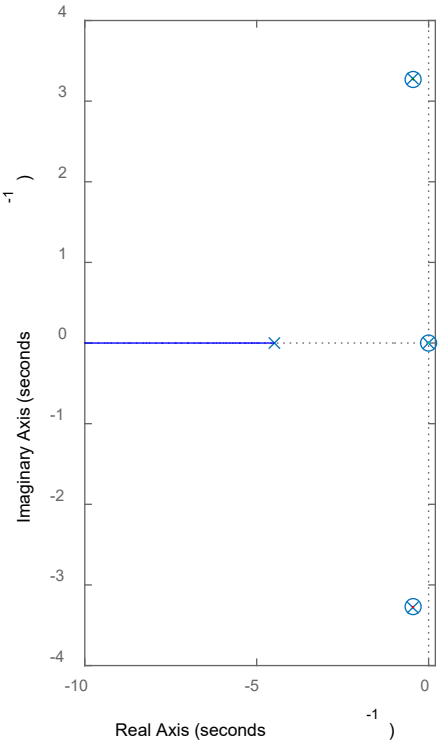


e) Transfer Function graphs (Lateral Dynamics)

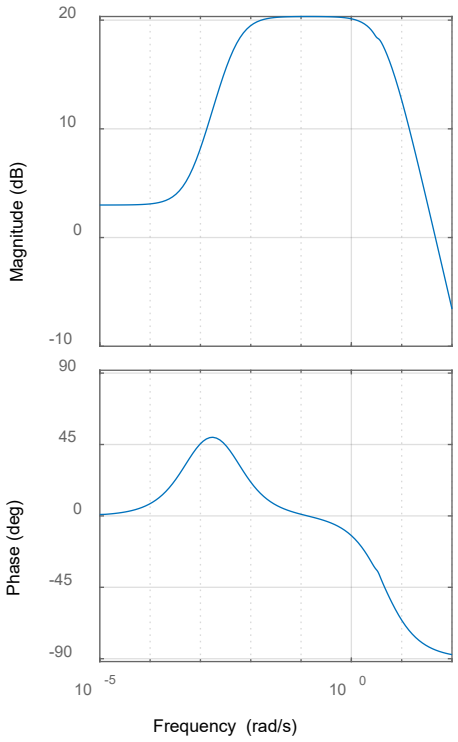




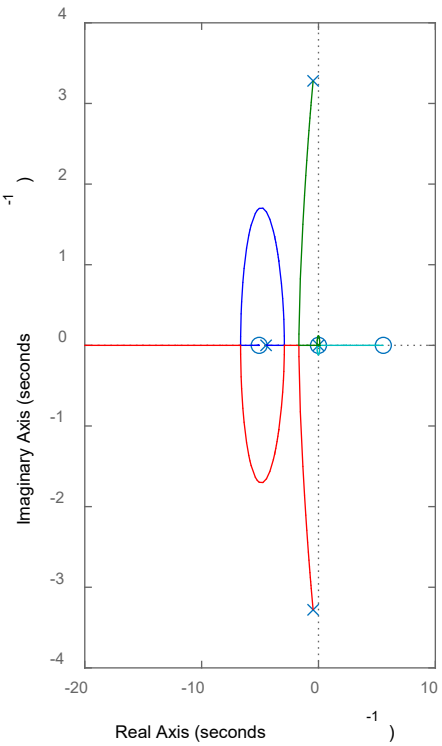
Lateral Full Linear root locus (p/δ_a)



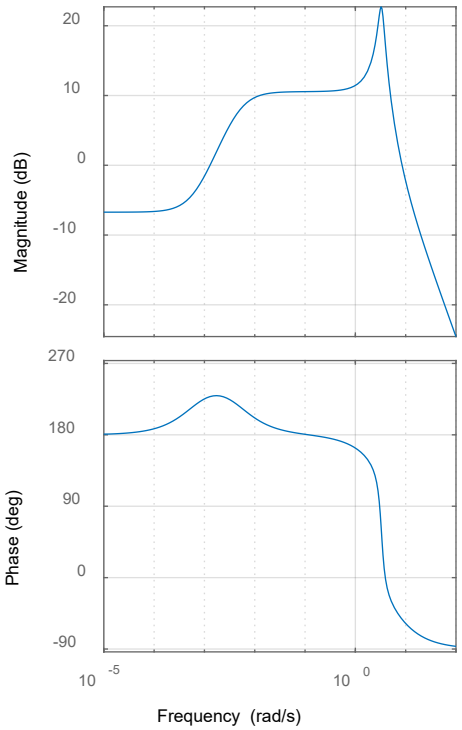
Lateral Full Linear bode plot (p/δ_a)

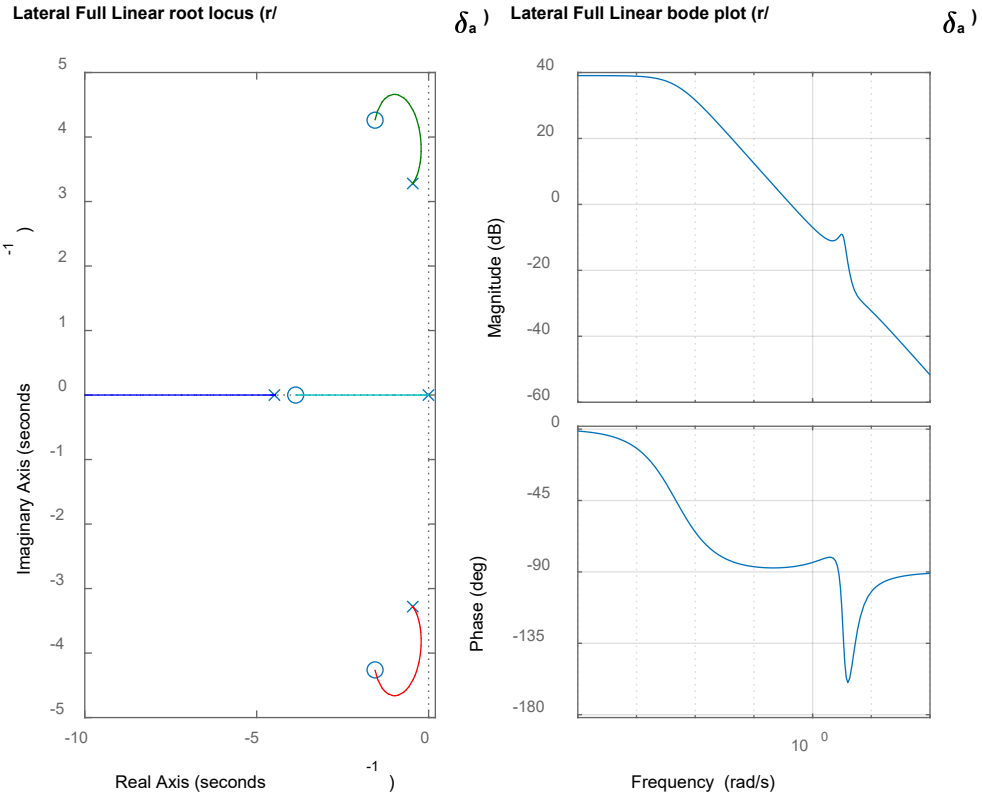


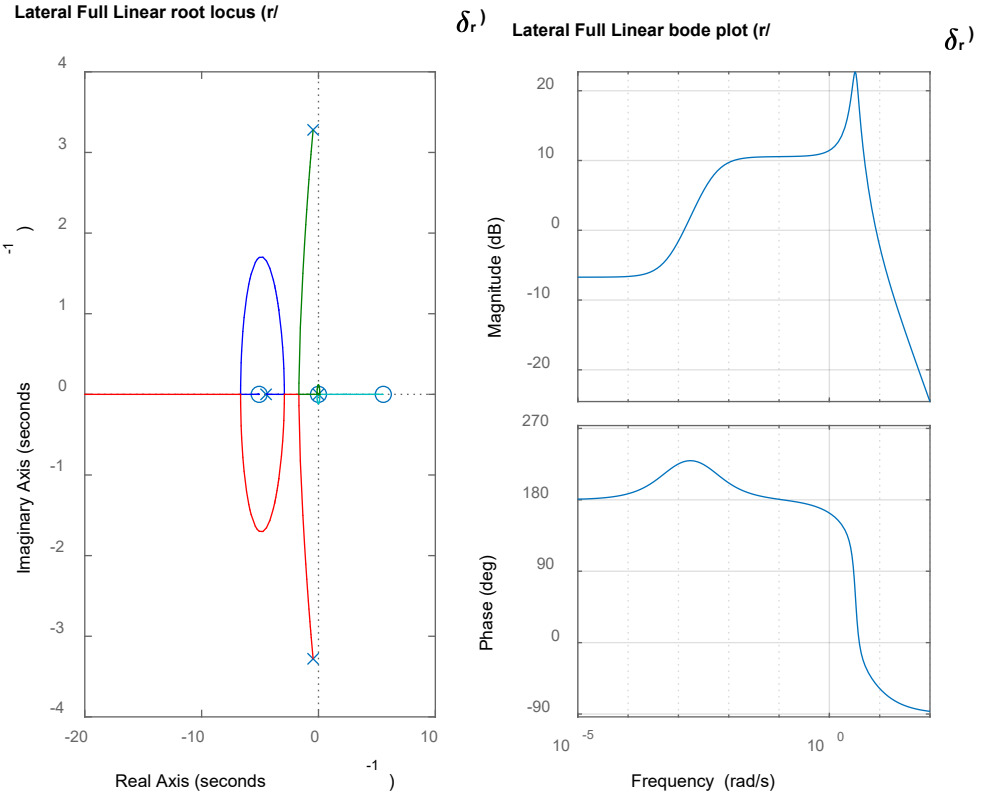
Lateral Full Linear root locus (p/δ_r)



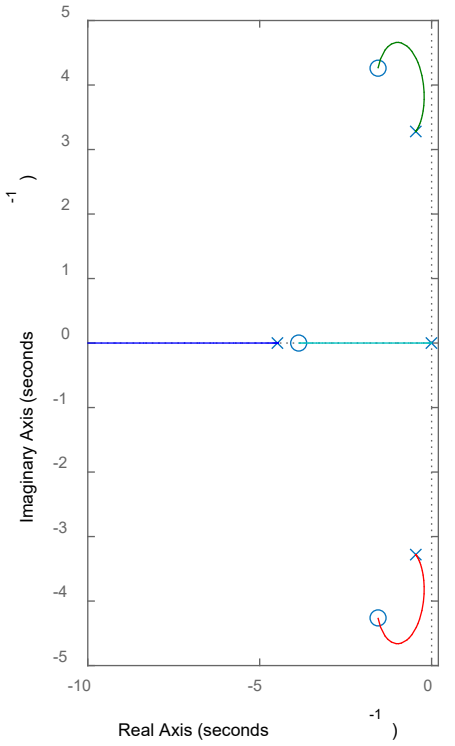
Lateral Full Linear bode plot (p/δ_r)



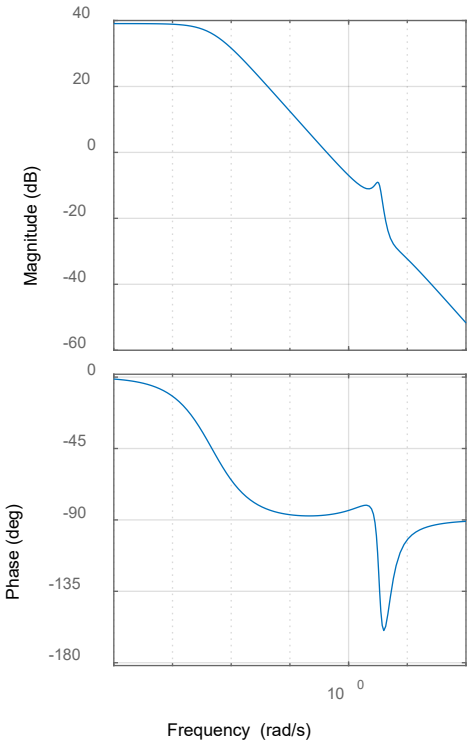

Lateral Full Linear root locus (r/ δa)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
Lateral Full Linear bode plot (r/ δa)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)


Lateral Full Linear root locus (r/ δr)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
Lateral Full Linear bode plot (r/ δr)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)

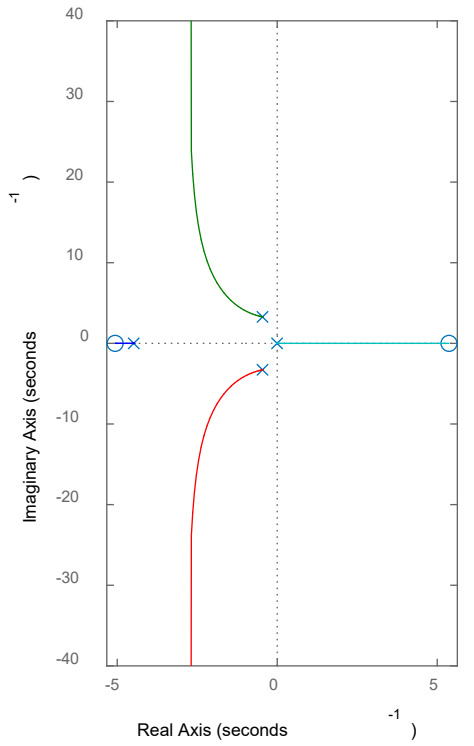
Lateral Full Linear root locus ($\phi' \delta_a$)



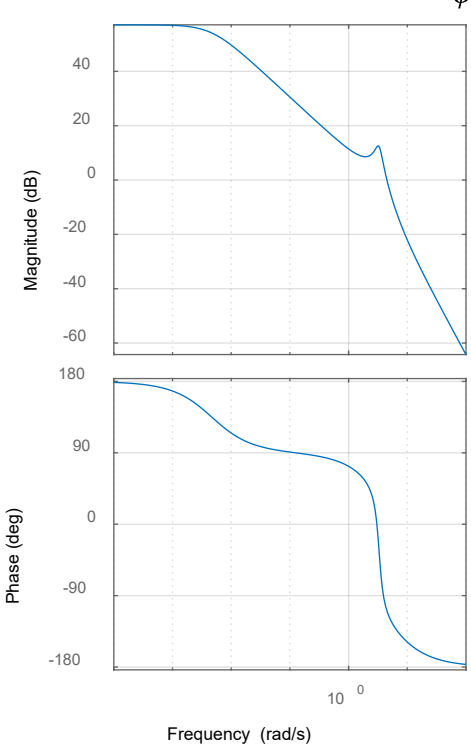
Lateral Full Linear bode plot ($\phi' \delta_a$)



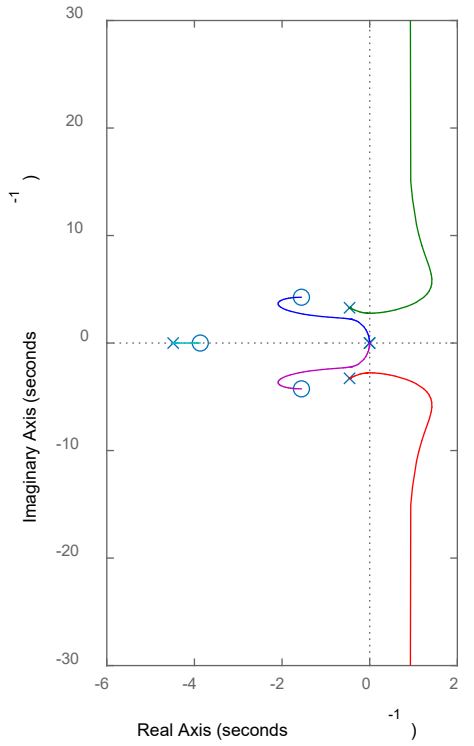
Lateral Full Linear root locus ($\phi' \delta_r$)



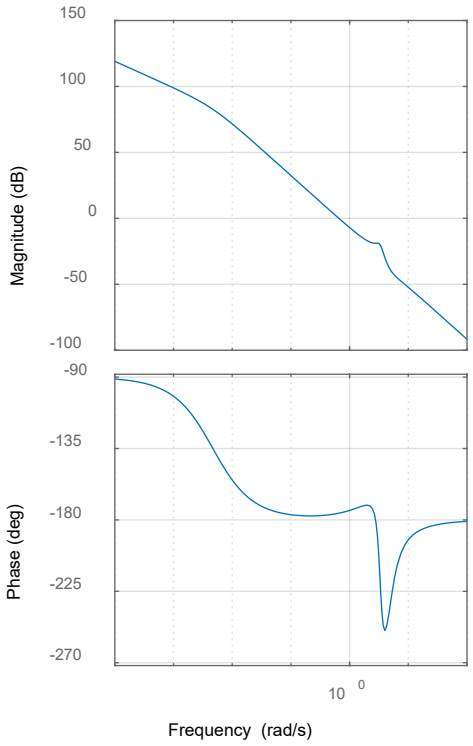
Lateral Full Linear bode plot ($\phi' \delta_r$)



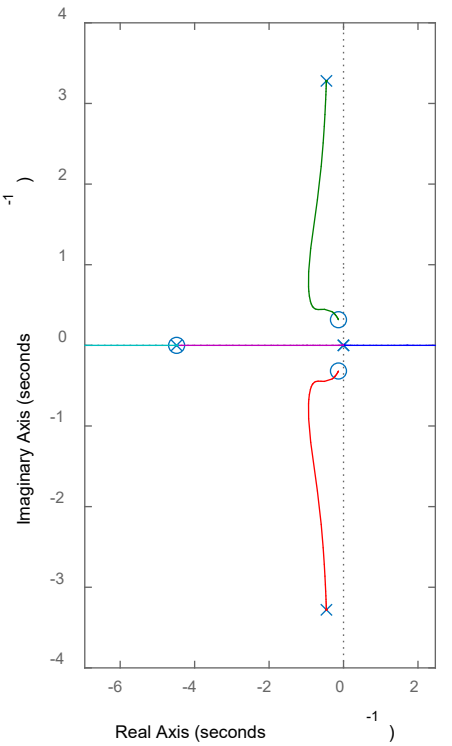
Lateral Full Linear root locus ($\psi' \delta_a$)



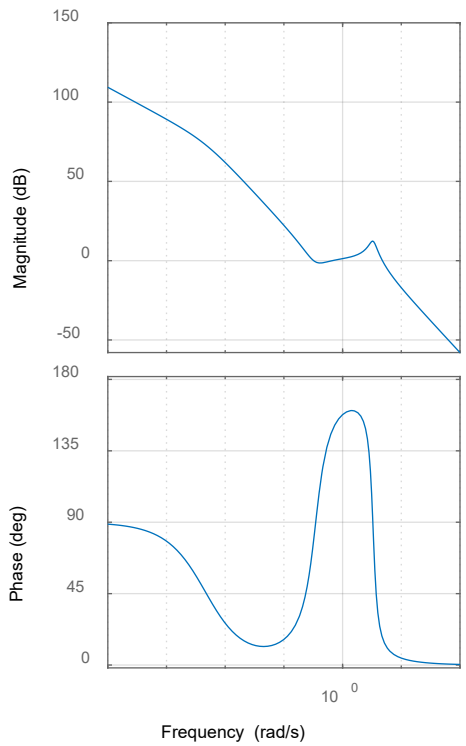
Lateral Full Linear bode plot ($\psi' \delta_a$)

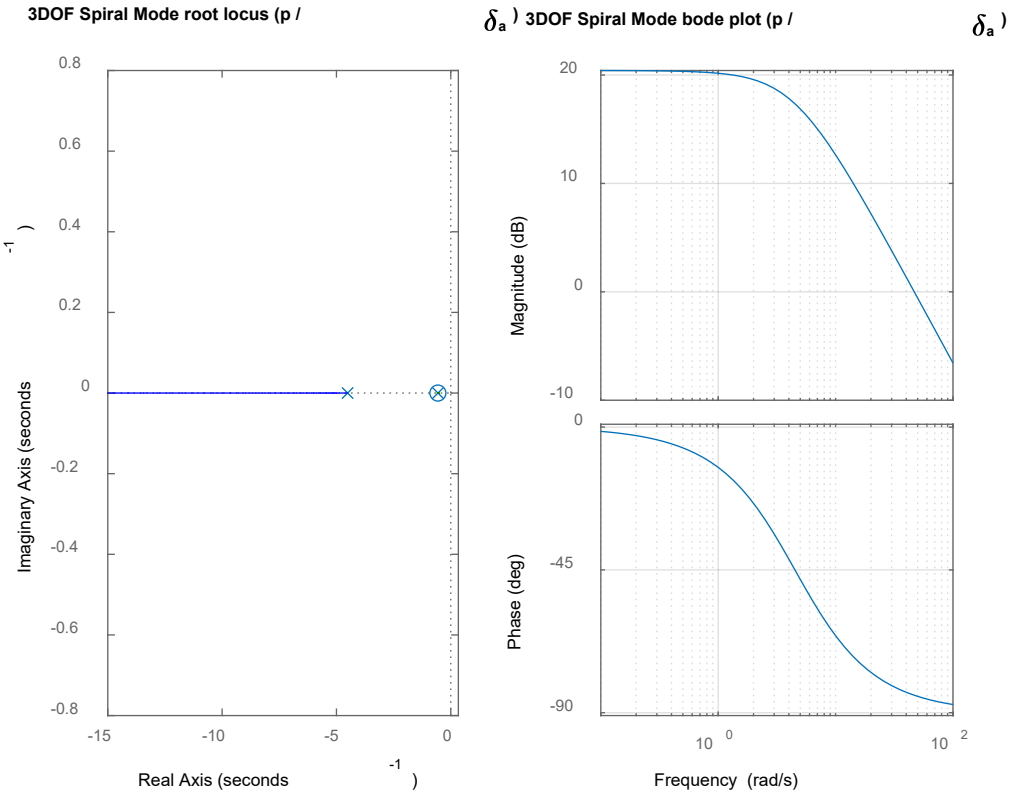


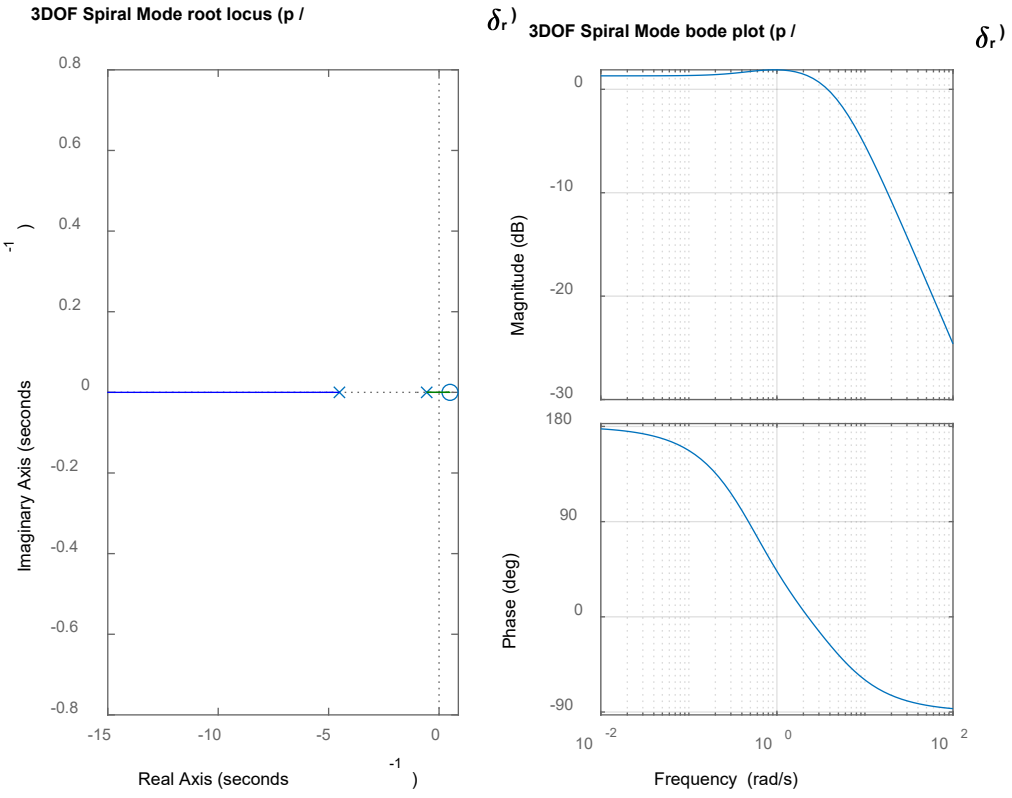
Lateral Full Linear root locus ($\psi' \delta_r$)

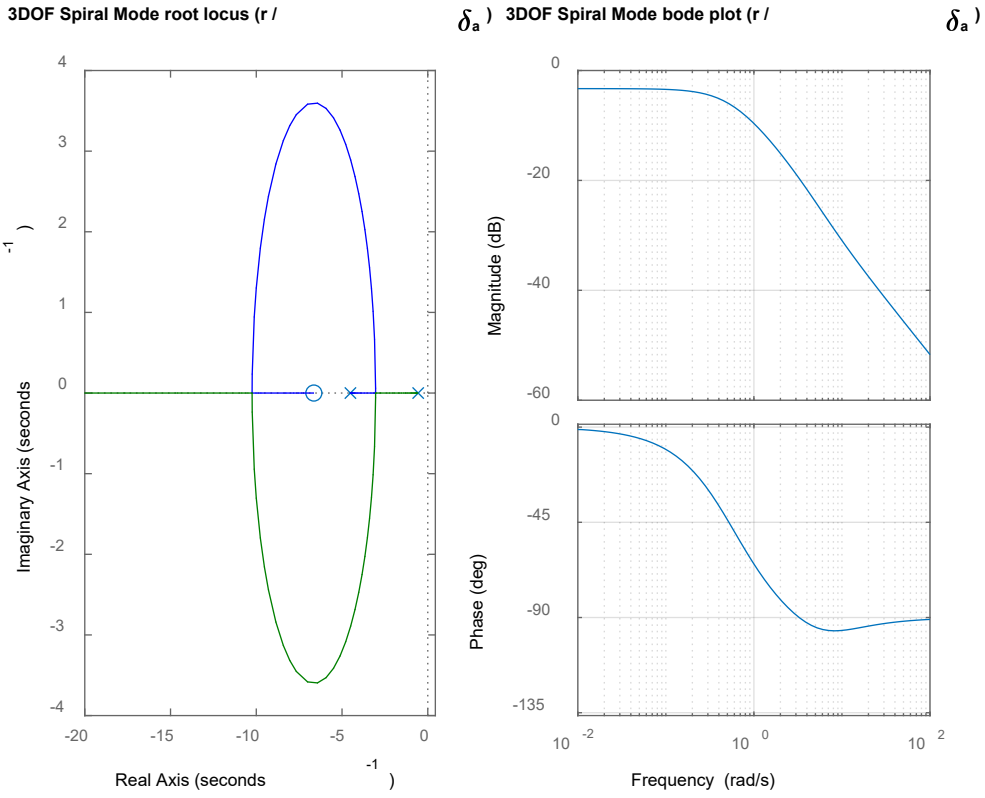


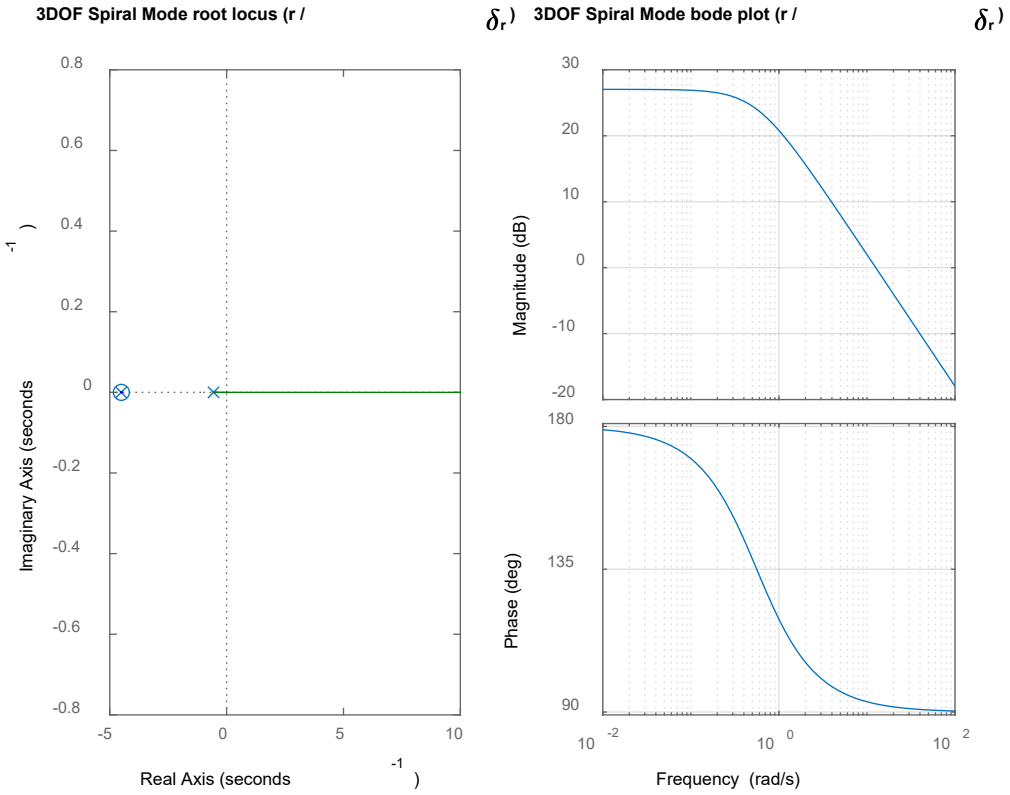
Lateral Full Linear bode plot ($\psi' \delta_r$)



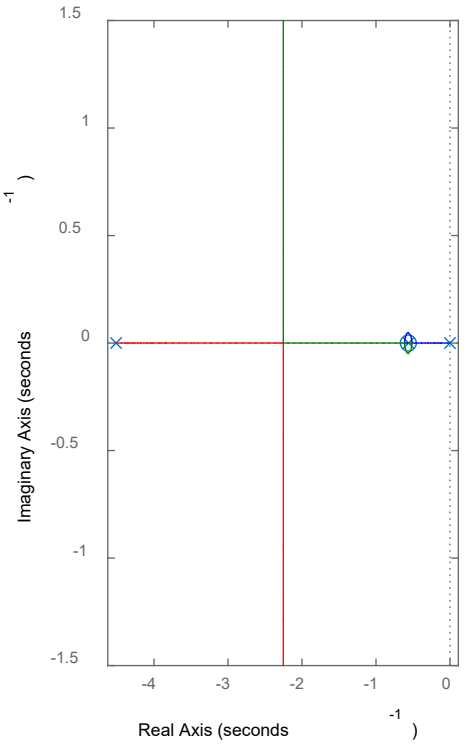

3DOF Spiral Mode root locus (p / δa)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-15
-10
-5
0
3DOF Spiral Mode bode plot (p / δa)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
20
10
0
-10
-45
-90
10 0
10 2


3DOF Spiral Mode root locus (p / δr)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-15
-10
-5
0
3DOF Spiral Mode bode plot (p / δr)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
0
-10
-20
-30
180
90
0
-90
10 -2
10 0
10 2

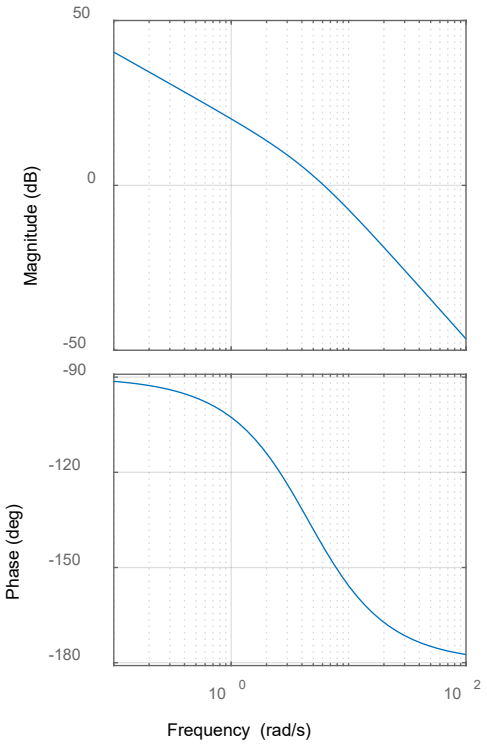

3DOF Spiral Mode root locus (r / δa)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
3DOF Spiral Mode bode plot (r / δa)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)


3DOF Spiral Mode root locus (r / δr)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
3DOF Spiral Mode bode plot (r / δr)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)

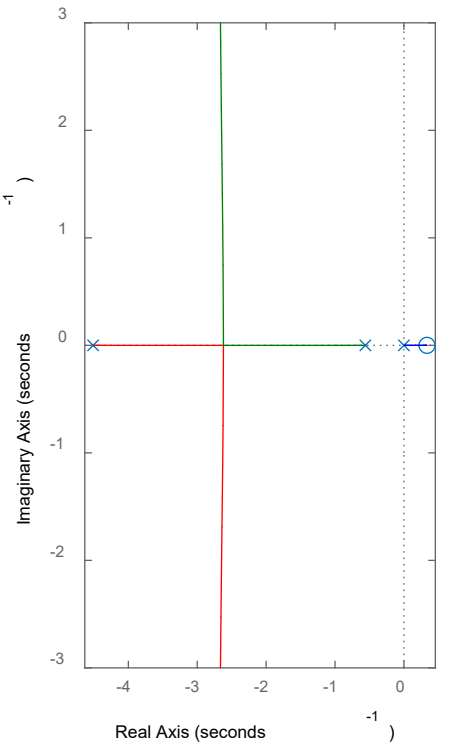
3DOF Spiral Mode root locus (ϕ / δ_a)



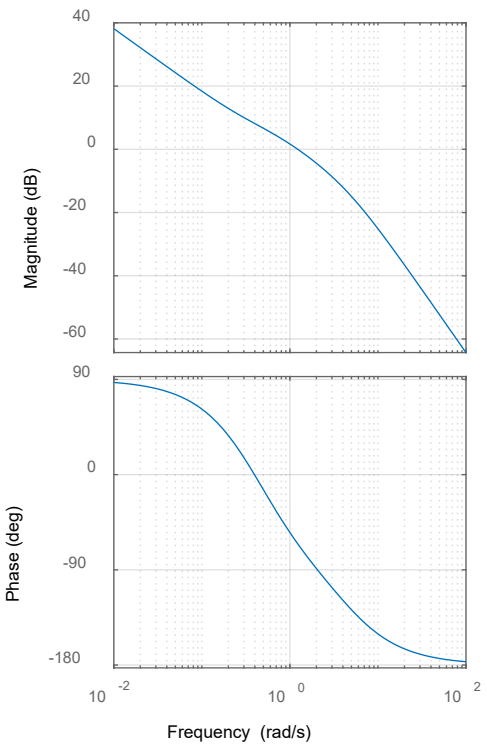
3DOF Spiral Mode bode plot (ϕ / δ_a)

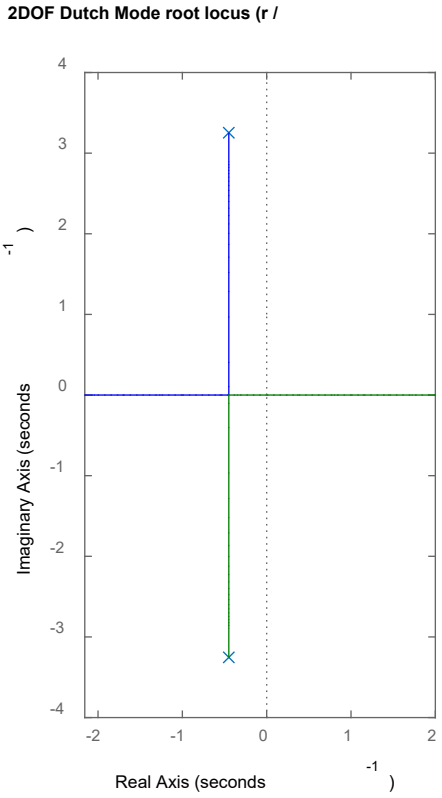


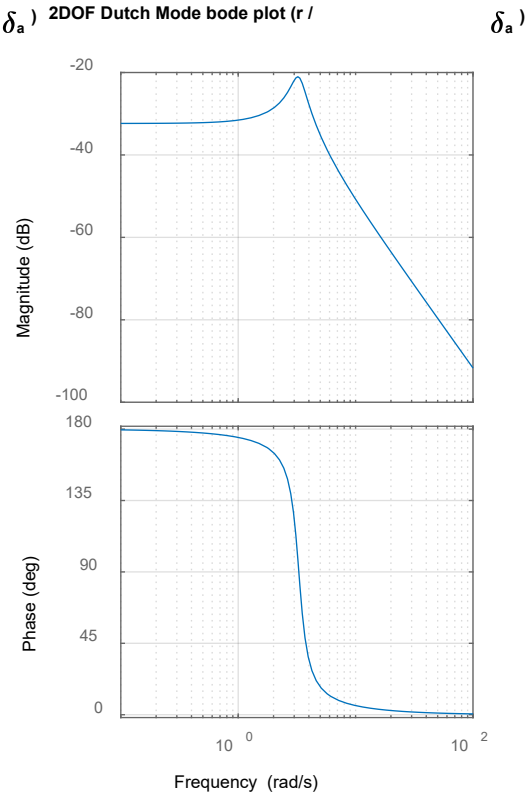
3DOF Spiral Mode root locus (ϕ / δ_r)

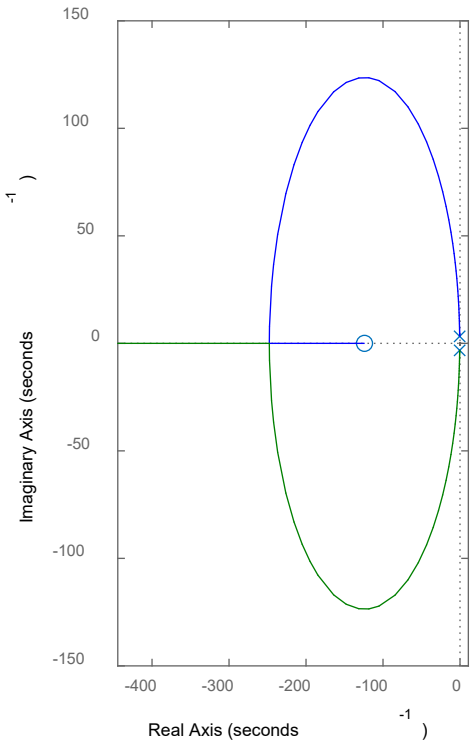


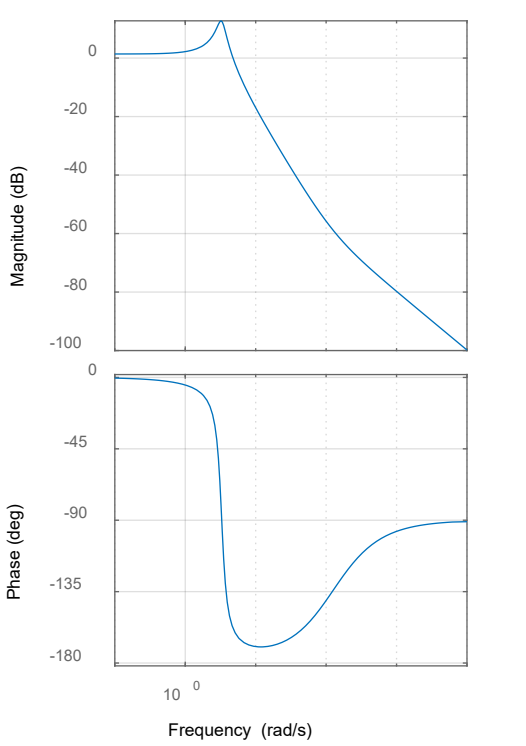
3DOF Spiral Mode bode plot (ϕ / δ_r)



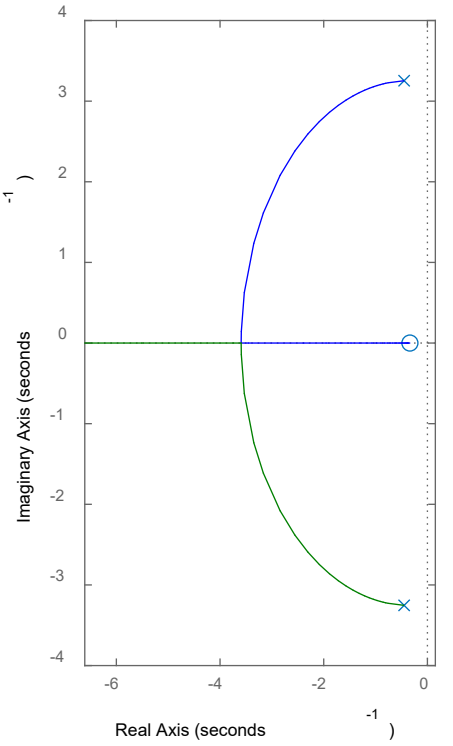

2DOF Dutch Mode root locus (r / δa)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
4
3
2
1
0
-1
-2
-3
-4
-2
-1
0
1
2


2DOF Dutch Mode bode plot (r / δa)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
-20
-40
-60
-80
-100
180
135
90
45
0
10 0
10 2

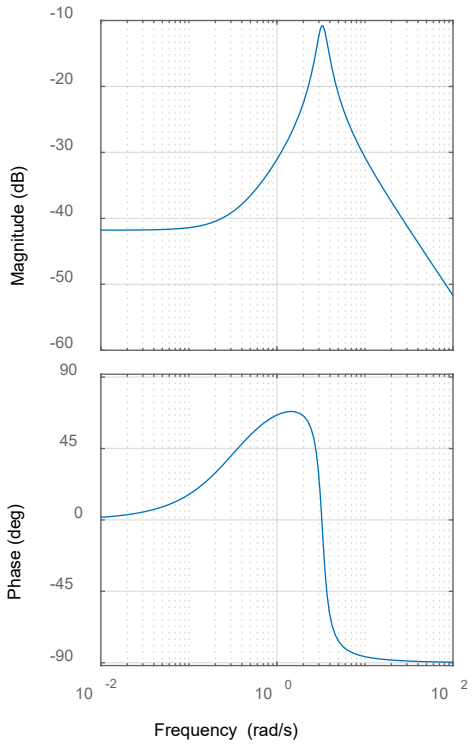

2DOF Dutch Mode root locus (r / δr)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
150
100
50
0
-50
-100
-150
-400
-300
-200
-100
0


2DOF Dutch Mode bode plot (r / δr)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
0
-20
-40
-60
-80
-100
0
-45
-90
-135
-180
10 0

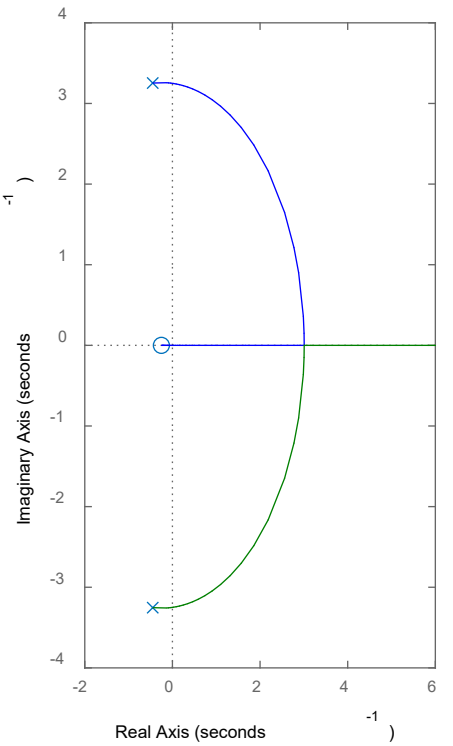
2DOF Dutch Mode root locus ($\beta' \delta_a$)



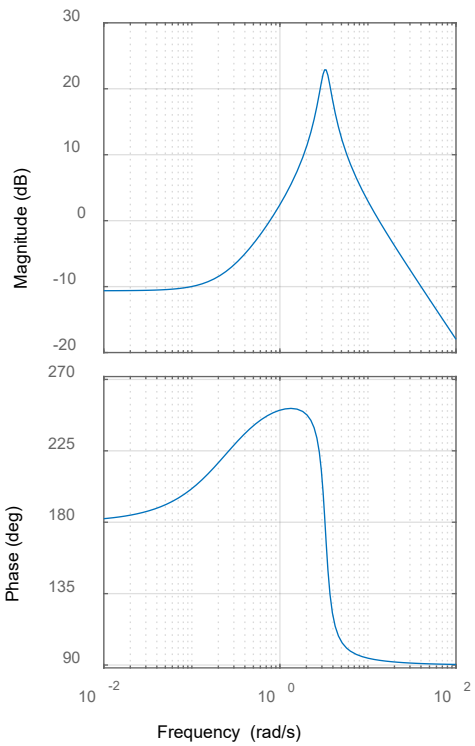
2DOF Dutch Mode bode plot ($\beta' \delta_a$)

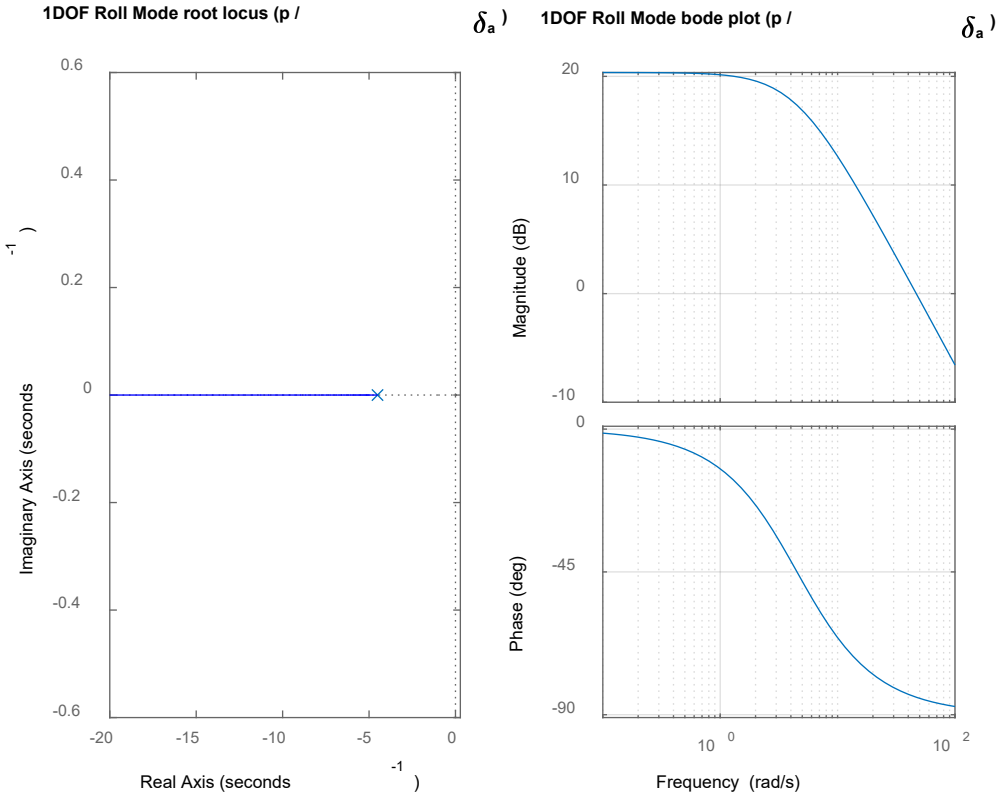


2DOF Dutch Mode root locus ($\beta' \delta_r$)



2DOF Dutch Mode bode plot ($\beta' \delta_r$)

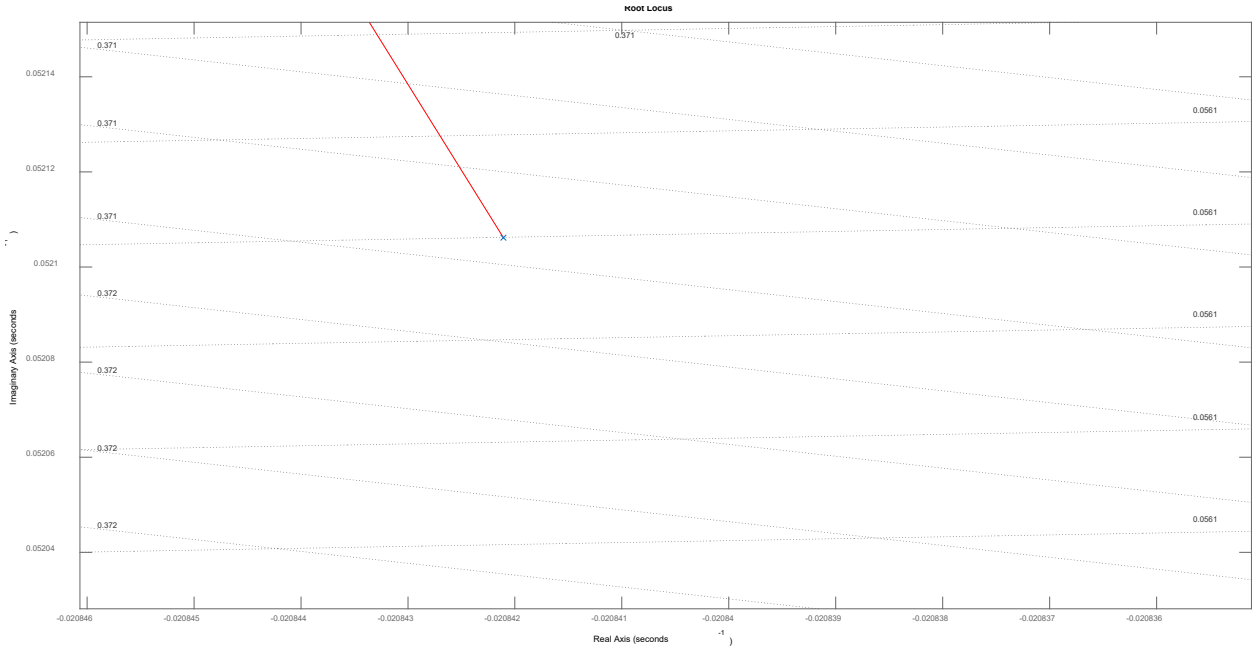



1DOF Roll Mode root locus (p / δa)
Imaginary Axis (seconds -1)
Real Axis (seconds -1)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-20
-15
-10
-5
0
1DOF Roll Mode bode plot (p / δa)
Magnitude (dB)
Phase (deg)
Frequency (rad/s)
20
10
0
-10
-45
-90
10 0
10 2

Task (5)

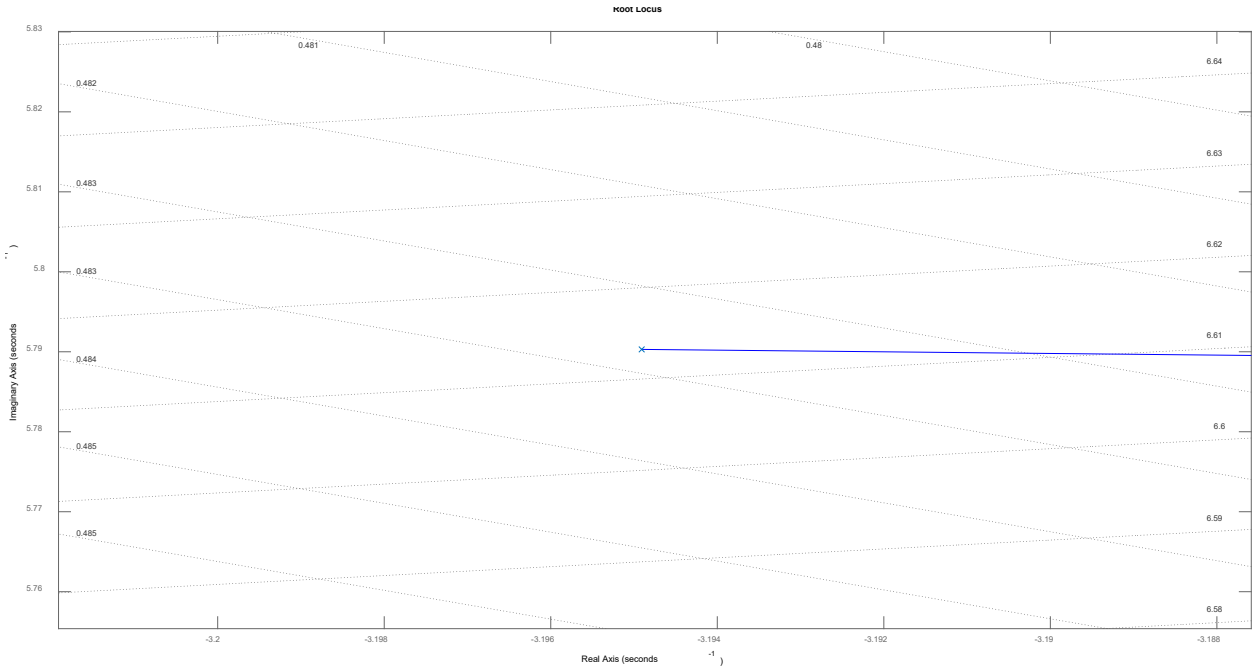
A. Design Pitch Controller with pitch rate feedback

Long period:



$$\xi = 0.3714 > 0.04$$

Short period



$$0.3 < \xi = 0.4831 < 2.0$$

Then the open loop transfer function is

```
OL_theta_thetacom =

                  527 s^2 + 1848 s + 74.13
      ------------------------------------------------------
      s^5 + 16.43 s^4 + 108.3 s^3 + 441.9 s^2 + 18.57 s + 1.377

Continuous-time transfer function.
```

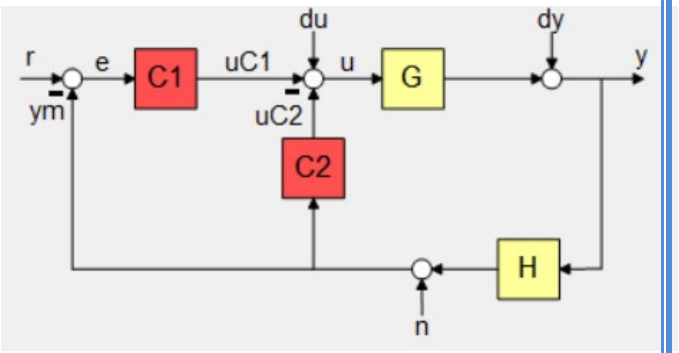
Design control loop with PD and PID:

```
PD_tf =

  0.085498 s

Name: C2

Continuous-time zero/pole/gain model.
```




```
PI_tf =

  0.75333 (s+0.6555)

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

          s

Name: C1

Continuous-time zero/pole/gain model.
```

Closed Loop transfer function:

```
CL_theta_thetacom_tf =


  From input "r" to output "y":

                 397 s^3 + 1653 s^2 + 968.6 s + 36.61

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

  s^6 + 16.43 s^5 + 153.4 s^4 + 996.9 s^3 + 1678 s^2 + 970 s + 36.61


Continuous-time transfer function.
```

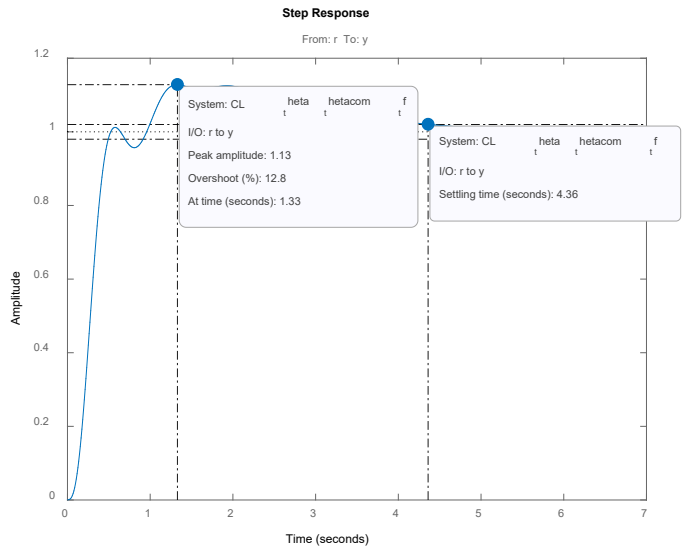


Figure 1. Response

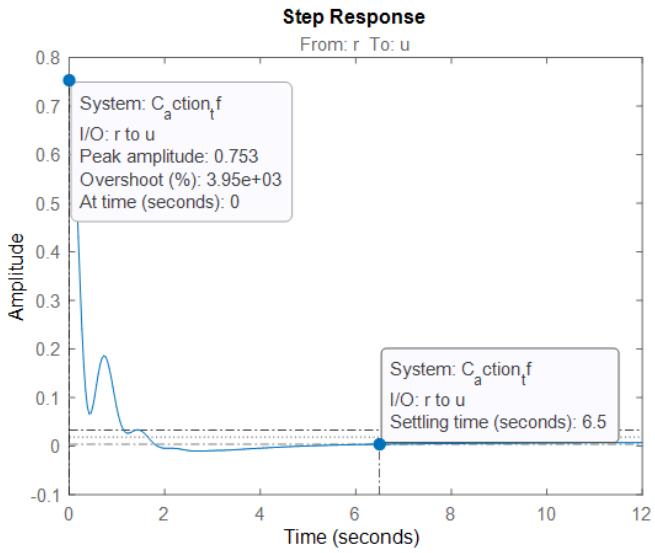


Figure 2. Control Action

Control action transfer function:

```
C_action_tf =


  From input "r" to output "u":

0.7533 s^6 + 12.87 s^5 + 89.72 s^4 + 386.4 s^3 + 232.2 s^2 + 10.21 s +
0.6802

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

       s^6 + 16.43 s^5 + 153.4 s^4 + 996.9 s^3 + 1678 s^2 + 970 s + 36.61


Continuous-time transfer function.
```

B. Testing Pitch Controller on the Full State Space Model

Simulink Simulation

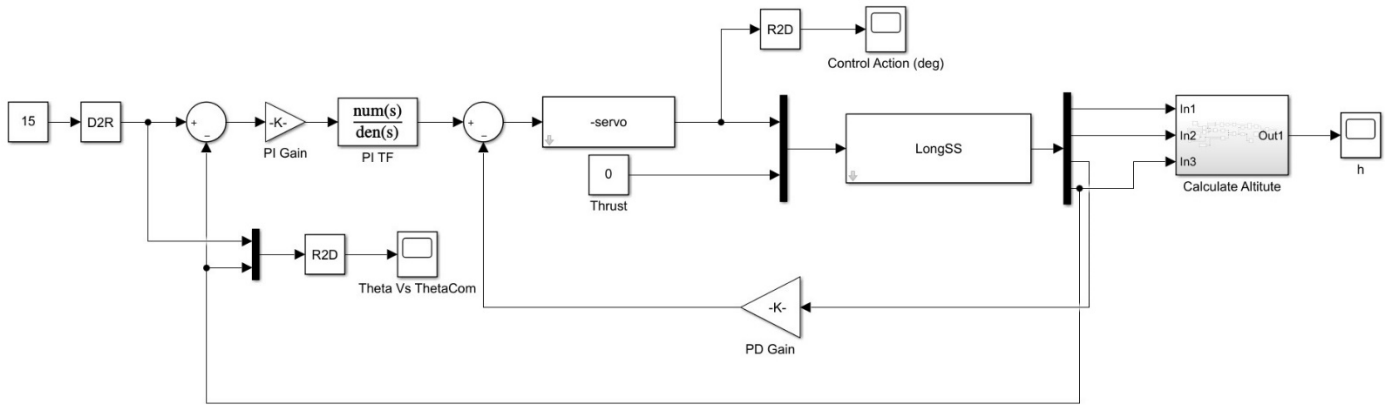


Figure 3. Simulink - Pitch Controller check

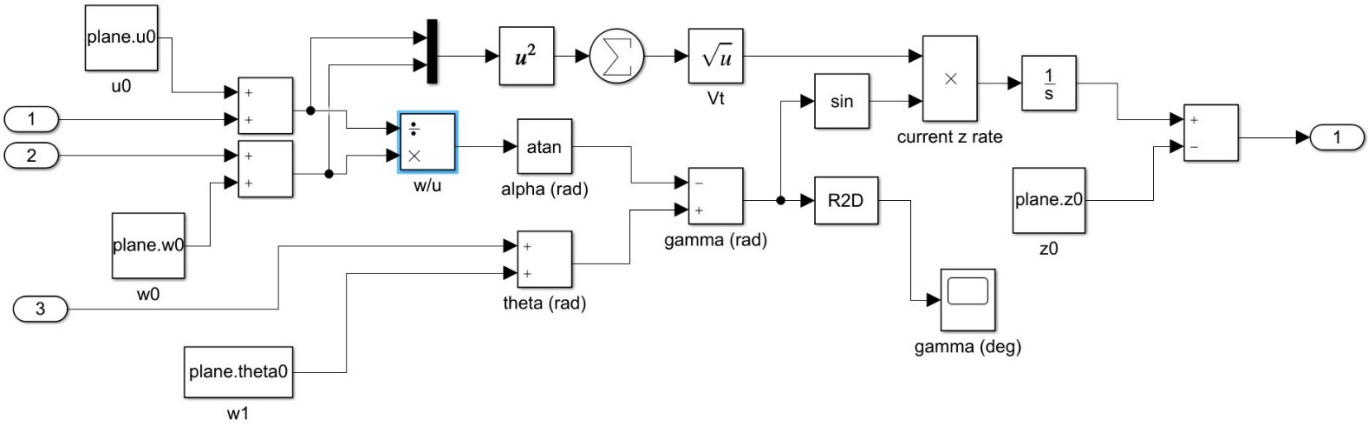


Figure 4. Simulink - Gamma & Altitude

Results

Command of 15-degree pitch angle

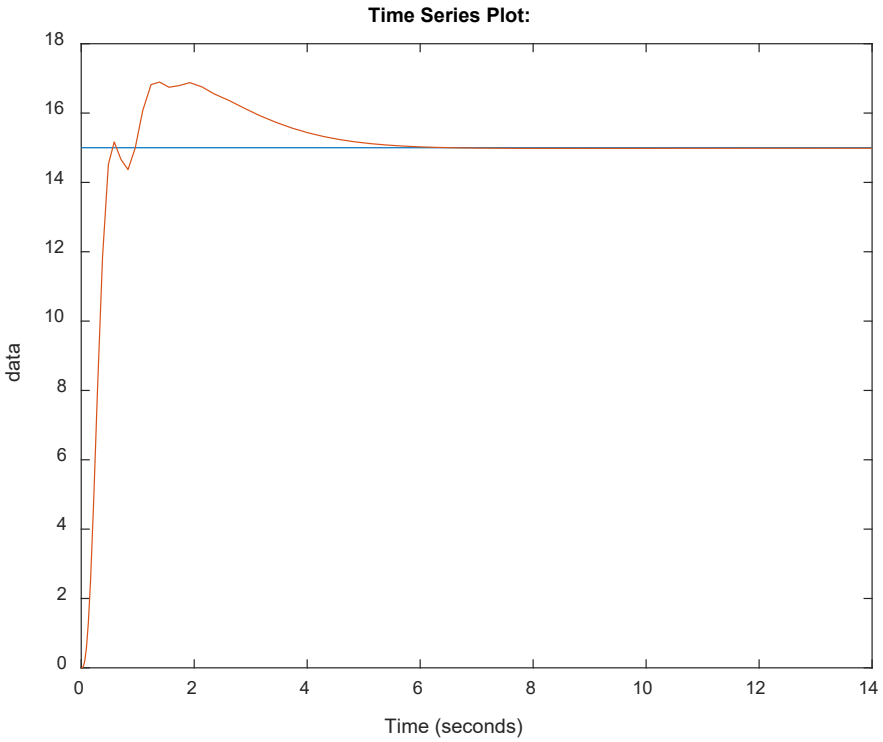


Figure 5. Theta response with Theta Command

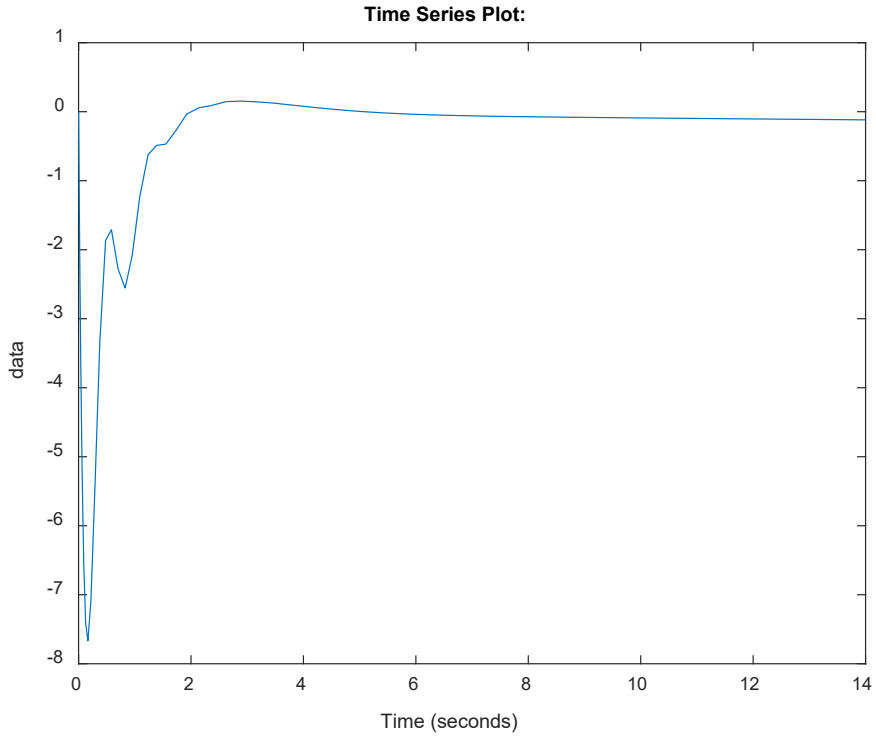


Figure 6. Control Action

C. Necessity of velocity control

when we input positive pitch angle, the action depends on the thrust if the thrust is big enough to climb upward the airplane will climb upward if the thrust is not enough the airplane would dive downward but logically, when the pilot input a positive pitch the airplane should climb upward.

$$\because \gamma = \theta - \alpha$$

$$tan\,(\alpha) = \frac{w}{u}$$

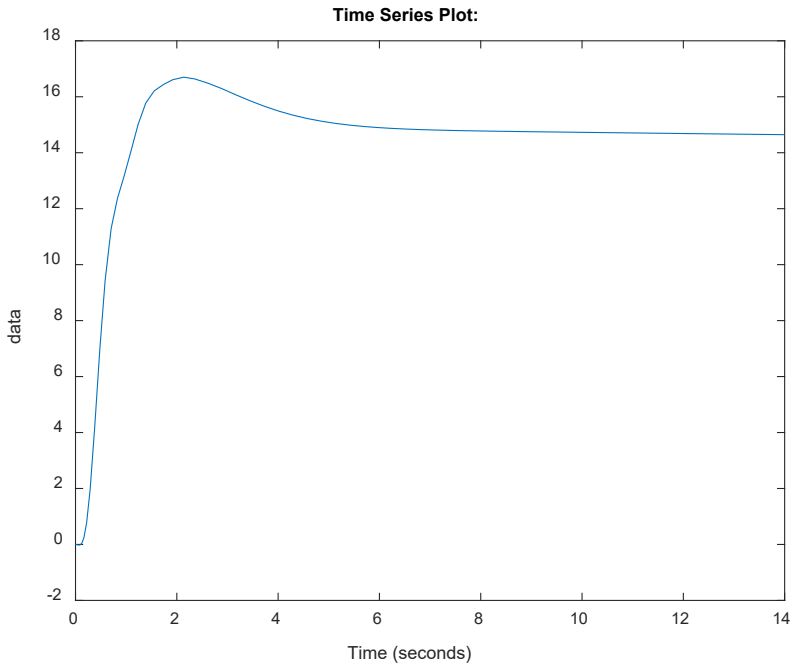


Figure 7. Gamma

We did use this relation to calculate the altitude rate of change and by integration we could obtain the current altitude value.

$$\because \dot{h} = V_{to} * Sin(\gamma)$$

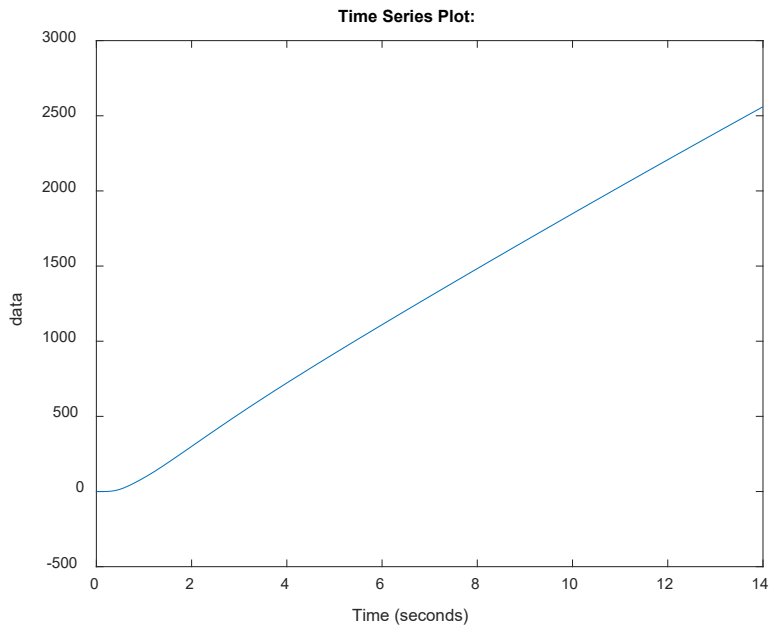


Figure 8. Altitude

Our plane climbs upward because it has enough thrust to accomplish this. But this not meaning that we control altitude because the airplane will climb to specific height that its thrust enough to reach, then the airplane would dive downward. And this specific height may be before or after that height we need to reach, so we need to control the velocity to reach to required altitude by change thrust (δ_{th})

D. Design a "Velocity Controller"

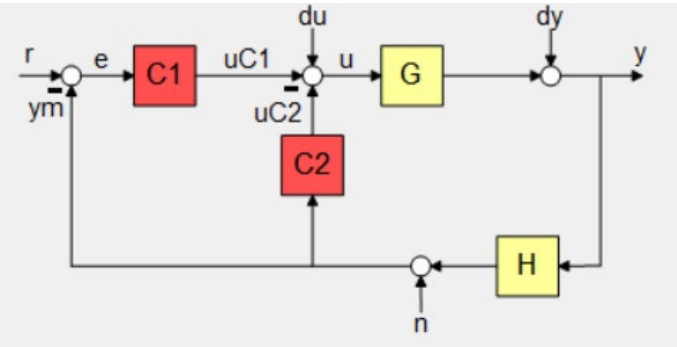
Then the open loop transfer function is

```
Ol_u_ucom =

              0.00235 s^3 + 0.015 s^2 + 0.1027 s - 5.831e-05
  ------------------------------------------------------------------
  s^6 + 16.53 s^5 + 110 s^4 + 452.7 s^3 + 62.76 s^2 + 3.234 s + 0.1377

Continuous-time transfer function.
```

Design control loop with PD and PID:




```
PD_tf_vel =


  15.095 (s+0.1)
  --------------
    (s+0.109)


Name: C2

Continuous-time zero/pole/gain model.


PI_tf_vel =


  32.915 (s+0.03648)
  ------------------
           S

Name: C1

Continuous-time zero/pole/gain model.
```

Closed Loop transfer function:

```
CL_u_ucom_tf =

  From input "r" to output "y":

 0.07735 s^5 + 0.5051 s^4 + 3.454 s^3 + 0.4919 s^2 + 0.01316 s- 7.629e-06
  ----------------------------------------------------------------------------

s^8 + 16.64 s^7 + 111.8 s^6 + 464.7 s^5 + 111.8 s^4 + 8.193 s^3 + 0.1525
s^2 + 0.001761 s + 7.629e-06


Continuous-time transfer function.
```

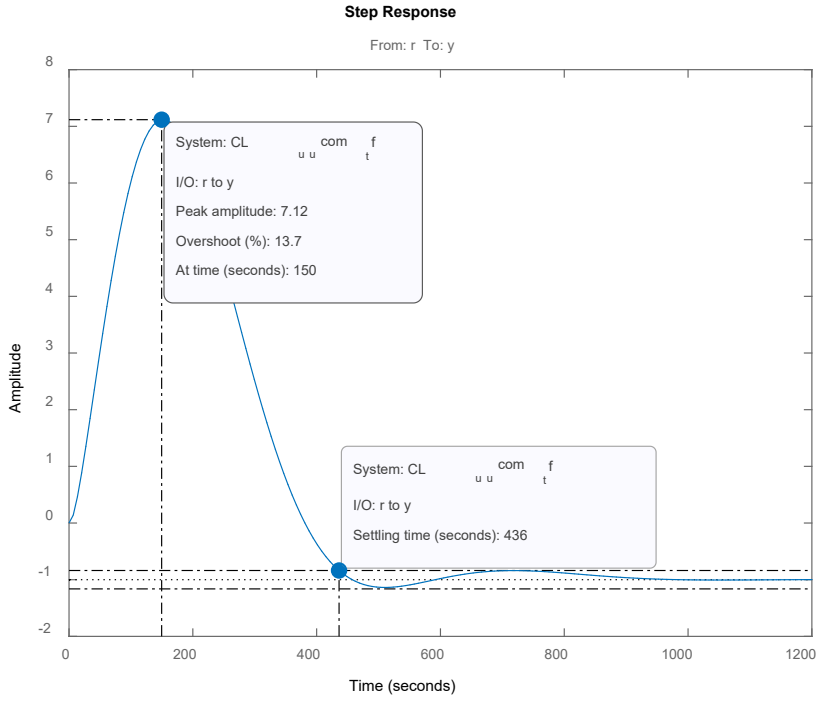


Figure 9 Response

Control action transfer function:

```
Con_action_tf_vel =


  From input "r" to output "u":

32.92 s^8 + 548.9 s^7 + 3699 s^6 + 1.543e04 s^5 + 4247 s^4 + 466.1 s^3 +
28.23 s^2 + 1.083 s + 0.01802

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

s^8 + 16.64 s^7 + 111.8 s^6 + 464.7 s^5 + 111.8 s^4 + 8.193 s^3 + 0.1525
s^2 + 0.001761 s + 7.629e-06


Continuous-time transfer function.
```

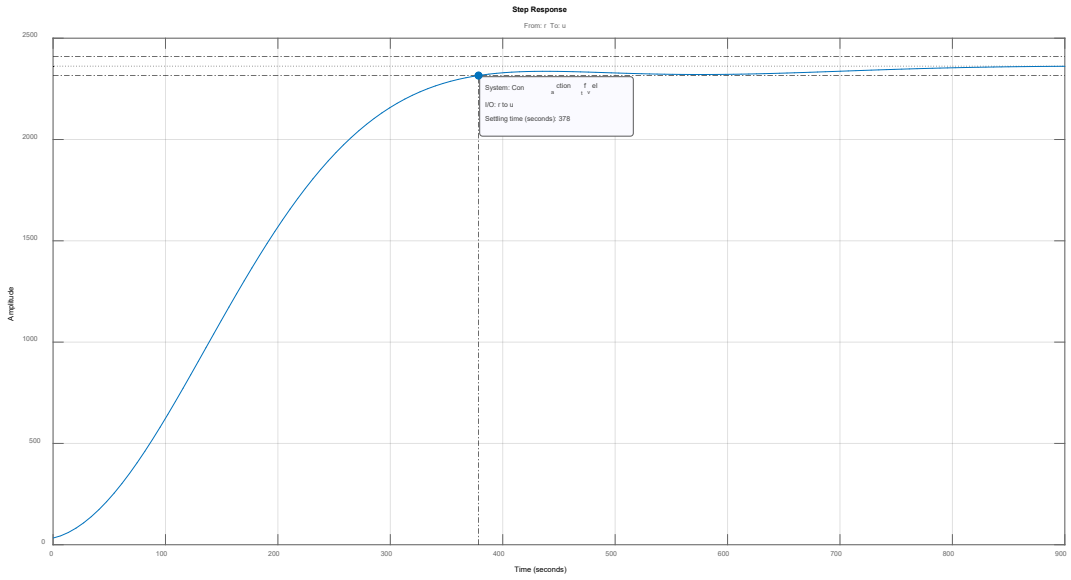


Figure 10 Control Action

F. Design “Altitude controller”

Then the open loop transfer function is

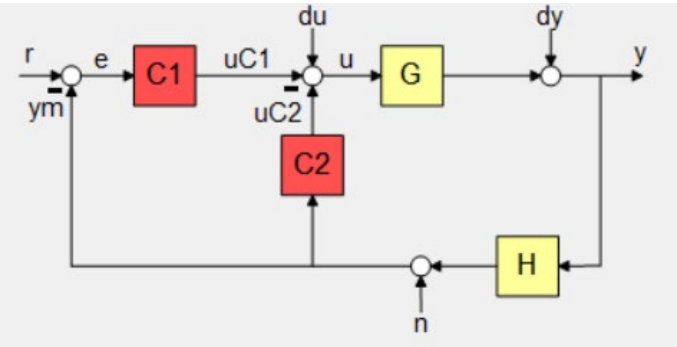
```
Ol_h_hcom =

  From input "r" to output:

 -1145 s^4 - 4001 s^3 + 1.073e06 s^2 + 7.461e05 s + 2.728e04
 ------------------------------------------------------------------------
  s^7 + 16.43 s^6 + 153.4 s^5 + 996.9 s^4 + 1678 s^3 + 970 s^2 + 36.61 s

Continuous-time transfer function.
```

Design control loop with PD and PID:




```
PD_tf_alt =


  0.0014691 (s+1.799)

Name: C2

Continuous-time zero/pole/gain model.


PI_tf_alt =


  0.00067403 (s+1.784)
  --------------------
            S


 Name: C1

Continuous-time zero/pole/gain model.
```

Closed Loop transfer function:

```
CL_h_hcom_tf =


  From input "r" to output "y":

  -0.7718 s^5 - 4.074 s^4 + 718.1 s^3 + 1793 s^2 + 915.7 s + 32.81

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

 s^8 + 16.43 s^7 + 151.7 s^6 + 987.2 s^5 + 3239 s^4 + 5620 s^3 + 3842 s^2 +
987.8 s + 32.81

Continuous-time transfer function.
```

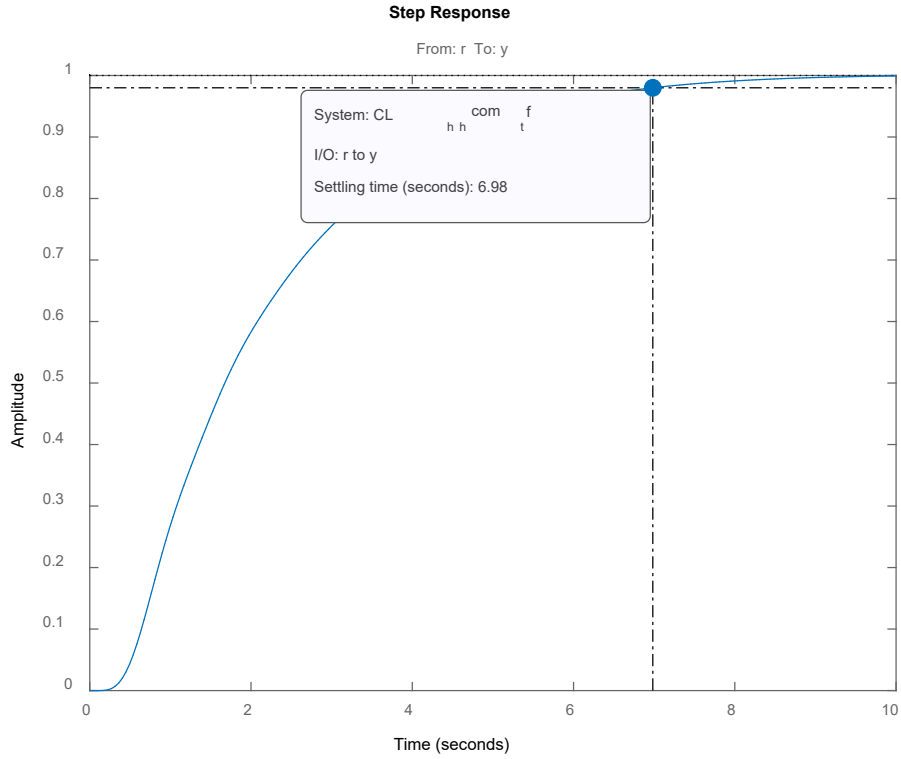


Figure 11 Response

Control action transfer function:

```
Con_action_tf_alt =


  From input "r" to output "u":

0.000674 s^8 + 0.01228 s^7 + 0.1231 s^6 + 0.8564 s^5 + 2.33 s^4 + 2.671 s^3
+ 1.191 s^2 + 0.04403 s + 5.584e-17

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

s^8 + 16.43 s^7 + 151.7 s^6 + 987.2 s^5 + 3239 s^4 + 5620 s^3 + 3842 s^2 +
987.8 s + 32.81

Continuous-time transfer function.
```

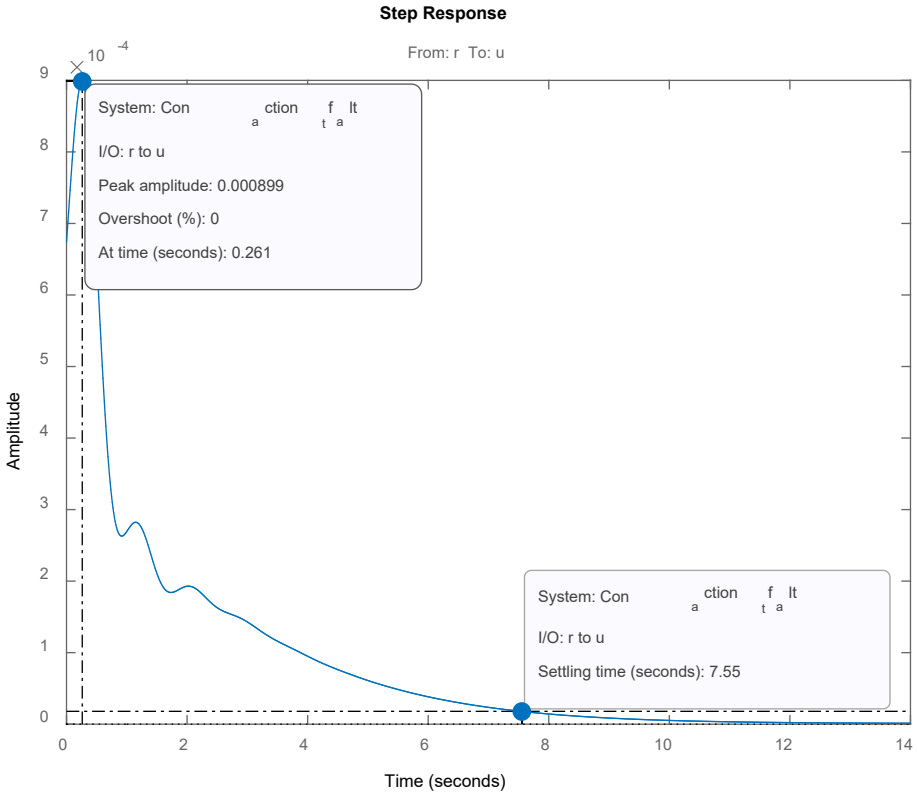
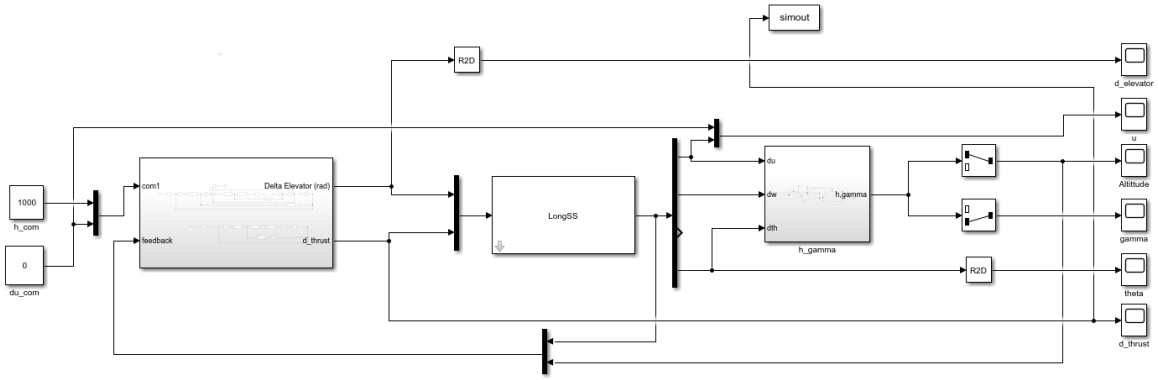


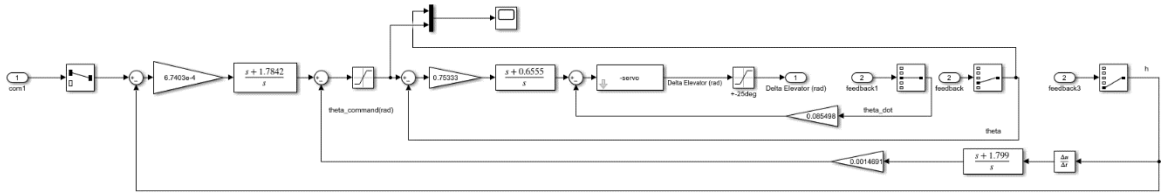
Figure 12 Control Action

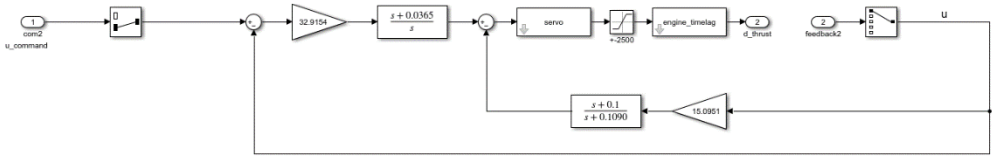
E. Test the (Pitch & Velocity) controllers on the full state space model and (g) Test the "Altitude & Velocity Controllers" together on the full state space model

Simulink Simulation

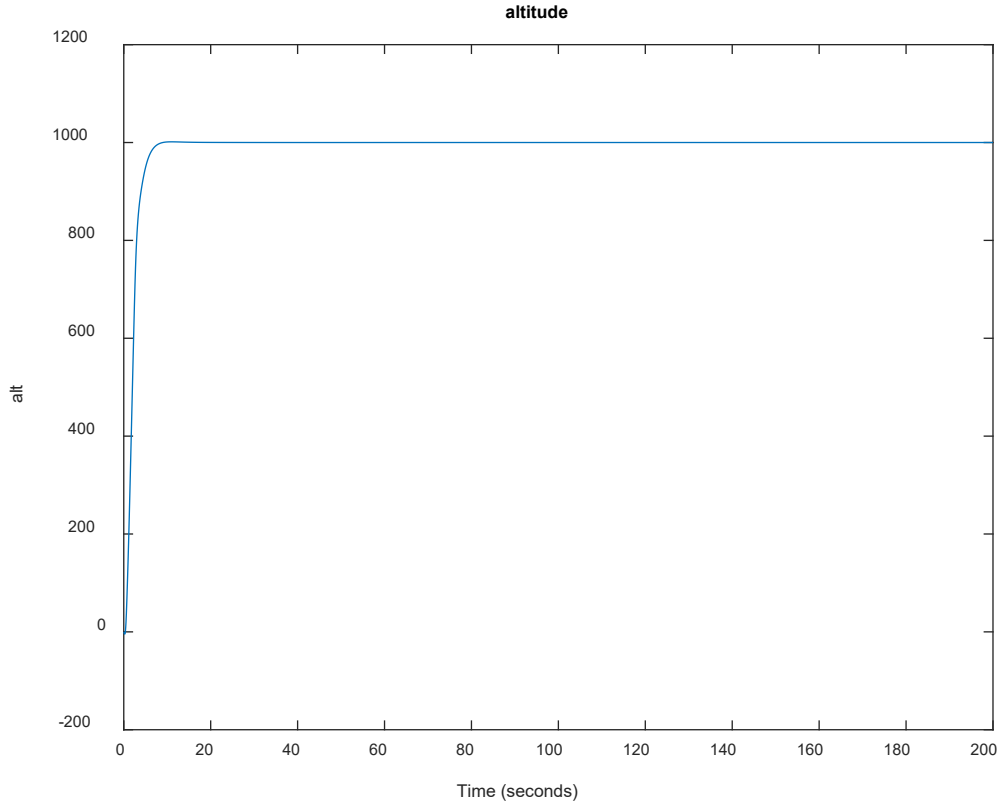


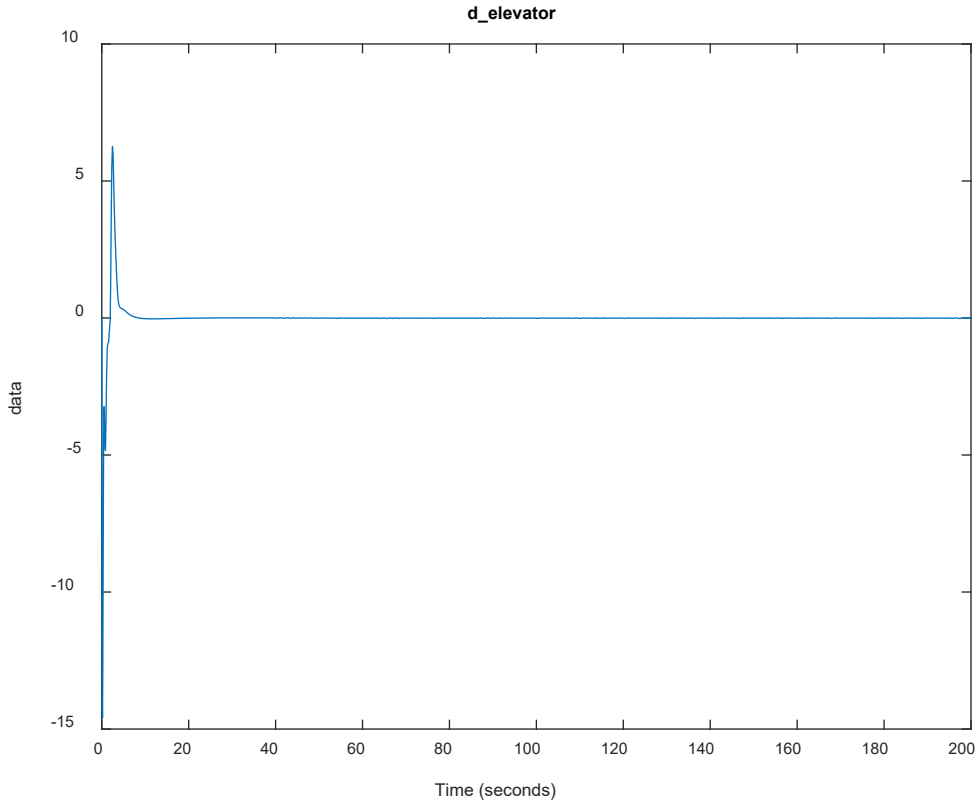
Subsystems:

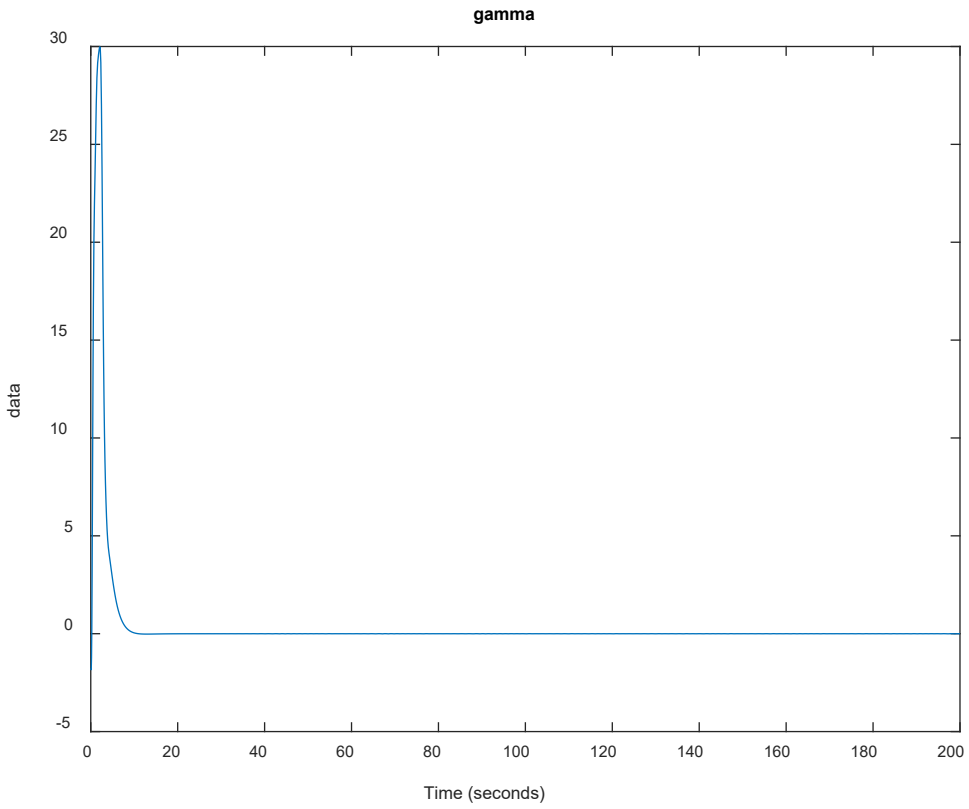

com1
6.7403e-4
s + 1.7842
s
theta_command(rad)
0.75393
s + 0.6555
s
-servo
Delta Elevator (rad)
+-25deg
Delta Elevator (rad)
feedback1
feedback
theta_dot
0.085498
theta
feedback3
h
0.001469
s + 1.799
s

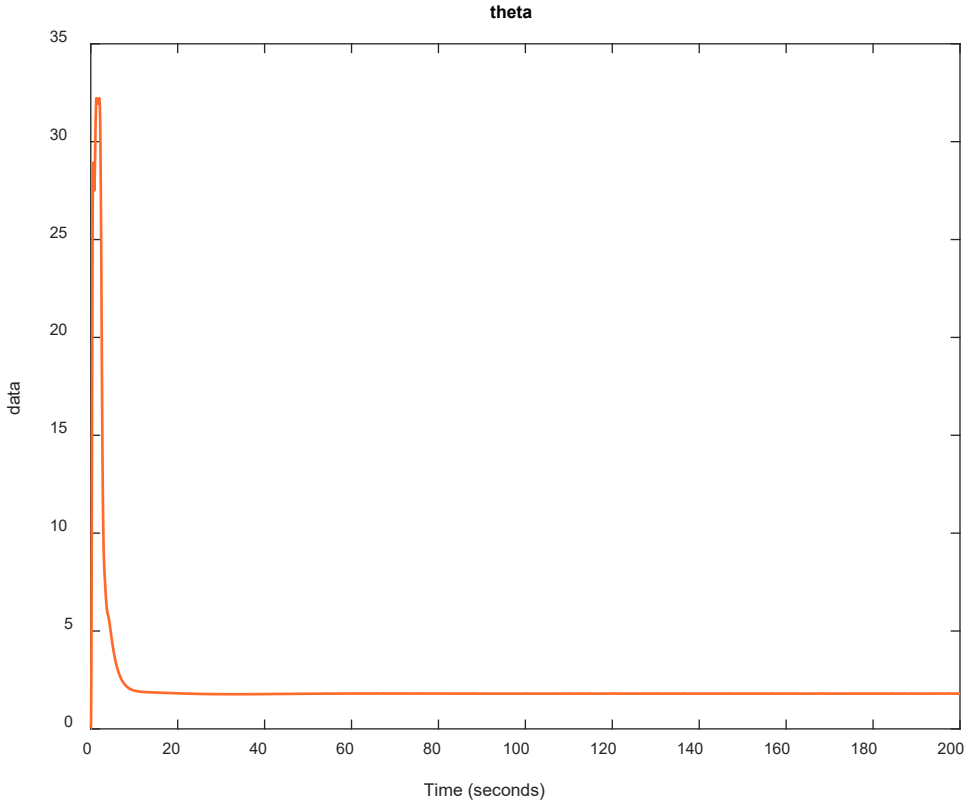

com2
u_command
32.9154
s + 0.0365
s
servo
+-2500
engine_timelag
d_thrust
feedback2
u
s + 0.1
s + 0.1090
15.0951

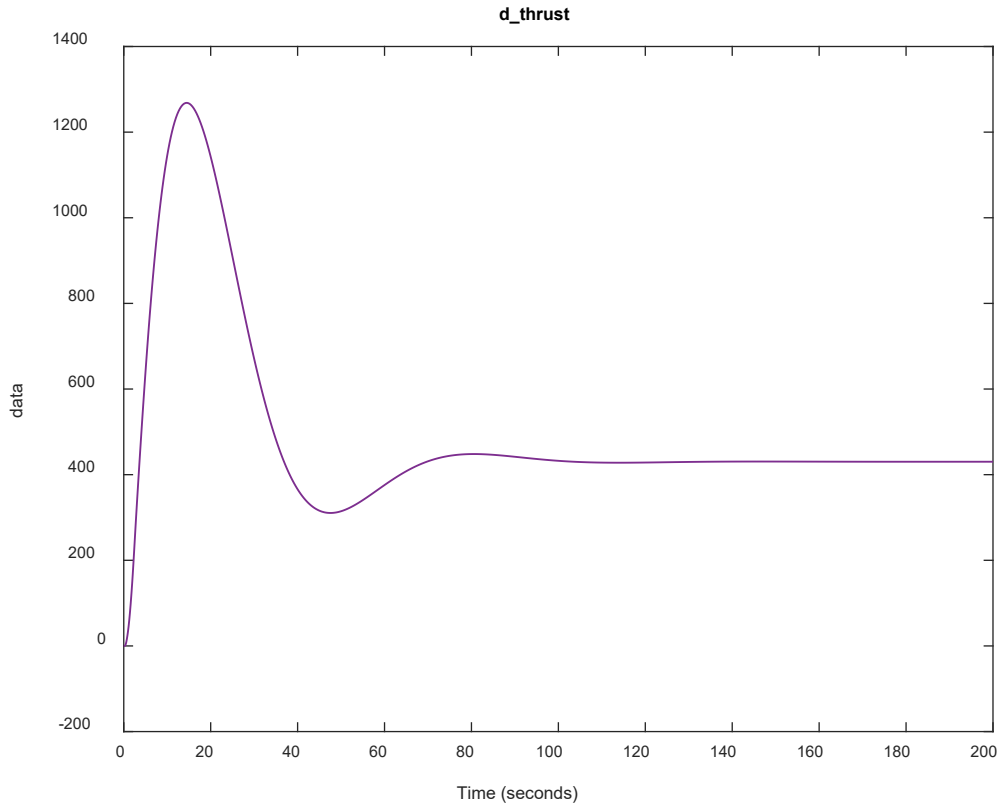
Results:

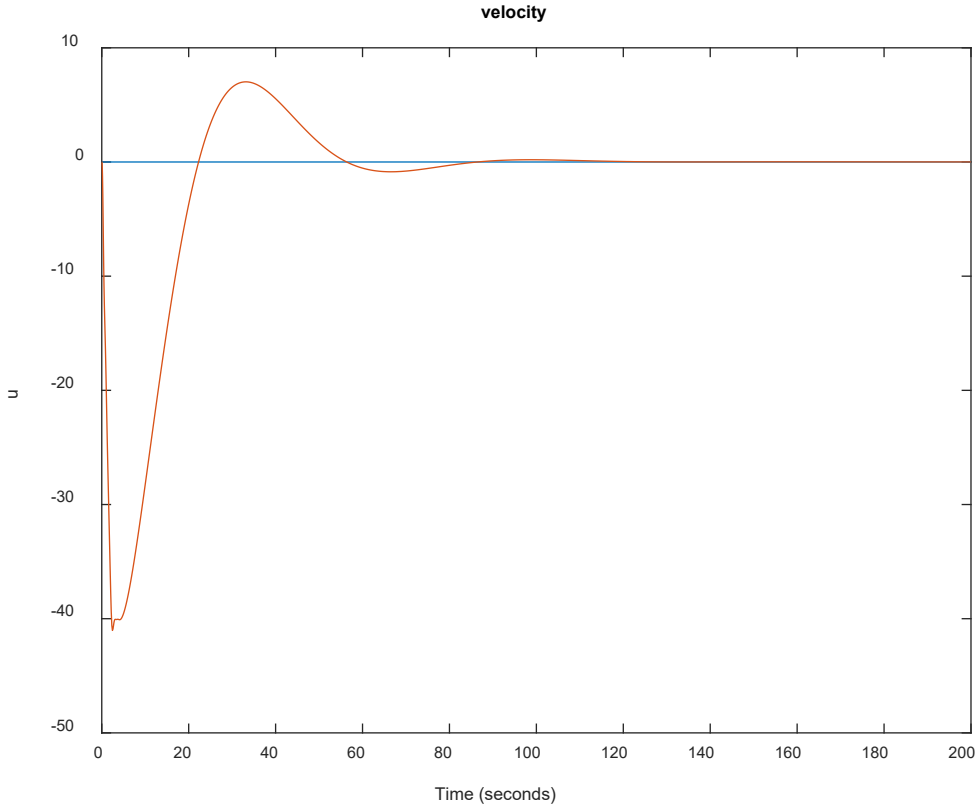

altitude
1200
1000
800
600
400
200
0
-200
alt
0
20
40
60
80
100
120
140
160
180
200
Time (seconds)


d_elevator
10
5
0
-5
-10
-15
data
0
20
40
60
80
100
120
140
160
180
200
Time (seconds)


gamma
data
Time (seconds)
30
25
20
15
10
5
0
-5
0
20
40
60
80
100
120
140
160
180
200


theta
data
Time (seconds)
35
30
25
20
15
10
5
0
0
20
40
60
80
100
120
140
160
180
200


d_thrust
data
Time (seconds)
1400
1200
1000
800
600
400
200
0
-200
0
20
40
60
80
100
120
140
160
180
200


velocity
u
Time (seconds)
10
0
-10
-20
-30
-40
-50
0
20
40
60
80
100
120
140
160
180
200

Task (6)

a) Design a "Yaw damper" for the Dutch Roll Mode

Design No. One

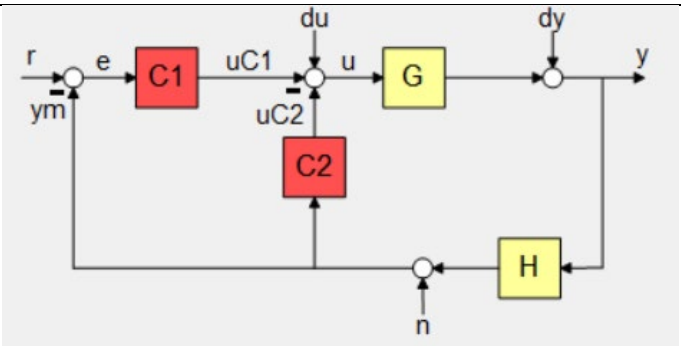
Then the open loop transfer function is

```
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
```

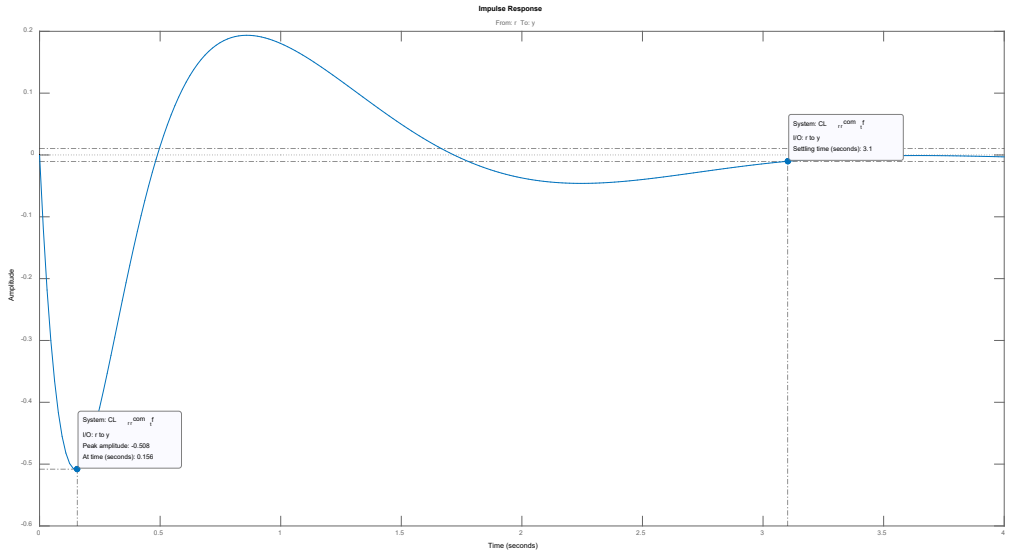


Figure 13. Design 1 - Response

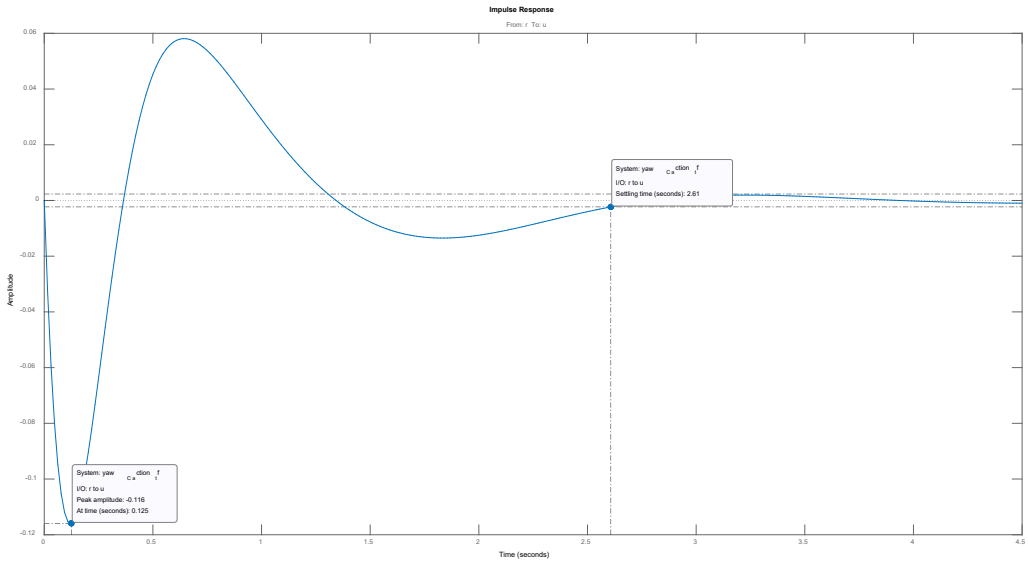


Figure 14. Design 1 - Control Action

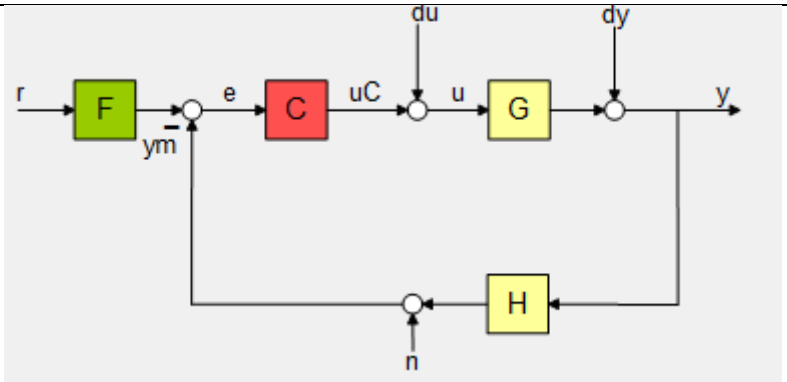
Then the open loop transfer function is

```
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
```

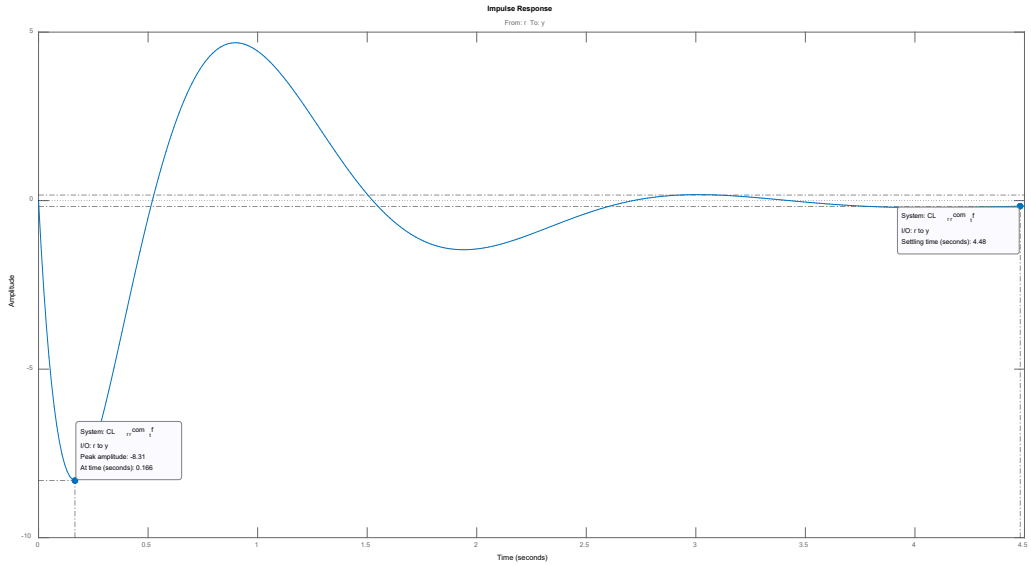


Figure 15. Design 2 - Response

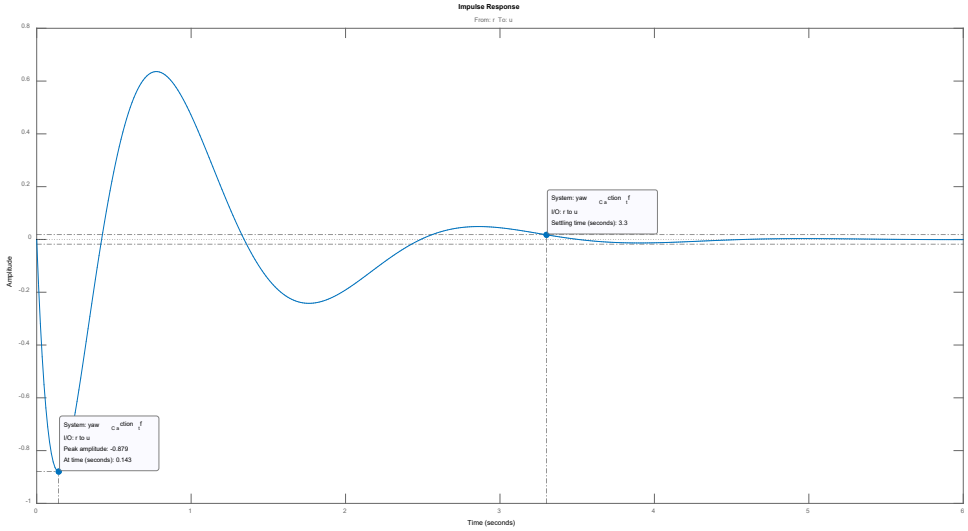
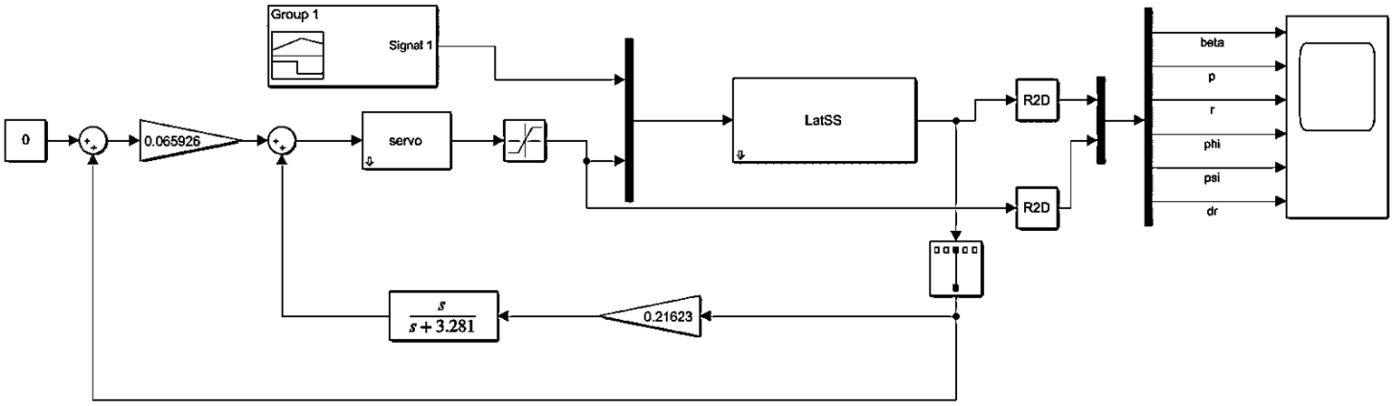


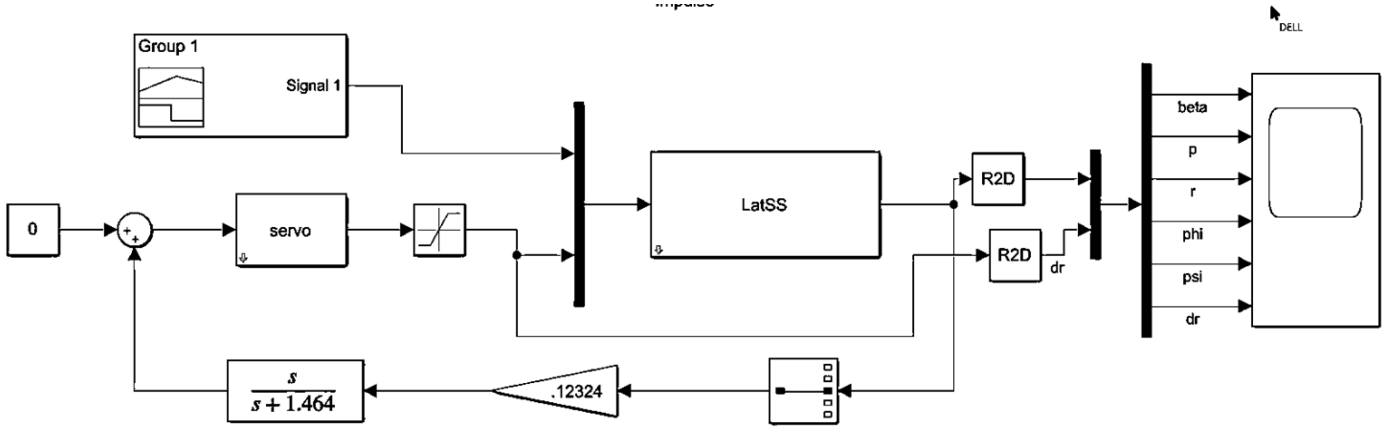
Figure 16. Design 2 - Control Action

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

Design No. One

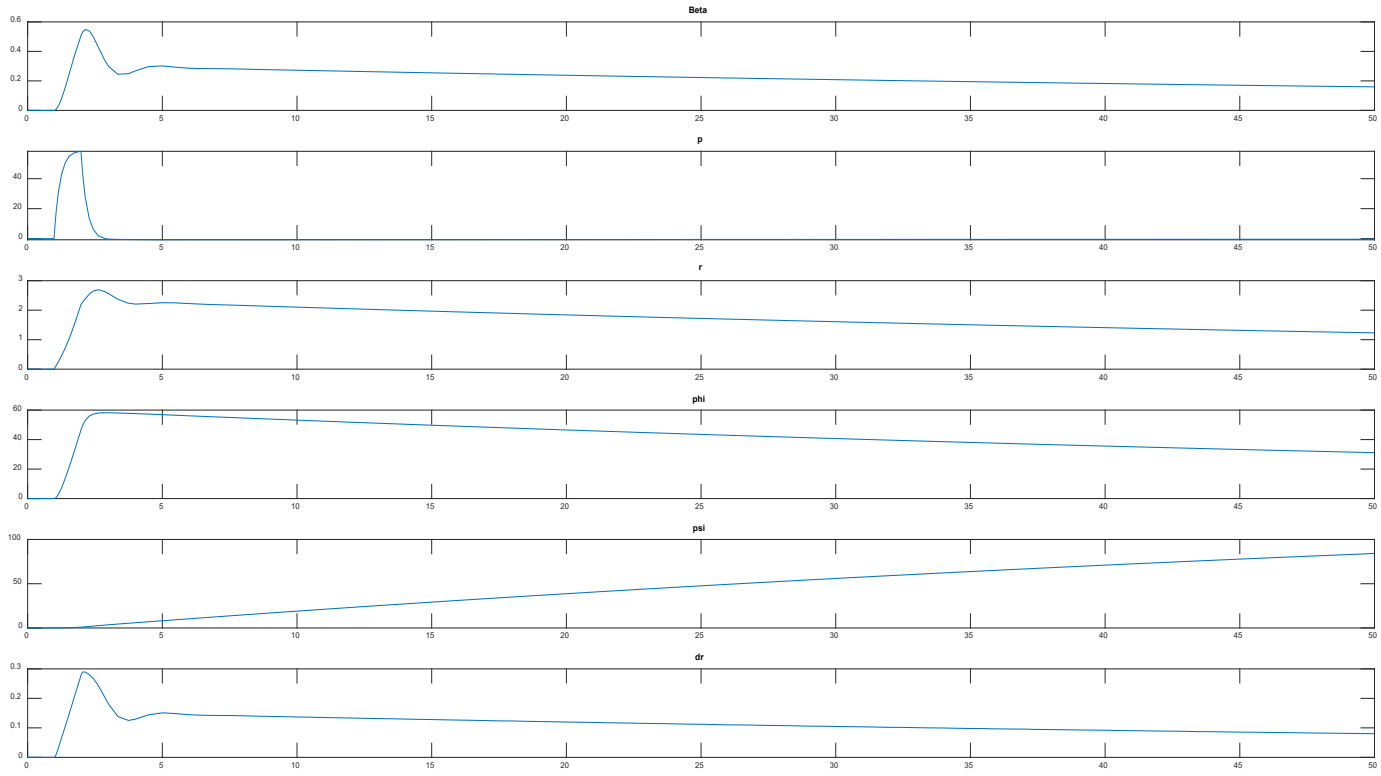


Design No. Two

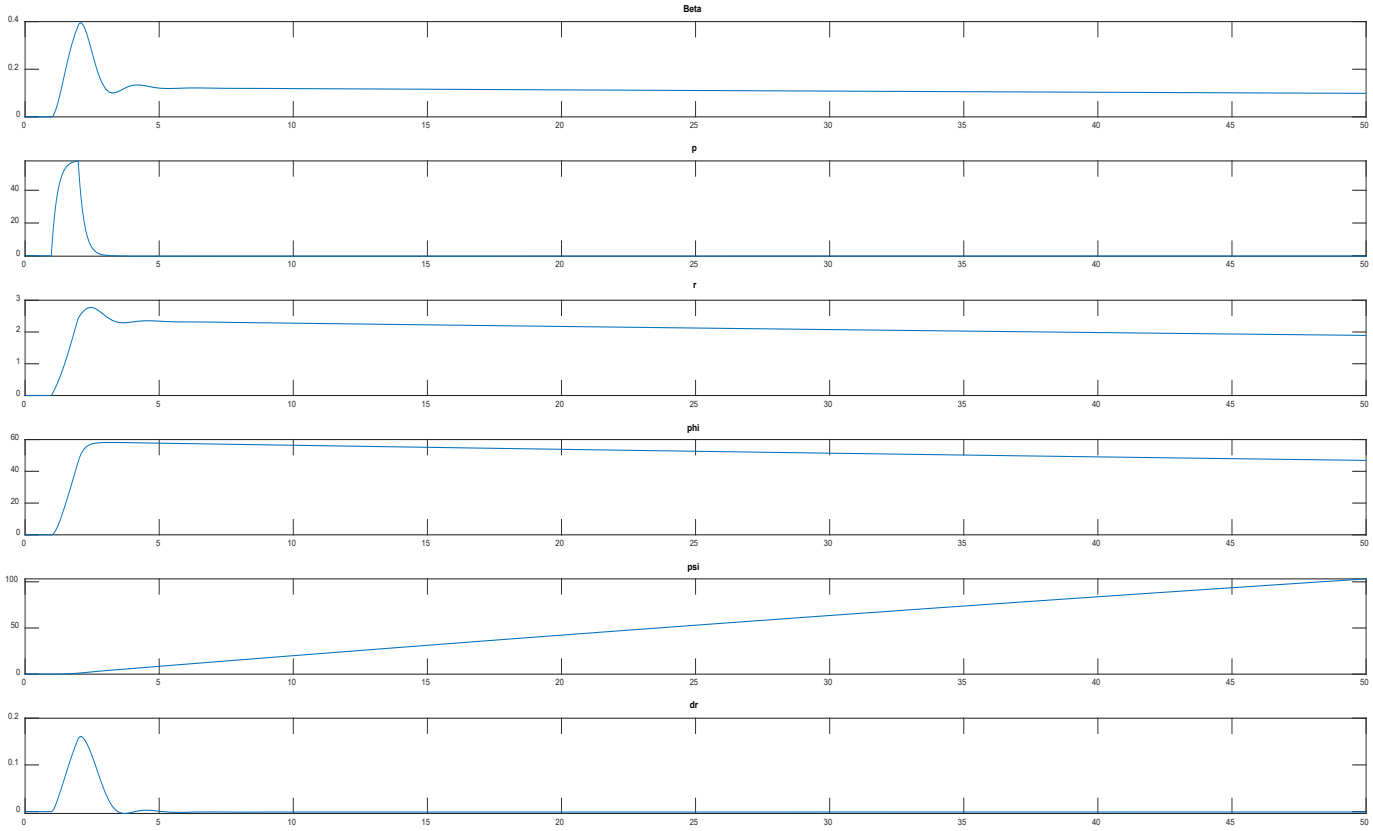


Results:

Design No. One



Design No. Two



c) Design "Roll Controller"

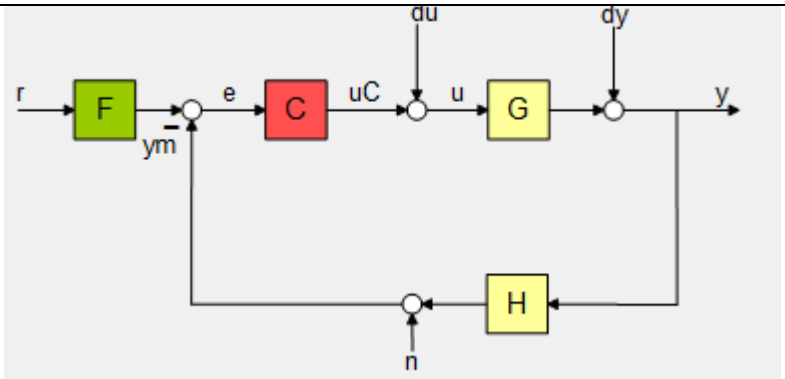
Then the open loop transfer function is

```
OL_phi_phicom =


4700 s^4 + 5.811e04 s^3 + 2.417e05 s^2 + 6.669e05 s + 7.489e05
-------------------------------------------------------------------------
s^8 + 36.87 s^7 + 544.7 s^6 + 4209 s^5 + 1.905e04 s^4 + 5.443e04 s^3
+ 9.535e04 s^2 + 7.327e04 s + 337.1


Continuous-time transfer function.
```

Design control loop:

roll_C_tf	Architecture
0.0078128 (s+0.02203) --------------------- s	

Closed Loop transfer function:

```
CL_roll_tf =

  From input "r" to output "y":


36.72 s^5 + 454.8 s^4 + 1898 s^3 + 5252 s^2 + 5966 s + 128.9

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

s^9 + 36.87 s^8 + 544.7 s^7 + 4209 s^6 + 1.908e04 s^5 + 5.489e04 s^4 +
9.725e04 s^3 + 7.853e04 s^2 + 6303 s + 128.9
```

Control action transfer function:

```
roll_C_action_tf =

  From input "r" to output "u":

0.007813 s^9 + 0.2882 s^8 + 4.262 s^7 + 32.97 s^6 + 149.5 s^5 + 428.5 s^4 +
754.3 s^3 + 588.9 s^2 + 15.24 s + 0.05802

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

  s^9 + 36.87 s^8 + 544.7 s^7 + 4209 s^6 + 1.908e04 s^5 + 5.489e04 s^4 +
9.725e04 s^3 + 7.853e04 s^2 + 6303 s + 128.9
```

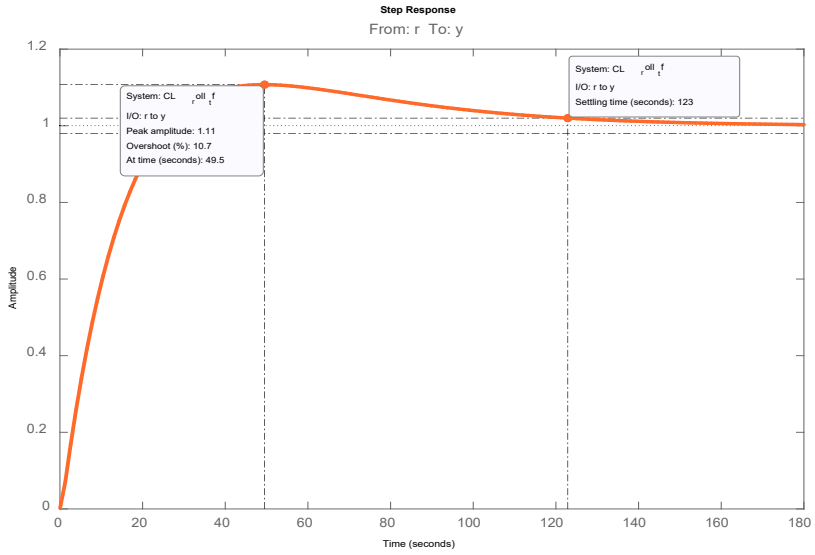


Figure 17. Response

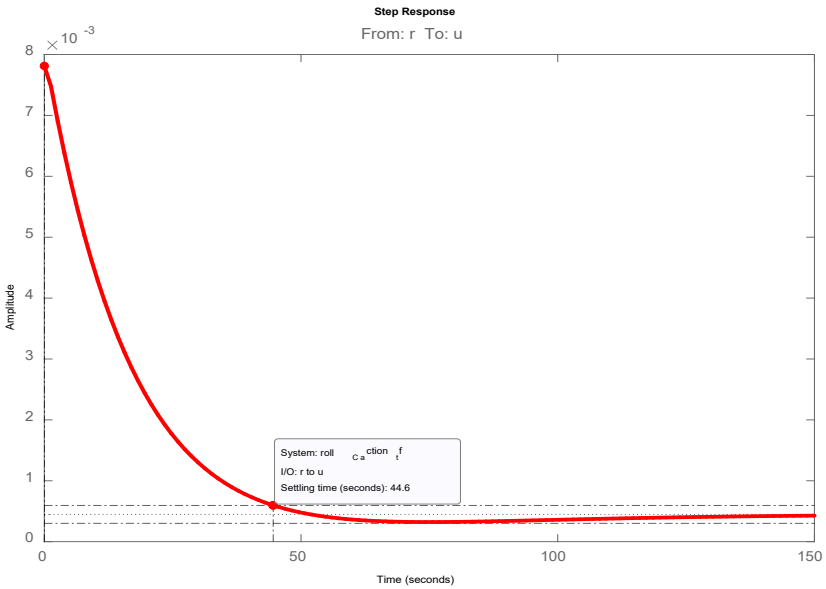
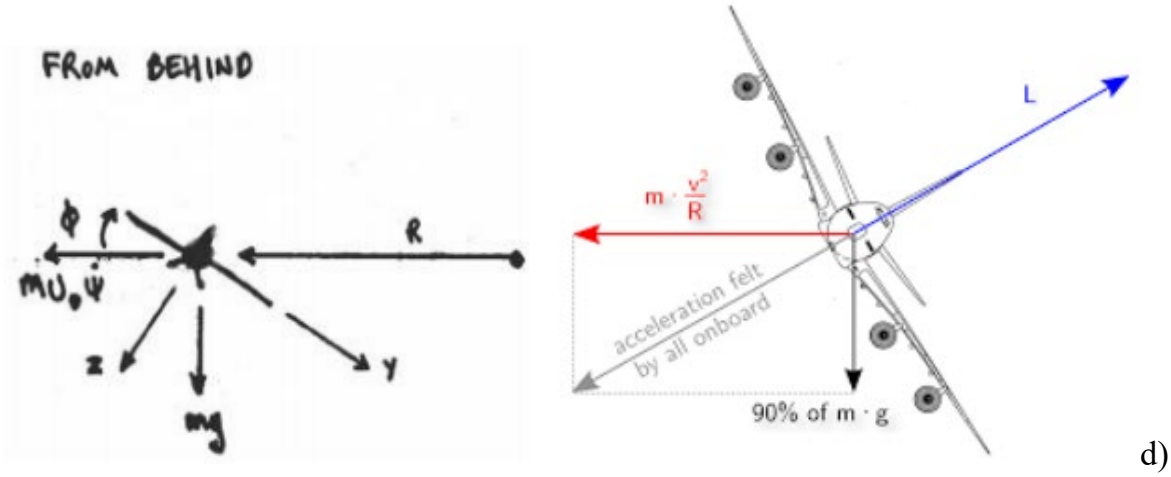


Figure 18. Control Action

d) Coordination



d)

As we can see from previous results that the sideslip angle (β) didn't settle at zero, therefore coordination is not achieved by default.

This can be achieved at a certain bank angle (**φ**) for a given Speed ($\boldsymbol{U_o}$) and turning rate $\dot{\boldsymbol{\psi}}$.This value of **φ** can be found from the free body diagram of the airplane during a turn.

Noting that the tangential Velocity is $\boldsymbol{U_o}$, **R** is the radius of the turn.

$$\therefore \boldsymbol{U_o = R\dot{\psi}}$$

From the free body diagram for a coordinated turn, we get the following:

$$L\cos(\phi) = mg$$

$$L\sin(\phi) = m\,U_o\dot{\psi}$$

By dividing the 2nd equation by the 1st one we get:

$$\boldsymbol{\tan(\phi) = \frac{U_o\dot{\psi}}{g}}$$

$$\boldsymbol{\therefore \phi \simeq \frac{U_o\dot{\psi}}{g}}$$

Therefore, in order to fly coordinated flight; for each velocity there is only one corresponding value of bank angle that satisfies the condition.

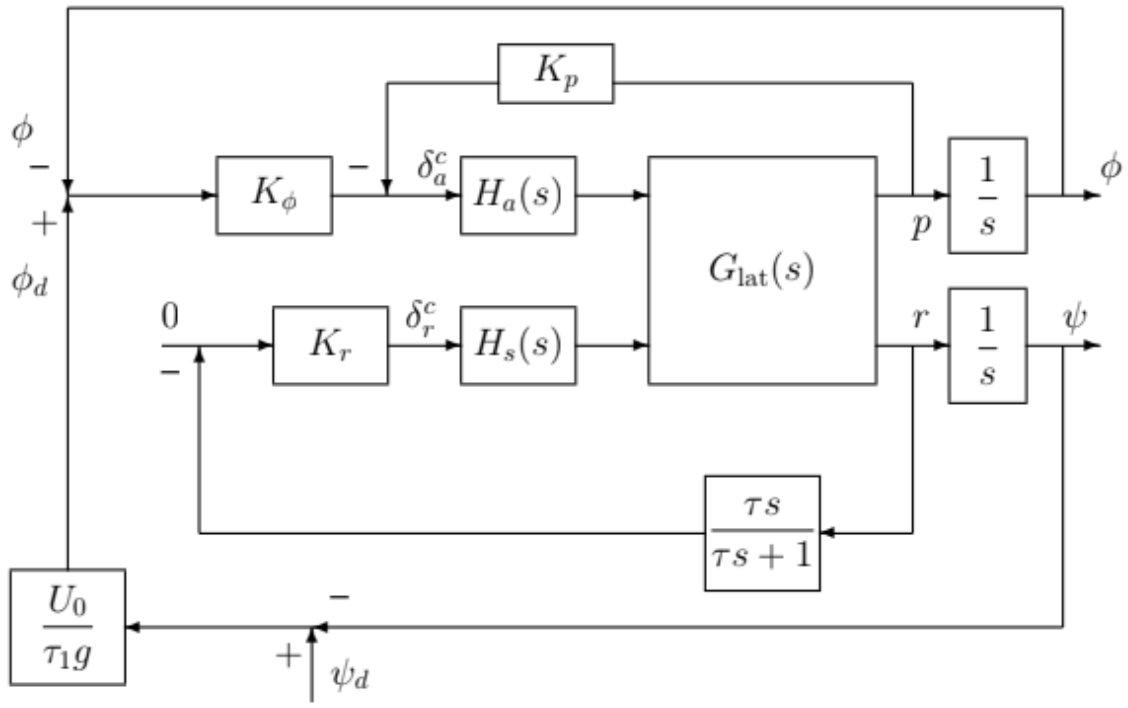
With:

$$\boldsymbol{{\psi}/{\psi_d} = \frac{1/\tau}{s+1/\tau}} \quad , \quad \boldsymbol{15 \le \tau \le 20}$$

$$\therefore \boldsymbol{\phi} = \frac{\boldsymbol{U_o}}{\boldsymbol{\tau\, g}}\,(\boldsymbol{\psi_{desired} - \psi})$$

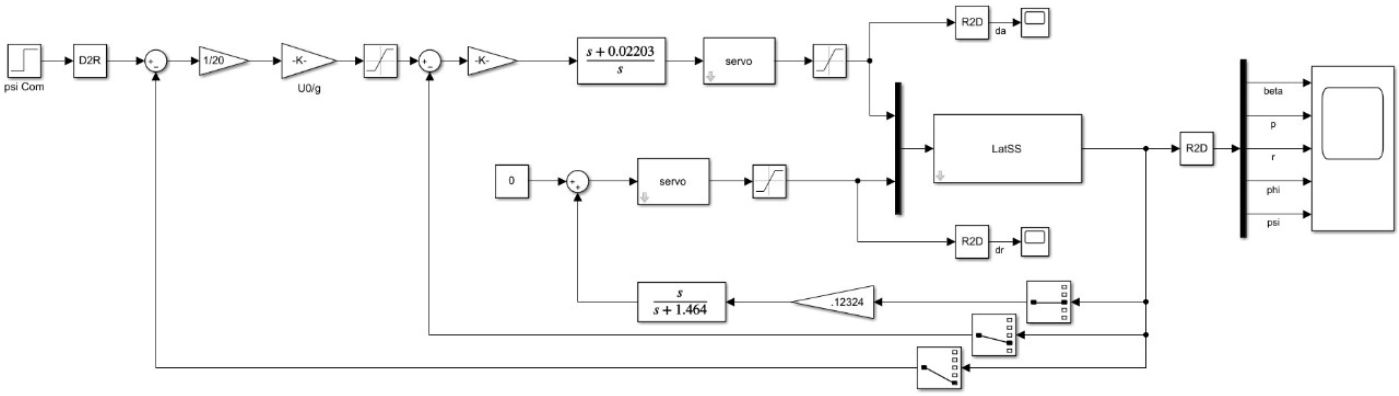
We choose $\boldsymbol{\tau = 15}$ & $\boldsymbol{\psi_d = 360\ degree}$ $(\boldsymbol{Complete\ turn})$

Final Controller after putting pieces together

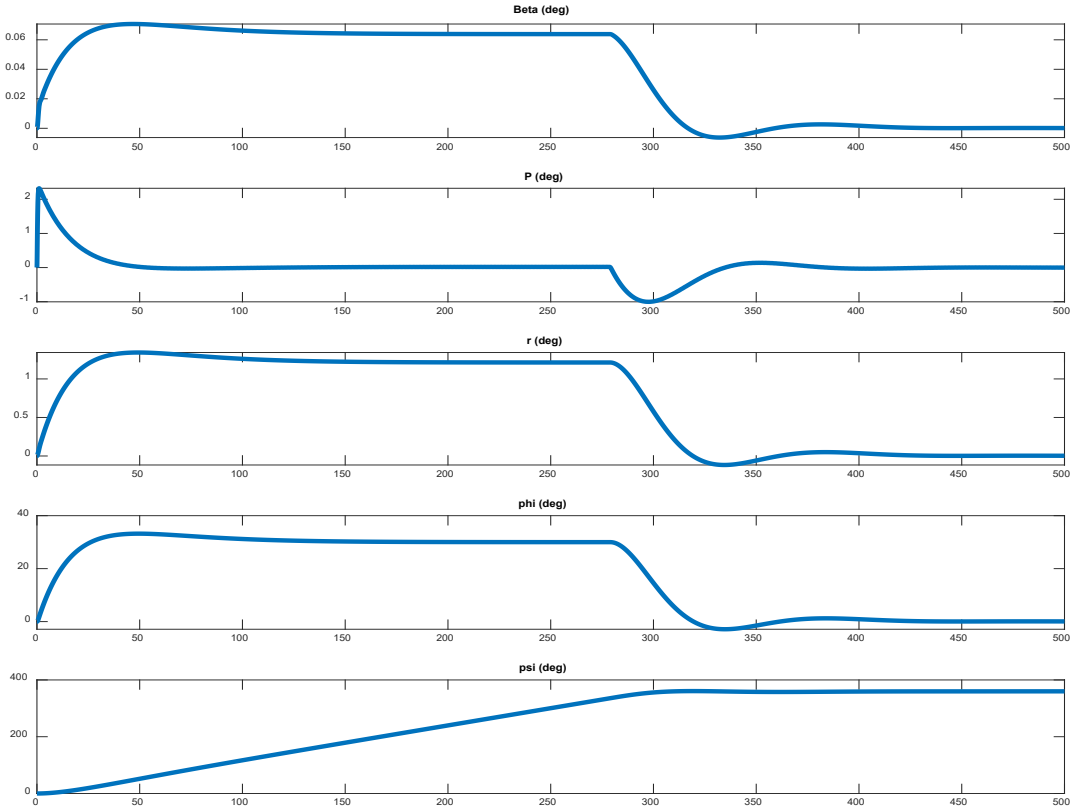


e) Test the designed controllers on the full state space model

Simulink

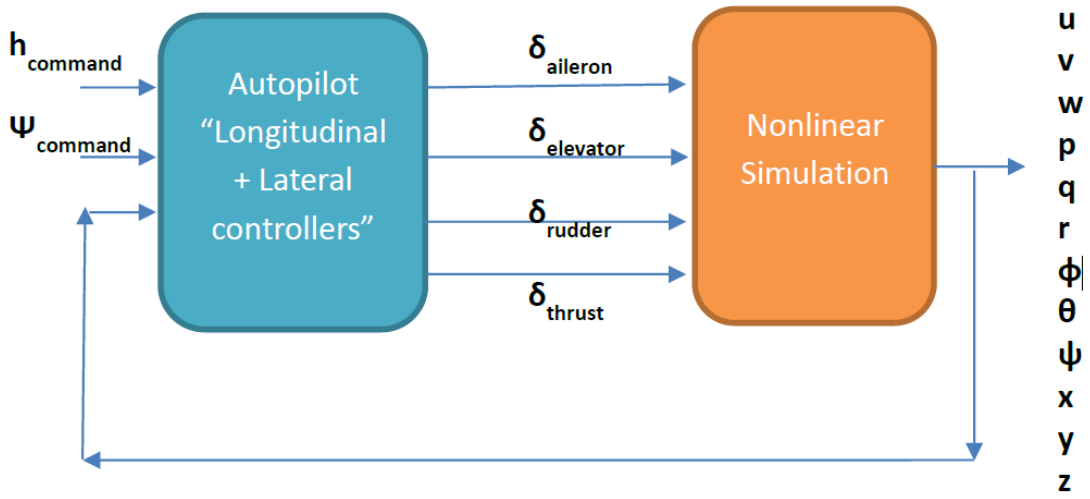


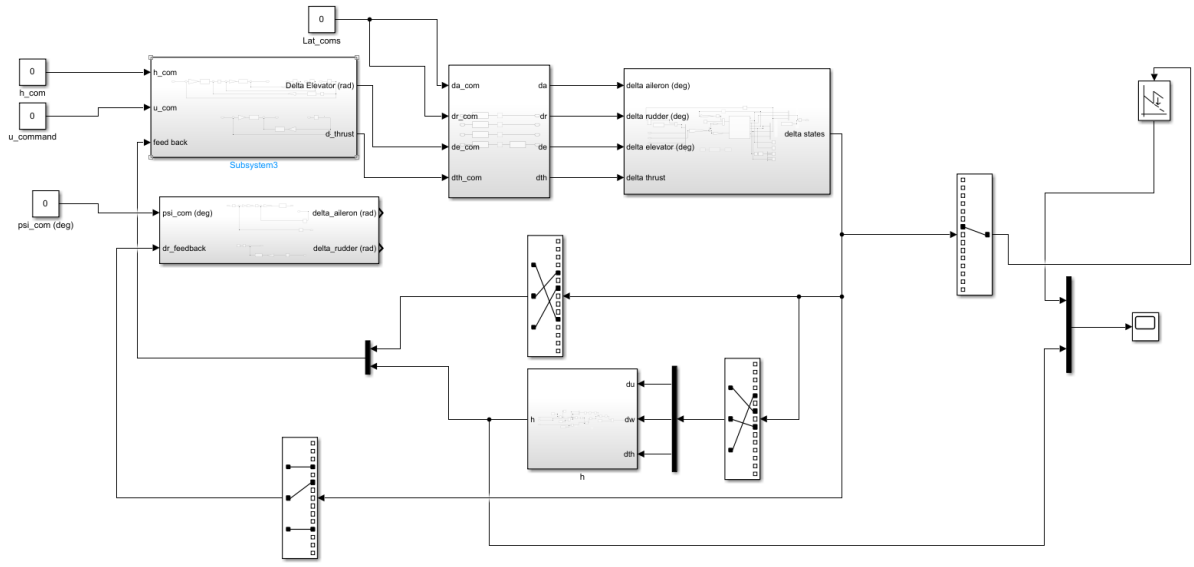
Results:

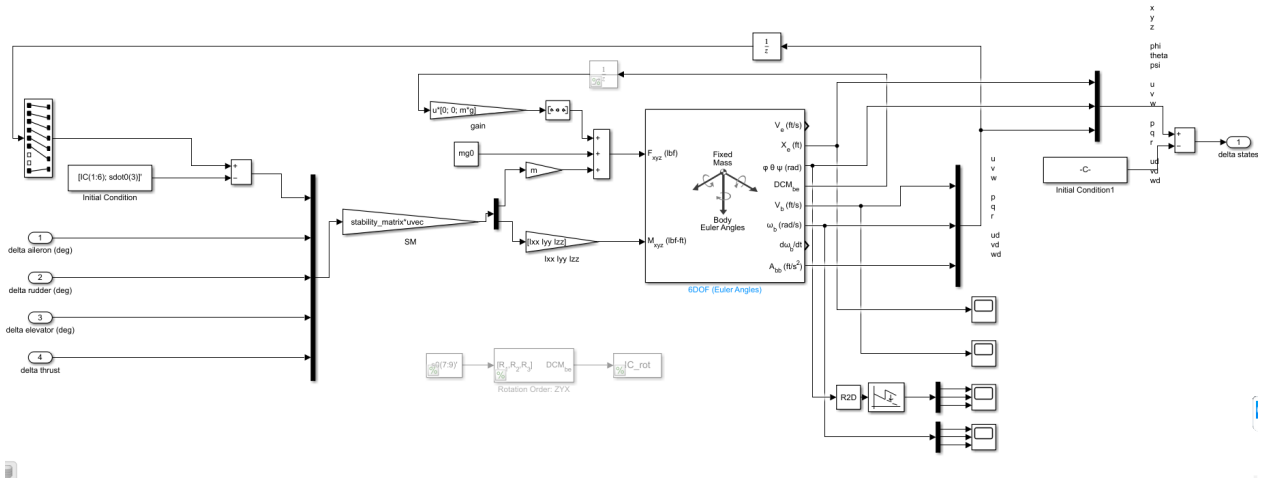


Task (7)

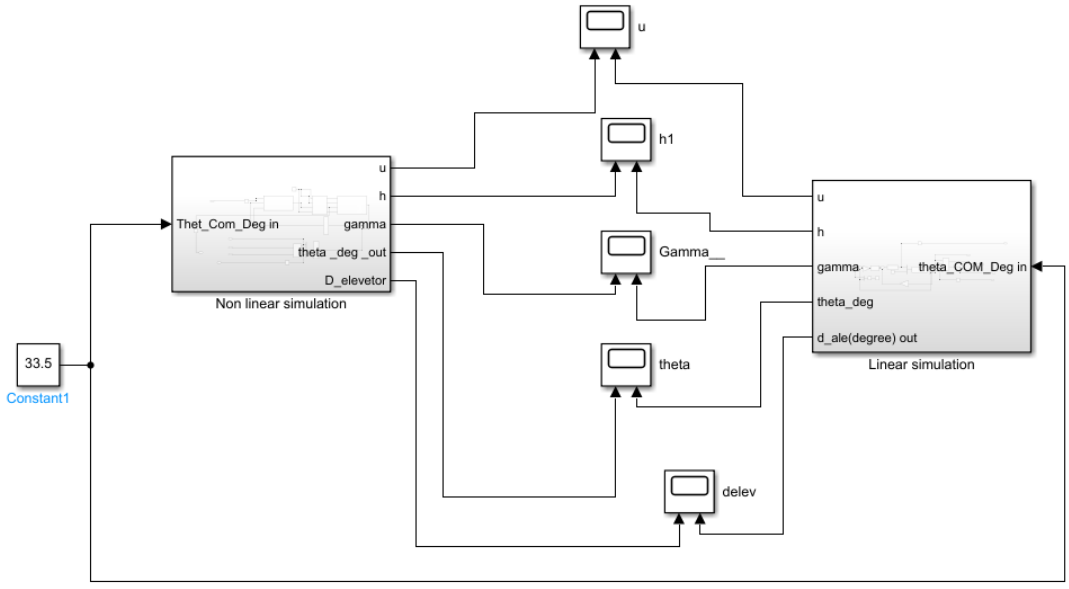
a) Develop the testing loop Containing the “Controller + Simulator”



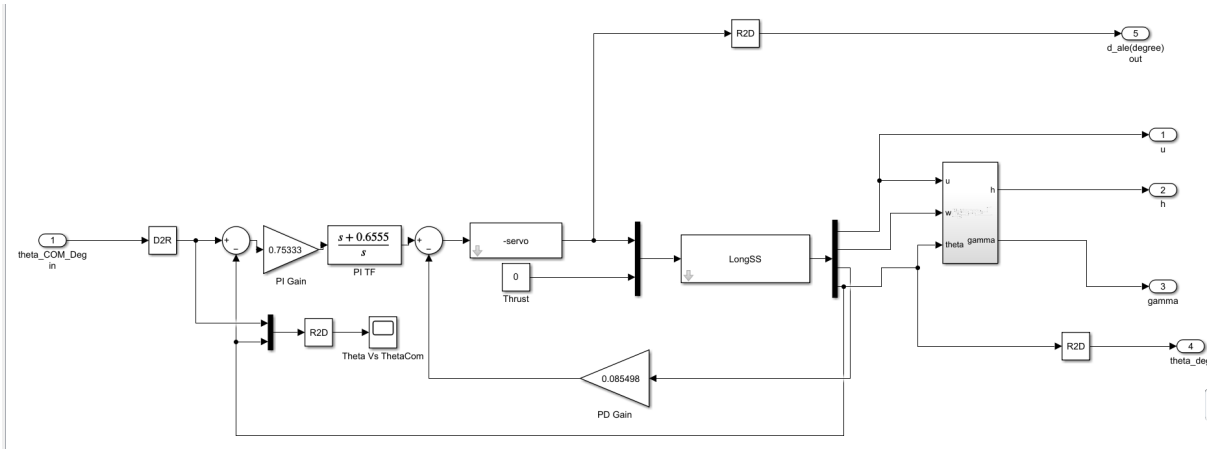




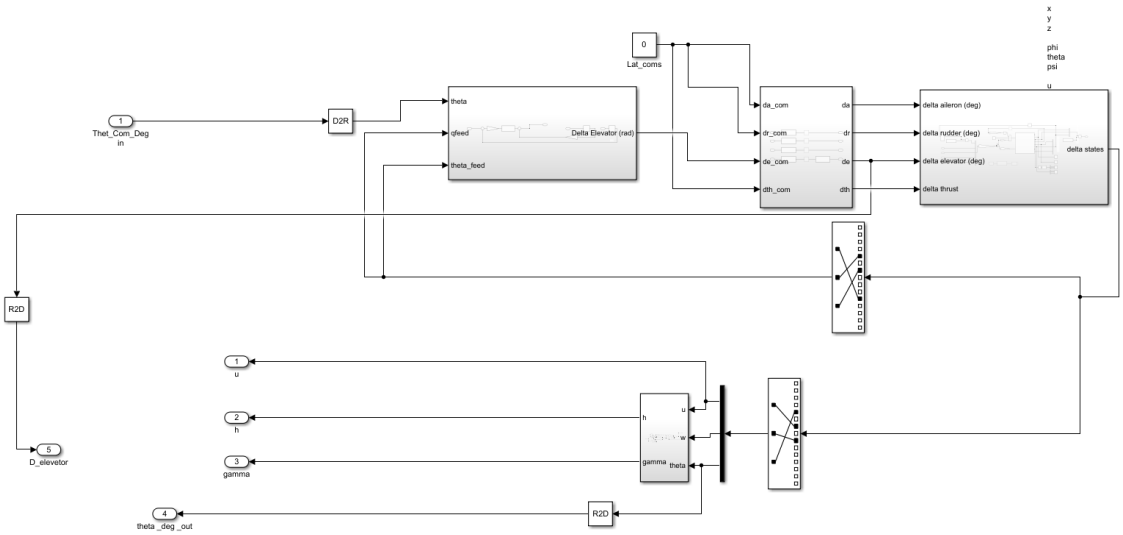
b) Test the “Pitch controller” and compare the response with the same test on the State space model



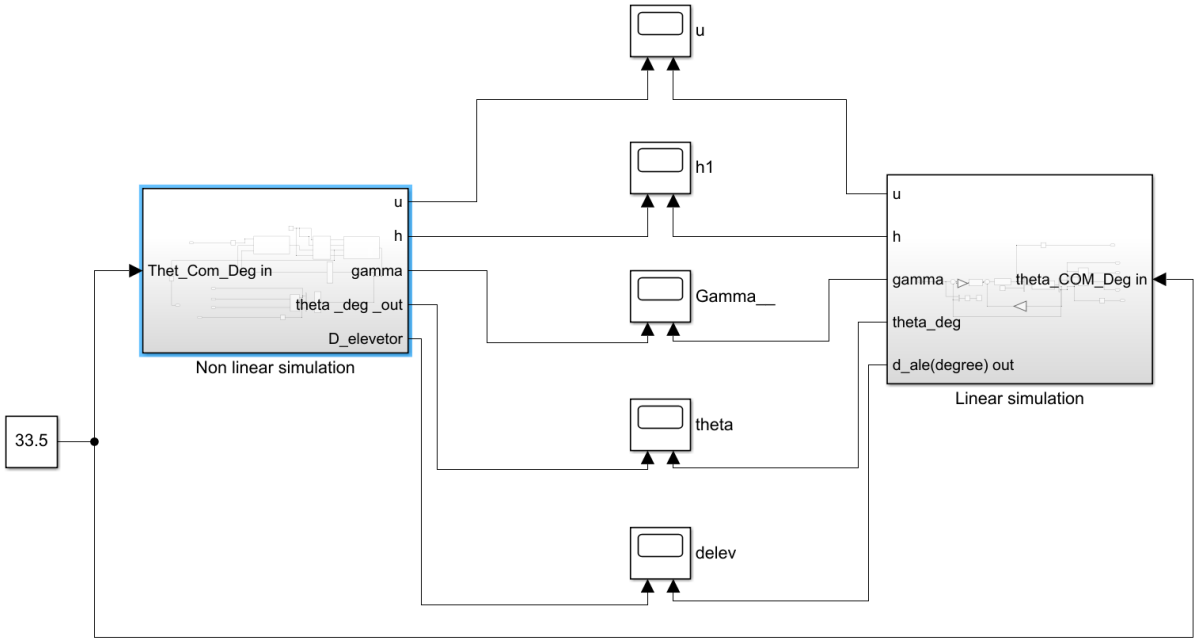
Linear block



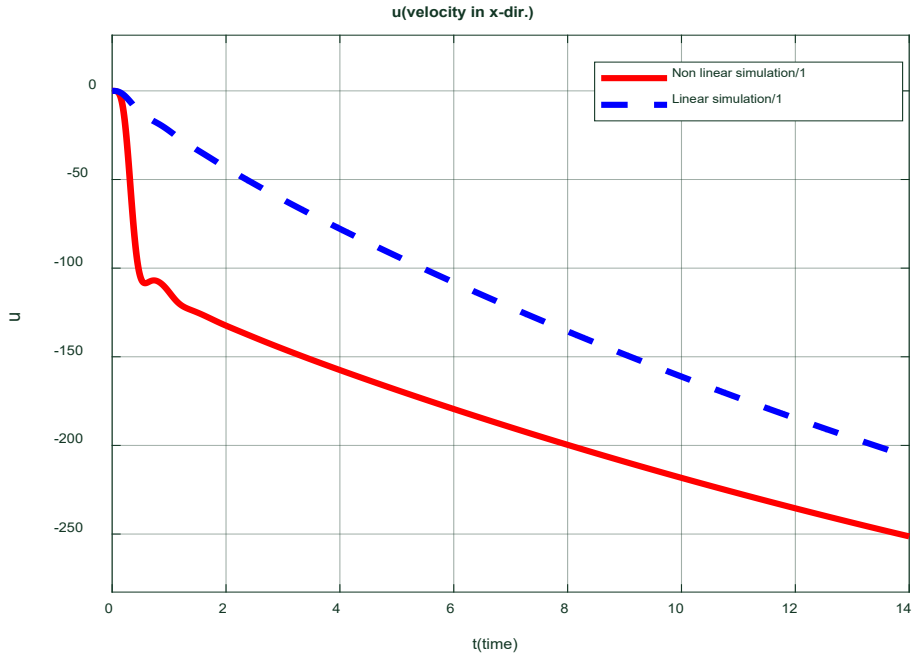
Nonlinear block



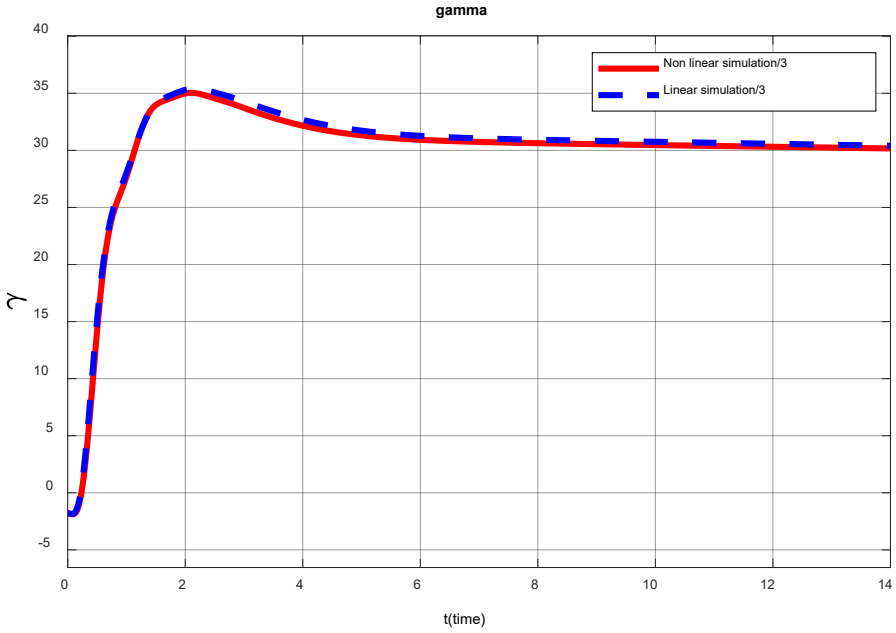
Results for 33.5 deg pitch angle



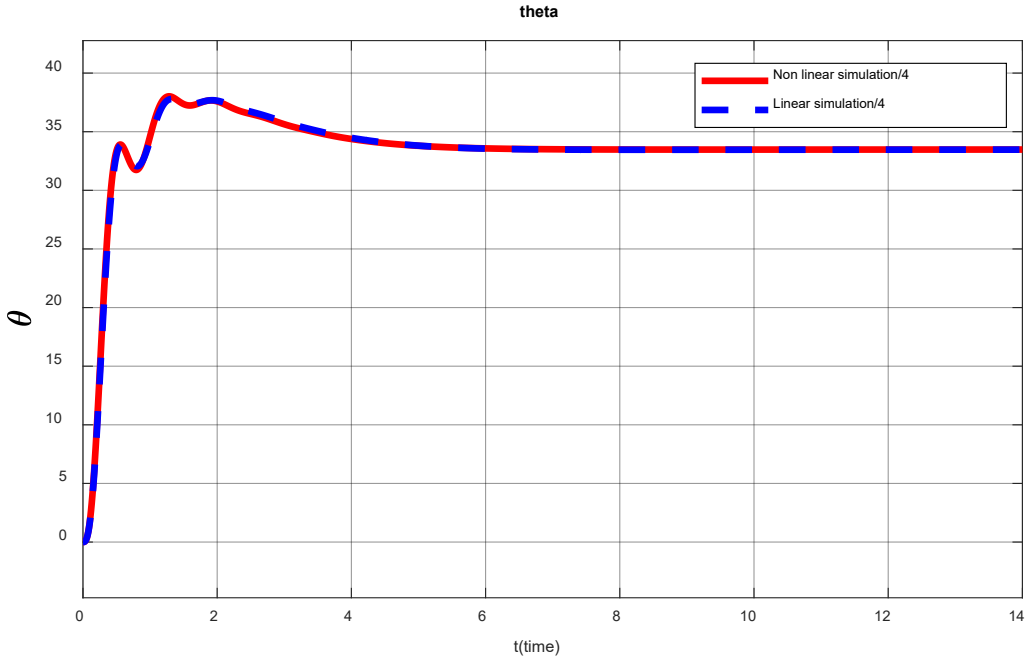
1-u



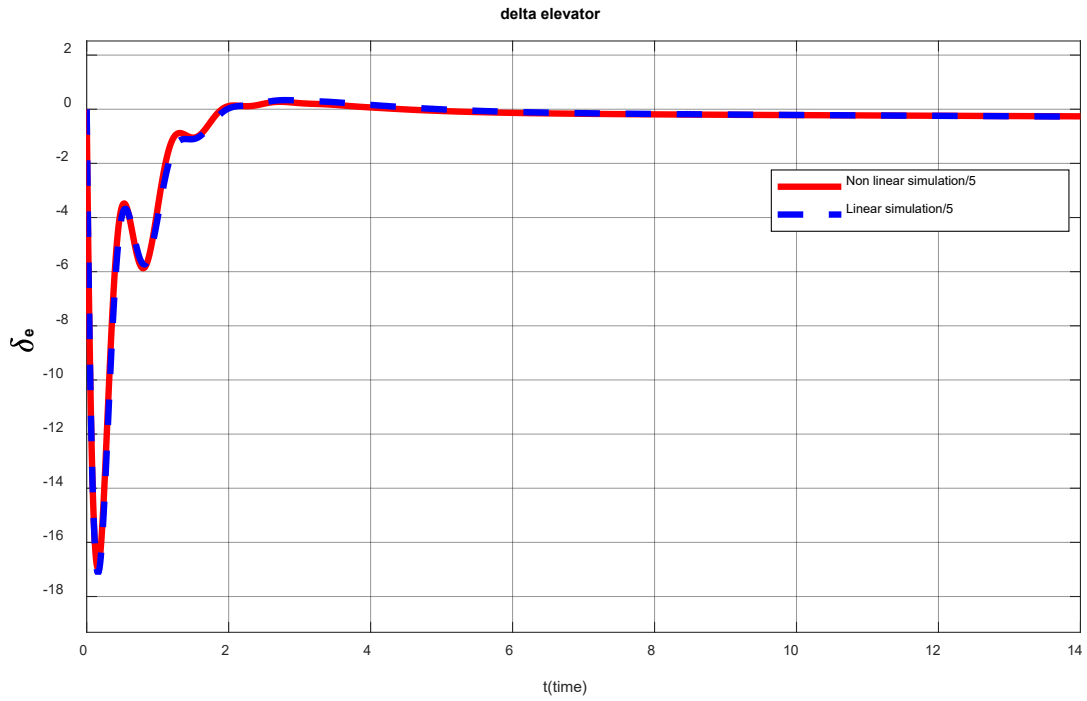
2-γ



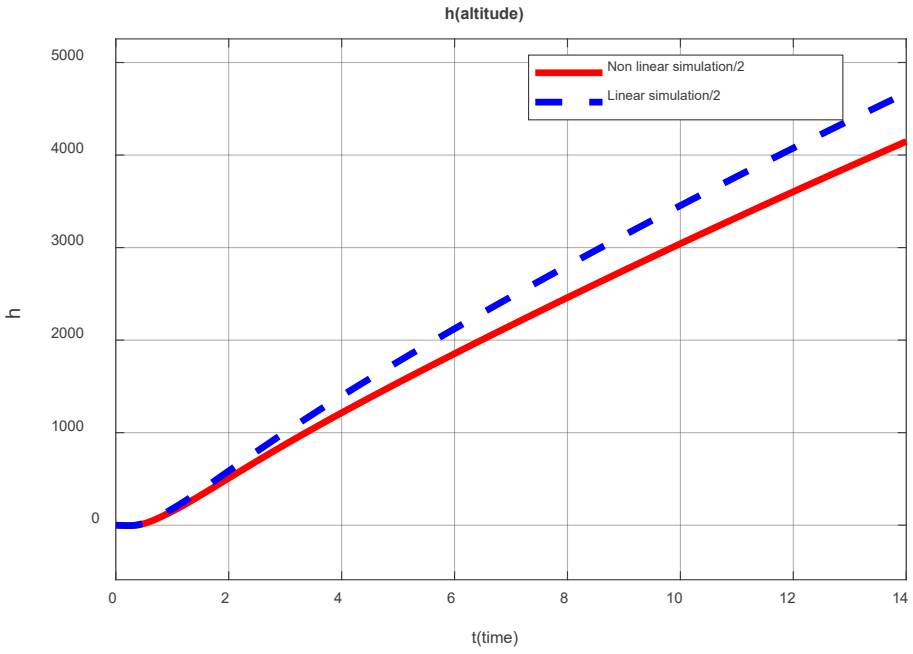
3-θ



4-$\delta_{elevator}$



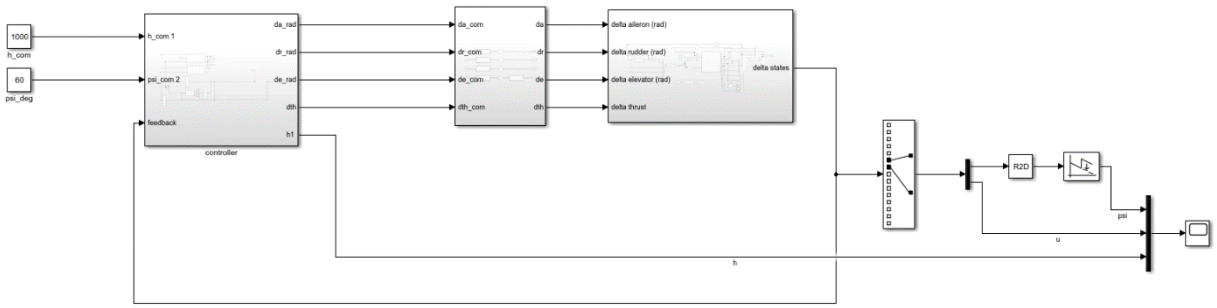
5-h

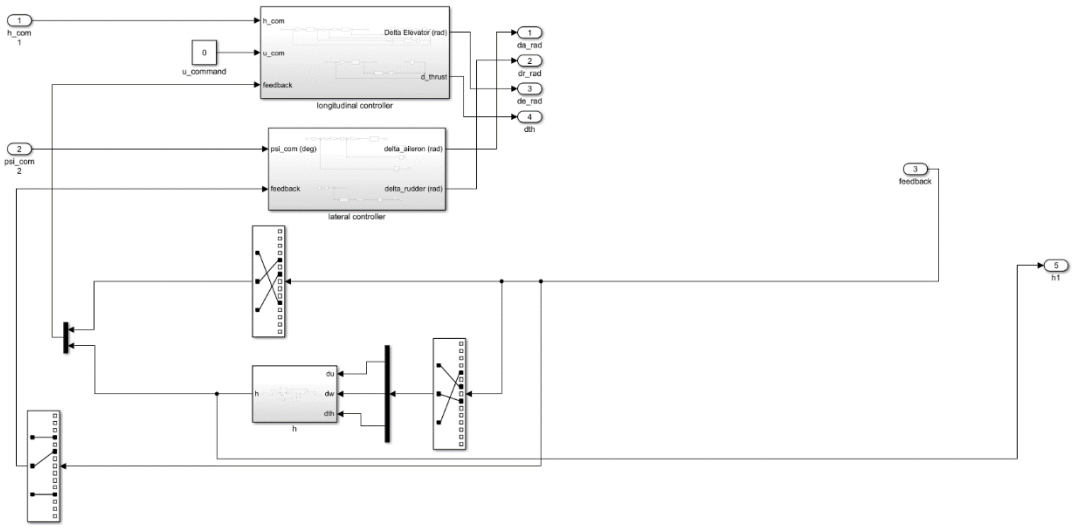


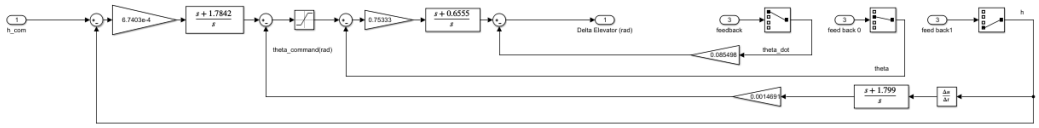
c) Test the "Pitch controller + Velocity controller" and compare the response with the same test on the State space model

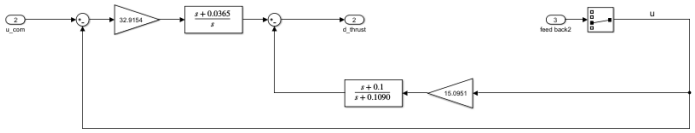
- Give a commanded input signal of pitch angle and observe the response and control action $(\theta, u, \gamma, h, \delta_e . \delta_{th})$.
- Perform the same test on the Linear Longitudinal state space model and observe the response and control action $(\theta, u, \gamma, h, \delta_e . \delta_{th})$
- Plot the results against each other.
- Note: in this test all the control actions are set to zero except the (δ_e , δ_{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.

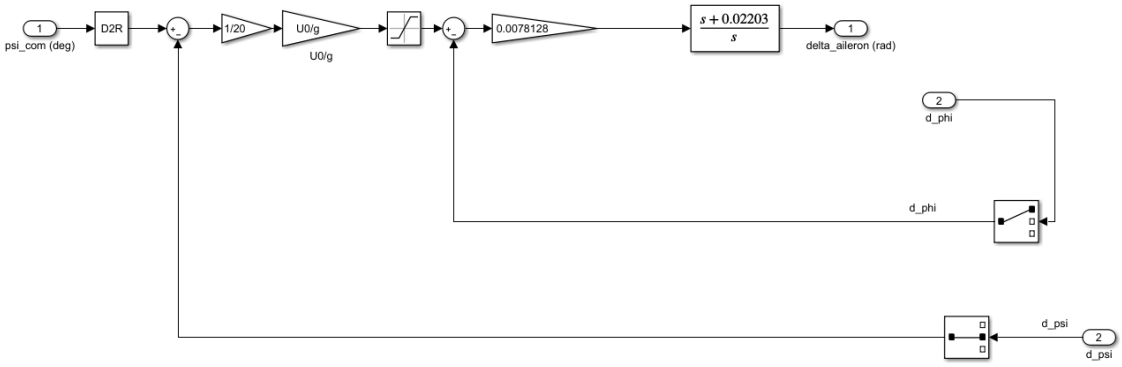
Simulink Non-linear simulator:

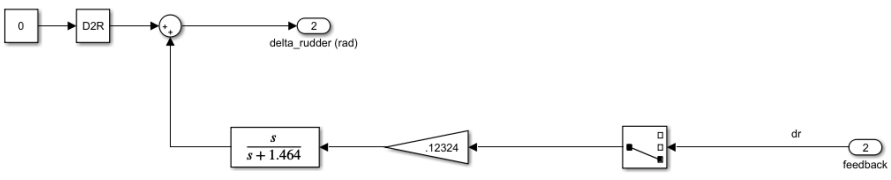



h_com
u_command
feedback
Delta Elevator (rad)
d_thrust
longitudinal controller
da_rad
dr_rad
de_rad
dth
psi_com (deg)
delta_aileron (rad)
delta_rudder (rad)
lateral controller
feedback
h1
du
dw
dth
h

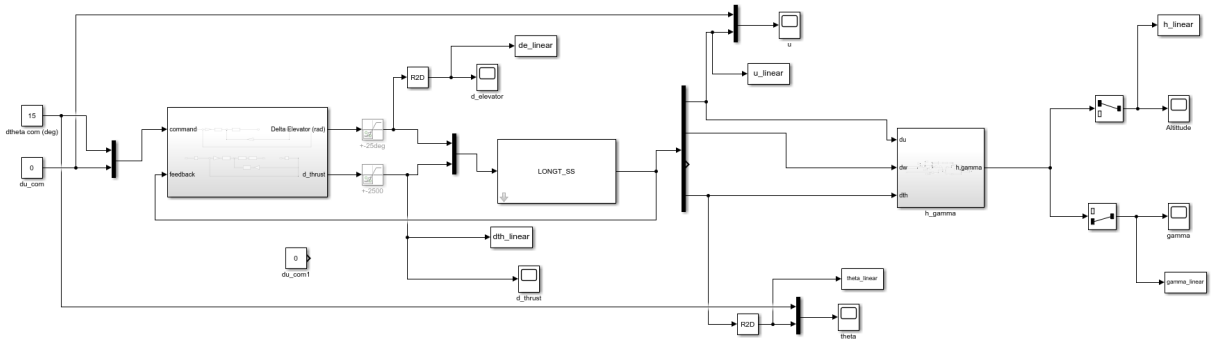

h_com
6.7403e-4
s + 1.7842
s
theta_command(rad)
0.75333
s + 0.6555
Delta Elevator (rad)
feedback
theta_dot
feed back 0
theta
feed back1
h
s + 1.799
0.001469

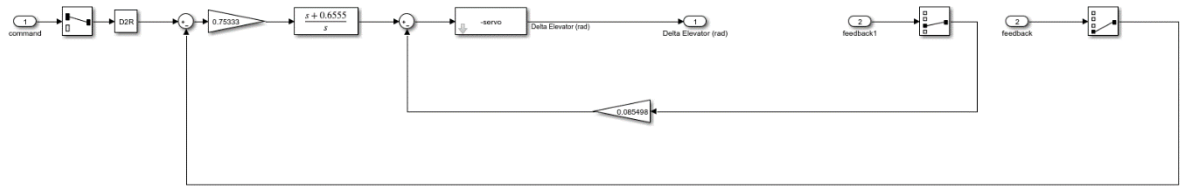

u_com
32.9154
s + 0.0365
d_thrust
feed back2
u
s + 0.1
s + 0.1090
15.0951

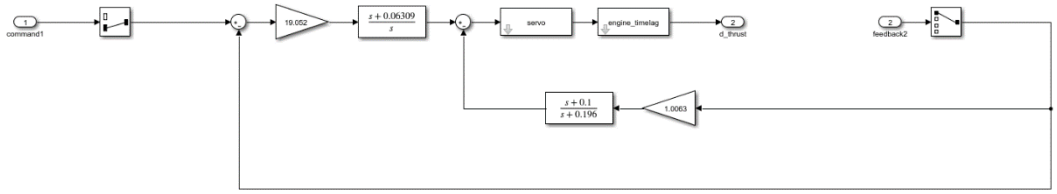

psi_com (deg)
D2R
1/20
U0/g
U0/g
0.0078128
s + 0.02203
s
delta_aileron (rad)
d_phi
d_phi
d_psi
d_psi

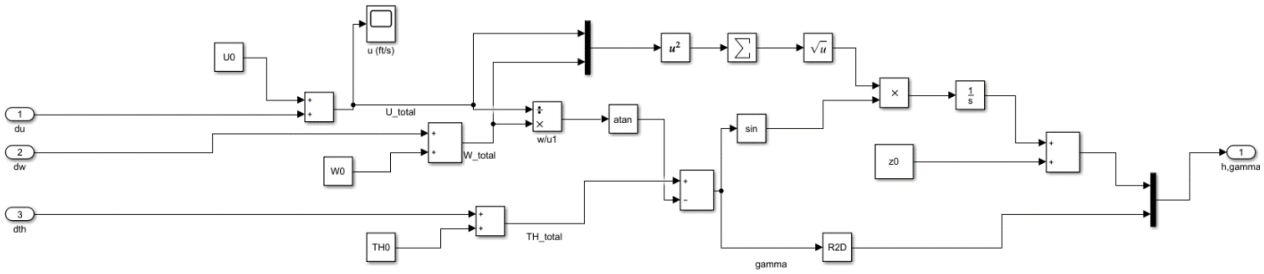

0
D2R
delta_rudder (rad)
s
s + 1.464
.12324
dr
feedback

Simulink linear simulator:

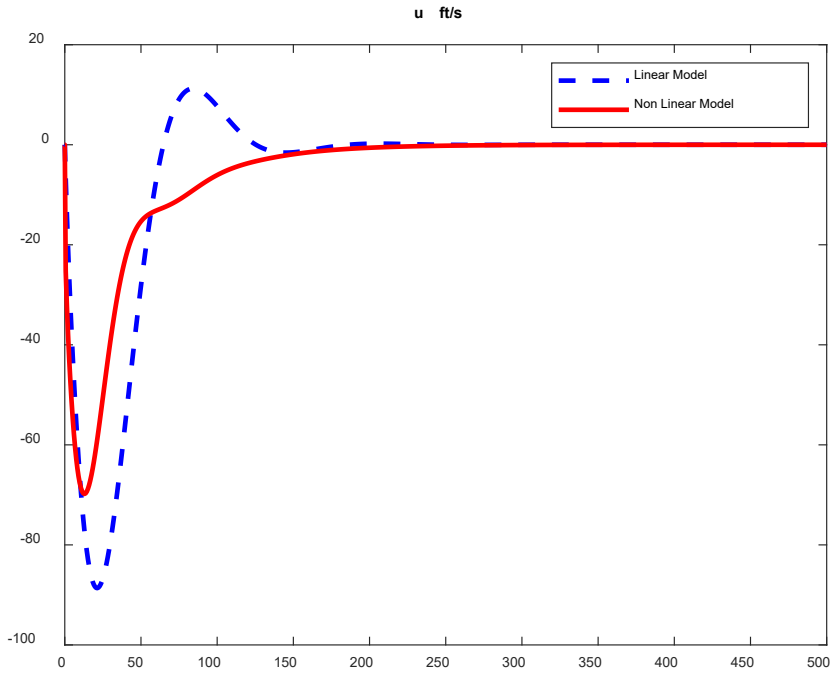


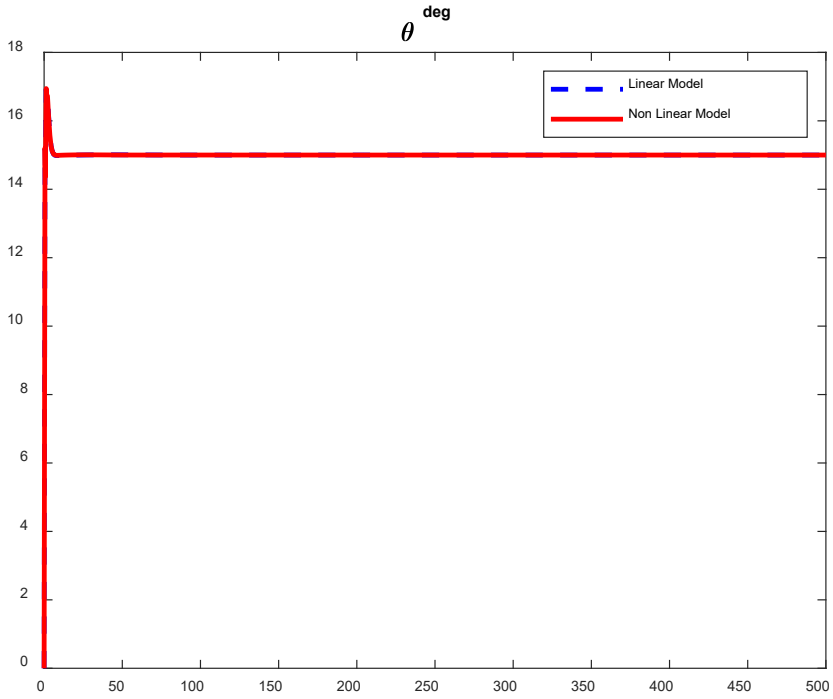


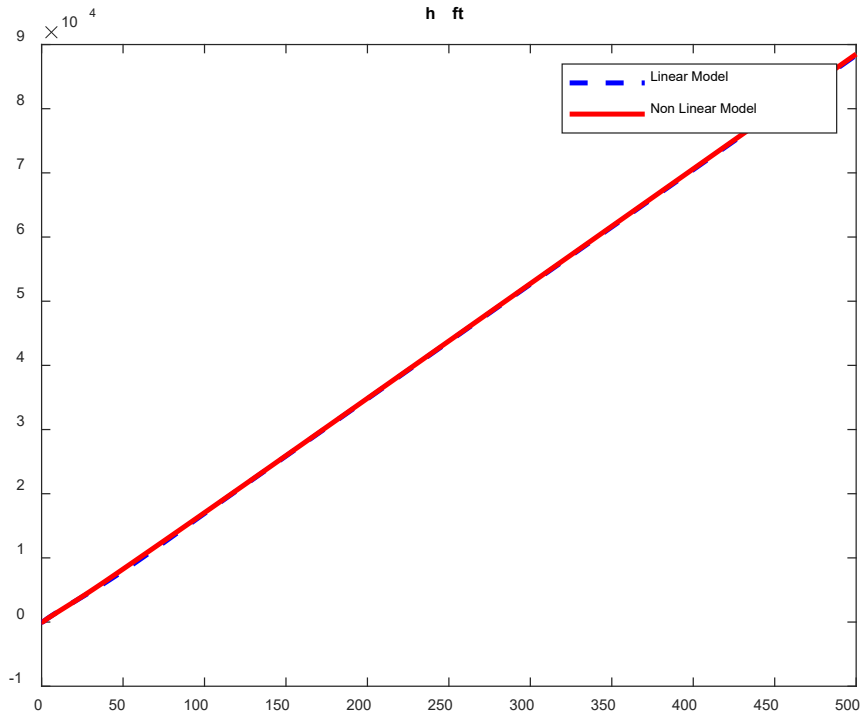


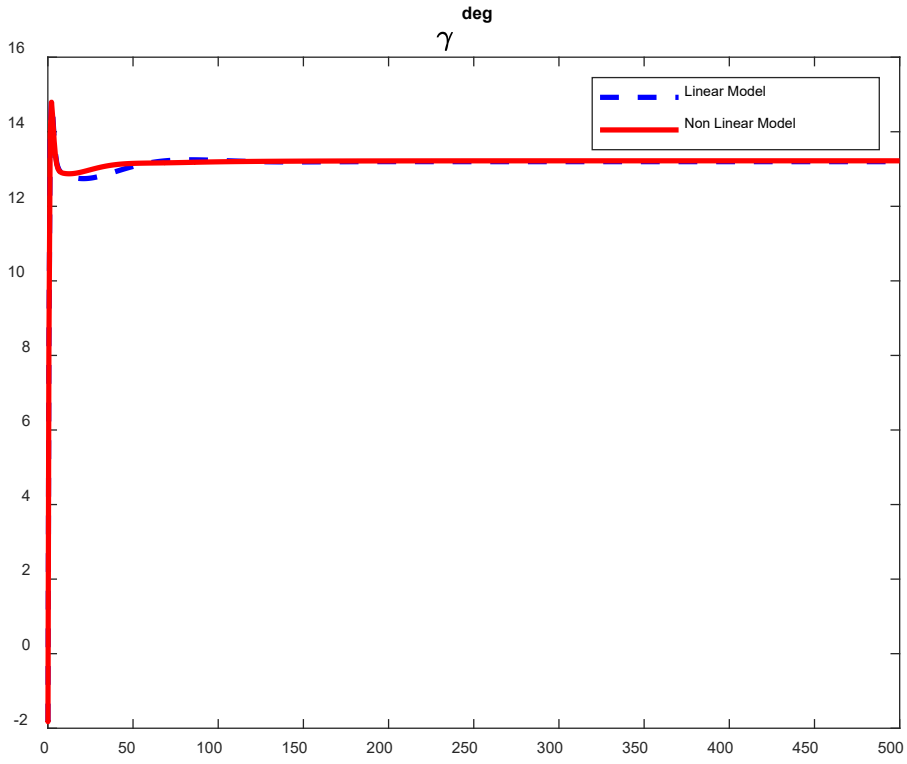


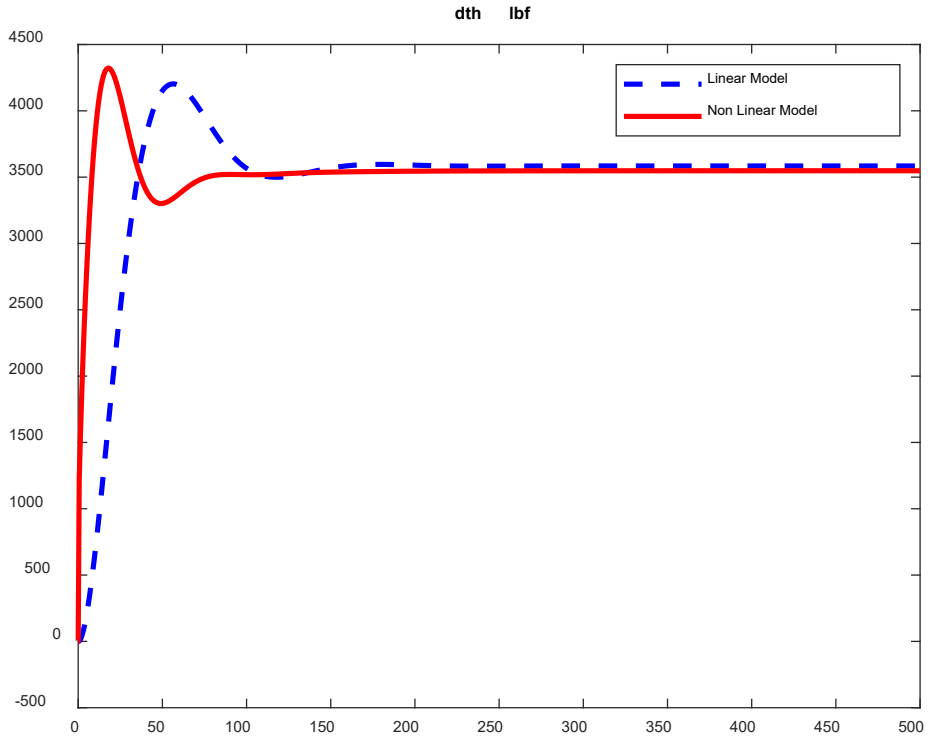
Simulation Results for 15° pitch command:

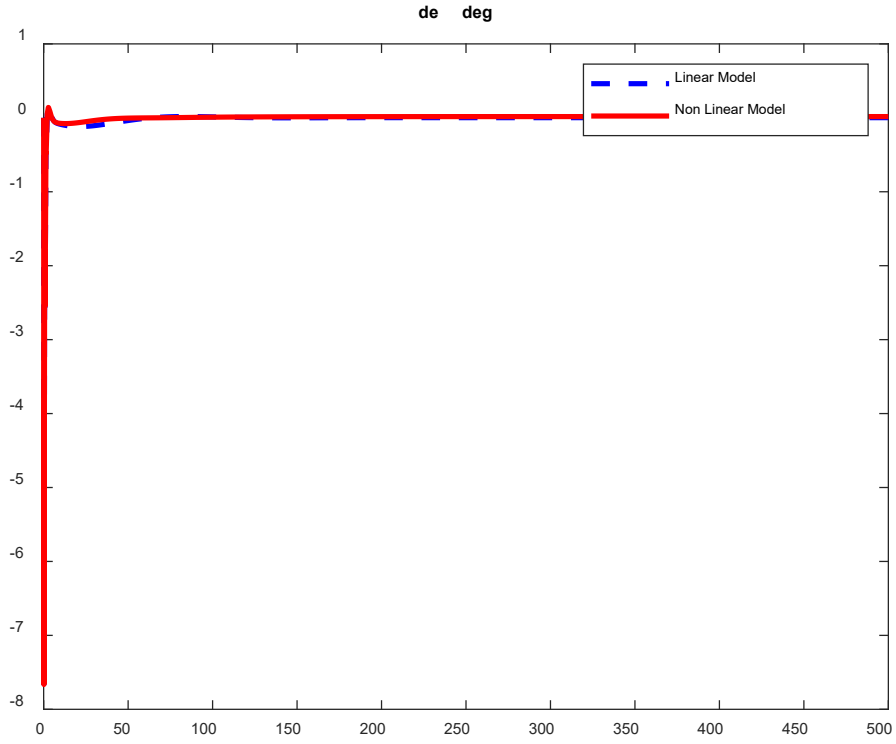





h ft
×10^4
Linear Model
Non Linear Model
9
8
7
6
5
4
3
2
1
0
-1
0
50
100
150
200
250
300
350
400
450
500

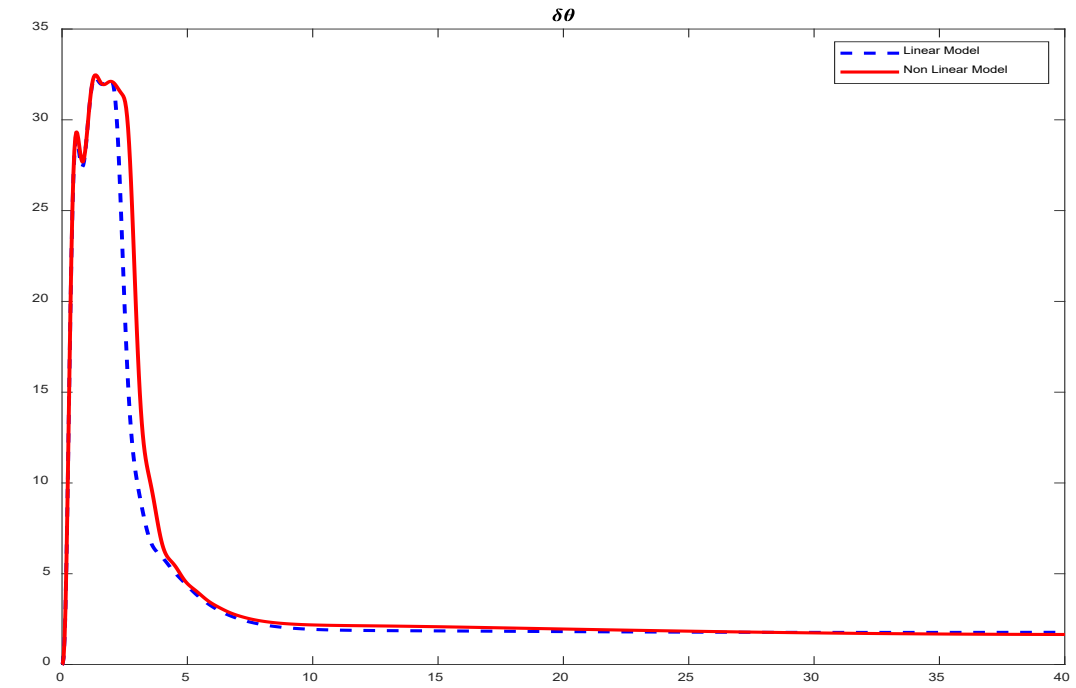

deg
γ
Linear Model
Non Linear Model
16
14
12
10
8
6
4
2
0
-2
0
50
100
150
200
250
300
350
400
450
500

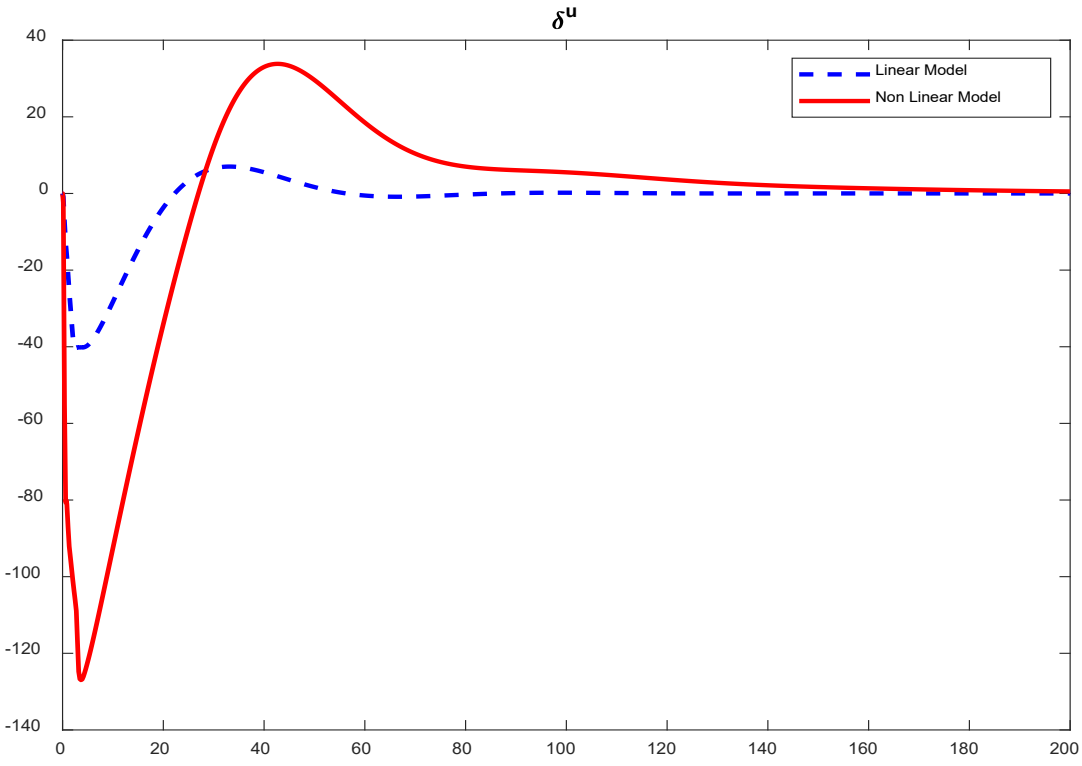

dth lbf
Linear Model
Non Linear Model
4500
4000
3500
3000
2500
2000
1500
1000
500
0
-500
0
50
100
150
200
250
300
350
400
450
500

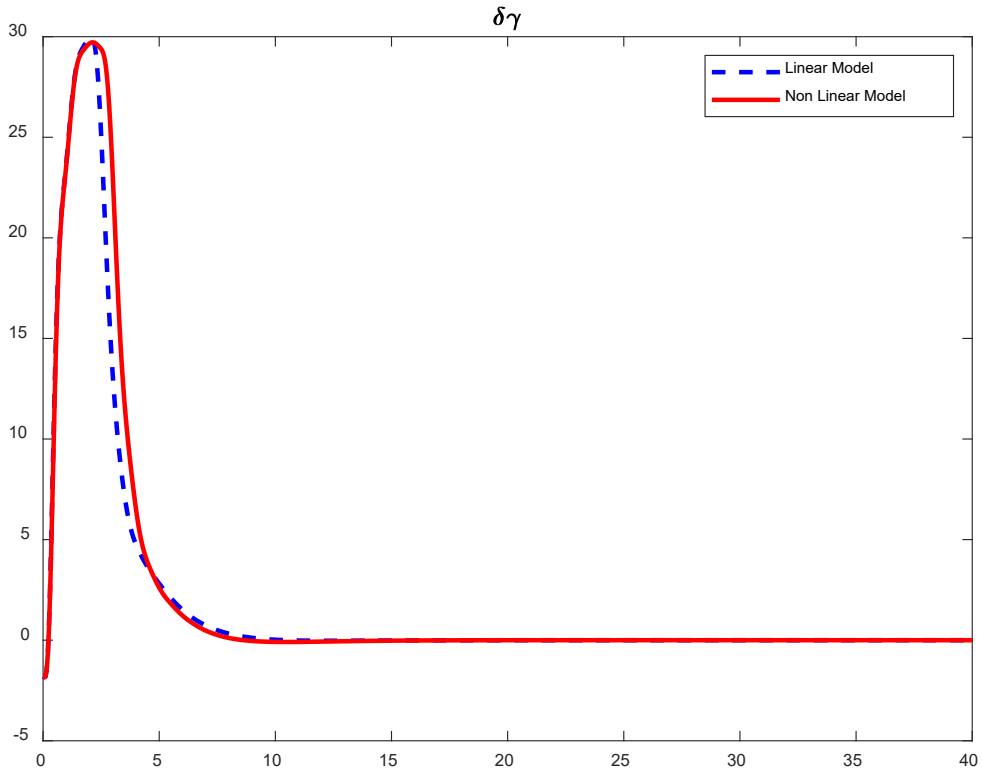

de deg
Linear Model
Non Linear Model
1
0
-1
-2
-3
-4
-5
-6
-7
-8
0
50
100
150
200
250
300
350
400
450
500

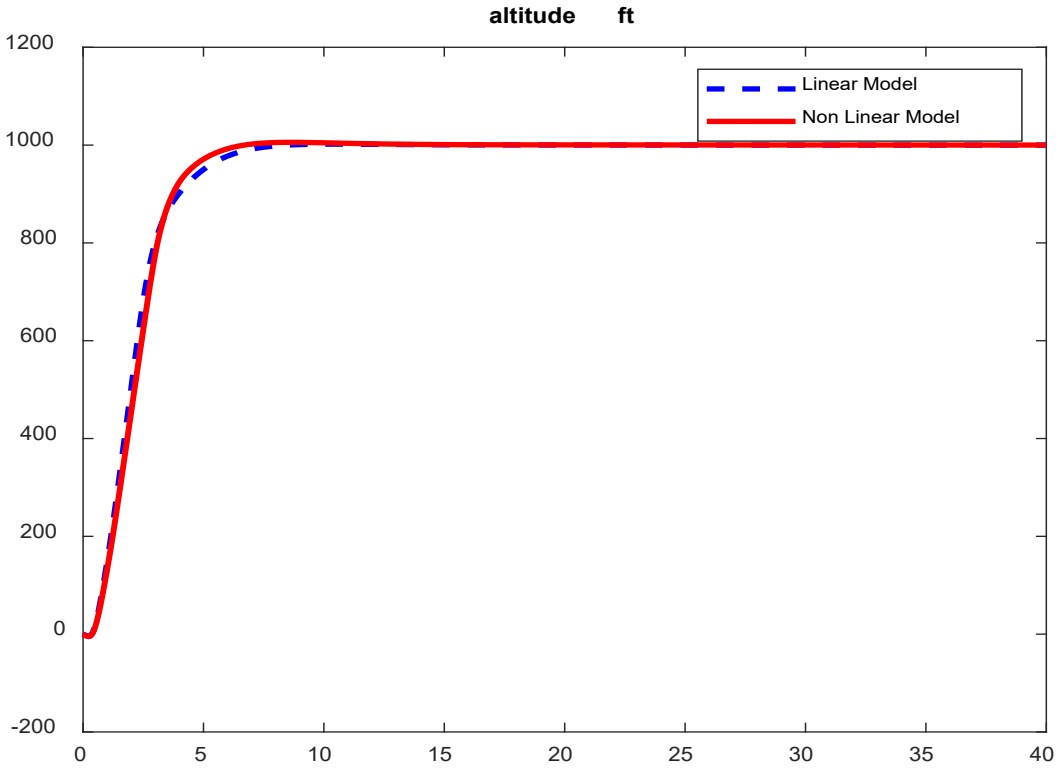
d) Test The "Altitude Hold" Controller and Compare the response with the same test on the state space model

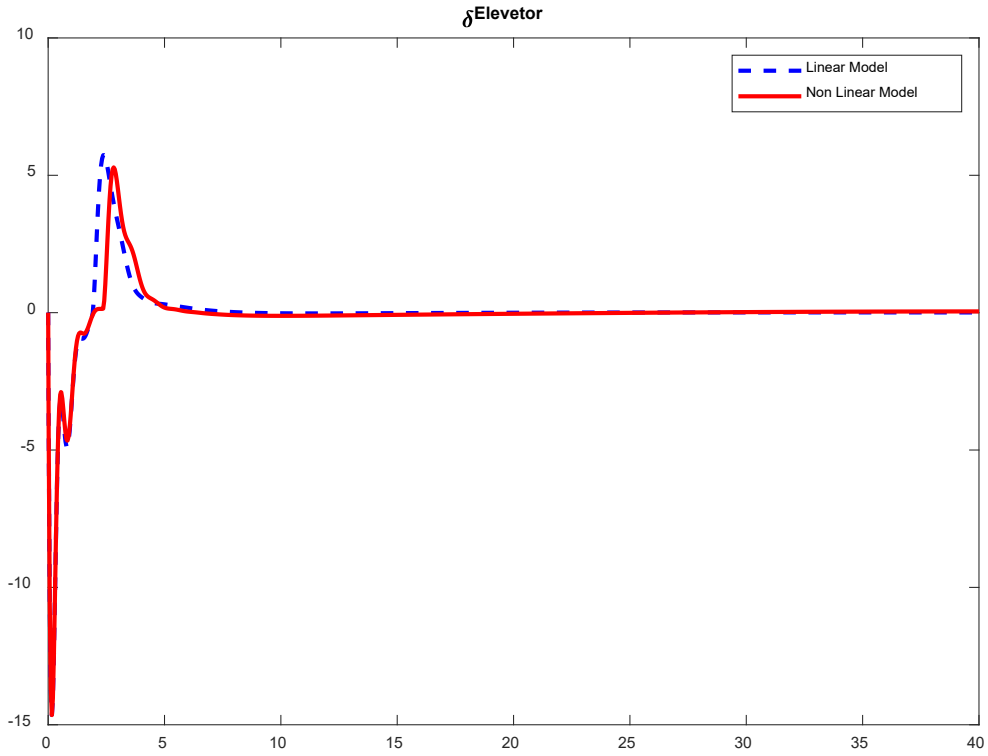
Results for input command of 1000ft.

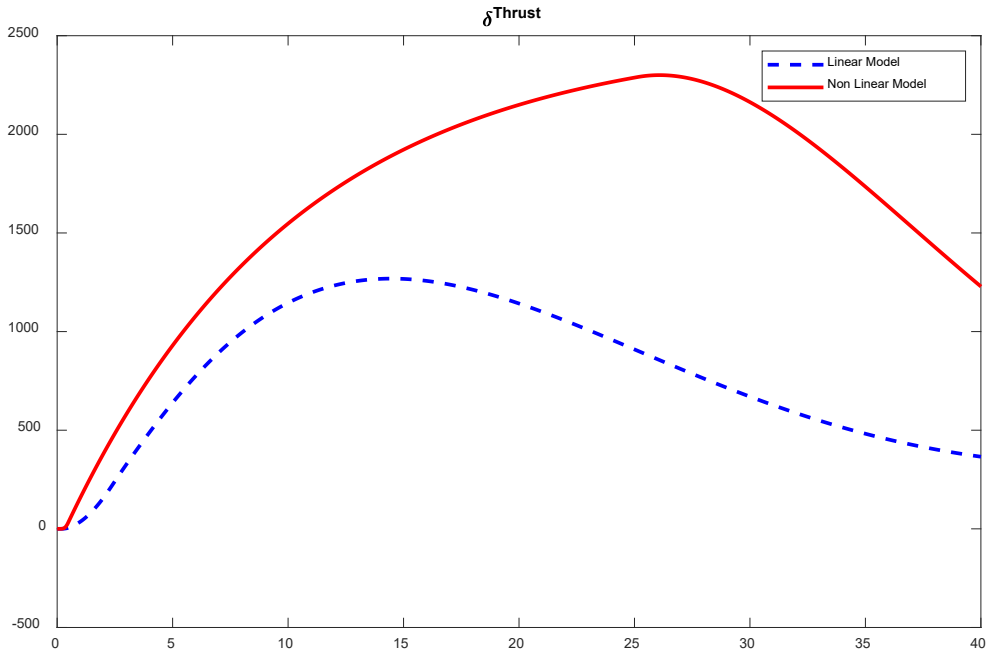




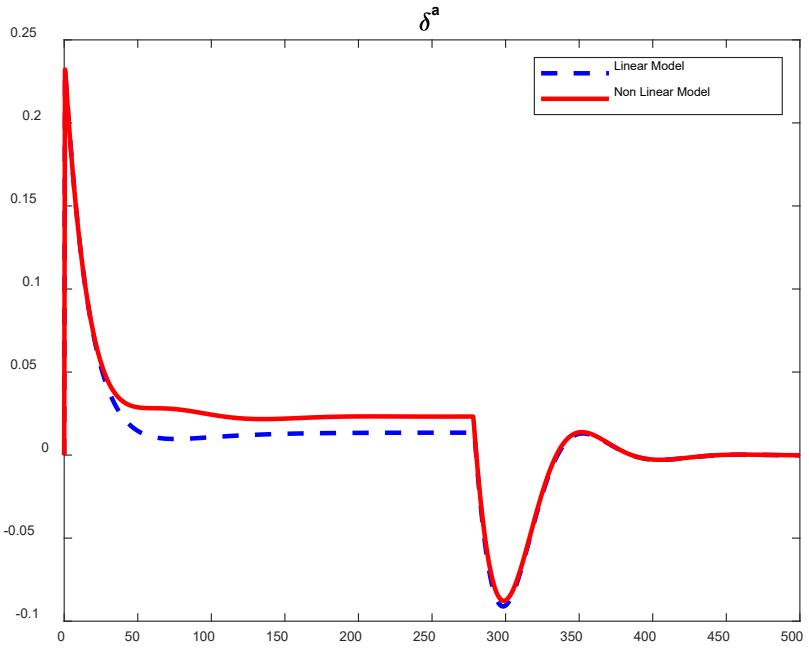

$\delta\gamma$
Linear Model
Non Linear Model
30
25
20
15
10
5
0
-5
0
5
10
15
20
25
30
35
40

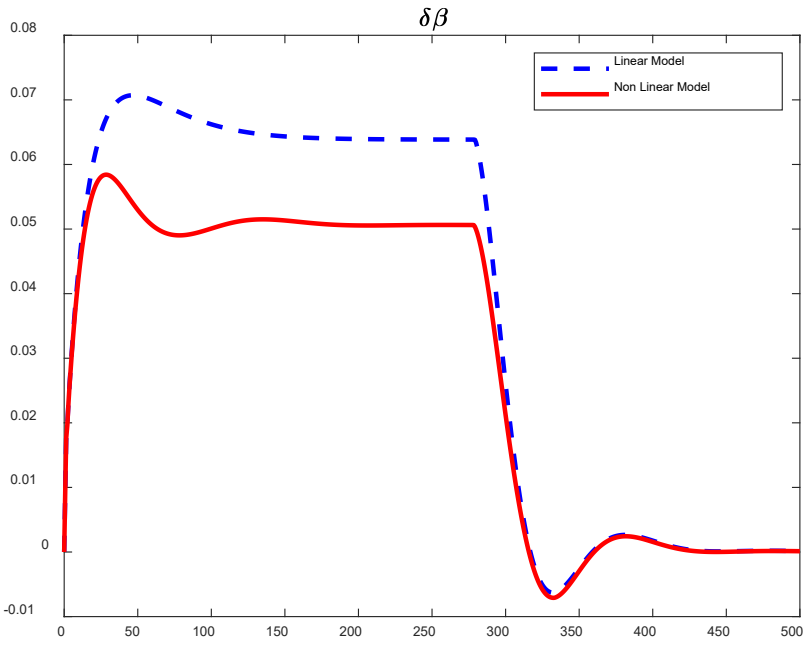

altitude ft
Linear Model
Non Linear Model
1200
1000
800
600
400
200
0
-200
0
5
10
15
20
25
30
35
40

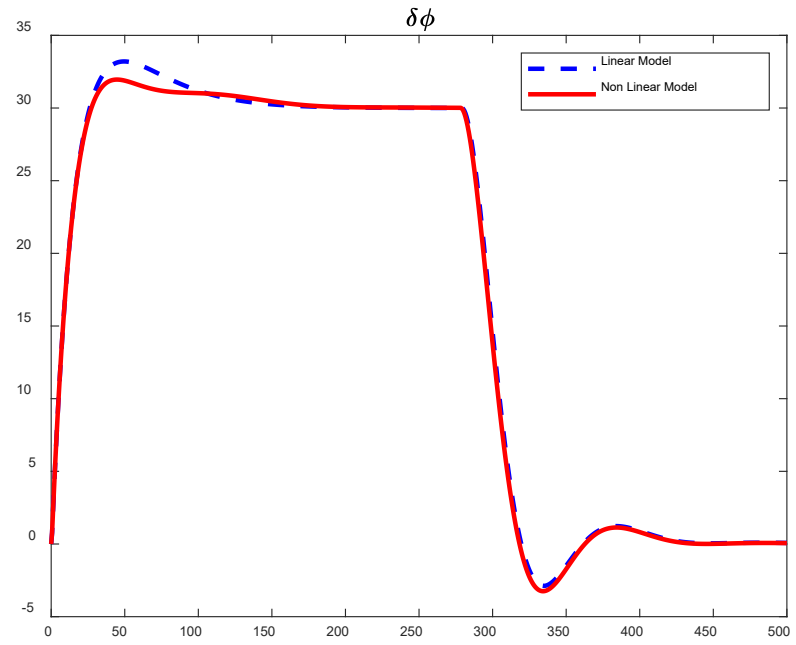

δElevetor
Linear Model
Non Linear Model
10
5
0
-5
-10
-15
0
5
10
15
20
25
30
35
40

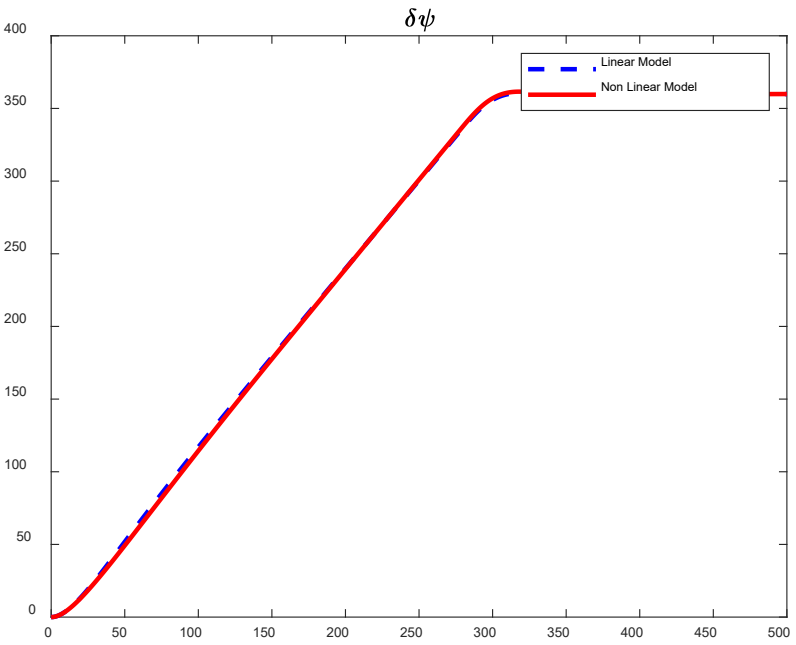

δThrust
Linear Model
Non Linear Model
2500
2000
1500
1000
500
0
-500
0
5
10
15
20
25
30
35
40

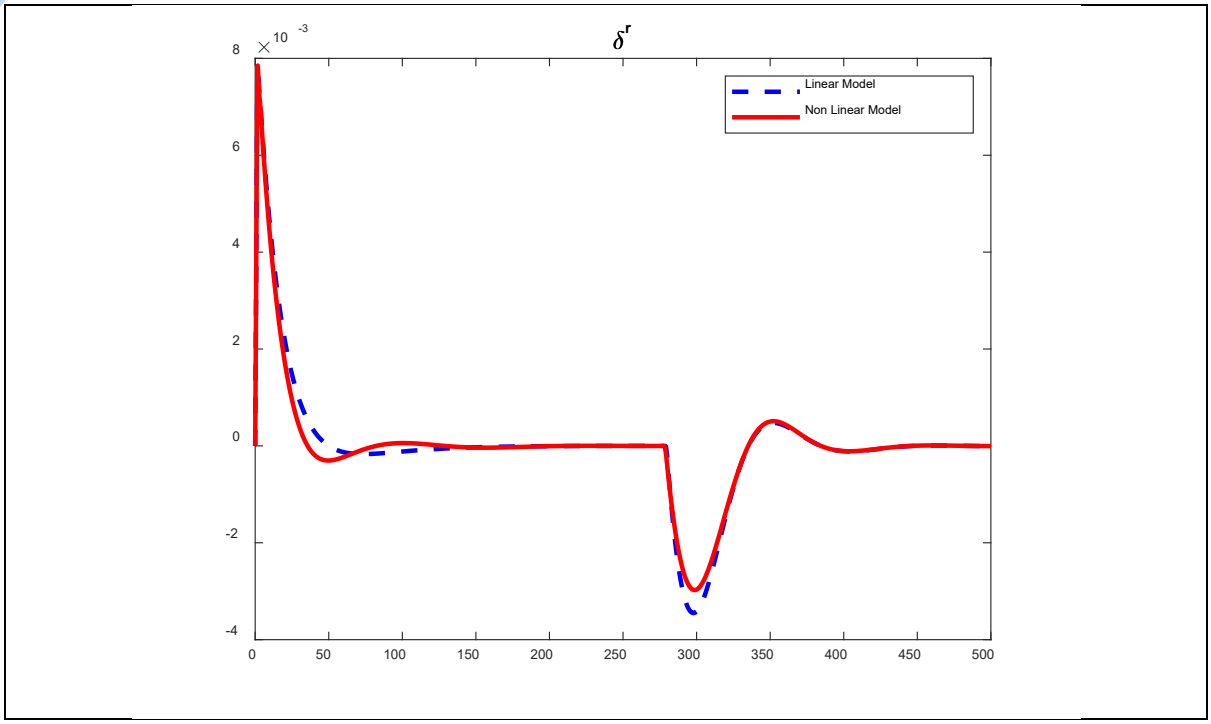
e) Perform a complete autonomous mission including “climb, cruise, turn, descent”



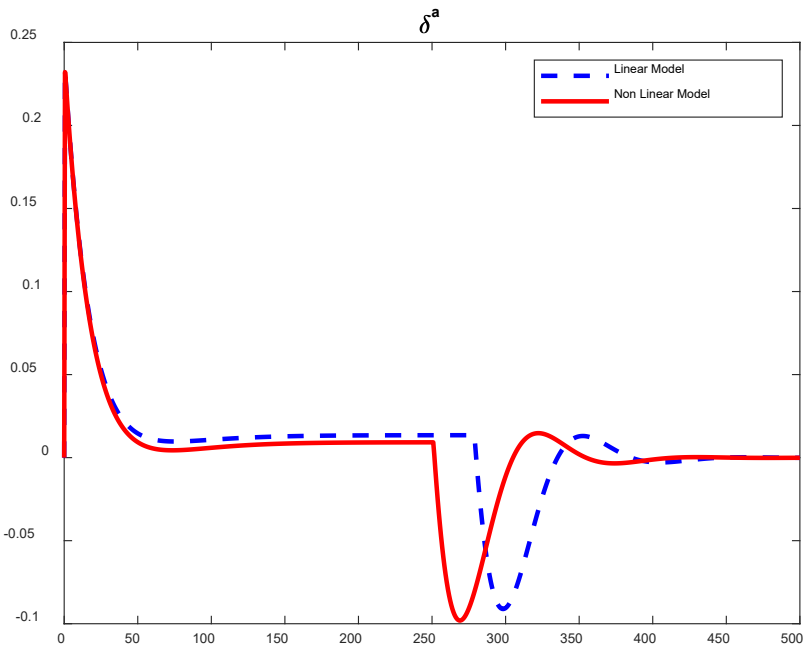


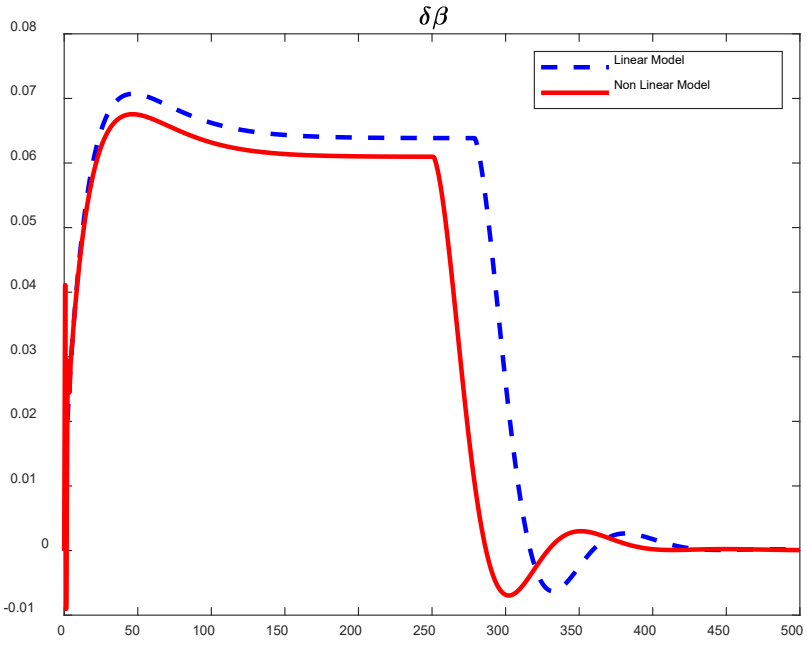

$\delta\phi$
Linear Model
Non Linear Model
35
30
25
20
15
10
5
0
-5
0
50
100
150
200
250
300
350
400
450
500

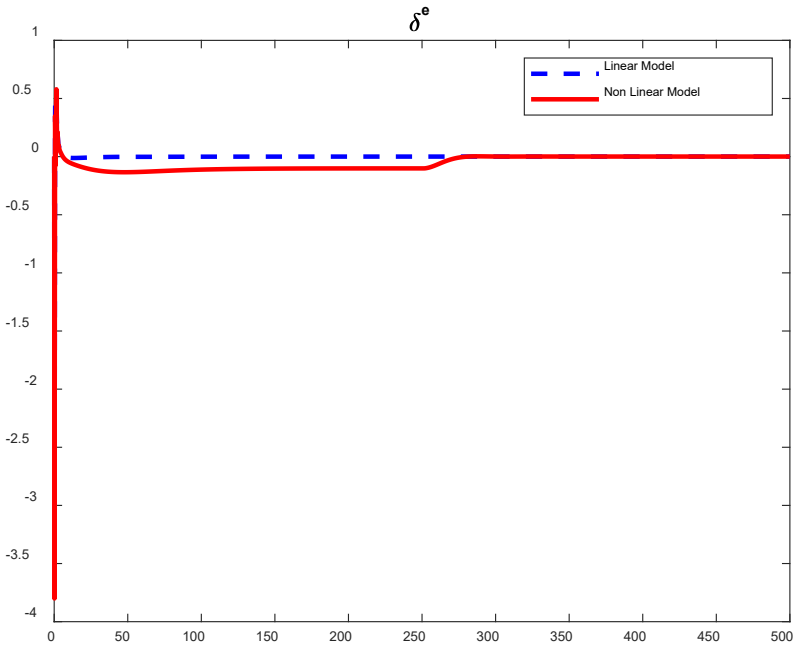

$\delta\psi$
Linear Model
Non Linear Model
400
350
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
400
450
500

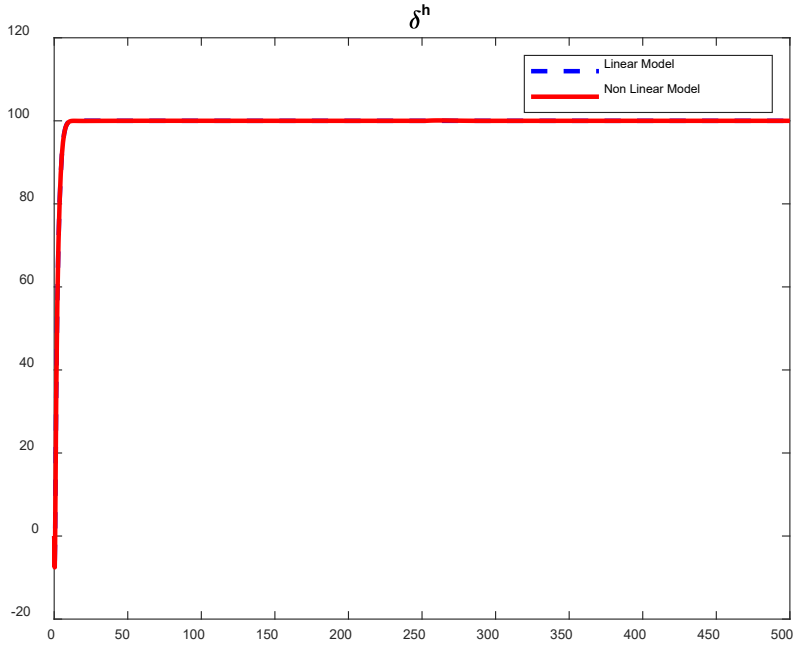

δ^r
$\times 10^{-3}$
Linear Model
Non Linear Model
8
6
4
2
0
-2
-4
0
50
100
150
200
250
300
350
400
450
500

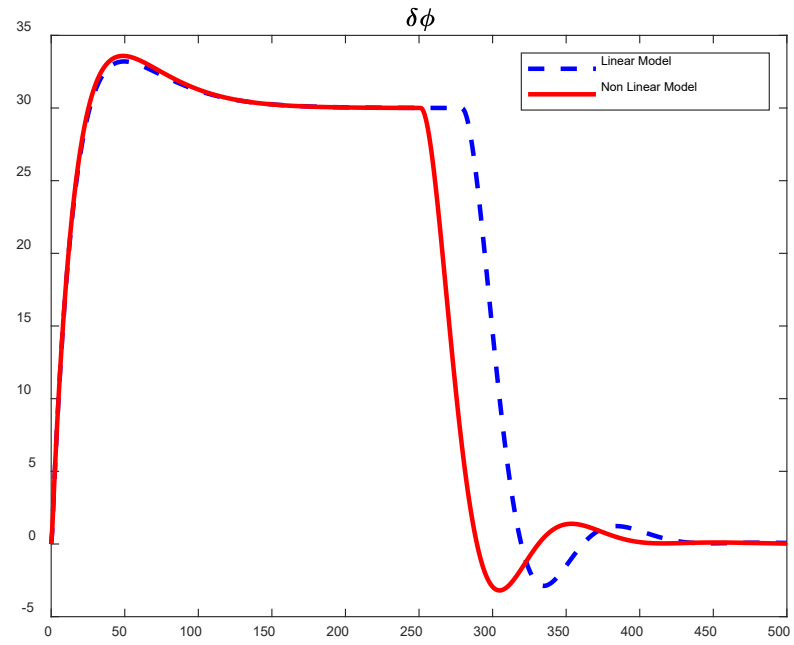
f) Test the "Lateral controller + Altitude Hold controller" and compare the response with the same test on the State space model

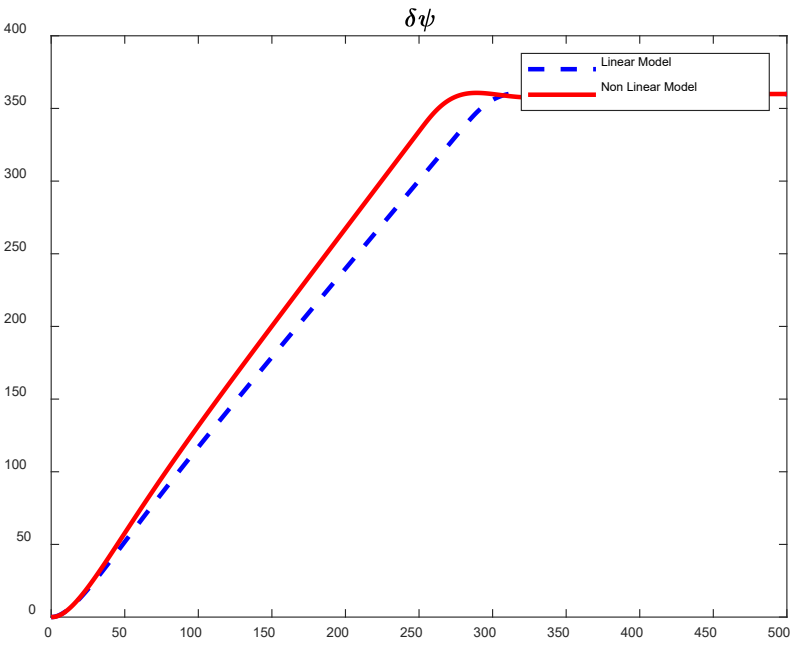


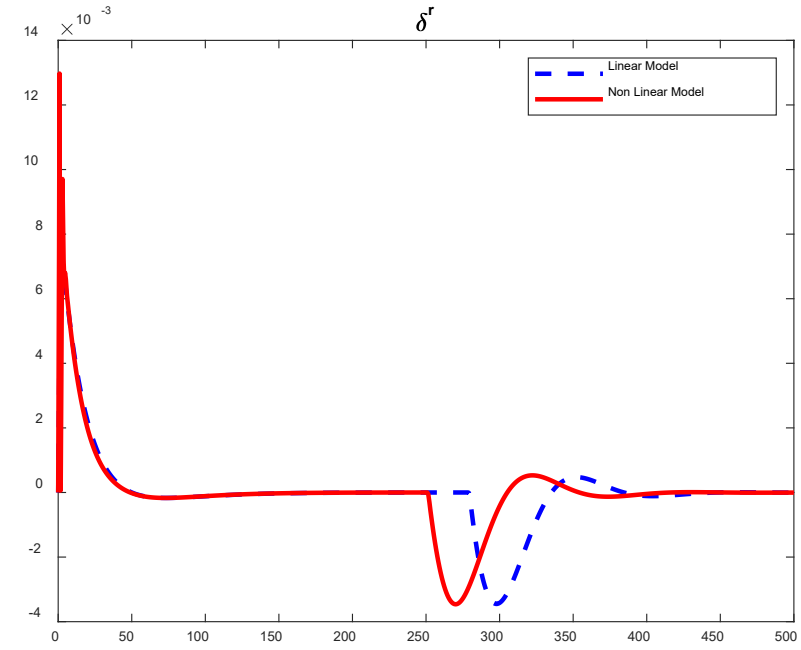


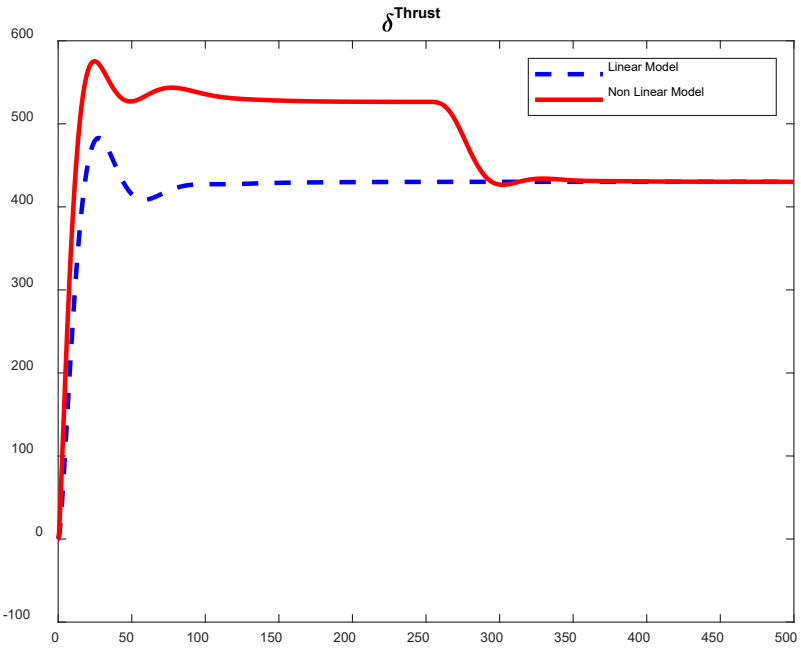

δ^e
Linear Model
Non Linear Model
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
0
50
100
150
200
250
300
350
400
450
500


δ^h
Linear Model
Non Linear Model
120
100
80
60
40
20
0
-20
0
50
100
150
200
250
300
350
400
450
500

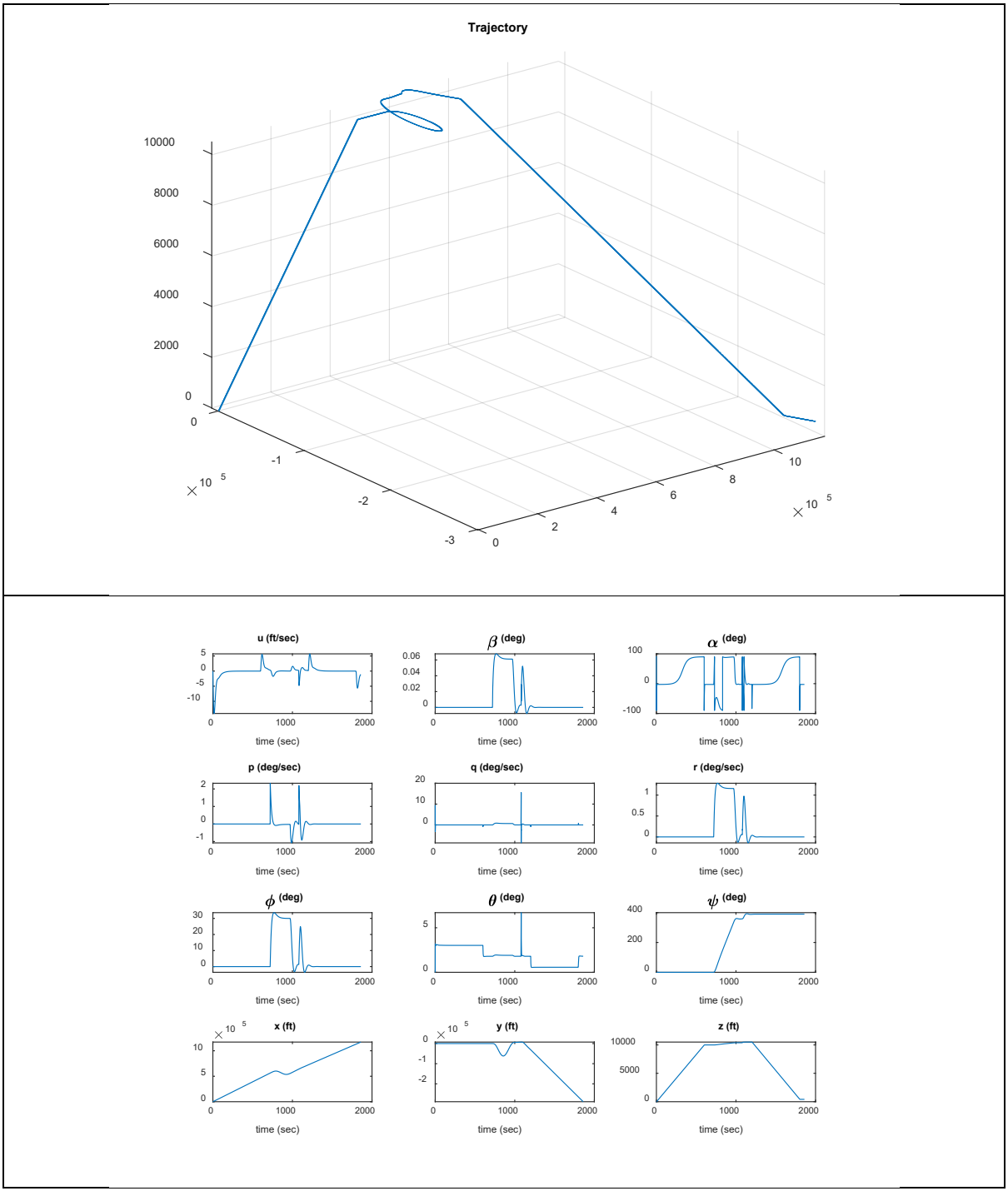

$\delta\phi$
Linear Model
Non Linear Model
35
30
25
20
15
10
5
0
-5
0
50
100
150
200
250
300
350
400
450
500

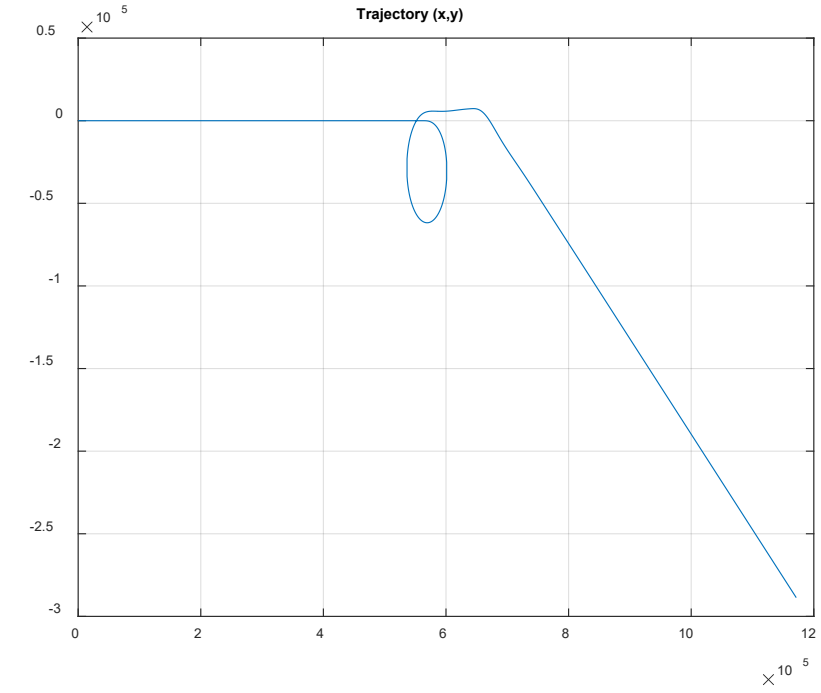

$\delta\psi$
Linear Model
Non Linear Model
400
350
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
400
450
500

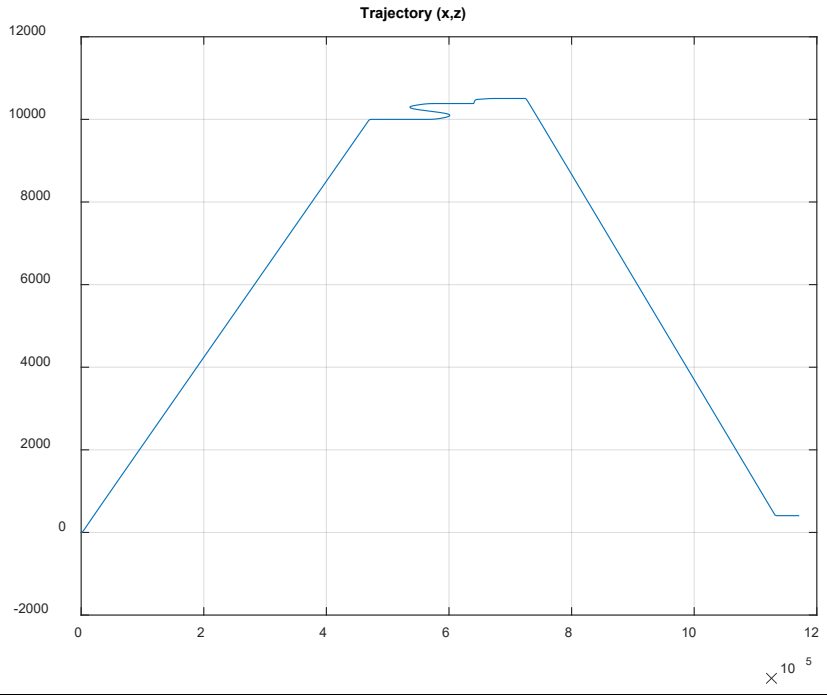

δ^r
$\times 10^{-3}$
Linear Model
Non Linear Model
14
12
10
8
6
4
2
0
-2
-4
0
50
100
150
200
250
300
350
400
450
500

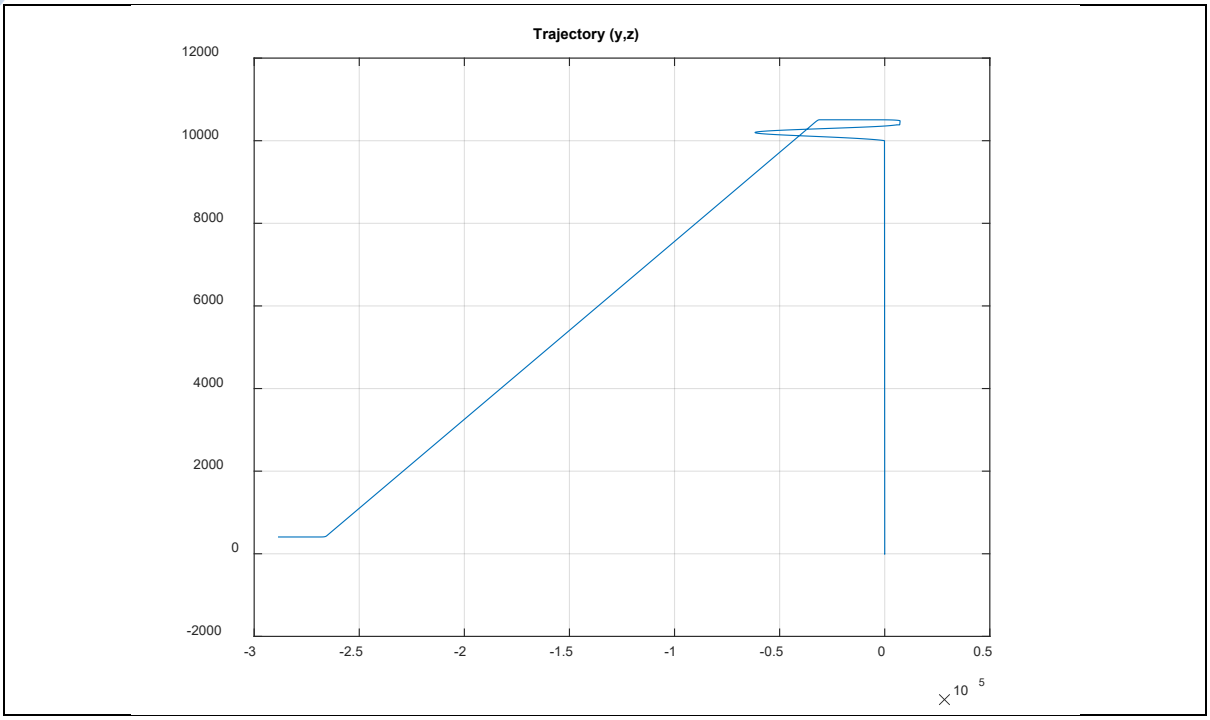

δ^{Thrust}
Linear Model
Non Linear Model
600
500
400
300
200
100
0
-100
0
50
100
150
200
250
300
350
400
450
500

g) Perform a complete autonomous mission including “climb, cruise, turn, descent”




Trajectory (x,y)
×10^5
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
0
2
4
6
8
10
12
×10^5


Trajectory (x,z)
12000
10000
8000
6000
4000
2000
0
-2000
0
2
4
6
8
10
12
×10^5


Trajectory (y,z)
12000
10000
8000
6000
4000
2000
0
-2000
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5
×10 5

Appendix: MATLAB code

Task (1)

```
clc
clear All
close All

t1=0;
t2=20;
n=100;
h=(t2-t1)/n;
t=(linspace(t1,t2,n+1))';
y1=zeros(length(t),1);
y2=zeros(length(t),1);
y2(1)=1;
y1(1)=-1;


for n=1:length(t)-1
k1=h*(cos(t(n))+sin(y2(n)));
q1=h*(sin(t(n))+cos(y1(n))+sin(y2(n)));

k2=h*(cos(t(n)+0.5*h)+sin(y2(n)+0.5*k1));
q2=h*(sin(t(n)+0.5*h)+cos(y1(n)+0.5*q1)+sin(y2(n)+0.5*k1));

k3=h*(cos(t(n)+0.5*h)+sin(y2(n)+0.5*k2));
q3=h*(sin(t(n)+0.5*h)+cos(y1(n)+0.5*q2)+sin(y2(n)+0.5*k2));

k4=h*(cos(t(n)+h)+sin(y2(n)+k3));
q4=h*(sin(t(n)+h)+cos(y1(n)+q3)+sin(y2(n)+k3));

y2(n+1)=y2(n)+(1/6)*(k1+2*k2+2*k3+k4);
y1(n+1)=y1(n)+(1/6)*(q1+2*q2+2*q3+q4);
end


figure
hold on
plot(t,y1,'r')
plot(t,y2,'b')
legend({'y(1)','y(2)'},...
'Location','southeast','FontSize',8,'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
%% Check using ode45
[tv,Yv]=ode45(@sys_fun,[0 20],[-1 1]);
plot(tv,Yv(:,1),'r--','LineWidth',1.5)
plot(tv,Yv(:,2),'k--','LineWidth',1.5)
legend({'y(1)','y(2)','y(1)ode45','y(2)ode45'},...
'Location','southeast','FontSize',8,'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
function f=sys_fun(t,Y)
f(1,1)=sin(t)+cos(Y(1))+sin(Y(2));
f(2,1)=cos(t)+sin(Y(2));
end
```

Task (2)

main.m

```
clc; clear; close all;

%% Inputs
% Forces, Moments and Inertia
[forces,Moments,Mass,I,timeSpan,dt,ICs] = Input();
g = 9.81;
Mg = Mass*g;
t_vec = timeSpan(1):dt:timeSpan(2);
n = length(t_vec);

%% Solving
Result = NaN(12,n);
Result(:,1) = ICs;
time = linspace(timeSpan(1), timeSpan(2), n);
for i =2:n
    Result(:, i) = RBDSolver(Result(:, i-1), dt);
end

% Rearranging Results
u_vec=Result(1,:);
v_vec=Result(2,:);
w_vec=Result(3,:);
p_vec=Result(4,:);
q_vec=Result(5,:);
r_vec=Result(6,:);
phi_vec=Result(7,:);
theta_vec=Result(8,:);
epsi_vec=Result(9,:);
xe_vec=Result(10,:);
ye_vec=Result(11,:);
ze_vec=Result(12,:);

%% Solving using ODE45

[t, states] = ode45(@DOF6,timeSpan,ICs);
t_ODE=t';
u_ODE=states(:,1)';
v_ODE=states(:,2)';
w_ODE=states(:,3)';
p_ODE=states(:,4)';
q_ODE=states(:,5)';
r_ODE=states(:,6)';
phi_ODE=states(:,7)';
theta_ODE=states(:,8)';
epsi_ODE=states(:,9)';
xe_ODE=states(:,10)';
ye_ODE=states(:,11)';
ze_ODE=states(:,12);

%% solving using simulink
%Run Smulink file
Sim_run=sim('simulink_test.slx');
```

```
%Extraxt Data from Simulink
%velocity body
Velocity_body=Sim_run.yout.getElement('velocity body');
%rotational speed body
Rotat_Velocity_body=Sim_run.yout.getElement('rotational speed
body');
%rotational speed earth
Rotat_Velocity_earth=Sim_run.yout.getElement('rotational speed
earth');
%positions earth
positions_earth=Sim_run.yout.getElement('positions earth');

t_sim=Velocity_body.Values.time;
u_sim=Velocity_body.Values.Data(:,1)';
v_sim=Velocity_body.Values.Data(:,2)';
w_sim=Velocity_body.Values.Data(:,3)';
p_sim=Rotat_Velocity_body.Values.Data(:,1)';
q_sim=Rotat_Velocity_body.Values.Data(:,2)';
r_sim=Rotat_Velocity_body.Values.Data(:,3)';
phi_sim=Rotat_Velocity_earth.Values.Data(:,1)';
theta_sim=Rotat_Velocity_earth.Values.Data(:,2)';
epsi_sim=Rotat_Velocity_earth.Values.Data(:,3)';
xe_sim=positions_earth.Values.Data(:,1)';
ye_sim=positions_earth.Values.Data(:,2)';
ze_sim=positions_earth.Values.Data(:,3)';

%% plots
%velocity body
figure
subplot(3, 1,1)
plot(t_vec,u_vec,'b',t_ODE,u_ODE,'g',t_sim,u_sim,'r')
legend({'$u(code)$','$u(ODE45)$','$u(sim)$'},'Location','southeast
','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$u$','Interpreter','latex','FontSize',13)
title('u(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 2)
plot(t_vec,v_vec,'b',t_ODE,v_ODE,'g',t_sim,v_sim,'r')
legend({'$v(code)$','$v(ODE45)$','$v(sim)$'},'Location','southeast
','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$v$','Interpreter','latex','FontSize',13)
title('v(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 3)
plot(t_vec,w_vec,'b',t_ODE,w_ODE,'g',t_sim,w_sim,'r')
legend({'$w(code)$','$w(ODE45)$','$w(sim)$'},'Location','southeast
','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$w$','Interpreter','latex','FontSize',13)
title('w(from the code ,ODE45 and Simulink)')
```

```
grid on


%rotational speed body
figure
subplot(3, 1,1)
plot(t_vec,p_vec,'b',t_ODE,p_ODE,'g',t_sim,p_sim,'r')
legend({'$p(code)$','$p(ODE45)$','$p(sim)$'},'Location','southeast
','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$p$','Interpreter','latex','FontSize',13)
title('p(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 2)
plot(t_vec,q_vec,'b',t_ODE,q_ODE,'g',t_sim,q_sim,'r')
legend({'$q(code)$','$q(ODE45)$','$q(sim)$'},'Location','southeast
','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$q$','Interpreter','latex','FontSize',13)
title('q(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 3)
plot(t_vec,r_vec,'b',t_ODE,r_ODE,'g',t_sim,r_sim,'r')
legend({'$r(code)$','$r(ODE45)$','$r(sim)$'},'Location','southeast
','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$r$','Interpreter','latex','FontSize',13)
title('r(from the code ,ODE45 and Simulink)')
grid on


%rotational speed earth
figure
subplot(3, 1,1)
plot(t_vec,phi_vec,'b',t_ODE,phi_ODE,'g',t_sim,phi_sim,'r')
legend({'$\phi(code)$','$\phi(ODE45)$','$\phi(sim)$'},'Location','
southeast','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$\phi$','Interpreter','latex','FontSize',13)
title('\phi(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 2)
plot(t_vec,theta_vec,'b',t_ODE,theta_ODE,'g',t_sim,theta_sim,'r')
legend({'$\theta(code)$','$\theta(ODE45)$','$\theta(sim)$'},'Locat
ion','southeast','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$\theta$','Interpreter','latex','FontSize',13)
title('\theta(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 3)
```

```
plot(t_vec,epsi_vec,'b',t_ODE,epsi_ODE,'g',t_sim,epsi_sim,'r')
legend({'$\psi(code)$','$\psi(ODE45)$','$\psi(sim)$'},'Location','
southeast','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$\psi$','Interpreter','latex','FontSize',13)
title('\psi(from the code ,ODE45 and Simulink)')
grid on


%positions earth
figure
subplot(3, 1,1)
plot(t_vec,xe_vec,'b',t_ODE,xe_ODE,'g',t_sim,xe_sim,'r')
legend({'$xe(code)$','$xe(ODE45)$','$xe(sim)$'},'Location','southe
ast','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$xe$','Interpreter','latex','FontSize',13)
title('xe(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 2)
plot(t_vec,ye_vec,'b',t_ODE,ye_ODE,'g',t_sim,ye_sim,'r')
legend({'$ye(code)$','$ye(ODE45)$','$ye(sim)$'},'Location','southe
ast','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$ye$','Interpreter','latex','FontSize',13)
title('ye(from the code ,ODE45 and Simulink)')
grid on
subplot(3,1, 3)
plot(t_vec,ze_vec,'b',t_ODE,ze_ODE,'g',t_sim,ze_sim,'r')
legend({'$ze(code)$','$ze(ODE45)$','$ze(sim)$'},'Location','southe
ast','FontSize',8,...
    'Interpreter','latex')
xlabel('$t$','Interpreter','latex','FontSize',13)
ylabel('$ze$','Interpreter','latex','FontSize',13)
title('ze(from the code ,ODE45 and Simulink)')
grid on
```

DOF6.m

```
% Function return the set of 12 state of the six degree of freedom
% system after sub., with given ICs
function F = DOF6(~, ICs)
    [forces,Moments,Mass,Inertia,~,~,~] = Input();
    % (Sin, Cos, Tan) of (phi, theta, epsi)
    [S, C, T] = SCT(ICs(7:9));

    Forces = forces + Mass*9.81*[
        -S.theta;
        S.phi*C.theta;
        C.phi*C.theta;
    ];
```

```
    % (u, v, w) dot
    F(1:3, 1) = Forces/Mass - cross(...
        ICs(4:6, 1), ICs(1:3, 1)...
    );

    % (p, q, r) dot
    F(4:6, 1) = Inertia\(Moments - cross(...
        ICs(4:6, 1), Inertia * ICs(4:6, 1)...
    ));

    % (phi, theta, epsi) dot
    F(7:9, 1) = [
        1, S.phi*T.theta, C.phi*T.theta;
        0, C.phi, -S.phi;
        0, S.phi/C.theta, C.phi/C.theta;
    ] * ICs(4:6, 1);

    % (x, y, z) dot
    F(10:12, 1) = [
        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
    ] * ICs(1:3, 1);

end
```

Input.m

```
function [Forces,Moments,Mass,I,timeSpan,dt,ICs] = Input()
    % Inputs
    % Forces, Moments and Inertia
    Forces = [10 5 9]';          % Vector
    Moments = [10 20 5]';        % Vector
    Mass = 15;
    I = [1 -2 -1;
        -2 5 -3;
        -1 -3 0.1];
    % Integration time span & Step
    timeSpan = [0 15];
    dt = 0.001;
    % 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];

end
```

RBDSolver.m

```
% RBD Solver is function that implements Runge-Kutta-4
% Algorithm in order to integrate 6 DOF set of equations
% with a give Inital Conditions ICs, for single dt time step.
function state = RBDSolver(ICs, dt)
    K = zeros(12, 4);

    K(:, 1) = dt*DOF6(0, ICs );
    K(:, 2) = dt*DOF6(0, ICs+0.5*K(:, 1) );
    K(:, 3) = dt*DOF6(0, ICs+0.5*K(:, 2) );
    K(:, 4) = dt*DOF6(0, ICs+K(:, 3) );

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

Task (3)

main.m

```
clc; clear; close all;

%% Inputs
% Forces, Moments and Inertia

[Mass, g, I, invI, timeSpan, dt, ICs, ICs_dot0, Vt0, ...
    dControl, SD_Long, SD_Lat, initialGravity] =
Input("excelsheet_data_modified.xlsx");

%% Solving
steps = (timeSpan(2) - timeSpan(1))/dt;
Result = NaN(12, steps);
Result(:,1) = ICs;
time_V = linspace(0, timeSpan(2), steps+1);
dForces = [0 ; 0; 0];
dMoments = [0 ; 0; 0];

for i =1:steps
    i
    Result(:, i+1) = RBDSolver(Result(:, i), dt, (initialGravity +
dForces), dMoments, Mass, I, invI, g);

    [dF, dM] = airFrame(SD_Long, SD_Lat, dControl, ICs, ICs_dot0,
Result(:, i+1) ,Vt0, ...
        (initialGravity + dForces), dMoments, Mass, I, invI, g);


    dForces = vpa(dF');
    dMoments = vpa(dM');

end

%% Plotting
% Rearranging Results
u = Result(1,:);
v = Result(2,:);
w = Result(3,:);
p = Result(4,:);
q = Result(5,:);
r = Result(6,:);
phi = Result(7,:);
theta = Result(8,:);
psi = Result(9,:);
x = Result(10,:);
y = Result(11,:);
z = Result(12,:);

beta_deg=asin(v/Vt0)*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(time_V,u)
title('u (ft/sec)')
xlabel('time (sec)')
subplot(4,3,2)
plot(time_V,beta_deg)
title('\beta (deg)')
xlabel('time (sec)')
subplot(4,3,3)
plot(time_V,alpha_deg)
title('\alpha (deg)')
xlabel('time (sec)')
subplot(4,3,4)
plot(time_V,p_deg)
title('p (deg/sec)')
xlabel('time (sec)')
subplot(4,3,5)
plot(time_V,q_deg)
title('q (deg/sec)')
xlabel('time (sec)')
subplot(4,3,6)
plot(time_V,r_deg)
title('r (deg/sec)')
xlabel('time (sec)')
subplot(4,3,7)
plot(time_V,phi_deg)
title('\phi (deg)')
xlabel('time (sec)')
subplot(4,3,8)
plot(time_V,theta_deg)
title('\theta (deg)')
xlabel('time (sec)')
subplot(4,3,9)
plot(time_V,psi_deg)
title('\psi (deg)')
xlabel('time (sec)')
subplot(4,3,10)
plot(time_V,x)
title('x (ft)')
xlabel('time (sec)')
subplot(4,3,11)
plot(time_V,y)
title('y (ft)')
xlabel('time (sec)')
subplot(4,3,12)
plot(time_V,z)
title('z (ft)')
xlabel('time (sec)')
```

airFrame.m

```
function [dForce, dMoment] = airFrame(SD_Long, SD_Lat, dControl,
state0, state0_dot, state, Vt0 ,forces, moments, Mass, I, invI, g)

    [YV, YB, LB, NB, LP, NP, LR, NR, YDA, YDR, LDA, NDA, LDR, NDR]
= feval(@(x) x{:}, num2cell(SD_Lat));
    [XU, ZU, MU, XW, ZW, MW, ZWD, ZQ, MWD, MQ, XDE, ZDE, MDE,
XD_TH, ZD_TH, MD_TH] = feval(@(x) x{:}, num2cell(SD_Long));
    [Da, Dr, De, Dth] = feval(@(x) x{:}, num2cell(dControl));

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

    state_dot = DOF6(state, forces, moments, Mass, I, invI, g);

    ds = state - state0;
    ds_dot = state_dot - state0_dot;

    beta0 = asin(state0(2)/Vt0);
    beta = asin(state(2)/Vt0);
    dbeta = beta-beta0;

    dX = Mass*(XU*ds(1)+XW*ds(3)+XDE*De+XD_TH*Dth);
    dY = Mass*(YV*ds(2)+YB*dbeta+YDA*Da+YDR*Dr);
    dZ =
Mass*(ZU*ds(1)+ZW*ds(3)+ZWD*ds_dot(3)+ZQ*ds(5)+ZDE*De+ZD_TH*Dth);

    dL = Ixx*(LB*dbeta+LP*ds(4)+LR*ds(6)+LDR*Dr+LDA*Da);
    dM =
Iyy*(MU*ds(1)+MW*ds(3)+MWD*ds_dot(3)+MQ*ds(5)+MDE*De+MD_TH*Dth);
    dN = Izz*(NB*dbeta+NP*ds(4)+NR*ds(6)+NDR*Dr+NDA*Da);

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

end
```

DOF6.m

```
% Function return the set of 12 state of the six degree of freedom
% system after sub., with given ICs
function F = DOF6(ICs, forces, Moments, Mass, Inertia, invI, g)

    % (Sin, Cos, Tan) of (phi, theta, epsi)
    [S, C, T] = SCT(ICs(7:9));

    Forces = forces + Mass*g*[
        -S.theta;
        S.phi*C.theta;
        C.phi*C.theta;
    ];

    % (u, v, w) dot
    F(1:3, 1) = Forces/Mass - cross(...
        ICs(4:6, 1), ICs(1:3, 1)...
```

```
    );

    % (p, q, r) dot
    F(4:6, 1) = invI*(Moments - cross(...
        ICs(4:6, 1), Inertia * ICs(4:6, 1)...
    ));

    % (phi, theta, epsi) dot
    F(7:9, 1) = [
        1, S.phi*T.theta, C.phi*T.theta;
        0, C.phi, -S.phi;
        0, S.phi/C.theta, C.phi/C.theta;
    ] * ICs(4:6, 1);

    % (x, y, z) dot
    F(10:12, 1) = [
        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
    ] * ICs(1:3, 1);

end
```

Input.m

```
function [Mass, g, I, invI, timeSpan, dt, ICs, ICs_dot0, Vt0, ...
    dc, SD_Long, SD_Lat, initialGravity] = Input(inputs_filename)
    % Inputs
    % here B2:B61 means read the excel sheet from cell B2 to cell
B61
    aircraft_data=xlsread(inputs_filename,'B2:B61');

    % Integration time span & Step
    dt = aircraft_data(1);
    tfinal = aircraft_data(2);
    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];
    ICs = aircraft_data(4:15);
    ICs_dot0 = zeros(12,1);
    Vt0 = sqrt(ICs(1)^2 + ICs(2)^2 + ICs(3)^2);    % Vto

    % control actions values
    % D_a, D_r, D_e, D_th
    dc = [ aircraft_data(57:59) * pi/180 ; aircraft_data(60)];

    % gravity, mass % inertia
    Mass = aircraft_data(51);
    g = aircraft_data(52);
    Ixx = aircraft_data(53);
    Iyy = aircraft_data(54);
```

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


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

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

    SD_Lat = LateralSD2BodyAxes(SD_Lat_dash, I);

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

end
```

LateralSD2BodyAxes.m

```
function SD_Lat = LateralSD2BodyAxes(SD_Lat_dash, I)

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

    Ixx = I(1);
    Izz = I(9);
    Ixz = -I(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 == LBd;
    eq2 = (NB+Ixz*LB/Izz)*G == NBd;
    eq3 = (LP+Ixz*NP/Ixx)*G == LPd;
    eq4 = (NP+Ixz*LP/Izz)*G == NPd;
    eq5 = (LR+Ixz*NR/Ixx)*G == LRd;
    eq6 = (NR+Ixz*LR/Izz)*G == NRd;
    eq7 = (LDR+Ixz*NDR/Ixx)*G == LDRd;
    eq8 = (NDR+Ixz*LDR/Izz)*G == NDRd;
    eq9 = (LDA+Ixz*NDA/Ixx)*G == LDAd;
    eq10 = (NDA+Ixz*LDA/Izz)*G == 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;

    SD_Lat = [
        YV YB ...
        X(1) X(6) X(2) X(7) ...
        X(3) X(8) ...
        YDA YDR ...
        X(5) X(10) X(4) X(9) ...
    ]';

    SD_Lat = vpa(SD_Lat);

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
```

RBDSolver.m

```
% RBD Solver is function that implements Runge-Kutta-4
% Algorithm in order to integrate 6 DOF set of equations
% with a give Inital Conditions ICs, for single dt time step.
function state = RBDSolver(ICs, dt, Force, Moments, Mass, I, invI, g)

    K = zeros(12, 4);

    K(:, 1) = dt*DOF6(ICs ,Force, Moments, Mass, I, invI, g);
    K(:, 2) = dt*DOF6(ICs+0.5*K(:, 1) ,Force, Moments, Mass, I, invI, g);
    K(:, 3) = dt*DOF6(ICs+0.5*K(:, 2) ,Force, Moments, Mass, I, invI, g);
    K(:, 4) = dt*DOF6(ICs+K(:, 3) ,Force, Moments, Mass, I, invI, g);

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

end
```

Task (4)

main.m

```
clc; clear; close all;

%% Inputs
% Forces, Moments and Inertia

[Mass, g, I, invI, timeSpan, dt, ICs, ICs_dot0, Vt0, ...
    dControl, SD_Long, SD_Lat,SD_Lat_dash, initialGravity] =
Input("NT-33A_4.xlsx");
steps = (timeSpan(2) - timeSpan(1))/dt;
Result = NaN(12, steps);
Result(:,1) = ICs;
time_V = linspace(0, timeSpan(2), steps+1);

%% Solving
%profile on;
dForces = [0 ; 0; 0];
dMoments = [0 ; 0; 0];

for i =1:steps

    Result(:, i+1) = RBDSolver(Result(:, i), dt, (initialGravity +
dForces), dMoments, Mass, I, invI, g);

    [dF, dM] = airFrame(SD_Long, SD_Lat, dControl, ICs, ICs_dot0,
Result(:, i+1) ,Vt0, ...
        (initialGravity + dForces), dMoments, Mass, I, invI, g);


    dForces = vpa(dF');
    dMoments = vpa(dM');

end
%profile viewer

%% Rearranging Results
u = Result(1,:);
v = Result(2,:);
w = Result(3,:);
p = Result(4,:);
q = Result(5,:);
r = Result(6,:);
phi = Result(7,:);
theta = Result(8,:);
psi = Result(9,:);
x = Result(10,:);
y = Result(11,:);
z = Result(12,:);

beta_deg=asin(v/Vt0)*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;

%% Longitudenal Full Linear Model
XU=SD_Long(1);
ZU=SD_Long(2);
MU=SD_Long(3);
XW=SD_Long(4);
ZW=SD_Long(5);
MW=SD_Long(6);
ZWD=SD_Long(7);
ZQ=SD_Long(8);
MWD=SD_Long(9);
MQ=SD_Long(10);
XDE=SD_Long(11);
ZDE=SD_Long(12);
MDE=SD_Long(13);
XDTH=SD_Long(14);
ZDTH=SD_Long(15);
MDTH=SD_Long(16);
u0 = ICs(1);
v0 = ICs(2);
w0 = ICs(3);
q0 = ICs(5);
theta0 = ICs(8);
A_long=[XU XW -w0 -g*cos(theta0)
    ZU/(1-ZWD) ZW/(1-ZWD) (ZQ+u0)/(1-ZWD) -g*sin(theta0)/(1-ZWD)
    MU+MWD*ZU/(1-ZWD) MW+MWD*ZW/(1-ZWD) MQ+MWD*(ZQ+u0)/(1-ZWD) -
MWD*g*sin(theta0)/(1-ZWD)
    0 0 1 0];
B_long=[XDE XDTH
    ZDE/(1-ZWD) ZDTH/(1-ZWD)
    MDE+MWD*ZDE/(1-ZWD) MDTH+MWD*ZDTH/(1-ZWD)
    0 0];
C_long=eye(4);
D_long=zeros(4,2);

% Two Inputs - Four Output Each
LongSS = ss(A_long, B_long, C_long, D_long);

%% APPROXIMATE
%% PHUGOID MODE (LONG PERIOD MODE)

A_phug=[XU -g
    -ZU/(u0+ZQ) 0];
B_phug=[XDE XDTH
    -ZDE/(ZQ+u0) -ZDTH/(ZQ+u0)];
C_phug=eye(2);D_phug=zeros(2,2);

PHUG_SS=ss(A_phug,B_phug,C_phug,D_phug);

%% SHORT PERIOD MODE
A_short=[ZW/(1-ZWD) (ZQ+u0)/(1-ZWD)
    (MW+ZW*MWD/(1-ZWD)) (MQ+MWD*(ZQ+u0)/(1-ZWD))];
```

```
B_short=[ZDE/(1-ZWD) ZDTH/(1-ZWD)
    MDE+MWD*ZDE/(1-ZWD) MDTH+MWD*ZDTH/(1-ZWD)];
C_short=eye(2);D_short=zeros(2,2);

SHORT_SS=ss(A_short,B_short,C_short,D_short);

%%  Longitudinal Full Linear Model Step Response
%%% Due to delta_elevetor or delta_thrust
dControl_long = dControl(3:4); % dE, dTh
opt = stepDataOptions;
opt.StepAmplitude = dControl_long;
[res, ~, ~] = step(LongSS, time_V, opt);
res_dE  =res(:,:,1);
res_dTh =res(:,:,2);
%% Longitudinal Approximate Models Step Response
%%% Due to delta_elevetor or delta_thrust
dControl_long = dControl(3:4); % dE, dTh
opt = stepDataOptions;
opt.StepAmplitude = dControl_long;
[APPres_PH, ~,~] = step(PHUG_SS, time_V, opt);
[APPres_SH, ~,~] = step(SHORT_SS, time_V, opt);

APPres_dE=zeros (length(time_V),4);
APPres_dE(:,[1,4])=APPres_PH(:,:,1);
APPres_dE(:,[2,3])=APPres_SH(:,:,1);

APPres_dTH=zeros (length(time_V),4);
APPres_dTH(:,[1,4])=APPres_PH(:,:,2);
APPres_dTH(:,[2,3])=APPres_SH(:,:,2);
%% u response Full Linear - Approximate - Non Linear
figure(1)
if(dControl_long(1) ~= 0)
    plot(time_V, res_dE(:, 1) + u0, '--', 'DisplayName', 'u (Full
Linear)');  % Full Linear Model
    hold on
    plot(time_V, APPres_dE(:, 1) + u0, '--', 'DisplayName', 'u
(Long Period Approximation)');  % Approximate (long period Mode)
elseif(dControl_long(2) ~= 0)
    plot(time_V, res_dTh(:, 1) + u0, '--', 'DisplayName', 'u (Full
Linear)');  % Full Linear Model
    hold on
    plot(time_V, APPres_dTH(:, 1) + u0, '--', 'DisplayName', 'u
(Long Period Approximation)');  % Approximate (long period Mode)
end

hold on
plot(time_V, u, '-', 'DisplayName', 'u (Non-Linear)');
% Non-Linear Model
title('u (ft/sec)'); xlabel('t (sec)');
legend('show');
grid on
%% w response Full Linear - Approximate - Non Linear
figure(2)
if(dControl_long(1) ~= 0)
```

```
    plot(time_V, res_dE(:, 2) + w0, '--', 'DisplayName', 'w (Full
Linear)');  % Full Linear Model
    hold on
    plot(time_V, APPres_dE(:, 2) + w0, '--', 'DisplayName', 'w
(short Period Approximation)');  % Approximate (short period Mode)
elseif(dControl_long(2) ~= 0)
    plot(time_V, res_dTh(:, 2) + w0, '--', 'DisplayName', 'w (Full
Linear)');  % Full Linear Model
    hold on
    plot(time_V, APPres_dTH(:, 2) + w0, '--', 'DisplayName', 'w
(short Period Approximation)');  % Approximate (short period Mode)
end

hold on
plot(time_V, w, '-', 'DisplayName', 'w (Non-Linear)');
% Non-Linear Model
title('w (ft/sec)'); xlabel('t (sec)');
legend('show');
grid on
%% q response Full Linear - Approximate - Non Linear
figure(3)
if(dControl_long(1) ~= 0)
    q_ = res_dE(:, 3) + q0;
    plot(time_V, q_*180/pi, '--', 'DisplayName', 'q (Full
Linear)');  % Full Linear Model
    hold on
    q_APP = APPres_dE(:, 3) + q0;
    plot(time_V, q_APP*180/pi, '--', 'DisplayName', 'q (short
Period Approximation)');  % Approximate (short period Mode)

elseif(dControl_long(2) ~= 0)
    q_ = res_dTh(:, 3) + q0;
    plot(time_V, q_*180/pi, '--', 'DisplayName', 'q (Full
Linear)');  % Full Linear Model
    hold on
    q_APP = APPres_dTH(:, 3) + q0;
    plot(time_V, q_APP*180/pi, '--', 'DisplayName', 'q (short
Period Approximation)');  % Approximate (short period Mode)
end

hold on
plot(time_V, q_deg , '-', 'DisplayName', 'q (Non-Linear)');
% Non-Linear Model
title('q (1/sec^2)'); xlabel('t (sec)');
legend('show');
grid on
%% theta response Full Linear - Approximate - Non Linear
figure(4)
if(dControl_long(1) ~= 0)
    theta_ = (res_dE(:, 4) + theta0)*180/pi;
    plot(time_V, theta_, '--', 'DisplayName', '\Theta (Full
Linear)');  % Full Linear Model
    hold on
    theta_APP = (APPres_dE(:, 4) + theta0)*180/pi;
```

```
    plot(time_V, theta_APP, '--', 'DisplayName', '\Theta (Long
Period Approximation)');  % Full Linear Model
elseif(dControl_long(2) ~= 0)
    theta_ = (res_dTh(:, 4) + theta0)*180/pi;
    plot(time_V, theta_, '--', 'DisplayName', '\Theta (Full
Linear)');  % Full Linear Model
    hold on
    theta_APP = (APPres_dTH(:, 4) + theta0)*180/pi;
    plot(time_V, theta_APP, '--', 'DisplayName', '\Theta (Long
Period Approximation)');  % Full Linear Model
end

hold on
plot(time_V, theta_deg, '-', 'DisplayName', '\Theta (Non-
Linear)');                  % Non-Linear Model
title('theta (deg/sec)'); xlabel('t (sec)');
legend('show');
grid on
%% Lateral Full Linear Model

U0=u0; W0=w0; TH0=theta0; psi0=ICs(9); phi0=ICs(7);
Vto = sqrt(ICs(1)^2 + ICs(2)^2 + ICs(3)^2);    % Vto

% stability derivatives Lateral motion

Yp=0;
Yr=0;
YDa_star=SD_Lat_dash(9);
YDr_star=SD_Lat_dash(10);
Yb=SD_Lat_dash(2);
YDa=SD_Lat_dash(9)*Vto;
YDr=SD_Lat_dash(10)*Vto;

Lbd=SD_Lat_dash(3);
Lpd=SD_Lat_dash(5);
Lrd=SD_Lat_dash(7);
LDrd=SD_Lat_dash(13);
LDad=SD_Lat_dash(11);

Nbd=SD_Lat_dash(4);
Npd=SD_Lat_dash(6);
Nrd=SD_Lat_dash(8);
NDrd=SD_Lat_dash(14);
NDad=SD_Lat_dash(12);


A_Lat=[Yb/Vto (Yp+W0)/Vto (Yr-U0)/Vto g*cos(TH0)/Vto 0;...
       Lbd Lpd Lrd 0 0;...
       Nbd Npd Nrd 0 0;...
       0 1 tan(TH0) 0 0;...
       0 0 1/cos(TH0) 0 0];
B_Lat=[YDa YDr;...
       LDad LDrd;...
       NDad NDrd;...
       0 0;0 0];
```

```
C_Lat=eye(5); D_Lat=zeros(5,2);


Lateral_SS = ss(A_Lat,B_Lat,C_Lat,D_Lat);
Lateral_TF = tf(Lateral_SS);

B_DA_L = Lateral_TF(1,1);
B_DR_L = Lateral_TF(1,2);

P_DA_L = Lateral_TF(2,1);
P_DR_L = Lateral_TF(2,2);

R_DA_L = Lateral_TF(3,1);
R_DR_L = Lateral_TF(3,2);

PHI_DA_L = Lateral_TF(4,1);
PHI_DR_L = Lateral_TF(4,2);

PSI_DA_L = Lateral_TF(5,1);
PSI_DR_L = Lateral_TF(5,2);


%% 3DOF Spiral Mode Approximation
A_Spiral = [Lpd Lrd 0;...
            Npd Nrd 0;...
            1 tan(TH0) 0];
B_Spiral = [LDad LDrd;NDad NDrd;0 0];
C_Spiral = eye(3); D_Spiral = zeros(3,2);

Spiral_SS = ss(A_Spiral,B_Spiral,C_Spiral,D_Spiral);
Spiral_TF = tf(Spiral_SS);

P_DA_S= Spiral_TF(1, 1);
P_DR_S = Spiral_TF(1, 2);

R_DA_S = Spiral_TF(2, 1);
R_DR_S = Spiral_TF(2, 2);

PHI_DA_S = Spiral_TF(3, 1);
PHI_DR_S = Spiral_TF(3, 2);


%% 2DOF Dutch Mode Approximation
A_Dutch = [Yb/Vto (Yr-U0)/Vto-tan(TH0)*(Yp+W0)/Vto;Nbd Nrd-
tan(TH0)*Npd];
B_Dutch = [YDa_star YDr_star;NDad NDrd];
C_Dutch = eye(2); D_Dutch = zeros(2,2);

Dutch_SS = ss(A_Dutch, B_Dutch, C_Dutch, D_Dutch);
Dutch_TF = tf(Dutch_SS);

B_DA_D = Dutch_TF(1, 1);
B_DR_D = Dutch_TF(1, 2);

R_DA_D = Dutch_TF(2, 1);
```

```
R_DR_D = Dutch_TF(2, 2);


%% 1DOF Roll Approximation
A_Roll = Lpd;
B_Roll = LDad;
C_Roll = eye(1); D_Roll = zeros(1, 1);

Roll_SS = ss(A_Roll, B_Roll, C_Roll, D_Roll);
Roll_TF = tf(Roll_SS);

P_DA_R = Roll_TF(1 ,1);

%%  Lateral Full Linear Model Step Response
%% Due to delta_elevetor or delta_thrust
dControl_latr = dControl(1:2); % dA, dR
opt = stepDataOptions;
opt.StepAmplitude = dControl_latr;
[Lat_res, ~, ~] = step(Lateral_SS, time_V, opt);
Lat_res_dA  =Lat_res(:,:,1);
Lat_res_dR =Lat_res(:,:,2);
%% Lateral Approximate Models Step Response

dControl_latr = dControl(1:2); % dA, dR
opt = stepDataOptions;
opt.StepAmplitude = dControl_latr;

[Spir_res, ~,~] = step(Spiral_SS, time_V, opt);
[Dutch_res, ~,~] = step(Dutch_SS, time_V, opt);

dC1= dControl(1); % dA, dR
opt = stepDataOptions;
opt.StepAmplitude = dC1;
[Roll_res, ~,~] = step(Roll_SS, time_V, opt);

%%% Spiral
Spir_res_dA=zeros (length(time_V),5);
Spir_res_dA(:,[2,3,4])=Spir_res(:,:,1);

Spir_res_dR=zeros (length(time_V),5);
Spir_res_dR(:,[2,3,4])=Spir_res(:,:,2);
%%% Dutch
Dutch_res_dA=zeros (length(time_V),5);
Dutch_res_dA(:,[1,3])=Dutch_res(:,:,1);

Dutch_res_dR=zeros (length(time_V),5);
Dutch_res_dR(:,[1,3])=Dutch_res(:,:,2);
%%% Roll
Roll_res_dA=zeros (length(time_V),5);
Roll_res_dA(:,2)=Roll_res;
%%
figure(5)
beta0=v0/Vt0;
if(dControl_latr(1) ~= 0)
    beta_ = (Lat_res_dA(:, 1) + beta0)*180/pi;
```

```
    plot(time_V, beta_, '--', 'DisplayName', '\beta (Full Linear
lateral)');  % Full Lateral Linear Model
    hold on
    beta_d = (Dutch_res_dA(:, 1) + beta0)*180/pi;
    plot(time_V, beta_d, '--', 'DisplayName', '\beta (dutch)');  %
spiral Linear Model
elseif(dControl_latr(2) ~= 0)
    beta_ = (Lat_res_dR(:, 1) + beta0)*180/pi;
    plot(time_V, beta_, '--', 'DisplayName', '\beta (Full Linear
lateral)');  % Full Lateral Linear Model
    hold on
    beta_d = (Dutch_res_dR(:, 1) + beta0)*180/pi;
    plot(time_V, beta_d, '--', 'DisplayName', '\beta (dutch)');  %
spiral Linear Model
end

hold on
plot(time_V, beta_deg, '-', 'DisplayName', '\beta (Non-Linear)');
% Non-Linear Model
title('\beta (deg/sec)'); xlabel('t (sec)');
legend('show');
grid on
%%
figure(6)
beta0=v0/Vt0;
if(dControl_latr(1) ~= 0)
    P_ = Lat_res_dA(:, 2) ;
    plot(time_V, P_, '--', 'DisplayName', 'P (Full Linear
lateral)');  % Full Lateral Linear Model
    hold on
    P_r = Roll_res_dA(:, 2);
    plot(time_V, P_r, '--', 'DisplayName', 'P (Roll)');  % spiral
Linear Model
    hold on
    P_s = Spir_res_dA(:, 2);
    plot(time_V, P_s, '--', 'DisplayName', 'P (Spiral)');  %
spiral Linear Model
elseif(dControl_latr(2) ~= 0)
    P_ = Lat_res_dR(:, 2) ;
    plot(time_V, P_, '--', 'DisplayName', 'P (Full Linear
lateral)');  % Full Lateral Linear Model
    hold on
    P_s = Spir_res_dR(:, 2);
    plot(time_V, P_s, '--', 'DisplayName', 'P (Spiral)');  %
spiral Linear Model
end

hold on
plot(time_V, p, '-', 'DisplayName', 'p (Non-Linear)');
% Non-Linear Model
title('p (1/sec^2)'); xlabel('t (sec)');
legend('show');
grid on
%%
figure(7)
```

```
if(dControl_latr(1) ~= 0)
    R_ = Lat_res_dA(:, 3) ;
    plot(time_V, R_, '--', 'DisplayName', 'R (Full Linear
lateral)');  % Full Lateral Linear Model
    hold on
    R_d = Dutch_res_dA(:, 3);
    plot(time_V, R_d, '--', 'DisplayName', 'R (dutch)');  % spiral
Linear Model
    hold on
    R_s = Spir_res_dA(:, 3);
    plot(time_V, R_s, '--', 'DisplayName', 'R (Spiral)');  %
spiral Linear Model
elseif(dControl_latr(2) ~= 0)
    R_ = Lat_res_dR(:, 3) ;
    plot(time_V, R_, '--', 'DisplayName', 'R (Full Linear
lateral)');  % Full Lateral Linear Model
    hold on
    R_d = Dutch_res_dR(:, 3);
    plot(time_V, R_d, '--', 'DisplayName', 'R (dutch)');  % spiral
Linear Model
    hold on
    R_s = Spir_res_dR(:, 3);
    plot(time_V, R_s, '--', 'DisplayName', 'R (Spiral)');  %
spiral Linear Model
end

hold on
plot(time_V, r, '-', 'DisplayName', 'r (Non-Linear)');
% Non-Linear Model
title('r (1/sec^2)'); xlabel('t (sec)');
legend('show');
grid on
%%
figure(8)
if(dControl_latr(1) ~= 0)
    phi_ = (Lat_res_dA(:, 4) + phi0)*180/pi;
    plot(time_V, phi_, '--', 'DisplayName', '\phi (Full Linear
lateral)');  % Full Lateral Linear Model
    hold on
    phi_s = (Spir_res_dA(:, 4) + phi0)*180/pi;
    plot(time_V, phi_s, '--', 'DisplayName', '\phi (spiral)');  %
spiral Linear Model
elseif(dControl_latr(2) ~= 0)
      phi_ = (Lat_res_dR(:, 4) + phi0)*180/pi;
    plot(time_V, phi_, '--', 'DisplayName', '\phi (Full Linear
lateral)');  % Full Linear Model
    hold on
    phi_s = (Spir_res_dR(:, 4) + phi0)*180/pi;
    plot(time_V, phi_s, '--', 'DisplayName', '\phi (spiral)');  %
Full Linear Model
end

hold on
plot(time_V, phi_deg, '-', 'DisplayName', '\phi (Non-Linear)');
% Non-Linear Model
```

```
title('\phi (deg/sec)'); xlabel('t (sec)');
legend('show');
grid on
%%
figure(9)
if(dControl_latr(1) ~= 0)
    psi_ = (Lat_res_dA(:, 5) + psi0)*180/pi;
    plot(time_V, psi_, '--', 'DisplayName', '\psi (Full Linear
lateral)');  % Full Linear Model

elseif(dControl_latr(2) ~= 0)
      psi_ = (Lat_res_dR(:, 5) + psi0)*180/pi;
    plot(time_V, psi_, '--', 'DisplayName', '\psi (Full Linear
lateral)');  % Full Linear Model

end

hold on
plot(time_V, phi_deg, '-', 'DisplayName', '\psi (Non-Linear)');
% Non-Linear Model
title('\psi (deg/sec)'); xlabel('t (sec)');
legend('show');
grid on

% export_figure(max(double(get(groot,'Children')))+[-8:0], '',
{'t1', 't2','t3','t4','t5','t6','t7','t8','t9'}, 600, {'emf',
'emf','emf','emf','emf','emf','emf','emf','emf'})
```

airFrame.m

```
function [dForce, dMoment] = airFrame(SD_Long, SD_Lat, dControl,
state0, state0_dot, state, Vt0 ,forces, moments, Mass, I, invI, g)

    [YV, YB, LB, NB, LP, NP, LR, NR, YDA, YDR, LDA, NDA, LDR, NDR]
= feval(@(x) x{:}, num2cell(SD_Lat));
    [XU, ZU, MU, XW, ZW, MW, ZWD, ZQ, MWD, MQ, XDE, ZDE, MDE,
XD_TH, ZD_TH, MD_TH] = feval(@(x) x{:}, num2cell(SD_Long));
    [Da, Dr, De, Dth] = feval(@(x) x{:}, num2cell(dControl));

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

    state_dot = DOF6(state, forces, moments, Mass, I, invI, g);

    ds = state - state0;
    ds_dot = state_dot - state0_dot;

    beta0 = asin(state0(2)/Vt0);
    beta = asin(state(2)/Vt0);
    dbeta = beta-beta0;

    dX = Mass*(XU*ds(1)+XW*ds(3)+XDE*De+XD_TH*Dth);
    dY = Mass*(YV*ds(2)+YB*dbeta+YDA*Da+YDR*Dr);
```

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

    dL = Ixx*(LB*dbeta+LP*ds(4)+LR*ds(6)+LDR*Dr+LDA*Da);
    dM =
Iyy*(MU*ds(1)+MW*ds(3)+MWD*ds_dot(3)+MQ*ds(5)+MDE*De+MD_TH*Dth);
    dN = Izz*(NB*dbeta+NP*ds(4)+NR*ds(6)+NDR*Dr+NDA*Da);

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

end
```

DOF6.m

```
% Function return the set of 12 state of the six degree of freedom
% system after sub., with given ICs
function F = DOF6(ICs, forces, Moments, Mass, Inertia, invI, g)

    % (Sin, Cos, Tan) of (phi, theta, epsi)
    [S, C, T] = SCT(ICs(7:9));

    Forces = forces + Mass*g*[
        -S.theta;
        S.phi*C.theta;
        C.phi*C.theta;
    ];

    % (u, v, w) dot
    F(1:3, 1) = Forces/Mass - cross(...
        ICs(4:6, 1), ICs(1:3, 1)...
    );

    % (p, q, r) dot
    F(4:6, 1) = invI*(Moments - cross(...
        ICs(4:6, 1), Inertia * ICs(4:6, 1)...
    ));

    % (phi, theta, epsi) dot
    F(7:9, 1) = [
        1, S.phi*T.theta, C.phi*T.theta;
        0, C.phi, -S.phi;
        0, S.phi/C.theta, C.phi/C.theta;
    ] * ICs(4:6, 1);

    % (x, y, z) dot
    F(10:12, 1) = [
        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
    ] * ICs(1:3, 1);

end
```

Input.m

```
function [Mass, g, I, invI, timeSpan, dt, ICs, ICs_dot0, Vt0, ...
    dc, SD_Long, SD_Lat, initialGravity] = Input(inputs_filename)
    % Inputs
    % here B2:B61 means read the excel sheet from cell B2 to cell
B61
    aircraft_data=xlsread(inputs_filename,'B2:B61');

    % Integration time span & Step
    dt = aircraft_data(1);
    tfinal = aircraft_data(2);
    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];
    ICs = aircraft_data(4:15);
    ICs_dot0 = zeros(12,1);
    Vt0 = sqrt(ICs(1)^2 + ICs(2)^2 + ICs(3)^2);    % Vto

    % control actions values
    % D_a, D_r, D_e, D_th
    dc = [ aircraft_data(57:59) * pi/180 ; aircraft_data(60)];

    % gravity, mass % inertia
    Mass = aircraft_data(51);
    g = aircraft_data(52);
    Ixx = aircraft_data(53);
    Iyy = aircraft_data(54);
    Izz = aircraft_data(55);
    Ixz = aircraft_data(56);
    Ixy=0;  Iyz=0;
    I = [Ixx , -Ixy , -Ixz ;...
        -Ixy , Iyy , -Iyz ;...
        -Ixz , -Iyz , Izz];
    invI = inv(I);


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

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

    SD_Lat = LateralSD2BodyAxes(SD_Lat_dash, I);

    % initial gravity force
    [S, C, ~] = SCT(ICs(7:9));
    initialGravity = Mass*g*[
        S.theta;
```

```
        -S.phi*C.theta;
        -C.phi*C.theta;
    ];

end
```

LateralSD2BodyAxes.m

```
function SD_Lat = LateralSD2BodyAxes(SD_Lat_dash, I)

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

    Ixx = I(1);
    Izz = I(9);
    Ixz = -I(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 == LBd;
    eq2 = (NB+Ixz*LB/Izz)*G == NBd;
    eq3 = (LP+Ixz*NP/Ixx)*G == LPd;
    eq4 = (NP+Ixz*LP/Izz)*G == NPd;
    eq5 = (LR+Ixz*NR/Ixx)*G == LRd;
    eq6 = (NR+Ixz*LR/Izz)*G == NRd;
    eq7 = (LDR+Ixz*NDR/Ixx)*G == LDRd;
    eq8 = (NDR+Ixz*LDR/Izz)*G == NDRd;
    eq9 = (LDA+Ixz*NDA/Ixx)*G == LDAd;
    eq10 = (NDA+Ixz*LDA/Izz)*G == 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;

    SD_Lat = [
        YV YB ...
        X(1) X(6) X(2) X(7) ...
        X(3) X(8) ...
        YDA YDR ...
        X(5) X(10) X(4) X(9) ...
    ]';

    SD_Lat = vpa(SD_Lat);

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
```

RBDSolver.m

```
% RBD Solver is function that implements Runge-Kutta-4
% Algorithm in order to integrate 6 DOF set of equations
% with a give Inital Conditions ICs, for single dt time step.
function state = RBDSolver(ICs, dt, Force, Moments, Mass, I, invI, g)

    K = zeros(12, 4);

    K(:, 1) = dt*DOF6(ICs ,Force, Moments, Mass, I, invI, g);
    K(:, 2) = dt*DOF6(ICs+0.5*K(:, 1) ,Force, Moments, Mass, I, invI, g);
    K(:, 3) = dt*DOF6(ICs+0.5*K(:, 2) ,Force, Moments, Mass, I, invI, g);
    K(:, 4) = dt*DOF6(ICs+K(:, 3) ,Force, Moments, Mass, I, invI, g);

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

end
```

Task (5)

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_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] = 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);

%% 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);

%% 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]);

%% 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_PD_tf = velocityControldesignValues.C2;
velocity_PI_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(altitude_C_action_tf)

%% Altitude Controller h/h_com
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_PD_tf = altitudeControldesignValues.C2;
altitude_PI_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)
```

Task (6)

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-Design2.mat");

yaw_C_tf = yawDamperControldesignValues.C;
CL_r_rcom_tf = tf(yawDamperControldesignValues.IOTransfer_r2y);
yaw_C_action_tf = tf(yawDamperControldesignValues.IOTransfer_r2u);
```

```
figure;
impulse(yaw_C_action_tf);
title('r/r_{com} Control Action');
figure;
impulse(CL_r_rcom_tf);

%% Design a "Roll Controller"

LatSSYawDamped = feedback(servo * LatSS, yaw_C_tf, 2, 3, 1);
LatTFYawDamped = tf(LatSSYawDamped);
checking_r_rcom_tf = LatTFYawDamped(3, 2);
hold on
impulse(checking_r_rcom_tf, 'r--');
title('r/r_{com} - With Controller Vs. New Lat SS');
hold off

OL_phi_phicom = minreal(servo * LatTFYawDamped(4, 1));

rollControldesignValues =
matfile("DesignValues/rollControllerValues.mat");

roll_C_tf = rollControldesignValues.C;
CL_roll_tf = tf(rollControldesignValues.IOTransfer_r2y);
roll_C_action_tf = tf(rollControldesignValues.IOTransfer_r2u);


%%
%%%%%%%%%%%%%%%%%%% 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)
```

Task (7)

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)')
```