Autopilot Project

Authors:
Abdelateef Khaled 20201344,
Alia Elhalawani 202000292,
Aya Atya 202000247,
Seif Raslan 202000797

Under the supervision of: Dr. Mohanned Draz

Contents

List of Figures

List of Tables

1 Research Questions

1.1 Autopilot definition

Autopilot is a system used to guide and control the airplane through its mission without the need for the direct control of a human. Its main objective is to lessen the workload on the crew during cruising, long flights, or adverse weather conditions; and to maintain control of altitude, heading, speed, and other parameters.

1.2 The first successful autopilot invented

After the development of the first human-carrying airplane by the Right Brothers, the autopilot systems were developed with the first successful system built in In 1912 by the Sperry brothers. The system was able to maintain heading, roll, and pitch angles. In order to test their design, one of the brothers, Lawrence Sperry, trimmed the airplane at a steady level flight and then activated the autopilot. He started creating some sort of disturbance by standing in the cockpit with his hands above his head, and his mechanic started walking along the wings to see if the autopilot would be able to go back to equilibrium. The autopilot was able to maintain the equilibrium by giving commands to change the aileron, rudder, and elevator deflections. Attached is a photo of the airplane tested with Lawrence Sperry and his mechanic inside of it.[1]

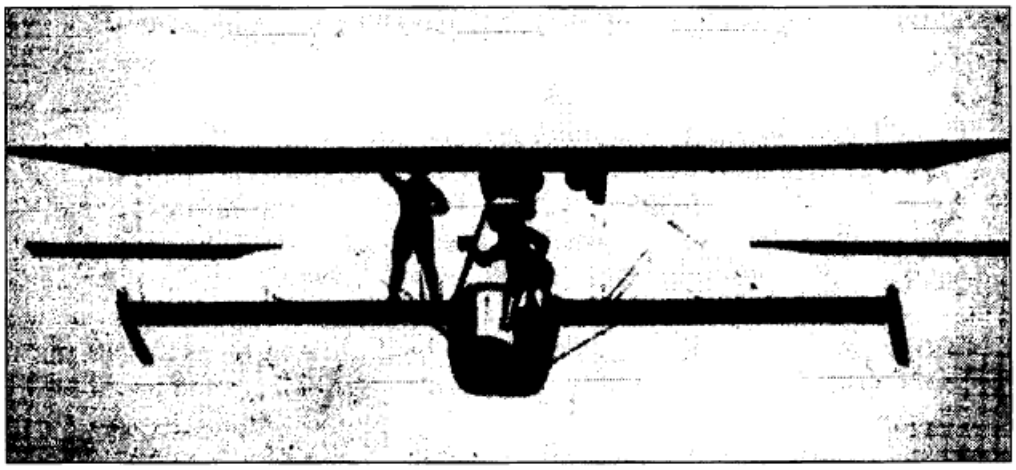


Figure 1: First successful autopilot mission

1.3 Inputs and outputs of autopilot system

Inputs:
-Control inputs: control surfaces (ailerons, rudder, elevator) positions to get feedback about how the deflection is changing.
-Navigation and performance data: coming from sensors such as GPS, Gyroscopes, and IMUs to detect the aircraft and wind speed, altitude, and heading.
Outputs:
-Control surface commands to maintain the desired orientation and direction.
-Flight path adjustments

1.4 Pilots job in the presence of autopilot systems

Pilots continuously monitor aircraft's systems such as engine performance, navigation, and altitude, to ensure that everything is operating properly. After all, the autopilot system is a world of human beings and it can be subjected to failure so in case that happens, the pilot must be there to ensure a safe flight. In addition, the pilots must be aware of the surroundings and keep track of any sudden changes in the environment or in the flight path. [2] Moreover, in normal cases, takeoff and landing is usually done by the pilot if the visibility is okay.

1.5 difference between an Autopilot SAS (stability augmentation system)

System	**Autopilot**	**SAS**
Objective	Automatic control of the flight by adjusting and maintaining flight parameters such as heading, altitude, navigation, and speed	Designed to mainly improve stability by damping the unwanted disturbances or oscillations caused during the flight and smoothing the flight route. However, it doesn't provide control for the aircraft by itself. It makes it easier to control the aircraft but doesn't replace the pilot control[**?**]
System	Control inputs usually connect to the aircraft control surfaces. It uses complex feedback loops with sensor inputs such as GPS and IMU. Typically has several subsystems and multiple modes of operation to account for every flight condition.	Limited to specific inputs. For example, a common type of SAS is the yaw damper which is used to reduce the Dutch roll. Sometimes the SAS systems, such as the yaw damper, can be part of the Autopilot system and sometimes they are standalone systems[**?**]. The system is usually much simpler than autopilot, with faster and more frequent feedback loops.
Application	Commonly used in civil and military aircraft.	Commonly used in helicopters, UAVs, and some modern jets.

Table 1: Comparison of Autopilot and Stability Augmentation System (SAS)

1.6 The role of the onboard sensors

Sensor	Function	Sample Rate
GPS	Determines the plane's position in space using satellites in its network to calculate its distance from it.	1570 1100 MHz
Gyroscopes	Provides stability and heading, and helps maintain the attitude. Their working principle depends on rotational inertia, so they resist changes to their orientation when spinning. This resistance can be used to measure changes in orientation.	400 750 Hz
Accelerometers	Detects changes in velocity by measuring the force exerted on an internal mass whilst the airplane is moving.	Low-frequency accelerometers: 0.1 500 Hz High-frequency accelerometers: 10 10 kHz
Altimeters	Measures the height of the aircraft above a certain reference point, usually sea level. It works by either measuring the ambient pressure or the time delay of a reflected wave.	4.2–4.4 GHz
Compasses	Determines the aircraft heading relative to Earth's magnetic poles.	–
Airspeed Indicators	Measures the aircraft's speed relative to air.	1 100 Hz
IMUs	Mainly consists of accelerometers and gyroscopes (sometimes magnetometers to provide additional information about the aircraft's heading). It is used to measure attitude (roll, pitch, yaw), velocity, altitude, and changes in gravitational forces.	100 1000 Hz

Table 2: Overview of sensors, functions, and sample rates used in aircraft.

1.7 How to measure a state that is not directly measured by a sensor

1. By calculating it using other states: for example, dynamic pressure is not directly sensed, but it is calculated using the data of airspeed measure from the pitot tube.
2. Using some estimation technique: such as the Kalman Filter which uses a series of observed noisy measurements over time, and produces estimates of unknown variables. For example, it can estimate the position and angular velocity of an aircraft by combining data from GPS, and inertial sensors such as accelerometers.[4]
3. Observers: An observer is a mathematical model that estimates the internal states of a system based on measured outputs and known inputs.

1.8 Fly-by-wire control system

As seen in Figure 2 It is a semi-automatic flight control system that is regulated using computers. The system basically uses computers to process control input whether made by the autopilot or the pilot himself, and then it sends electrical signals that correspond to these inputs to the control surface actuators in order to adjust them automatically. [6]
The traditional mechanical systems use levers and cables which the pilot moves to adjust the control surfaces. Although this might give the pilot a direct sense of how every control surface is adjusted, it is complicated and needs consistent monitoring.[7] Its main advantage is that it replaces the direct mechanical linkage making the linkage much lighter, and improves safety and maneuverability.

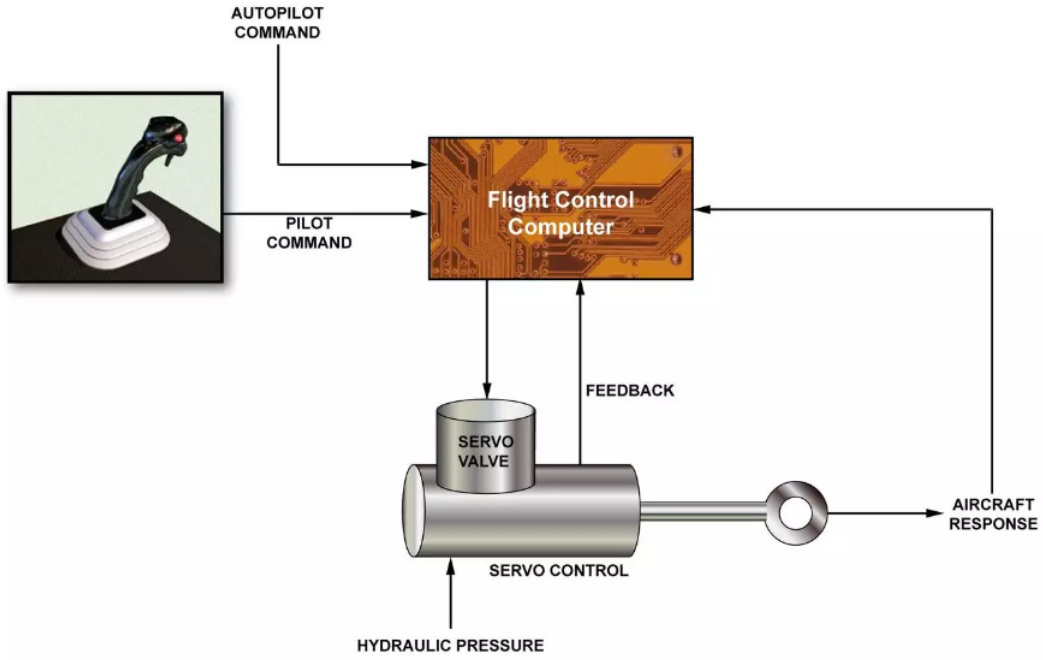


Figure 2: Fly-By-Wire system

1.9 open-source autopilots for UAVs

Pixhawk: used in a wide range of applications from drones and UAVs to industrial applications. [8]
ArduPilot: With the first autopilot open code created in 2009 it now supports many vehicle types: multi-copters, traditional helicopters, fixed-wing aircraft, VTOLs, and others. It is constantly developed and enhanced based on the feedback of users.[9]
Cube Orange+: Advanced open-source autopilot for UAVs. [10]

2 Flight Mechanics Review

2.1 General rigid body dynamics (RBD) equations in 3D space

In 3D space, the equations can be classified as translational or rotational motion.

Body Axes	**Linear Velocities**	**Forces**	**Angular Velocities**	**Moments**
x_b	u	$F_x(X)$	p	$\mathcal{L}$ (rolling)
y_b	v	$F_y(Y)$	q	M (pitching)
z_b	w	$F_z(Z)$	r	N (yawing)

Table 3: Body Axes, Velocities, Forces, Angular Velocities, and Moments

Translational Motion: Governed by Newton's second law:

$$F = ma$$

$$F = ma = \frac{d}{dt}(m\vec{V}) = m\frac{d\vec{V}}{dt}, \text{ where } \frac{d\vec{V}}{dt} \text{ is } \dot{\vec{V}} + \vec{W} \times \vec{V}$$

Where:

- $\vec{F}$ = sum of external forces acting on the aircraft.
- m = mass of the aircraft.
- a = acceleration of the aircraft.
- $\vec{V}$ and $\vec{W}$ are the velocity and angular velocity vectors, respectively.

Thus in matrix form:

$$\begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = m \left[\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} + \begin{pmatrix} qw - rv \\ ru - pw \\ pv - qu \end{pmatrix} \right]$$

Rotational Motion: Described by Euler's equations for the rotational dynamics of a rigid body:

$$M = I\frac{d\vec{\omega}}{dt} + \vec{\omega} \times (I\vec{\omega})$$

Where:

- $\vec{M}$ = sum of external moments acting on the aircraft.
- I = Aircraft's Moment of Inertia.
- $\vec{\omega}$ = Angular velocity vector of aircraft.

Thus in matrix form:

$$\begin{bmatrix} \mathcal{L} \\ M \\ N \end{bmatrix} = I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Where:

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

Assumptions to be used:

- **Inertial Frame Assumption**: The equations are derived in an inertial frame (Earth or space-fixed), assuming no rotation or acceleration of the Earth, which simplifies the dynamics.
- **Symmetric Aircraft**: The aircraft is considered to be symmetric about the longitudinal plane, X-Z plane, which simplifies the inertia. (i.e., $I_{xy} = I_{xz} = I_{yz} = I_{zx} \approx 0$).
- **Constant Mass and Inertia**: Fuel consumption and variations in payload are not taken in account to calculate the mass and inertia properties of the aircraft.

thus we get the following:

$$\begin{pmatrix} \mathcal{L} \\ M \\ N \end{pmatrix} = \begin{pmatrix} I_x\dot{p} - I_{xz}\dot{r} \\ I_y\dot{q} \\ I_z\dot{r} - I_{xz}\dot{p} \end{pmatrix} + \begin{pmatrix} qr(I_z - I_y) - I_{xz}pq \\ pr(I_x - I_z) + I_{xz}(p^2 - r^2) \\ pq(I_y - I_x) + I_{xz}qr \end{pmatrix}$$

Euler rates and Euler angles could be presented by the following steps respectively:

- we could use **Rotation Matrices** To convert from inertial frame (I) to the body frame (B):

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}^{(B)} = \left[R_\phi^x R_\theta^y R_\psi^z\right] \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}^{(I)}$$

  Where R_{total} *is* $\left[R_\phi^x R_\theta^y R_\psi^z\right]$.

- To get body angular velocities in terms of Euler angles and Euler rates:

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix} = \begin{pmatrix} \dot{\phi} - \dot{\psi}\sin\theta \\ \dot{\theta}\cos\phi + \dot{\psi}\cos\theta\sin\phi \\ \dot{\psi}\cos\theta\cos\phi - \dot{\theta}\sin\phi \end{pmatrix}$$

- To get the time derivatives of Euler angles:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} p + (q\sin\phi + r\cos\phi)\tan\theta \\ q\cos\phi - r\sin\phi \\ \frac{q\sin\phi + r\cos\phi}{\cos\theta} \end{pmatrix}$$

To express Earth-fixed velocities in terms of body velocities:

$$\begin{pmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{pmatrix} = [T]_{EB} \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

Hence:

$$\begin{pmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{pmatrix} = \begin{pmatrix} u^E\cos\theta\cos\psi + v^E(\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi) + w^E(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) \\ u^E\cos\theta\sin\psi + v^E(\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi) + w^E(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \\ -u^E\sin\theta + v^E\sin\phi\cos\theta + w^E\cos\phi\cos\theta \end{pmatrix}$$

2.2 Classification of Equations into Kinematics and Kinetics

- **Kinetics**:
  - **Translational equations**: Newton's second law for forces acting on the aircraft.
  - **Rotational equations**: Euler's equations for moments and angular accelerations.
- **Kinematics**:
  - **Position and orientation**: The position (x, y, z) of the aircraft's center of mass and the Euler angles ϕ (roll), θ (pitch), and ψ (yaw), characterize the aircraft's 3D orientation.
  - **Velocity and angular velocity**: Linear velocities u, v, w and angular velocities p, q, r.

2.3 Assumptions

- **Rigid Body Assumption**: The aircraft is considered as a rigid body not deforming under any condition of aerodynamic or inertial force.
- **Inertial Frame Assumption**: The equations are derived in an inertial frame (Earth or space-fixed), assuming no rotation or acceleration of the Earth, which simplifies the dynamics.
- **Small Angle Approximations (for Stability Analysis)**: For stability analysis, small angle approximations of the type $\sin(\theta) \approx 0$ and $\cos(\theta) \approx 1$ are normally used while analyzing small perturbations around a trim state (steady flight).
- **Symmetric Aircraft**: The aircraft is considered to be symmetric about the longitudinal plane, X-Z plane, which simplifies the inertia. (i.e., $I_{xy} = I_{xz} = I_{yz} = I_{zx} \approx 0$).
- **Constant Mass and Inertia**: Fuel consumption and variations in payload are not taken in account to calculate the mass and inertia properties of the aircraft.
- **Gravity as a Constant Force**: Gravity is modeled as a constant vertical force, with no variation due to altitude or geographic location.
- **No External Disturbances**: External forces such as wind gusts, atmospheric disturbances, and turbulence are neglected for simplicity.
- **Neglecting Higher-Order Effects**: The non-linearities such as aerodynamic hysteresis or higher order terms of the Taylor series expansion are neglected. Dynamic effects higher than second-order are usually neglected, such as unsteady aerodynamics or large deflections.

2.4 The set of equations added to the (RBD) equations to form the Fixed Wing Airplanes (EOM)

1. **Gravitational Forces:**

$$\mathbf{F_g} = mg \begin{pmatrix} -\sin\theta \\ \cos\theta \sin\phi \\ \cos\theta \cos\phi \end{pmatrix}$$

2. **Body Forces (Aerodynamics, etc.):**

$$\mathbf{F} = \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

3. **Thrust Forces:**

   The thrust T_x for a jet engine can be approximated as:

$$T_x = \dot{m} \cdot (V_e - V_0) + A_e \cdot (P_e - P_0)$$

   Where:

   - $\dot{m}$ is the mass flow rate of the air through the engine.
   - V_e is the exhaust velocity of the jet.
   - V_0 is the velocity of the aircraft (true airspeed).
   - A_e is the exit area of the engine.
   - P_e is the pressure at the exhaust.
   - P_0 is the ambient pressure.

Now, the thrust force vector can be written as:

$$\mathbf{F_T} = \begin{pmatrix} T_x \\ 0 \\ 0 \end{pmatrix}$$

Assuming that Thrust acts in a translational motion (x axis), and there are components in the lateral or vertical directions(as the engine is not tilted at an angle).

2.5 Aircraft's EOM Mathematical Classification.

The equations of motion can be classified into the following based on the mathematical form of the variables involved:

Nonlinear Coupled Differential Equations

The full EOM of an aircraft consists of **nonlinear, coupled, ordinary differential equations** that relate the velocities and accelerations in both the translational and rotational domains. These include:

- **Six degrees of freedom (6-DOF):**
  - **Three translational** equations for motion in the x, y, z directions.
  - **Three rotational** equations for the roll, pitch, and yaw motions.
- **Nonlinearities:**
  - The equations are **nonlinear** due to the coupling terms involving the rotational rates and translational velocities (e.g.: qr, pr, $qw - rv$, $pv - qu$, etc.).
  - These **coupling** terms introduce **Coriolis effects** and **Gyroscopic effects**, which indicates the influence of rotational motion over translational motion.
  - The dependence of aerodynamic forces and moments on the angle of attack α, sideslip angle β, and control surface deflections, which are nonlinear functions.

Linearized Approximation for Small Disturbances

The nonlinear EOM are **linearized** around a steady-state flight condition. The linearized equations are valid for small perturbations around this equilibrium state and take the form:

$$A\Delta x + B\Delta u = \dot{x}$$

Where:

- x is a state vector, containing small perturbations in velocity, angular rate, etc.
- u is the *input vector*, representing small control inputs.
- A is the *system matrix*, and B is the *input matrix*.

Decoupled Modes:

The linearized equations can often be decoupled into two sets of modes:

Longitudinal modes: Related to pitch, altitude, and speed (involving u, w, q, θ).

- Short-period mode.
- Phugoid mode.

Lateral-directional modes: Related to yaw, roll, and sideslip (involving v, p, r, ϕ, ψ).

- Dutch roll mode.
- Spiral mode.
- Roll subsidence.

2.6 The difference between the (Body axes) and the (earth or inertial axes)

Body Axes System:

- It's fixed to the aircraft and constantly moves and rotates with it.
- It's useful for expressing forces (like aerodynamic forces), moments, and velocities relative to the aircraft itself.

Earth (Inertial) Axes System:

- It's fixed relative to the ground and provides an inertial reference.
- When considering aircraft motion, this frame is typically assumed to be non-rotating and non-accelerating, which simplifies calculations.

2.7 Difference between the pitch angle (θ) and the angle of attack (α), and between the sideslip angle (β) and the heading angle (ψ)

Pitch Angle (θ) vs Angle of Attack (α):

- **Pitch Angle (θ):**
  - The pitch angle is the angle between the aircraft's longitudinal axis and the *local horizontal plane* (horizontal plane in the inertial frame).
  - It represents the nose-up or nose-down attitude of the aircraft relative to the Earth's horizontal plane.
- **Angle of Attack (α):**
  - The angle of attack is the angle between the aircraft's longitudinal axis and the *relative airflow* or *flight path vector*.
  - It also reflects how the aircraft is interacting with the airstream, which is important for lift generation.

Relationship:

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

Where:

- γ = flight path angle (angle between the velocity vector and the horizontal plane).

Sideslip Angle (β) vs Heading Angle (ψ):

- **Sideslip Angle (β):**
  - The sideslip angle is the angle between the aircraft's longitudinal axis (X-body axis) and the *relative airflow*.
  - It measures the extent to which the aircraft is "sliding" sideways relative to the airflow.
- **Heading Angle (ψ):**
  - The heading angle is the angle between the aircraft's longitudinal axis and *true north* on the horizontal plane. It represents the aircraft's orientation.

2.8 Different attitude representations

There are alternative ways to represent aircraft orientation or attitude other than Euler angles. Each has its advantages and disadvantages.

1. Direction Cosine Matrix (DCM)

A **Direction Cosine Matrix (DCM)** is a 3×3 matrix that transforms vectors from one coordinate frame (e.g., body frame) to another (e.g., Earth frame). It contains the cosine of the angles between each pair of coordinate axes.

- **Advantages:**
  - Provides an exact and continuous representation of orientation.
  - Directly used to convert vectors between coordinate frames.
  - No singularities (unlike Euler angles).
- **Disadvantages:**
  - Requires 9 components (matrix elements) to describe orientation, compared to only 3 angles for Euler.
  - It is prone to numerical drift (loss of orthogonality) during integration, and orthogonality must be enforced periodically.

2. Quaternions

Quaternions are a compact way to represent orientation using four numbers (scalar + 3-vector). A quaternion can be represented as:

$$q = [q_0, q_1, q_2, q_3] = \left[\cos\left(\frac{\theta}{2}\right), u_x \sin\left(\frac{\theta}{2}\right), u_y \sin\left(\frac{\theta}{2}\right), u_z \sin\left(\frac{\theta}{2}\right)\right]$$

Where θ is the rotation angle and (u_x, u_y, u_z) is the unit vector around which the body rotates.

- **Advantages:**
  - No Singularity (*gimbal lock*): Quaternions avoid the singularity problem of Euler angles.
  - Compact: Requires only 4 components.
  - Efficient: Multiplying quaternions is computationally more efficient than multiplying rotation matrices.
- **Disadvantages:**
  - More complex and less intuitive compared to Euler angles.
  - Requires normalization during numerical integration to maintain unit magnitude.

3. Axis-Angle Representation

In **axis-angle representation**, orientation is described by a single *rotation axis* and a *rotation angle*.
The orientation is given by rotating the body around the unit vector $\mathbf{u} = (u_x, u_y, u_z)$ by an angle θ.

- **Advantages:**
  - Simple and intuitive for representing a single rotation.
  - Compact (3 numbers: axis + angle).
- **Disadvantages:**
  - Similar to quaternions, normalization is needed to keep the unit vector magnitude.
  - Not efficient for composing multiple rotations.
  - More cumbersome to integrate and use in sequential rotations.

Each representation is suitable for different applications, and in aerospace engineering, **Quaternions** and **DCM** are often preferred for their robustness and lack of singularities.

3 Numerical Solution of ODEs

3.1 Numerical Solving Algorithms for ODEs

Euler's Method: Is a simple and fast numerical method used frequently for its simple and fast approach. The method mainly depends on calculating the next point using the current slope of the point that it is currently on.

$$y_{n+1} = y_n + hf\left(x_n, y_n\right)$$

Heun's Method: Similar to Euler's method, It's a second order method that uses the average of the slopes at the beginning and the end of the step.

$$y_{n+1} = y_n + \frac{h}{2}\left(f\left(x_n, y_n\right) + f\left(x_{n+1}, y_n + hf\left(x_n, y_n\right)\right)\right)$$

Runge-Kutta 2nd Order (RK2): The Runge-Kutta is a method used widely in numerical algorithms. It is a secound order method that uses an intermediate step and has higher accuracy than Euler's method

$$y_{n+1} = y_n + hk_2$$

$$k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$

Runge-Kutta 4th Order (RK4): The Runge-Kutta 4th order (RK4) method is a highly accurate algorithm that efficiently estimates solutions with fewer time steps, offering a balance between precision and computational speed.

$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

$$k_1 = h \cdot f\left(x_0, y_0\right).$$

$$k_2 = h \cdot f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right).$$

$$k_3 = h \cdot f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}\right)$$

$$k_4 = h \cdot f\left(x_0 + h, y_0 + k_3\right).$$

3.2 Algorithm for Solving the Airplane's EOM

The most efficient algorithm that can be used in such case we will use the RK4 method. We need to state the initial conditions in order to solve the differential equation. Firstly, we will assume that the mass m and the inertia matrix I are constant so we don't need to include initial states for these states. The initial states can be shown as follows:

Initial Position: (x_0, y_0, z_0)
Initial Velocity: (u_0, v_0, w_0)
Initial orientation: $(\phi_0, \theta_0, \psi_0)$
Initial angular velocity:(p_0, q_0, r_0)

By identifying the initial conditions we are set to use the RK4 method and to achieve this we need to perform a set of iterations in order to identify $k_1, k_2, k_3 and k_4$ and the following states are needed:

current Position: (x_n, y_n, z_n)
current Velocity: (u_n, v_n, w_n)
current orientation: $(\phi_n, \theta_n, \psi_n)$
current angular velocity:(p_n, q_n, r_n)

The output that will be calculated is the updated states by using the new k_n. Hence, we can list the outputs as follows:

$$k_1 = h \cdot f\left(x_0, y_0\right).$$

$$k_2 = h \cdot f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right).$$
$$k_3 = h \cdot f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}\right)$$
$$k_4 = h \cdot f\left(x_0 + h, y_0 + k_3\right).$$

We will plug the current states in the RK4 formula to determine the new states in which we will compare them with the current states and see if the change is less than a the tolerance we defined then we will stop iterating

$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

and for the states:

New position: $(x_{n+1}, y_{n+1}, z_{n+1})$.
New velocity: $(u_{n+1}, v_{n+1}, w_{n+1})$.
New orientation (Euler angles): $(\phi_{n+1}, \theta_{n+1}, \psi_{n+1})$.
New angular velocity: $(p_{n+1}, q_{n+1}, r_{n+1})$.

3.3 The solution of first order system of ODEs

The system of first-order ordinary differential equations (ODEs) to be solved is given by:

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

with the initial conditions:

$$\text{At } t = 0, \quad \mathbf{y}(0) = \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

The goal is to solve the system from $t = 0$ to $t = 20$ using 100 time steps (i.e., $n = 100$).

3.3.1 Solution Algorithm

To solve this system of ODEs, we use the fourth-order Runge-Kutta (RK4) method. The Runge-Kutta method is an iterative method that calculates the solution at each time step based on the following equations:

$$k_1 = h \cdot f\left(t_i, \mathbf{y}_i\right),$$
$$k_2 = h \cdot f\left(t_i + \frac{h}{2}, \mathbf{y}_i + \frac{k_1}{2}\right),$$
$$k_3 = h \cdot f\left(t_i + \frac{h}{2}, \mathbf{y}_i + \frac{k_2}{2}\right),$$
$$k_4 = h \cdot f\left(t_i + h, \mathbf{y}_i + k_3\right),$$

where h is the time step size, and $f(t, \mathbf{y})$ represents the system of ODEs.
The solution at the next time step $\mathbf{y}_{i+1}$ is then updated as follows:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

We solve the system using the RK4 method with the following parameters:

$$t_0 = 0, \quad t_f = 20, \quad n = 100.$$

The step size is calculated as:

$$h = \frac{t_f - t_0}{n} = \frac{20 - 0}{100} = 0.2.$$

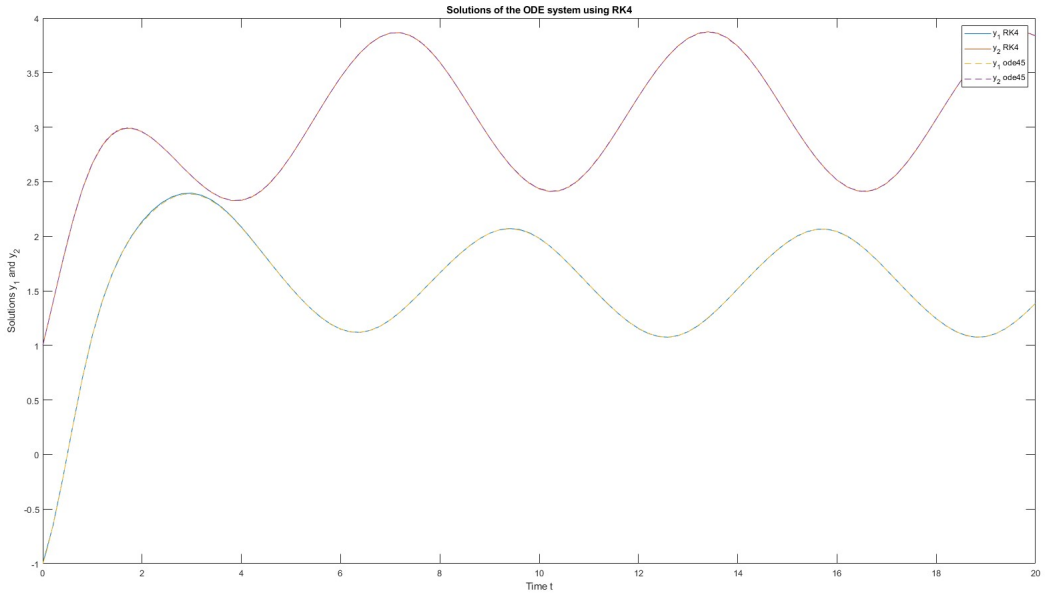


Figure 3: Numerical Solution of the ODE System Using RK4 and Validation with ode45

3.3.2 Solution Results and Validation

The MATLAB implementation for this numerical method is shown in Appendix A, and the results of the system of ODEs are plotted in Figure 224.
We validate the solution by comparing it with MATLAB's built-in `ode45` solver. As seen in Figure 224, the RK4 method provides results that closely match those from `ode45`, demonstrating the accuracy and reliability of our implementation.

4 Airplane Simulator Part I

Rigid Body Dynamics (RBD) equations describe the motion of a rigid body under the influence of external forces and moments. These equations are nonlinear, making analytical solutions impractical for most real-world scenarios. To handle this complexity, numerical methods are employed to approximate the solutions. One of the most widely used methods for solving such differential equations is the Runge-Kutta 4th order (RK4) method, known for its accuracy and efficiency.
In this section, we focus on solving the RBD equations for a rigid body subjected to constant external forces and moments. The RK4 method is particularly suitable for this purpose because it provides a good balance between computational cost and precision. By applying RK4, we can simulate the evolution of the rigid body's orientation, angular velocity, and linear velocity over time.

4.1 Initial Conditions Configuration

We aim to solve the Rigid Body Dynamics (RBD) equations for a rigid body under the influence of constant external forces and moments using the Runge-Kutta 4th order (RK4) method. The simulation will be carried out over a time span of 25 seconds, with the following constants:

$$t_{\text{final}} = 25\,\text{sec},$$
$$\text{Forces} = [10; 5; 3]\,\text{N},$$
$$\text{Moments} = [10; 15; 20]\,\text{N} \cdot \text{m},$$
$$\text{Mass} = 30\,\text{kg},$$
$$I = \begin{bmatrix} 1 & -2 & -1 \\ -2 & 5 & -4 \\ -1 & -4 & 0.2 \end{bmatrix}\,\text{kg} \cdot \text{m}^2$$

The initial conditions for the linear and angular velocities, Euler angles, and position are defined as follows:

$$[u, v, w, p, q, r, \phi, \theta, \psi, x, y, z]_{t=0} =$$
$$[10, 2, 0, 2 \cdot \frac{\pi}{180}, 1 \cdot \frac{\pi}{180}, 0 \cdot \frac{\pi}{180}, 20 \cdot \frac{\pi}{180}, 15 \cdot \frac{\pi}{180}, 30 \cdot \frac{\pi}{180}, 2, 4, 7]$$

Here, u, v, w represent the initial linear velocities along the body-fixed axes, while p, q, r are the initial angular velocities. The Euler angles ϕ, θ, ψ represent the roll, pitch, and yaw angles, respectively, and x, y, z denote the initial position of the rigid body.
Using these initial conditions and the given external forces and moments, the RK4 method will be applied to compute the rigid body's motion over the simulation time.

4.2 Airplane Equations of Motion

The equations of motion for a rigid body are governed by Newton's second law for both translational and rotational motion. The following set of equations describe the dynamics of a rigid body in terms of the forces, moments, and inertial properties. These equations take into account the body's mass and inertia tensor to calculate linear and angular accelerations.
The translational motion is governed by the following equation, where F_x, F_y, F_z are the external forces acting on the body, m is the mass of the body, and u, v, w are the velocities in the body-fixed frame. Additionally, p, q, r represent the angular velocities about the x, y, z-axes:

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \left(\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \right)$$

The rotational dynamics are described by the moments acting on the body, L, M, N, and the inertia matrix I:

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Where the inertia tensor I is given as:

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

The orientation of the body is described by the Euler angles ϕ (roll), θ (pitch), and ψ (yaw). The rates of change of these angles are related to the angular velocities p, q, r as follows:

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = [J] \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Finally, the translational motion of the body in the inertial frame (x_E, y_E, z_E) is related to the body frame velocities (u, v, w) by the following transformation matrix $[T]_{EB}$:

$$\begin{bmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{bmatrix} = [T]_{EB} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

Where the transformation matrix is given by:

$$[T]_{EB} = \begin{bmatrix} \cos(\theta)\cos(\psi) & -\cos(\phi)\sin(\psi) + \cos(\psi)\sin(\theta)\sin(\phi) & \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\theta)\sin(\phi)\sin(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

4.3 Evaluation of Rigid Body Dynamics Solutions: A Simulink and MATLAB Comparison

The rigid body dynamics equations were solved using both a Simulink model and a custom MATLAB script that implements the fourth-order Runge-Kutta (RK4) method. The three sets of results—velocities, angles, and angular velocities—show excellent agreement between Simulink and MATLAB outputs, which verifies the accuracy of both approaches.

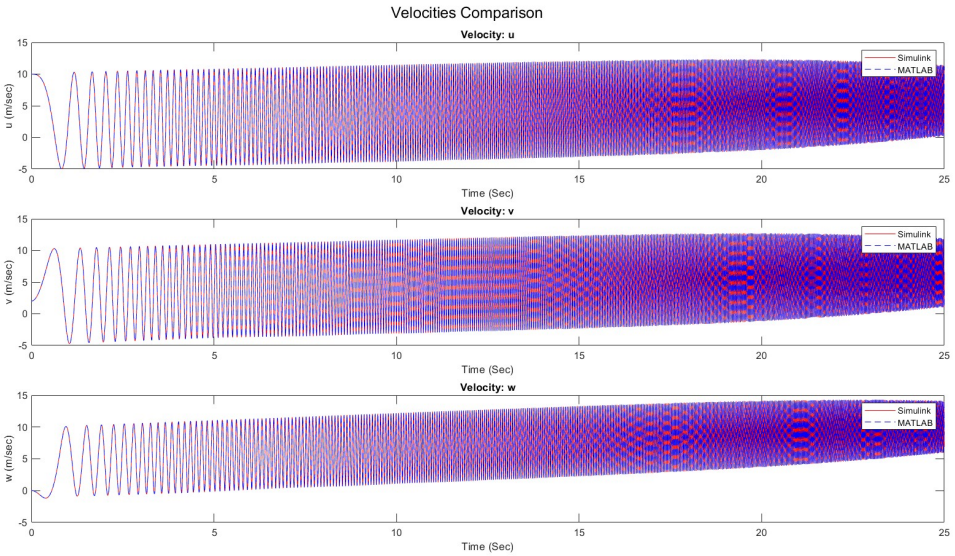


Figure 4: Comparison of aircraft velocity components (u, v, w) between MATLAB and Simulink results

Velocities: The time evolution of the linear velocities u, v, and w is presented in the first plot. Both the Simulink and MATLAB results match closely, showing that all velocity components increase over time. The initial oscillations gradually diminish, leading to smoother behavior as the system stabilizes. This indicates that the dynamic responses of the system are being captured accurately in both simulation environments.

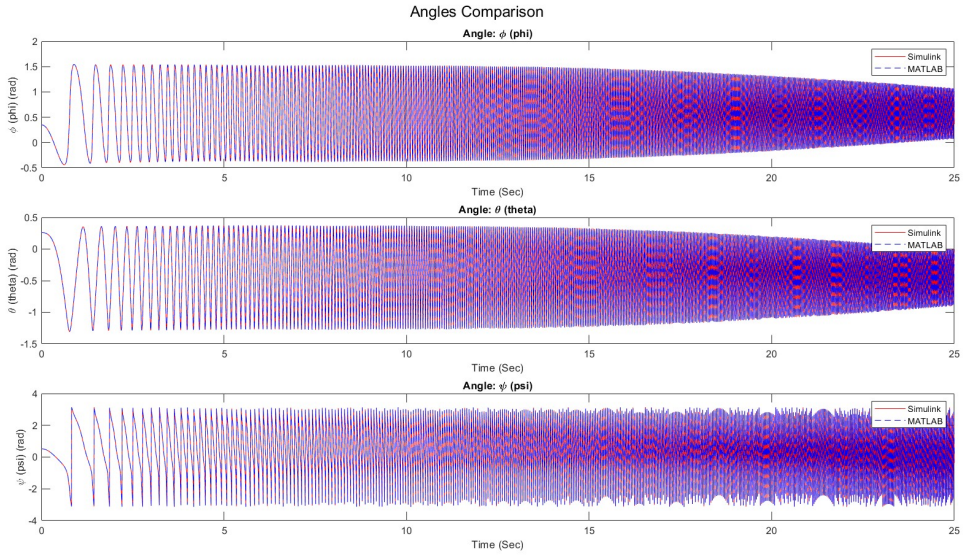


Figure 5: Comparison of aircraft orientation angles (ϕ, θ, ψ) between MATLAB and Simulink results.

Angles: The second plot shows the evolution of the Euler angles ϕ (roll), θ (pitch), and ψ (yaw) over time. The oscillations in the angles during the early phase of the simulation eventually stabilize, reflecting the evolution of the body's rotational dynamics. Once again, the results from both Simulink and MATLAB are in excellent agreement, which validates the correctness of the transformations applied in both methods.

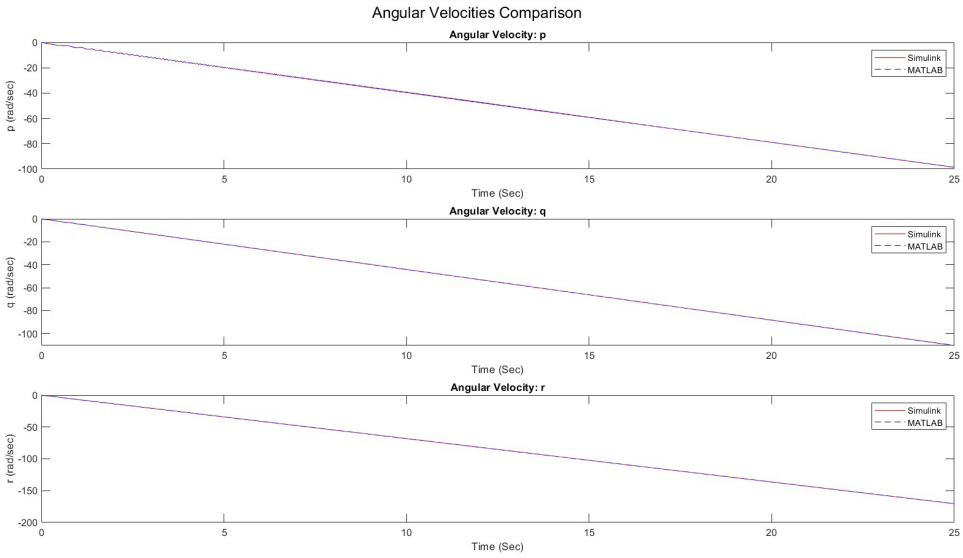


Figure 6: Comparison of aircraft angular velocity components (p, q, r) between MATLAB and Simulink results

Angular Velocities: The third plot depicts the time history of angular velocities p, q, and r. These values decrease over time in a nearly linear fashion. Both Simulink and MATLAB produce identical results, confirming that the inertial dynamics have been properly modeled and integrated using the RK4 method.

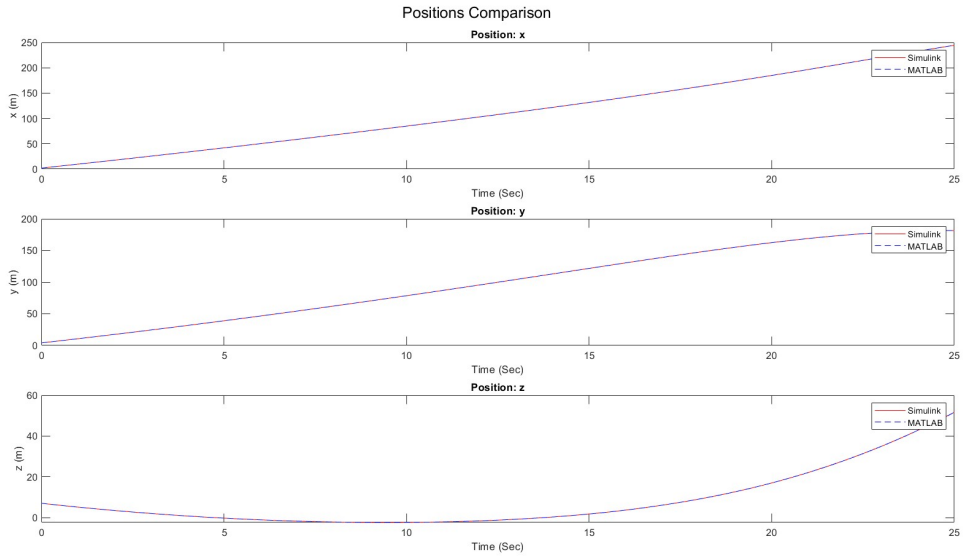


Figure 7: Comparison of aircraft position components (x, y, z) between MATLAB and Simulink results.

Positions: The final plot illustrates the evolution of the position coordinates x, y, and z over time. The positions show a continuous trend, indicating that the body is following a well-defined trajectory.

5 Airplane Simulator Part II

In the continuation of the airplane simulator project, we now focus on incorporating pilot input signals to calculate the aerodynamic and thrust forces and moments acting on the airplane. These calculations will then be used to build the complete non-linear flight simulator by combining them with the Rigid Body Dynamics (RBD) solver developed earlier. The simulator will evaluate the aircraft's behavior in response to control surface deflections and thrust variations.

5.1 Forces and Moments Calculation

The calculation of aerodynamic and thrust forces, as well as moments acting on an airplane, is a critical aspect of the airplane's response to control surface inputs such as aileron, rudder, elevator, and thrust adjustments. These forces and moments result from changes in airflow over the aircraft and engine output, and their effects on the aircraft's dynamics are governed by stability and control derivatives. The stability derivatives describe how small changes in state variables, such as velocity or angular velocity, influence the aerodynamic forces and moments.

5.1.1 Stability and Control Derivatives

The aircraft's response to control inputs is modeled using stability and control derivatives, which are essentially sensitivity factors that describe how the forces and moments change with respect to changes in the aircraft's state or control inputs. These derivatives are usually obtained from wind tunnel tests or flight data and are listed in tables such as those found in the *NASA CR-2144* report.
For example:

- **Stability derivatives** ($\frac{\partial X}{\partial u}$, $\frac{\partial Y}{\partial v}$, etc.) indicate how forces and moments change due to variations in the aircraft's velocities (e.g., longitudinal, lateral, vertical).

- **Control derivatives** ($\frac{\partial X}{\partial \delta_e}$, $\frac{\partial Y}{\partial \delta_r}$, $\frac{\partial Z}{\partial \delta_a}$) show how forces and moments change in response to pilot inputs (e.g., elevator, rudder, and aileron deflections).

5.1.2 Perturbation from Trim Condition

When the aircraft is in steady flight, known as the trim condition, forces and moments are balanced. Any deviation from this steady flight due to control inputs or changes in velocity leads to perturbations in the aerodynamic and thrust forces and moments. These perturbations are calculated using linearized models around the trim point, as follows:

$$\Delta X = \frac{\partial X}{\partial u}\Delta u + \frac{\partial X}{\partial w}\Delta w + \frac{\partial X}{\partial \delta_e}\Delta \delta_e + \frac{\partial X}{\partial \delta_T}\Delta \delta_T$$

$$\Delta Y = \frac{\partial Y}{\partial v}\Delta v + \frac{\partial Y}{\partial p}\Delta p + \frac{\partial Y}{\partial r}\Delta r + \frac{\partial Y}{\partial \delta_r}\Delta \delta_r$$

$$\Delta Z = \frac{\partial Z}{\partial u}\Delta u + \frac{\partial Z}{\partial w}\Delta w + \frac{\partial Z}{\partial \dot{q}}\Delta \dot{q} + \frac{\partial Z}{\partial q}\Delta q + \frac{\partial Z}{\partial \delta_e}\Delta \delta_e + \frac{\partial Z}{\partial \delta_T}\Delta \delta_T$$

These equations indicate how the aerodynamic forces in the longitudinal, lateral, and vertical directions change as a result of variations in velocity components (u, v, w) and control surface deflections (δ_e, δ_r, δ_a, δ_T). The perturbation values $\Delta u, \Delta v, \Delta w, \ldots$ represent deviations from the values at the trim condition, i.e., the steady flight condition where the forces and moments are balanced.

5.1.3 Moment Calculations

Similar to the aerodynamic forces, the moments acting on the airplane (roll L, pitch M, and yaw N) are calculated as a function of perturbations in the angular velocities (p, q, r) and control surface deflections. These moments determine the rotational behavior of the aircraft and are essential for simulating how the airplane will roll, pitch, or yaw in response to pilot commands.
The equations for the moments are:

$$\Delta L = \frac{\partial L}{\partial v}\Delta v + \frac{\partial L}{\partial p}\Delta p + \frac{\partial L}{\partial r}\Delta r + \frac{\partial L}{\partial \delta_r}\Delta \delta_r + \frac{\partial L}{\partial \delta_a}\Delta \delta_a$$

$$\Delta M = \frac{\partial M}{\partial u}\Delta u + \frac{\partial M}{\partial w}\Delta w + \frac{\partial M}{\partial \dot{q}}\Delta \dot{q} + \frac{\partial M}{\partial q}\Delta q + \frac{\partial M}{\partial \delta_e}\Delta \delta_e + \frac{\partial M}{\partial \delta_T}\Delta \delta_T$$

$$\Delta N = \frac{\partial N}{\partial v}\Delta v + \frac{\partial N}{\partial p}\Delta p + \frac{\partial N}{\partial r}\Delta r + \frac{\partial N}{\partial \delta_r}\Delta \delta_r + \frac{\partial N}{\partial \delta_a}\Delta \delta_a$$

In these equations:

- ΔL represents the perturbation in roll moment, influenced by changes in lateral velocity (v), roll rate (p), yaw rate (r), and control inputs such as rudder and aileron deflections.
- ΔM refers to the perturbation in pitch moment, which is driven by longitudinal velocity (u), vertical velocity (w), pitch rate (q), and elevator deflection.
- ΔN captures the perturbation in yaw moment, depending on lateral velocity, roll and yaw rates, and control inputs.

5.1.4 Absolute Force and Moment Calculation

The perturbations in forces and moments calculated above are not the final forces acting on the airplane. These perturbations must be added to the forces and moments at the trim condition (steady flight) to obtain the total forces and moments. For example, the total longitudinal force X is given by:

$$X = X_0 + \Delta X = \Delta X + mg\sin(\theta_0)$$

Here, X_0 is the trim force in the longitudinal direction, and ΔX is the perturbation calculated using the derivatives. Similarly, the lateral and vertical forces are given by:

$$Y = Y_0 + \Delta Y = \Delta Y - mg\cos(\theta_0)\sin(\phi_0)$$
$$Z = Z_0 + \Delta Z = \Delta Z - mg\cos(\theta_0)\cos(\phi_0)$$

The total forces F_x, F_y, F_z acting on the airplane are then fed into the RBD equations to update the airplane's position and orientation at each time step in the simulation.

5.1.5 Gravitational Forces

In addition to the aerodynamic and thrust forces, gravitational forces must be included in the total force calculation. Gravity acts on the airplane in a fixed direction relative to the earth, while the aerodynamic and thrust forces act in the body-fixed frame. Therefore, the gravitational forces are projected into the body frame to compute their contribution to the total force acting on the airplane.
The gravitational forces are given by:

$$F_x = mg\sin(\theta)$$
$$F_y = mg\cos(\theta)\sin(\phi)$$
$$F_z = mg\cos(\theta)\cos(\phi)$$

These gravitational forces are added to the aerodynamic and thrust forces before passing them to the RBD solver.

5.2 Integration with RBD Solver

Once the aerodynamic and thrust forces and moments are calculated, these forces must be integrated with the Rigid Body Dynamics (RBD) solver to simulate the aircraft's motion. The RBD solver uses these forces and moments to determine how the airplane's orientation, velocity, and position evolve over time. The integration of these forces with the RBD solver is essential to accurately model the nonlinear flight dynamics of the airplane.

5.2.1 Total Forces Acting on the Airplane

The total forces acting on the airplane consist of three primary components:

1. **Aerodynamic forces**: These are generated by the airplane's interaction with the airflow, which depend on control surface deflections (e.g., aileron, rudder, elevator) and the aircraft's velocity components (e.g., longitudinal, lateral, vertical).
2. **Thrust force**: This is produced by the aircraft's engines, typically aligned with the longitudinal axis of the airplane. Thrust is influenced by the pilot's throttle input.
3. **Gravitational force**: This is the weight of the airplane acting in the vertical direction relative to the Earth's frame of reference, which changes its direction when projected into the body frame due to the aircraft's orientation.

These total forces are then used to compute the translational motion of the airplane by solving Newton's second law for each axis in the airplane's body frame.
The total forces in the longitudinal (X), lateral (Y), and vertical (Z) directions are given by the following equations:

$$F_x = X - mg\sin(\theta) = \Delta X + mg\sin(\theta_0) - mg\sin(\theta)$$
$$F_y = Y + mg\cos(\theta)\sin(\phi) = \Delta Y + mg\cos(\theta)\sin(\phi)$$
$$F_z = Z + mg\cos(\theta)\cos(\phi) = \Delta Z - mg\cos(\theta_0)\cos(\phi_0) + mg\cos(\theta)\sin(\phi)$$

In these equations:

- X, Y, Z are the aerodynamic forces acting in the body frame.
- F_x, F_y, F_z are the total forces acting on the airplane, which include gravitational forces.
- mg represents the airplane's weight, where m is the mass and g is the gravitational acceleration.
- θ and ϕ are the pitch and roll angles of the airplane, respectively.
- $\Delta X, \Delta Y, \Delta Z$ represent the perturbations in the aerodynamic forces from the trim condition.

5.2.2 Equations of Motion for Translation

The translational equations of motion are derived from Newton's second law and account for both the external forces and the airplane's mass.The Aerodynamic Forces F_x, F_y, F_z are used to calculate the change in velocity components u, v, and w (longitudinal, lateral, and vertical velocities in the body frame)

5.2.3 Equations of Motion for Rotation

The rotational dynamics of the airplane are determined by the moments acting on the airplane and are described by the rotational form of Newton's second law. The moments L, M, N (roll, pitch, and yaw moments) are used to calculate the angular accelerations.

5.2.4 Integration Using Runge-Kutta 4th Order (RK4) Method

The translational and rotational equations of motion, derived from the forces and moments, are integrated over time using the Runge-Kutta 4th order (RK4) method. The RK4 method is a numerical integration technique that provides a good balance between computational efficiency and accuracy, particularly for nonlinear differential equations such as the RBD equations.
At each time step, the RK4 method computes the next state (positions, velocities, angular velocities) of the airplane by solving the following system of equations iteratively:

1. Compute the forces and moments based on the current state of the airplane and the pilot's inputs.
2. Use these forces and moments to solve the equations of motion for translational and rotational accelerations.
3. Update the state of the airplane (position, velocity, orientation) by integrating these accelerations using the RK4 method.
4. Repeat this process for each time step until the desired simulation time is reached.

The RK4 method's iterative process ensures that the nonlinear dynamics of the airplane are captured accurately over time, allowing for realistic predictions of how the airplane responds to pilot inputs.

5.2.5 Converting Between Body-Fixed and Inertial Frames

The integration process involves updating the airplane's position and orientation in the inertial (earth-fixed) frame. However, since the forces and moments are computed in the body-fixed frame, a transformation is necessary to convert these values between frames. The transformation matrix $[T]_{EB}$ relates the body-fixed velocities to the inertial frame:

$$\begin{bmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{bmatrix} = [T]_{EB} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

Where:

- $\dot{x}_E, \dot{y}_E, \dot{z}_E$ are the velocity components in the inertial frame.
- u, v, w are the velocity components in the body frame.
- $[T]_{EB}$ is the transformation matrix that accounts for the airplane's orientation in space, given by the Euler angles ϕ, θ, ψ (roll, pitch, yaw).

This transformation ensures that the airplane's position and velocity in the inertial frame are updated correctly during the simulation.

5.3 Validation

The validation process is essential to ensure the accuracy of the non-linear airplane flight simulator developed.

5.3.1 Validation Using Benchmark Test Data

The first key validation method involves comparing the simulator results with benchmark test data for the same aircraft under similar conditions. Benchmark Test data for the B-747 aircraft at Flight Condition 5 is used. The benchmark test includes response plots for various inputs, such as aileron, rudder, and elevator deflections. These plots provide a standard for validating the simulator's results under similar conditions.
The simulator results are compared against these benchmark plots to evaluate how well the non-linear flight dynamics model captures the real-world behavior of the aircraft. The following variables are validated:

- Linear velocities: u, v, and w (in the body frame).
- Angular velocities: p, q, and r (roll, pitch, and yaw rates).
- Euler angles: ϕ, θ, and ψ (roll, pitch, and yaw angles).
- Trajectory: Position coordinates x_E, y_E, and z_E (in the inertial frame).

5.3.2 Validation with Matlab Code

The following key cases are used to compare the simulator's outputs against the benchmark data:

- **Response for no input**: No affects induced, all plots show excellent agreement.

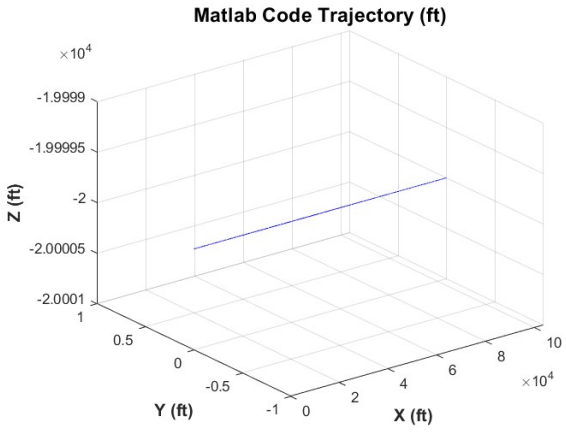


Figure 8: Matlab Simulator's plot

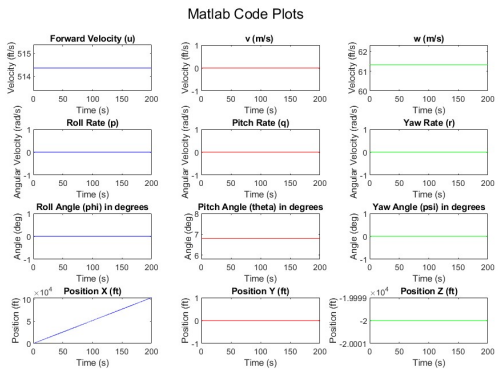


Figure 9: Matlab Simulator's Plots

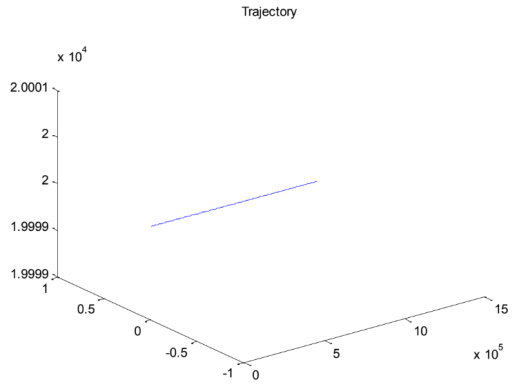


Figure 10: Reference (Benchmark) Trajectory Plot

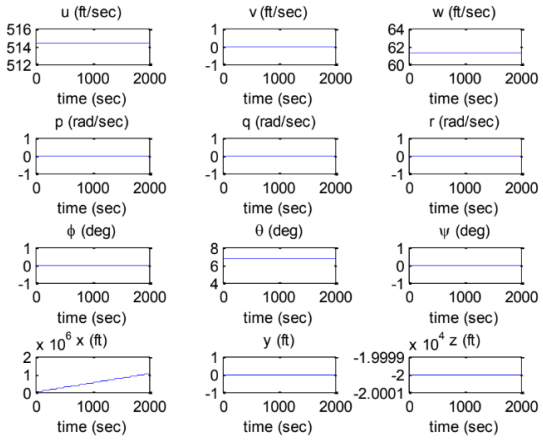


Figure 11: Reference Plots

- **+5° Aileron Deflection**: Primarily affects the roll rate (p) and yaw rate (r), inducing a rolling motion with slight yaw coupling.

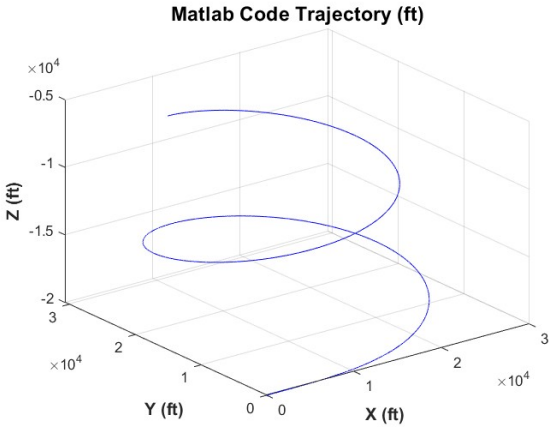


Figure 12: Simulator's Trajectory plot

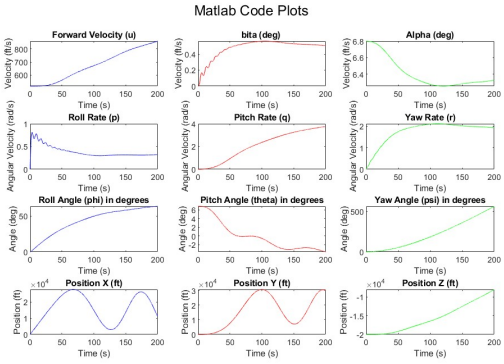


Figure 13: Simulator's Plots

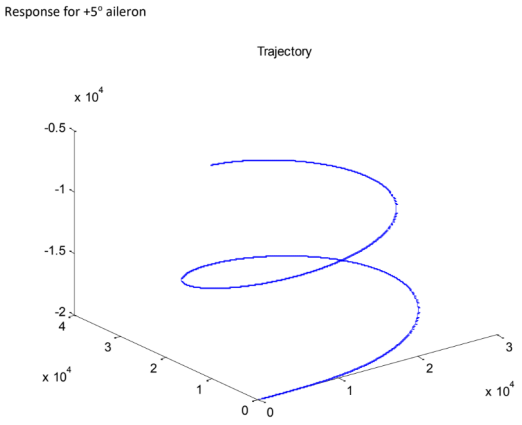


Figure 14: Reference (Benchmark) Trajectory Plot

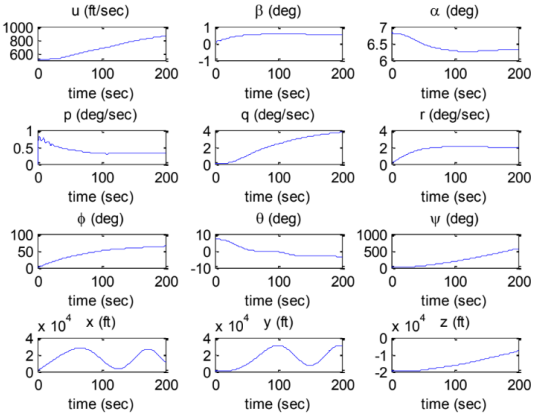


Figure 15: Reference Plots

- **-5° Aileron Deflection**: Primarily affects the roll rate (p) and yaw rate (r), inducing a rolling motion with slight yaw coupling.

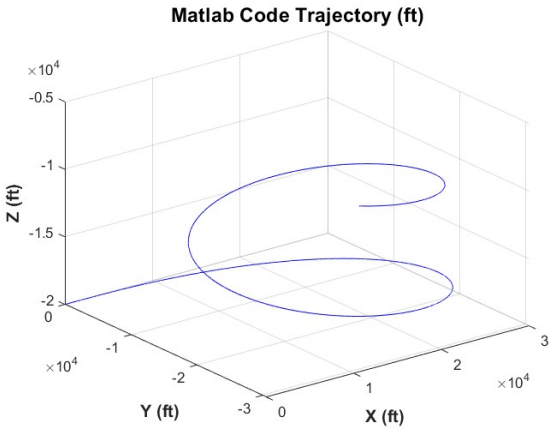


Figure 16: Simulator's Trajectory plot

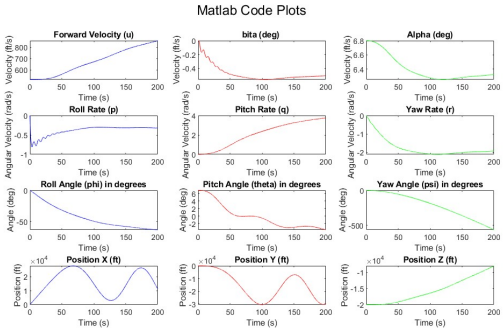


Figure 17: Simulator's Plots

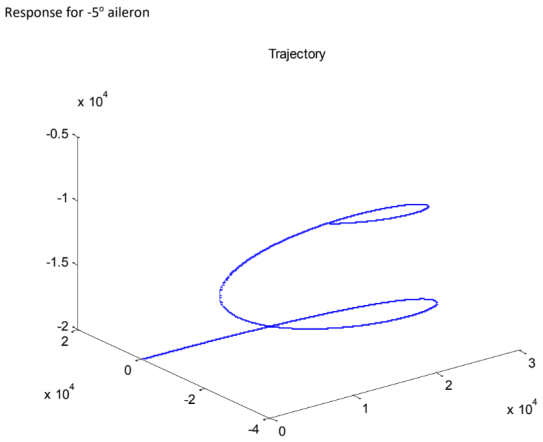


Figure 18: Reference (Benchmark) Trajectory Plot

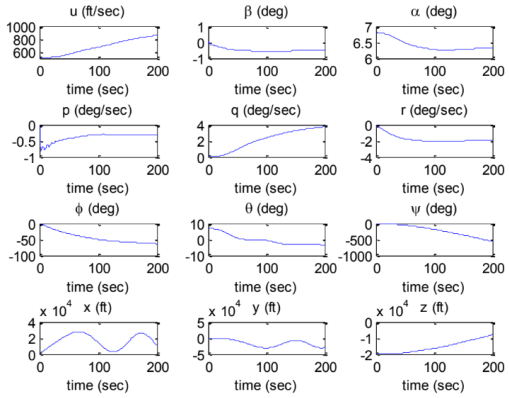


Figure 19: Reference Plots

- **+5° Rudder Deflection**: Primarily affects the yaw rate (r) and lateral velocity (v), inducing a yawing motion.

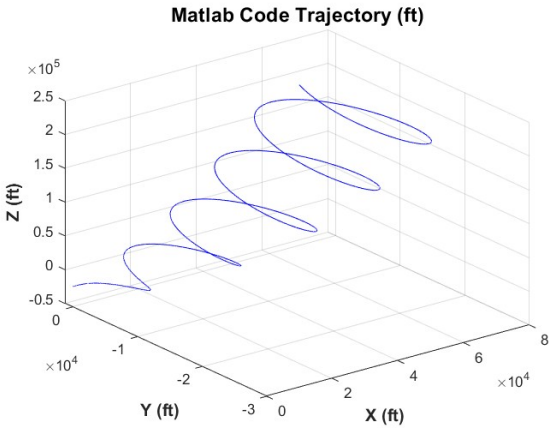


Figure 20: Simulator's Trajectory plot

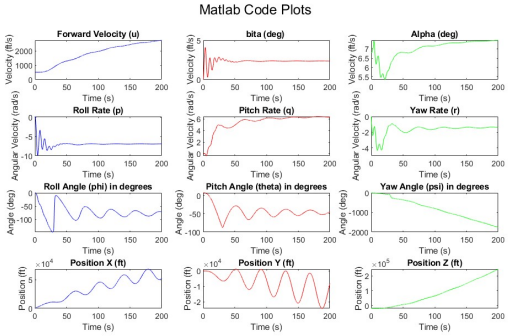


Figure 21: Simulator's Plots

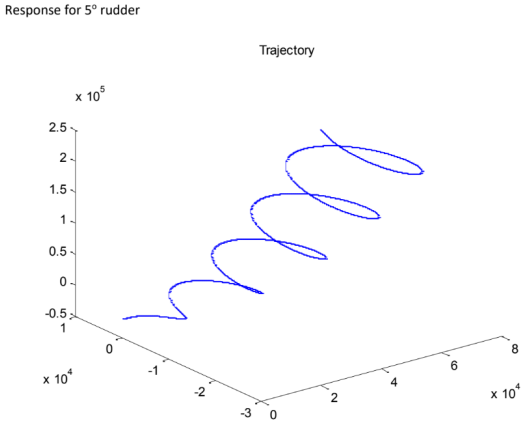


Figure 22: Reference (Benchmark) Trajectory Plot

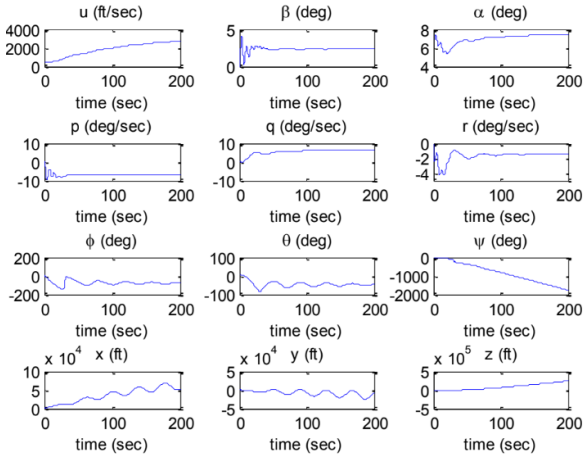


Figure 23: Reference Plots

• **-5° Rudder Deflection**: Primarily affects the yaw rate (r) and lateral velocity (v), inducing a yawing motion.

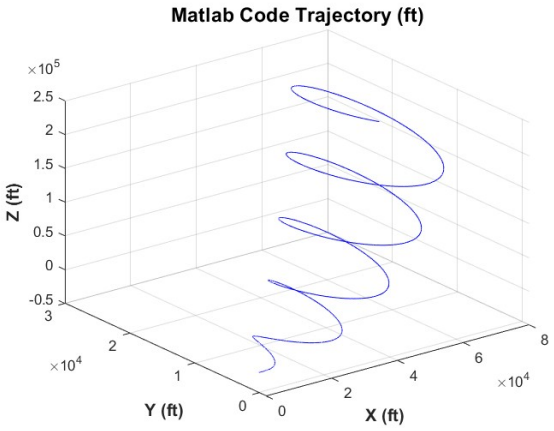


Figure 24: Simulator's Trajectory plot

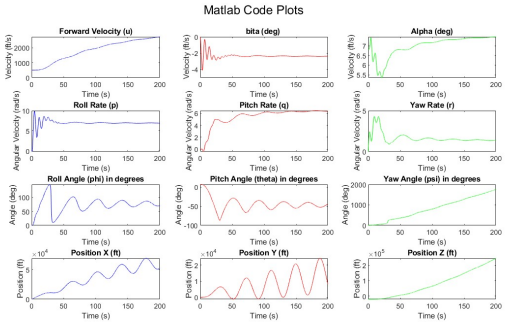


Figure 25: Simulator's Plots

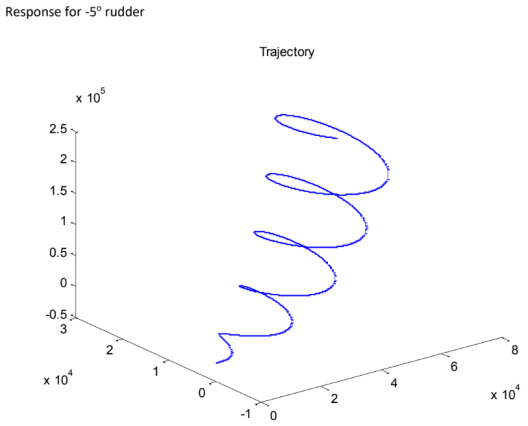


Figure 26: Reference (Benchmark) Trajectory Plot

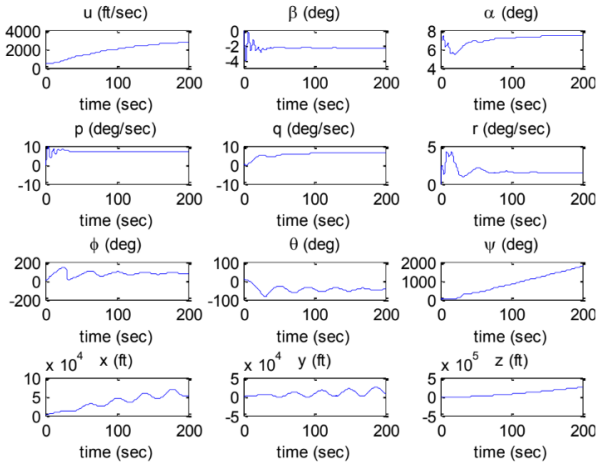


Figure 27: Reference Plots

- **+5° Elevator Deflection**: Primarily affects the pitch rate (q) and vertical velocity (w), inducing a pitching motion.

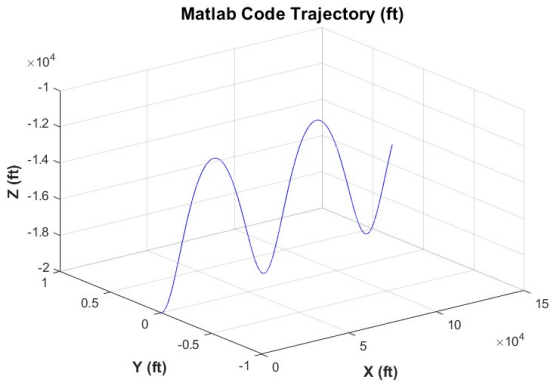


Figure 28: Simulator's Trajectory plot

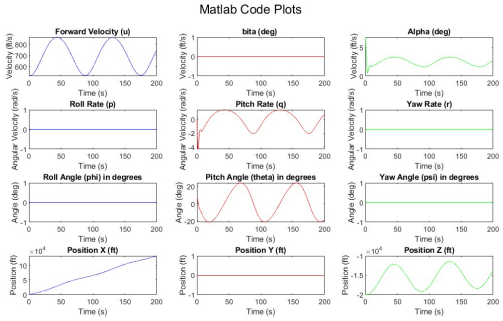


Figure 29: Simulator's Plots

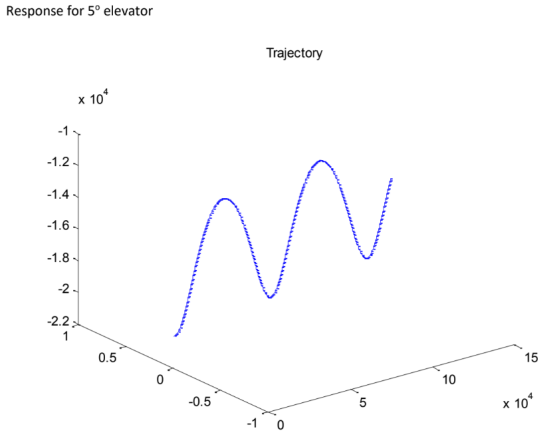


Figure 30: Reference (Benchmark) Trajectory Plot

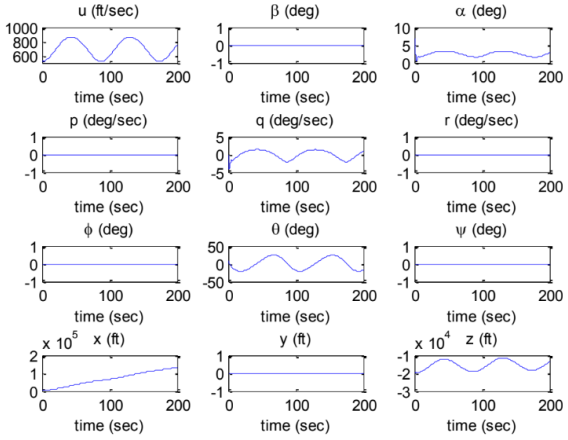


Figure 31: Reference Plots

- **-5° Elevator Deflection**: Primarily affects the pitch rate (q) and vertical velocity (w), inducing a pitching motion.

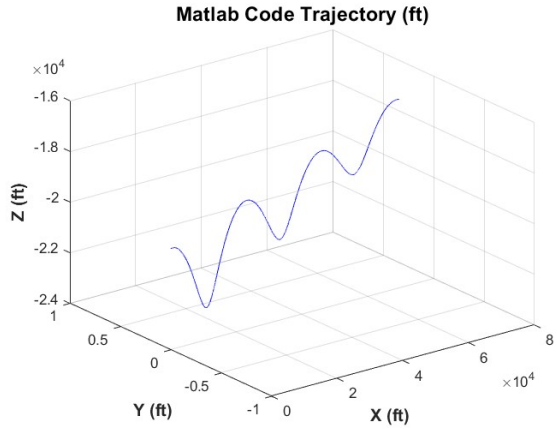


Figure 32: Simulator's Trajectory plot

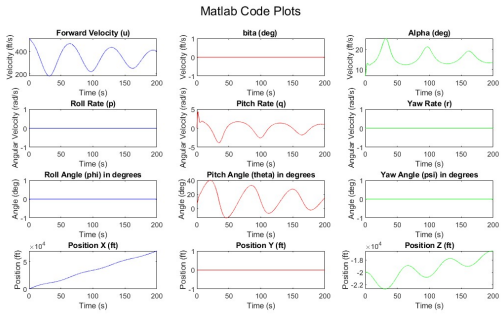


Figure 33: Simulator's Plots

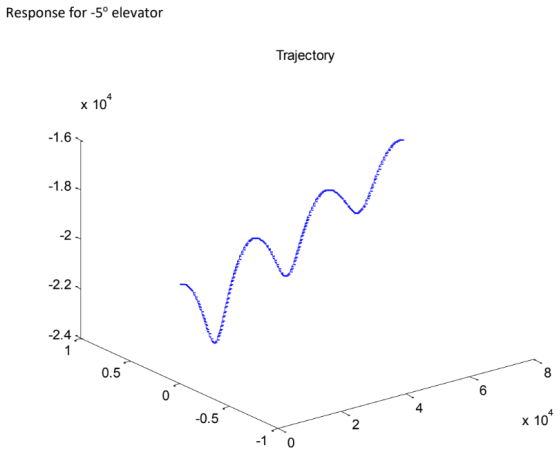


Figure 34: Reference (Benchmark) Trajectory Plot

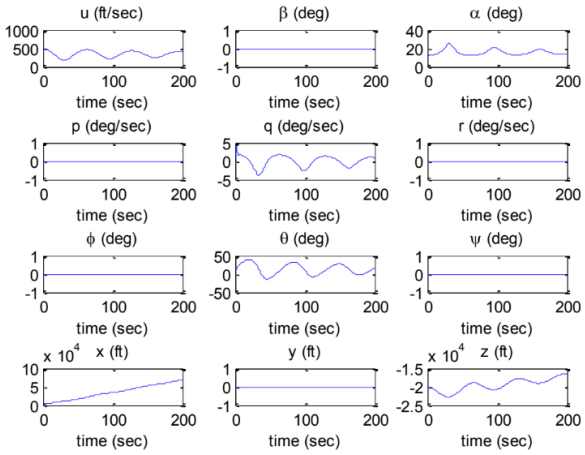


Figure 35: Reference Plots

- **Thrust Variation**: Primarily affects the forward velocity (u).
  - **Response for 10000 in thrust**

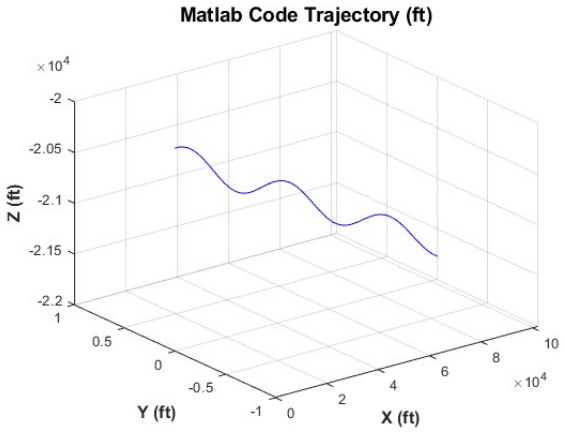


Figure 36: Simulator's Trajectory plot

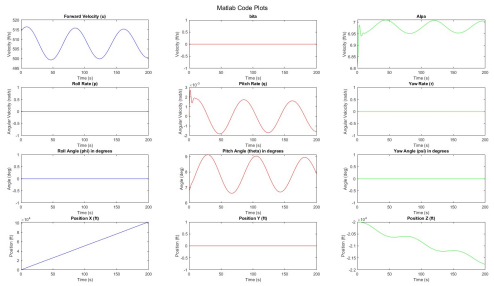


Figure 37: Simulator's Plots

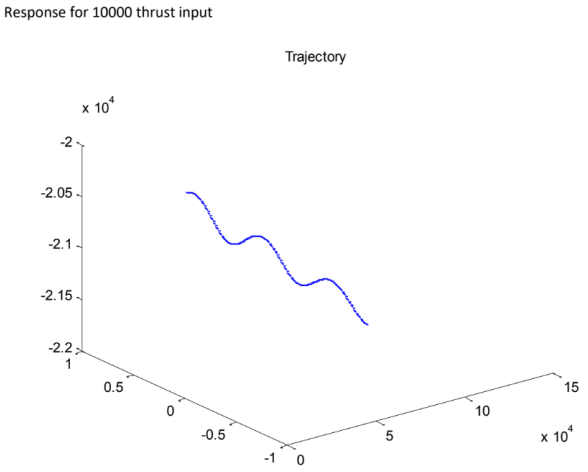


Figure 38: Reference (Benchmark) Trajectory Plot

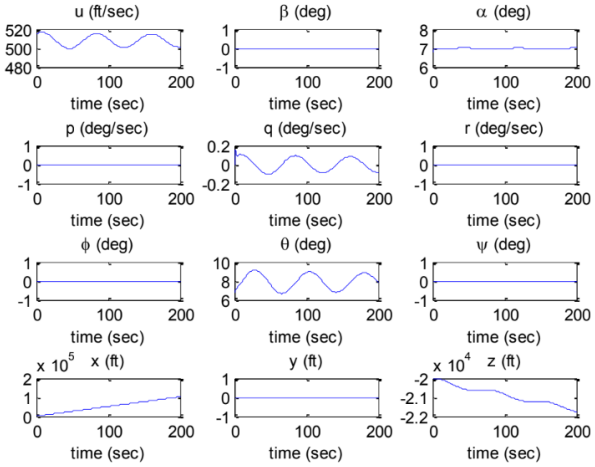


Figure 39: Reference Plots

– **Thrust Variation**: Primarily affects the forward velocity (u).

∗ **Response for 10000 in thrust**

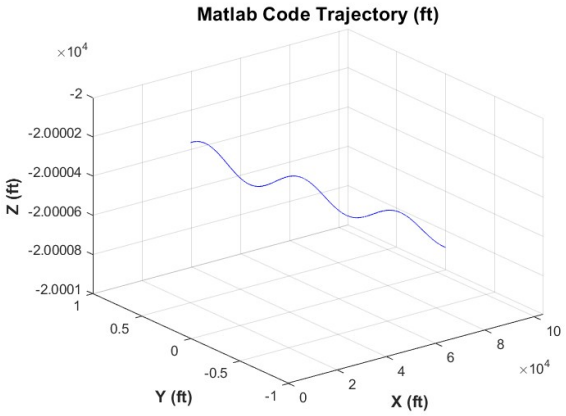


Figure 40: Simulator's Trajectory plot

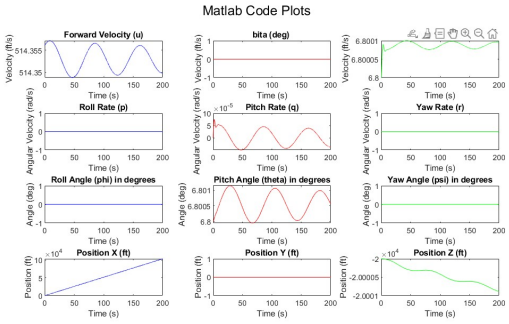


Figure 41: Simulator's Plots

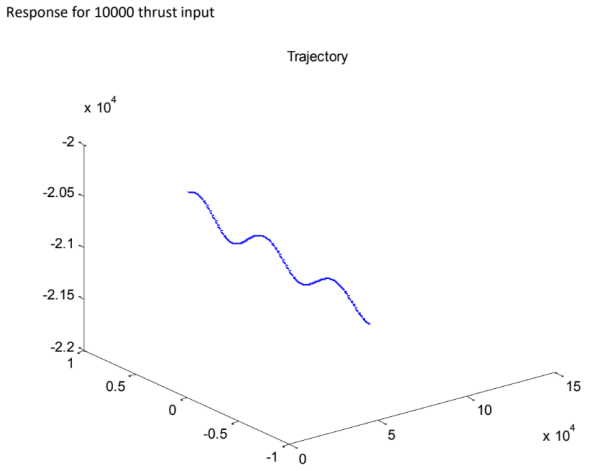


Figure 42: Reference (Benchmark) Trajectory Plot

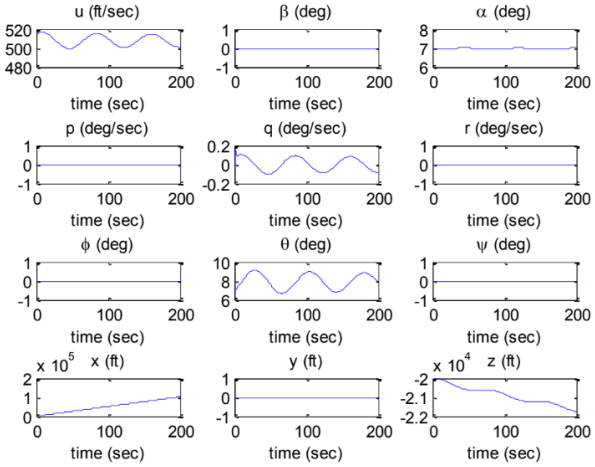


Figure 43: Reference Plots

5.3.3 Validation with Simulink

The following key cases are used to compare the simulator's outputs against the benchmark data:

* **Response for no input**: No affects induced, all plots show excellent agreement.

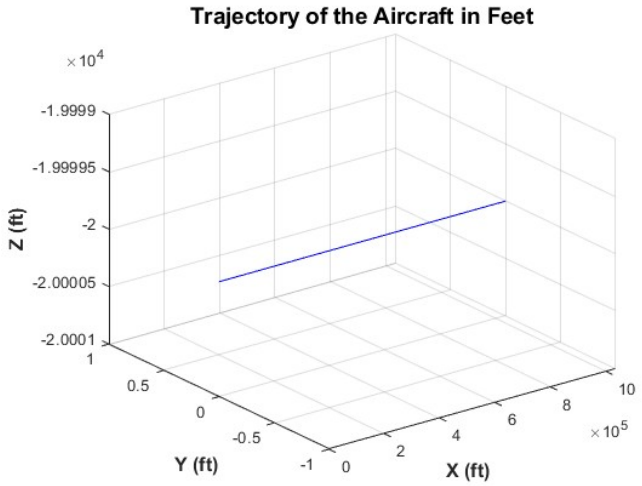


Figure 44: Simulink Simulator's plot

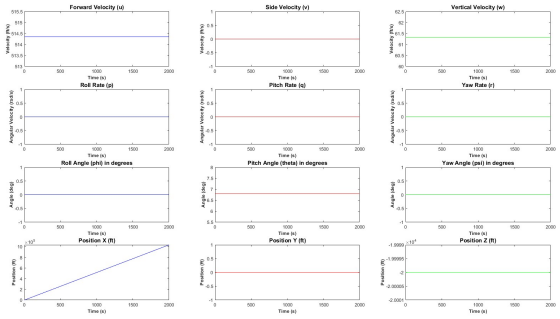


Figure 45: Simulink Simulator's Plots

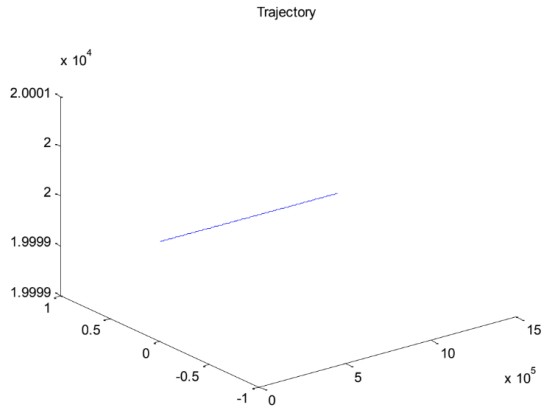


Figure 46: Reference (Benchmark) Trajectory Plot

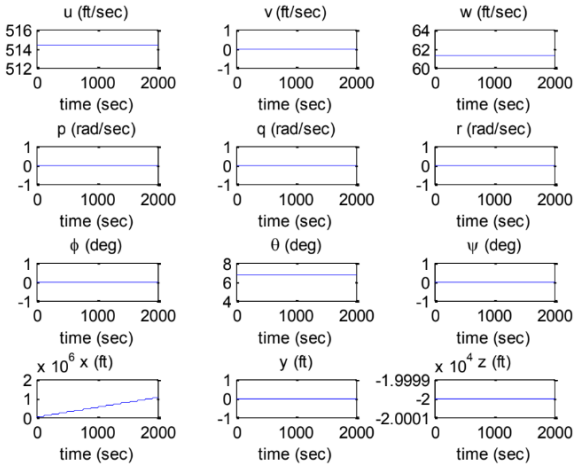


Figure 47: Reference Plots

* **+5° Aileron Deflection**: Primarily affects the roll rate (p) and yaw rate (r), inducing a rolling motion with slight yaw coupling.

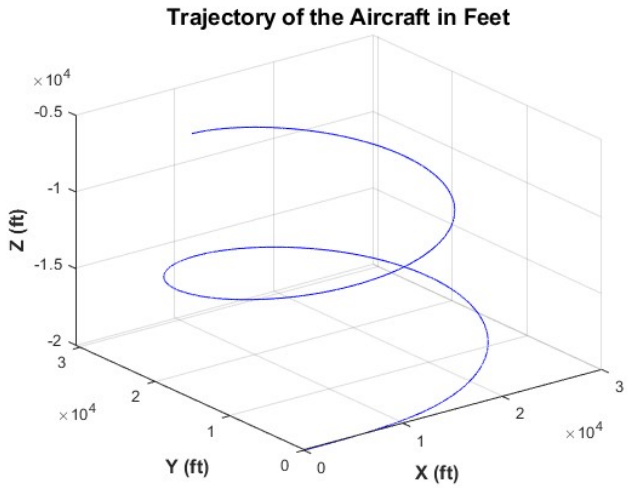


Figure 48: Simulink Simulator's Trajectory plot

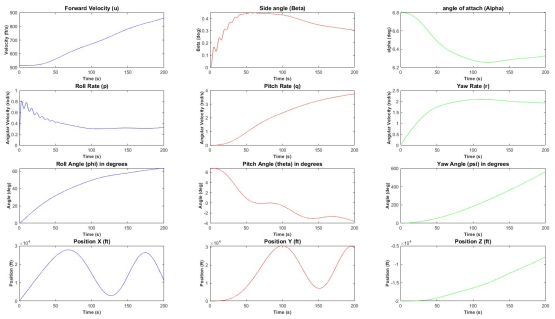


Figure 49: Simulink Simulator's Plots

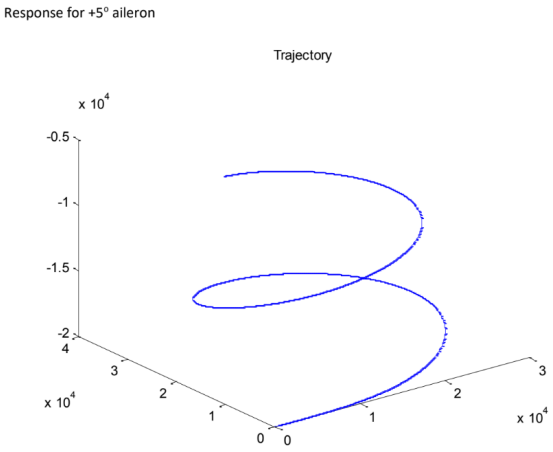


Figure 50: Reference (Benchmark) Trajectory Plot

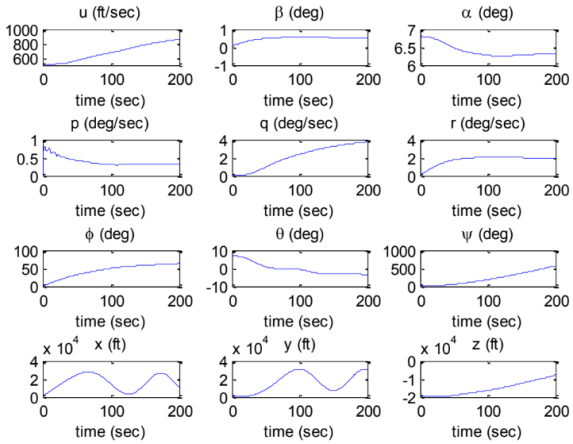


Figure 51: Reference Plots

∗ **-5° Aileron Deflection**: Primarily affects the roll rate (p) and yaw rate (r), inducing a rolling motion with slight yaw coupling.

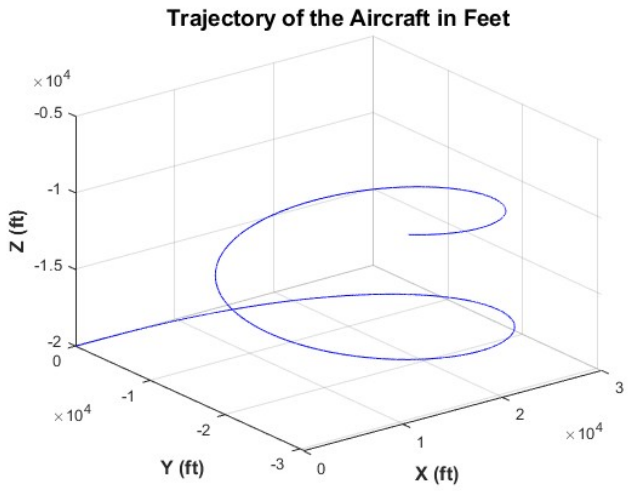


Figure 52: Simulink Simulator's Trajectory plot

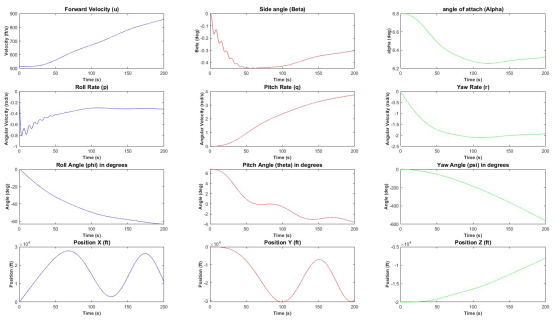


Figure 53: simulink Simulator's Plots

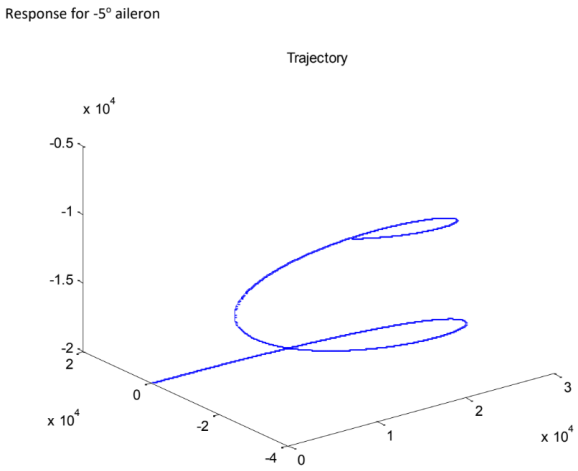


Figure 54: Reference (Benchmark) Trajectory Plot

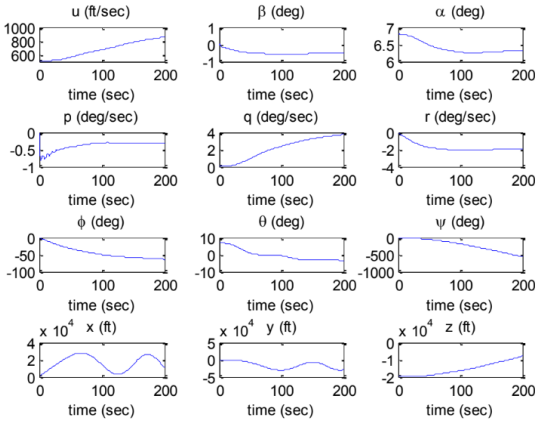


Figure 55: Reference Plots

∗ **+5° Rudder Deflection**: Primarily affects the yaw rate (r) and lateral velocity (v), inducing a yawing motion.

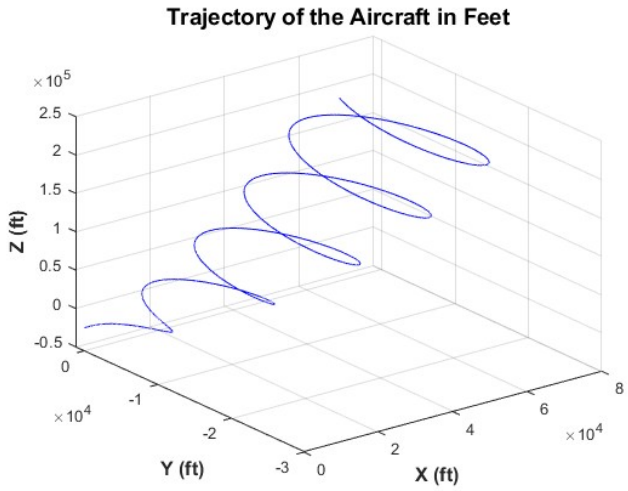


Figure 56: Simulink Simulator's Trajectory plot

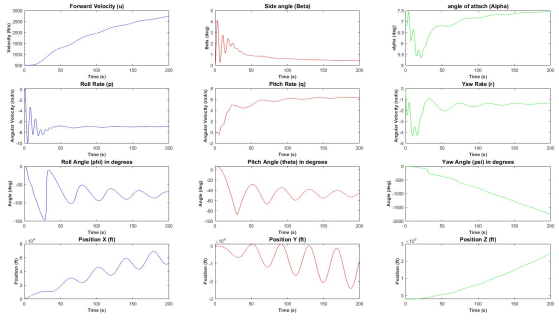


Figure 57: Simulink Simulator's Plots

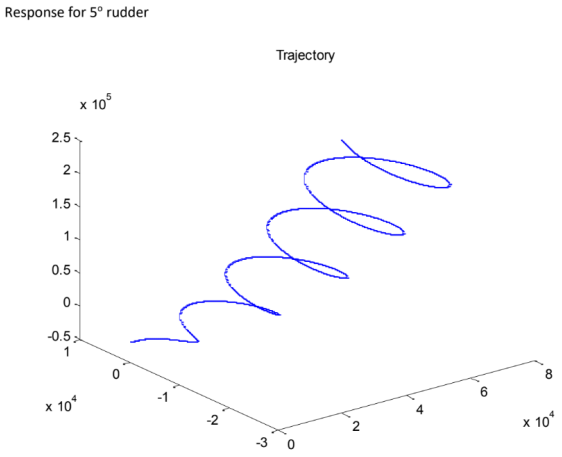


Figure 58: Reference (Benchmark) Trajectory Plot

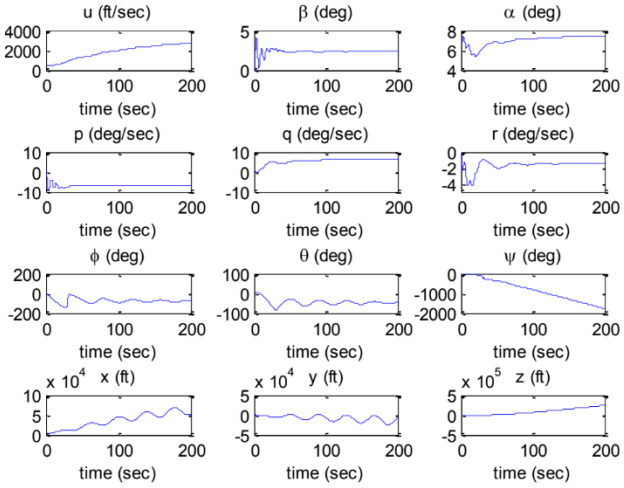


Figure 59: Reference Plots

∗ **-5° Rudder Deflection**: Primarily affects the yaw rate (r) and lateral velocity (v), inducing a yawing motion.

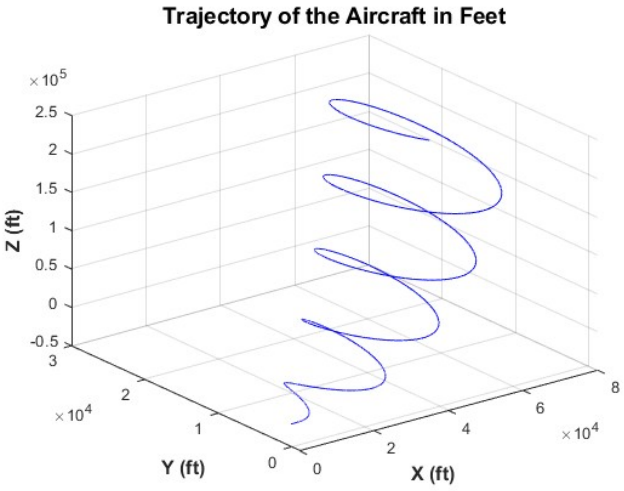


Figure 60: Simulink Simulator's Trajectory plot

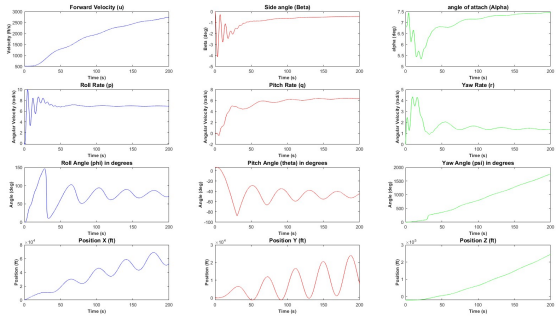


Figure 61: Simulink Simulator's Plots

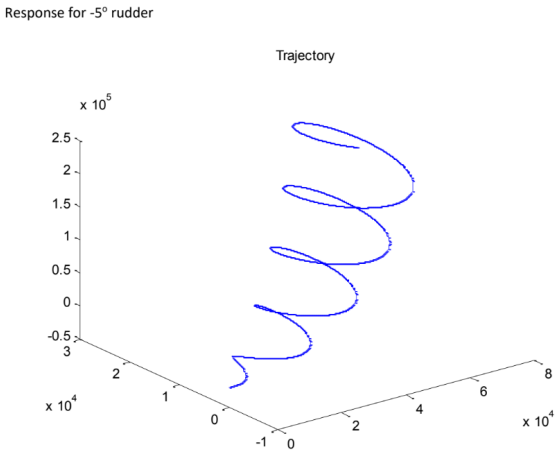


Figure 62: Reference (Benchmark) Trajectory Plot

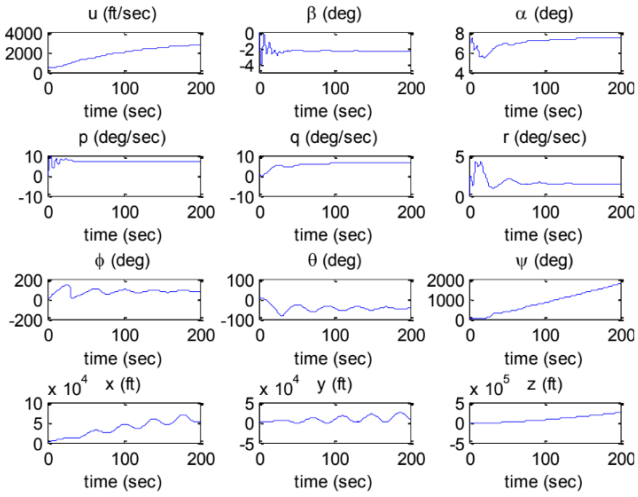


Figure 63: Reference Plots

∗ **+5° Elevator Deflection**: Primarily affects the pitch rate (q) and vertical velocity (w), inducing a pitching motion.

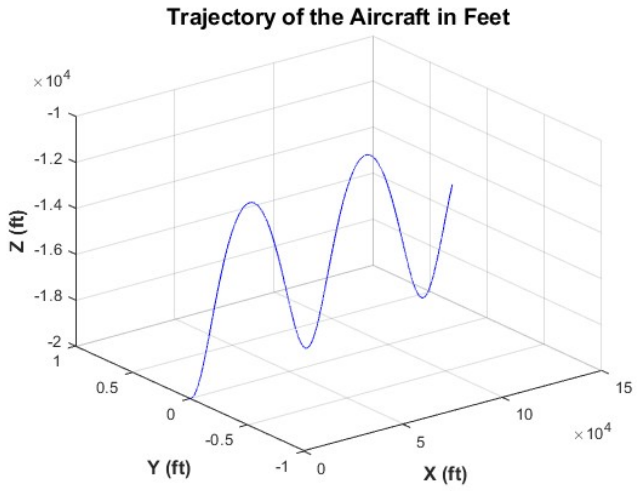


Figure 64: Simulink Simulator's Trajectory plot

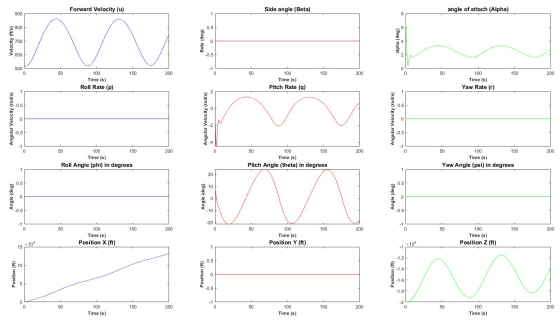


Figure 65: Simulink Simulator's Plots

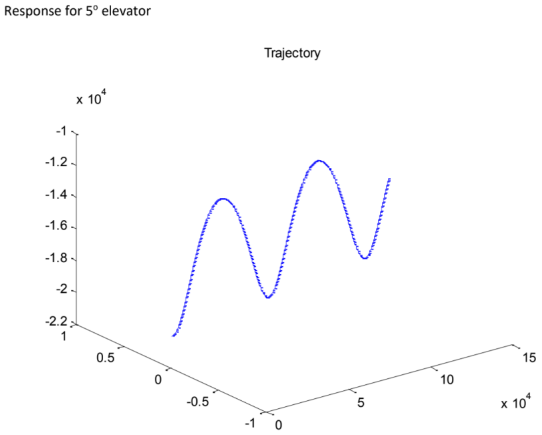


Figure 66: Reference (Benchmark) Trajectory Plot

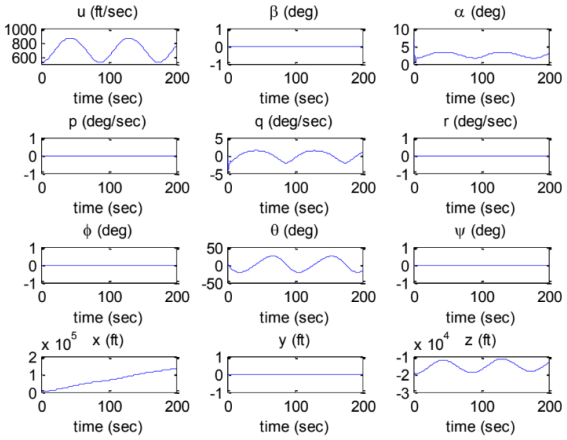


Figure 67: Reference Plots

∗ **-5° Elevator Deflection**: Primarily affects the pitch rate (q) and vertical velocity (w), inducing a pitching motion.

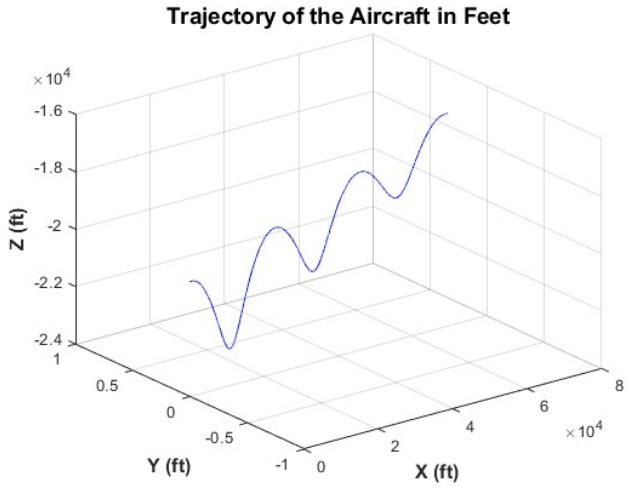


Figure 68: Simulink Simulator's Trajectory plot

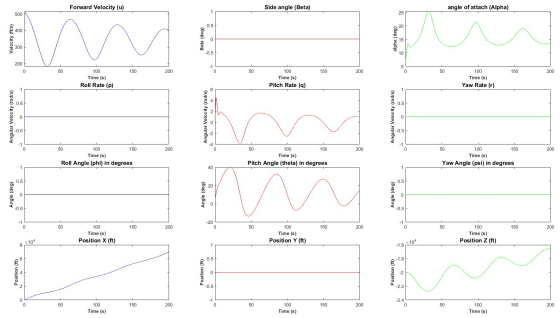


Figure 69: Simulink Simulator's Plots

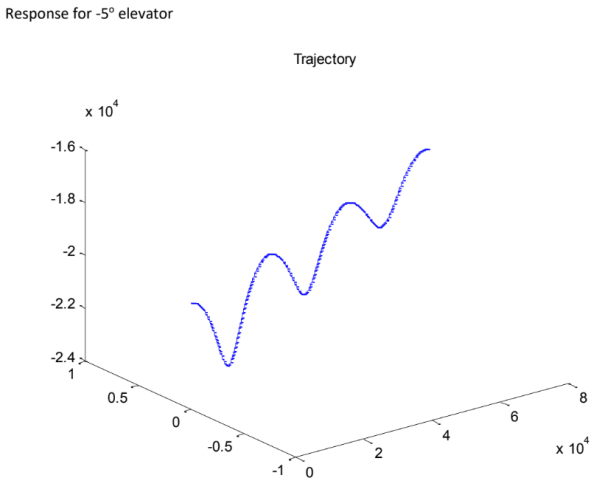


Figure 70: Reference (Benchmark) Trajectory Plot

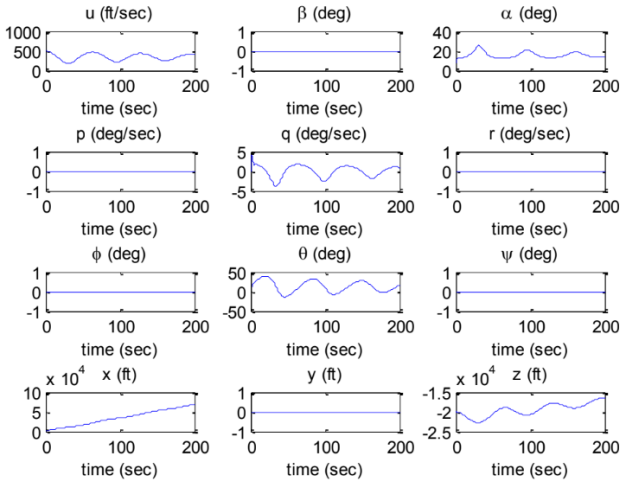


Figure 71: Reference Plots

* **Thrust Variation**: Primarily affects the forward velocity (u).
  - **Response for 1000 in thrust**

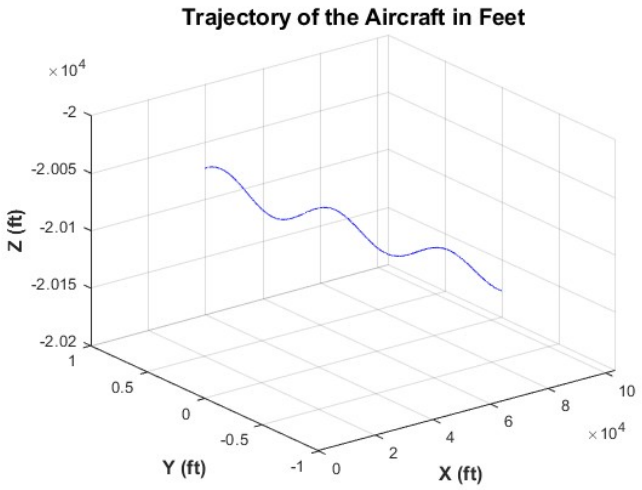


Figure 72: Simulator's Trajectory plot

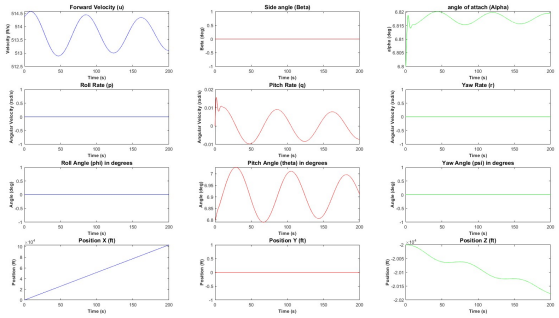


Figure 73: Simulator's Plots

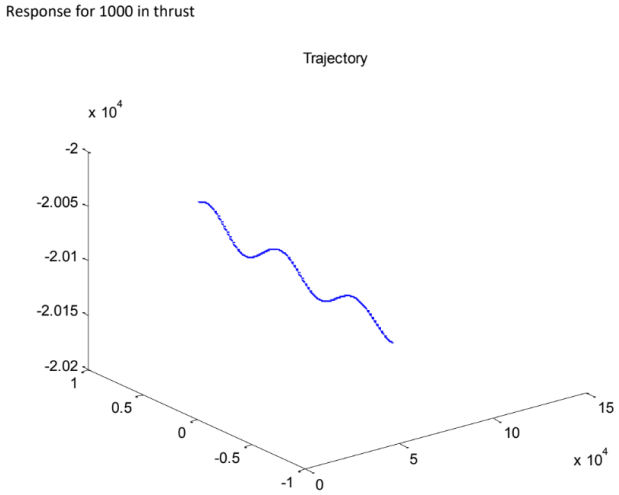


Figure 74: Reference (Benchmark) Trajectory Plot

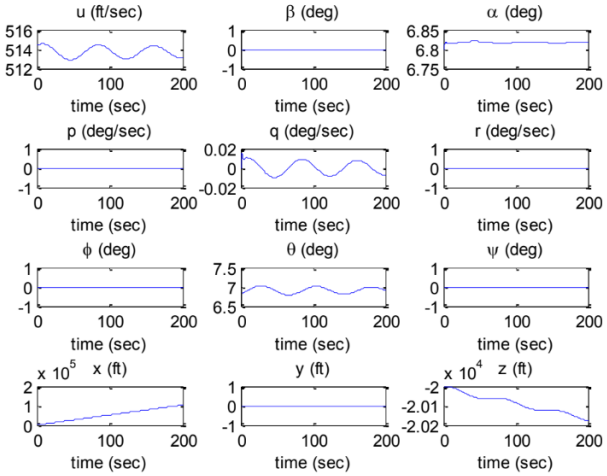


Figure 75: Reference Plots

· **Thrust Variation**: Primarily affects the forward velocity (u).
· **Response for 10000 in thrust**

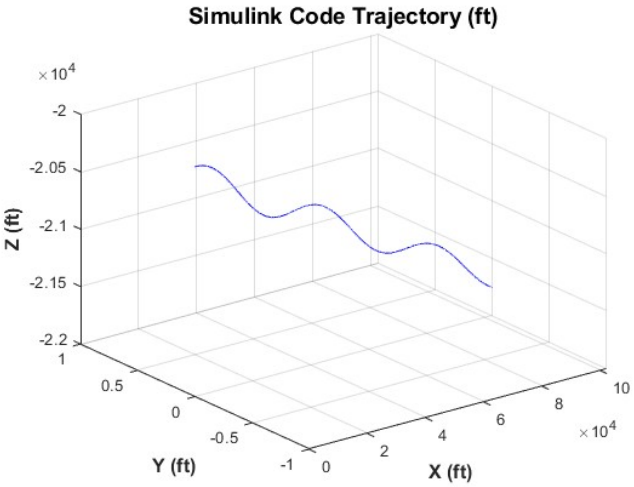


Figure 76: Simulator's Trajectory plot

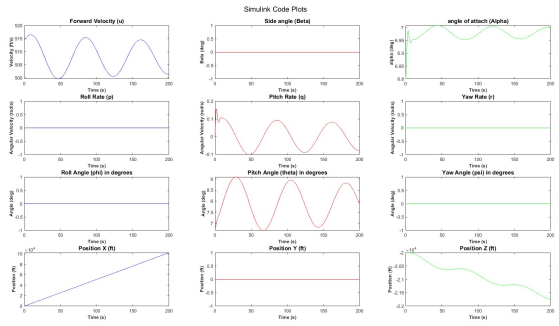


Figure 77: Simulator's Plots

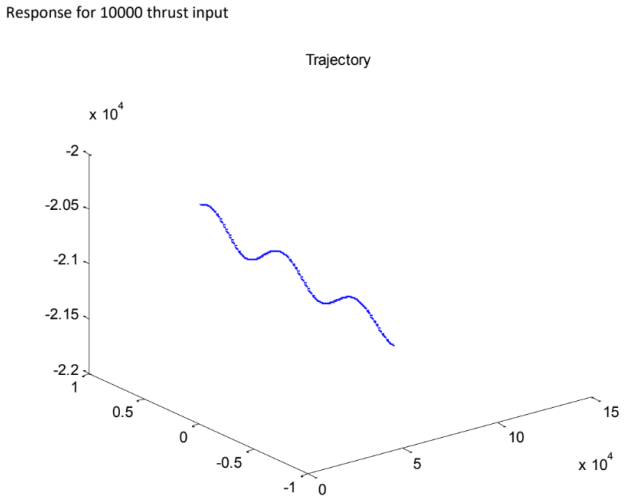


Figure 78: Reference (Benchmark) Trajectory Plot

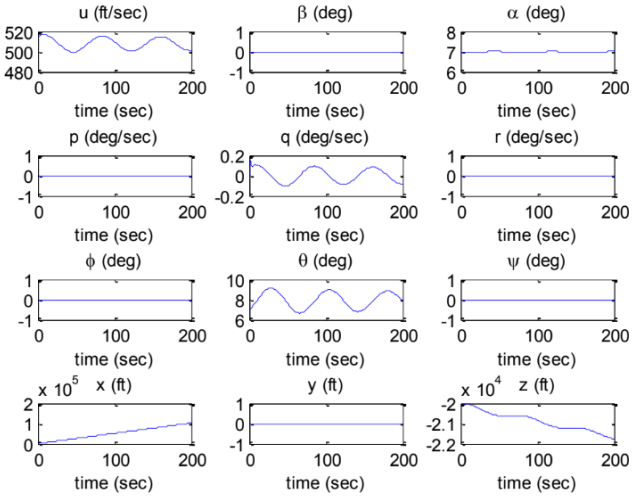


Figure 79: Reference Plots

5.4 B747-FC3 flight Condition

The below are the results for aircraft states at the flight condition number 3 at NASA Report.

- **Response for no input**: No affects induced, all plots show excellent agreement.

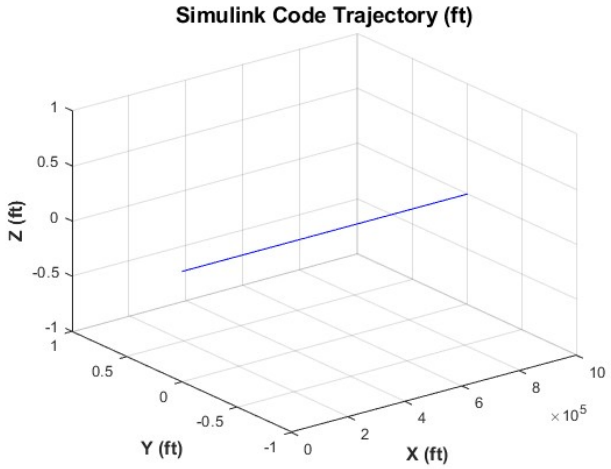


Figure 80: Simulink Trajectory Plot

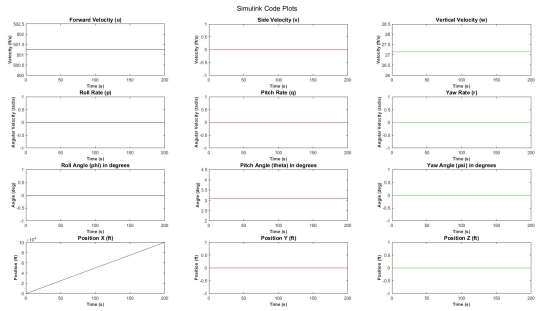


Figure 81: Simulink States Plots

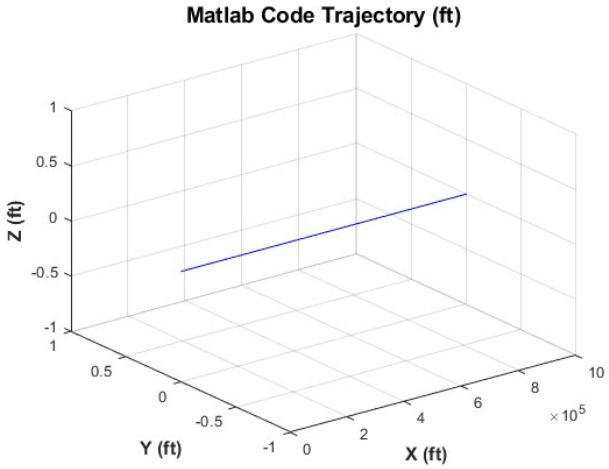


Figure 82: Matlab Trajectory Plot

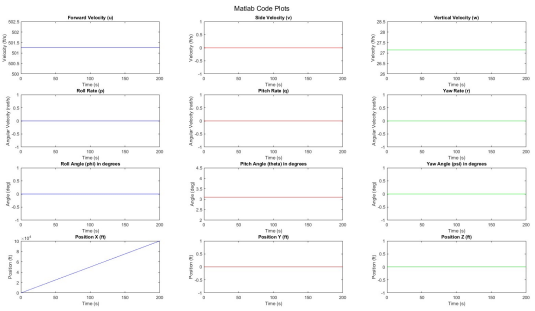


Figure 83: Matlab States Plots

· **+5° Aileron Deflection**: Primarily affects the roll rate (p) and yaw rate (r), inducing a rolling motion with slight yaw coupling.

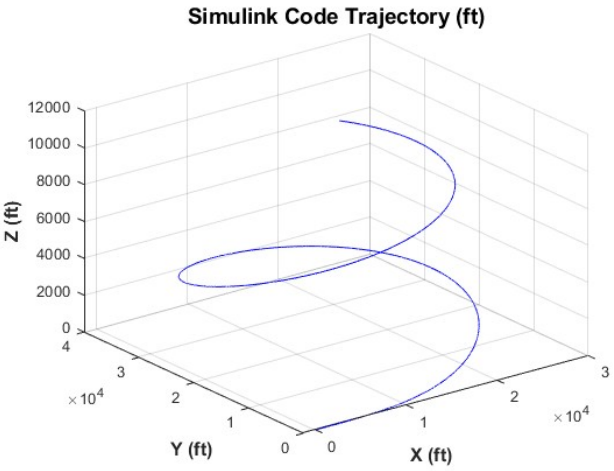


Figure 84: Simulink Trajectory Plot

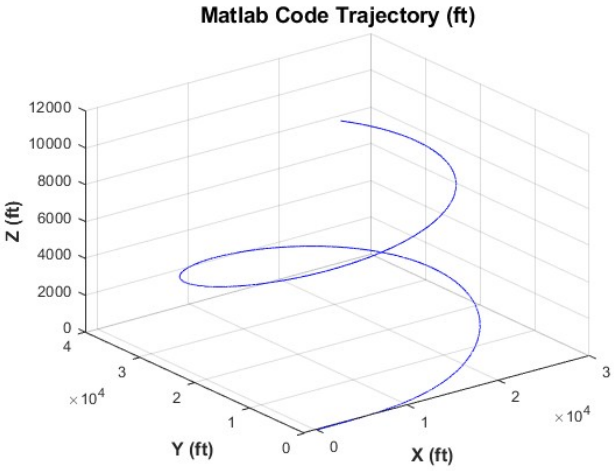


Figure 86: Matlab Trajectory Plot

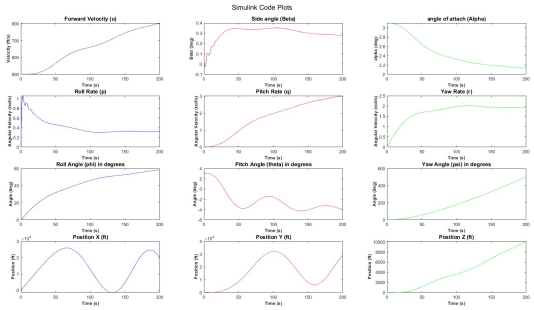


Figure 85: Simulink States Plots

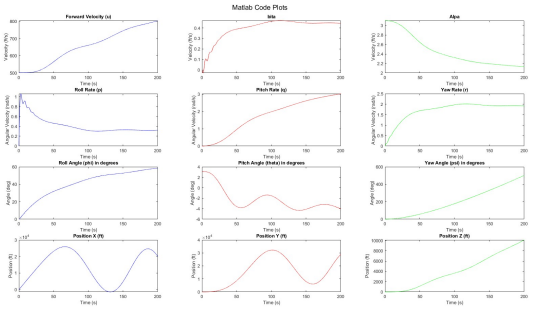


Figure 87: Matlab States Plots

· **-5° Aileron Deflection**: Primarily affects the roll rate (p) and yaw rate (r), inducing a rolling motion with slight yaw coupling.

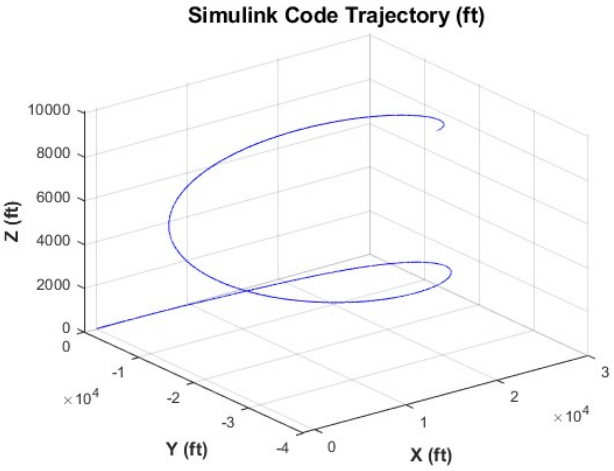


Figure 88: Simulink Trajectory Plot

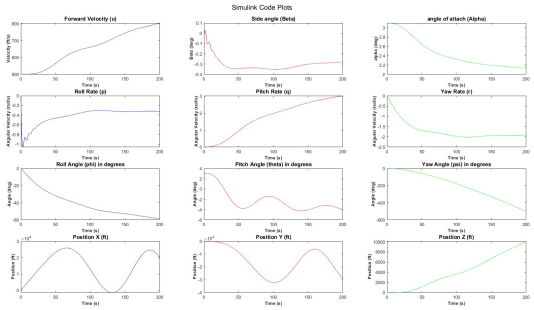


Figure 89: Simulink States Plots

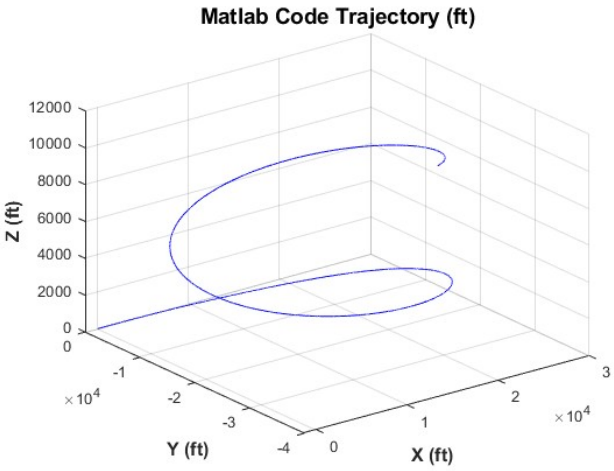


Figure 90: Matlab Trajectory Plot

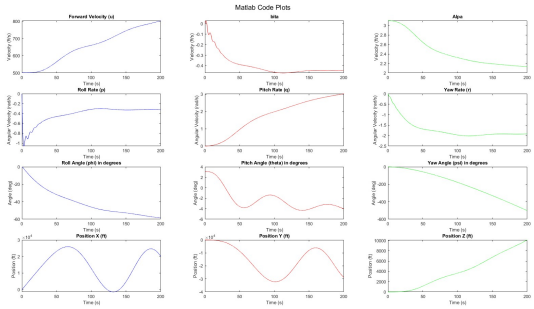


Figure 91: Matlab States Plots

· **+5° Rudder Deflection**: Primarily affects the yaw rate (r) and lateral velocity (v), inducing a yawing motion.

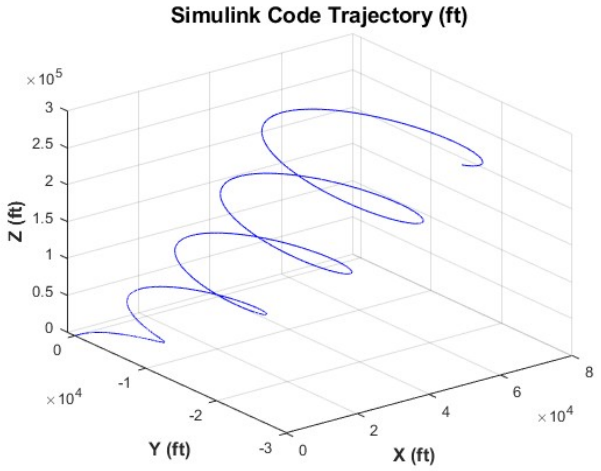


Figure 92: Simulink Trajectory Plot

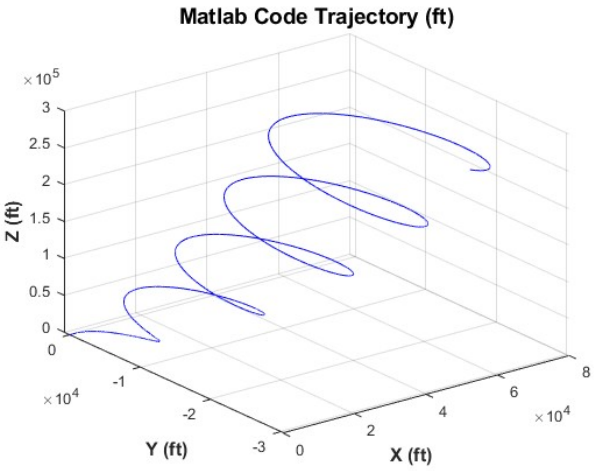


Figure 94: Matlab Trajectory Plot

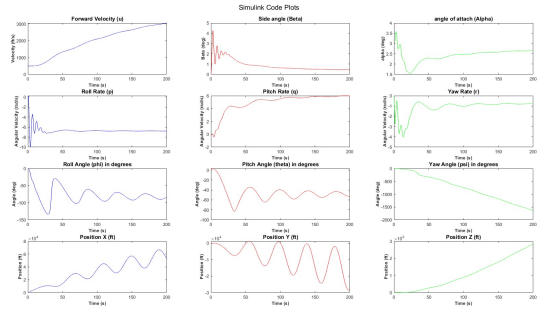


Figure 93: Simulink States Plots

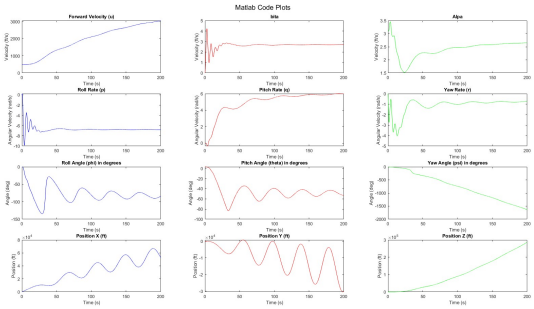


Figure 95: Matlab States Plots

· **-5° Rudder Deflection**: Primarily affects the yaw rate (r) and lateral velocity (v), inducing a yawing motion.

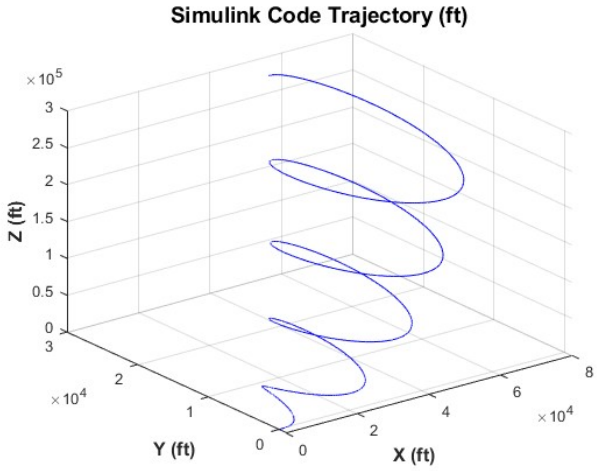


Figure 96: Simulink Trajectory Plot

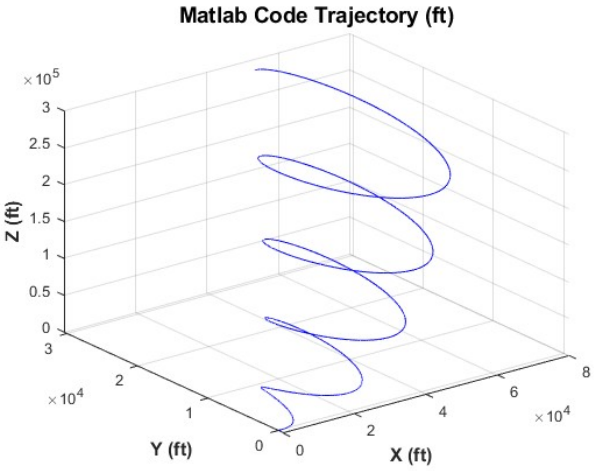


Figure 98: Matlab Trajectory Plot

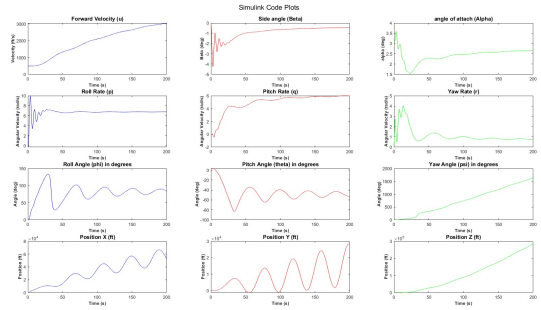


Figure 97: Simulink States Plots

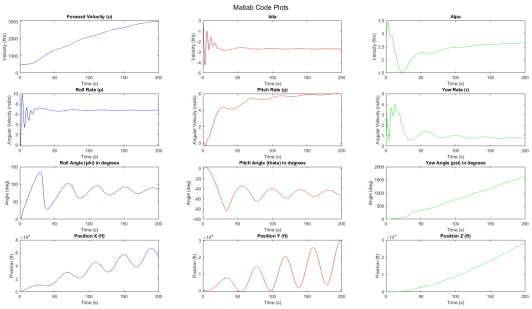


Figure 99: Matlab States Plots

· **+5° Elevator Deflection**: Primarily affects the pitch rate (q) and vertical velocity (w), inducing a pitching motion.

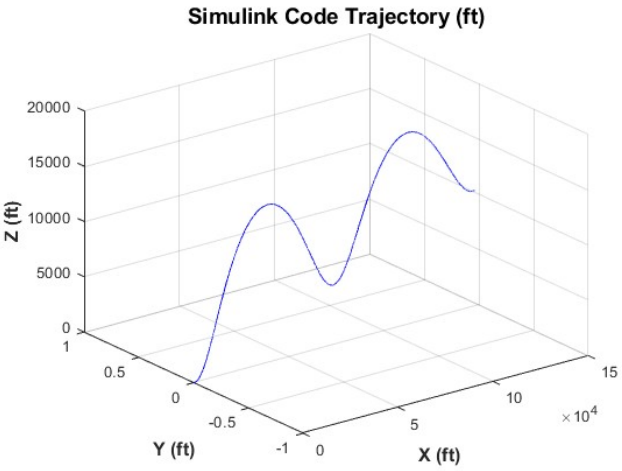


Figure 100: Simulink Trajectory Plot

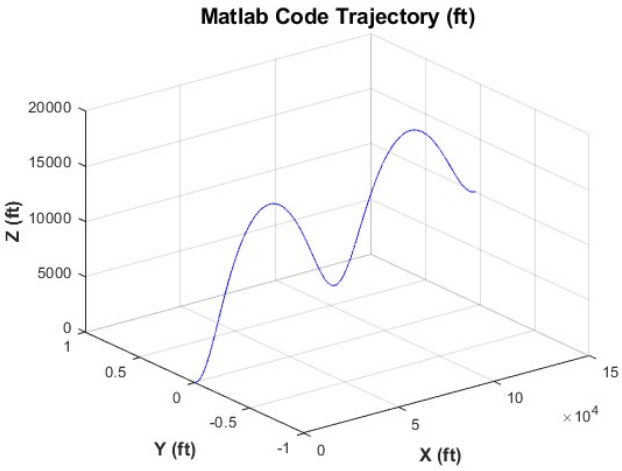


Figure 102: Matlab Trajectory Plot

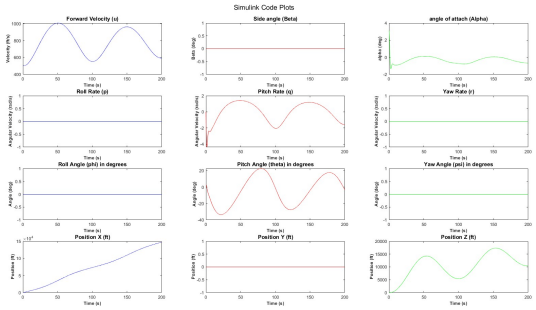


Figure 101: Simulink States Plots

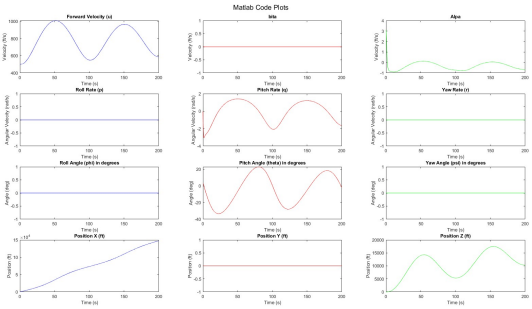


Figure 103: Matlab States Plots

· **-5° Elevator Deflection**: Primarily affects the pitch rate (q) and vertical velocity (w), inducing a pitching motion.

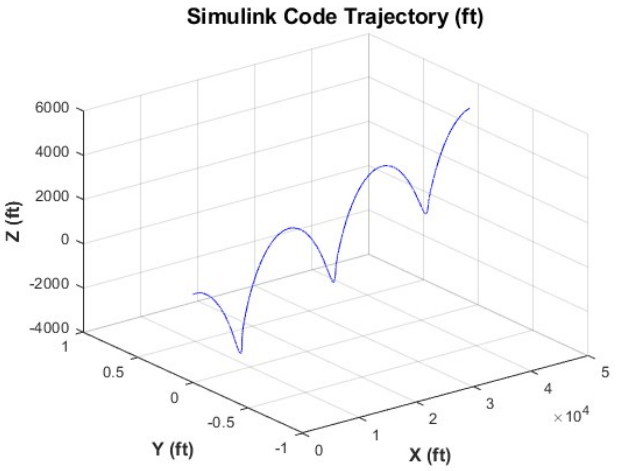


Figure 104: Simulink Trajectory Plot

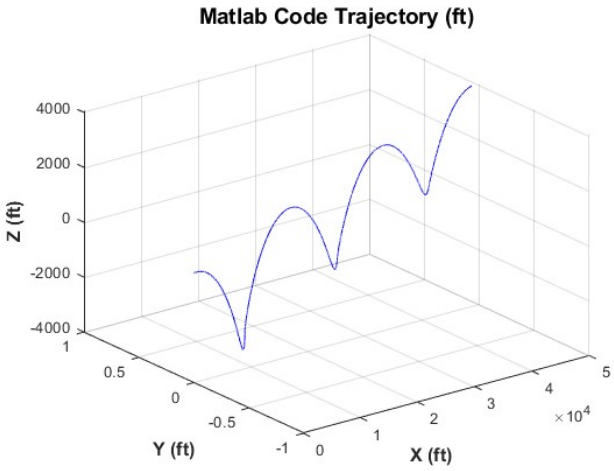


Figure 106: Matlab Trajectory Plot

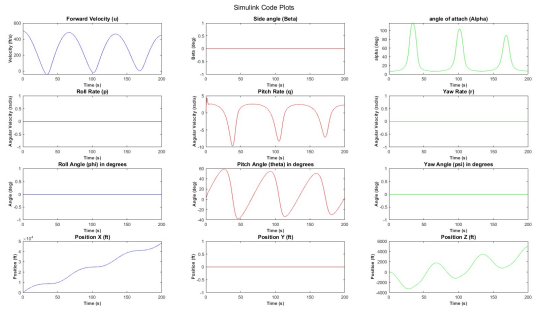


Figure 105: Simulink States Plots

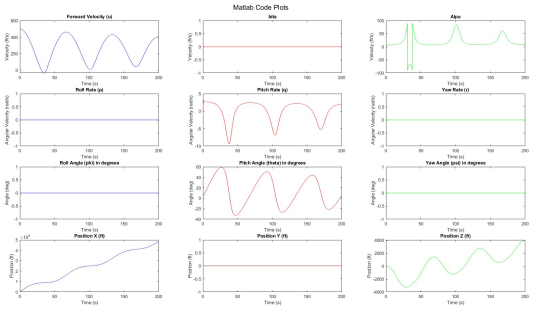


Figure 107: Matlab States Plots

· **Thrust Variation**: Primarily affects the forward velocity (u).
· **Response for 1000 in thrust**

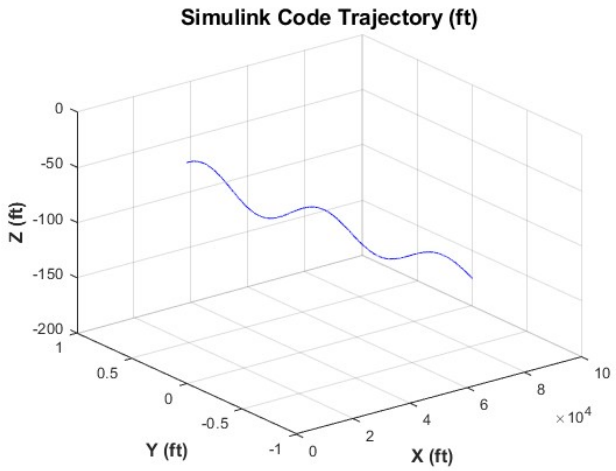


Figure 108: Simulink Trajectory Plot

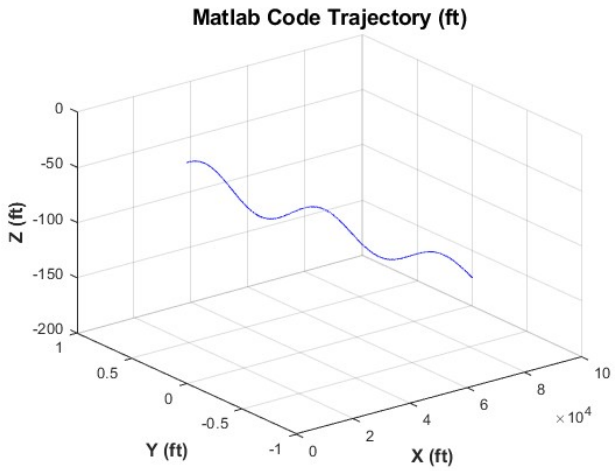


Figure 110: Matlab Trajectory Plot

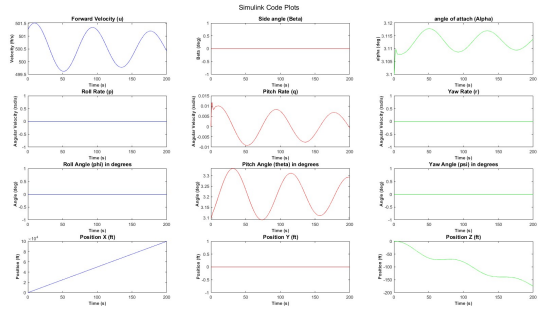


Figure 109: Simulink States Plots

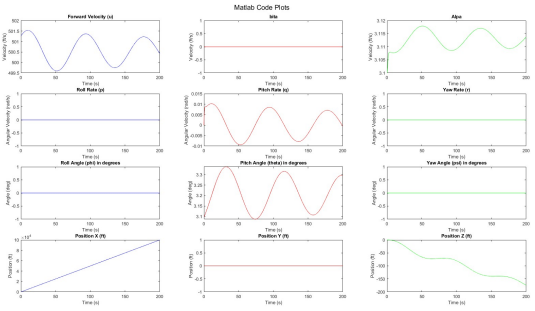


Figure 111: Matlab States Plots

· **Thrust Variation**: Primarily affects the forward velocity (u).
· **Response for 10000 in thrust**

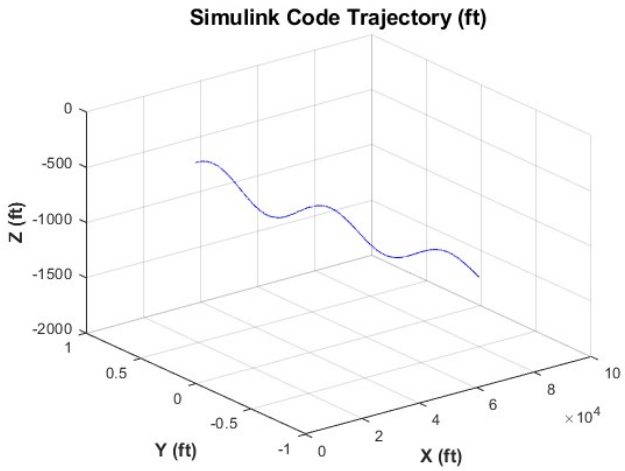


Figure 112: Simulink Trajectory Plot

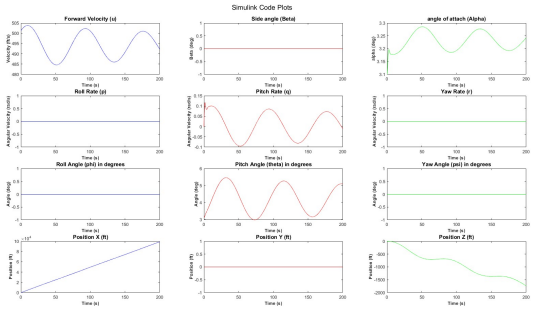


Figure 113: Simulink States Plots

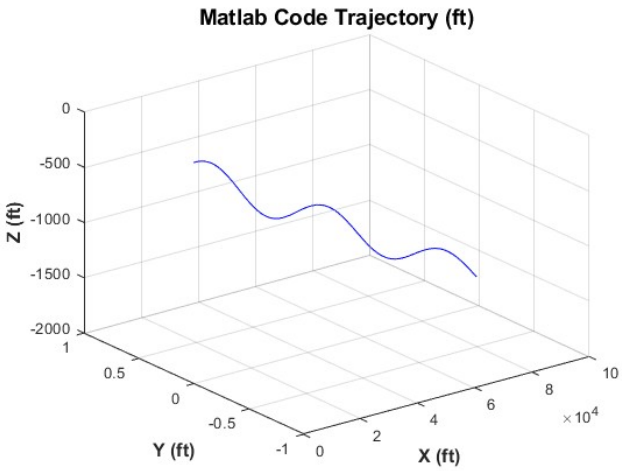


Figure 114: Matlab Trajectory Plot

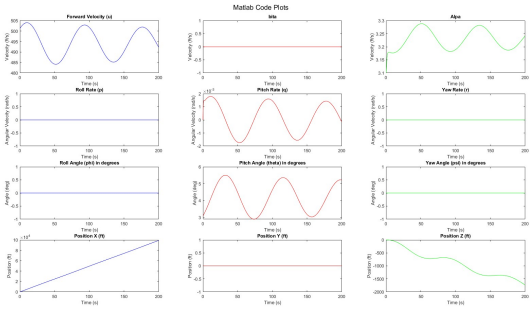


Figure 115: Matlab States Plots

6 Linearization and approximation

6.1 Steps

1-Equilibruim balance
analyse the equation in the equilibrium state and determine which state will be zero.
2-Small disturbance theory
assume each state is equivalent to the initial state + the change that happened in that state.
3-Make approximations

$$\cos(\Delta x) = 1$$
$$\sin(\Delta x) = \Delta x$$
$$\Delta x \Delta y = 0$$
$$\text{where} \quad \Delta x \quad \text{and} \quad \Delta y \text{ are small}$$

4-Taylor series
use Taylor series to represent the change in the state due to each of the other states

6.2 Lateral linearization

6.2.1

$$Y = mg\cos(\theta)\sin(\phi) = m(\dot{\nu} + ru - pw)$$

1. equilibrium balance

   $$Y_0 = mg\cos(\theta_0)\sin(\phi_0) = m(\dot{\nu}_0 + r_0 u_0 - p_0 w_0)$$

   where $Y_0 = 0$, $\phi_0 = 0$, $\dot{\nu}_0 = 0$, $r_0 = 0$, and $p_0 = 0$. we get:

   $$0 = mg\cos(\theta_0)\sin(0) = m(0 + 0 \cdot u_0 - 0 \cdot w_0)$$

   which simplifies to:

   $$0 = 0$$

   Thus, the equation reduces to a trivial identity under these conditions.
2. Small distrubance and approximations
   Starting with the equation:

   $$Y_0 + \Delta Y + mg\cos(\theta_0 + \Delta\theta)\sin(\phi_0 + \Delta\phi) = m\left(\dot{\nu}_0 + \Delta\dot{\nu} + (r_0 + \Delta r)(u_0 + \Delta u) - (p_0 + \Delta p)(w_0 + \Delta w)\right)$$

   with the conditions $Y_0 = 0$, $\phi_0 = 0$, $\dot{\nu}_0 = 0$, $r_0 = 0$, and $p_0 = 0$.
   This becomes:

   $$\Delta Y + mg\,\Delta\phi\left[\cos(\theta_0)\cos(\Delta\theta) - \sin(\theta_0)\sin(\Delta\theta)\right] = m\left(\Delta\dot{\nu} + \Delta r u_0 + \Delta p \omega_0\right)$$

   Now, assuming $\cos(\Delta\theta) = 1$ and $\sin(\Delta\theta) = \Delta\theta$, we get:

   $$\Delta Y + mg\,\Delta\phi\cos(\theta_0) - mg\,\Delta\phi\sin(\theta_0)\Delta\theta = m\left(\Delta\dot{\nu} + \Delta r u_0 - \Delta p \omega_0\right)$$

   Then, if $\Delta\theta\,\Delta\phi = 0$, it further simplifies to:

   $$\Delta Y + mg\,\Delta\phi\cos(\theta_0) = m\left(\Delta\dot{\nu} + \Delta r u_0 - \Delta p \omega_0\right)$$

3. Taylor series
   Starting with the equation:

   $$\frac{\Delta Y}{m} = Y_\nu \Delta\nu + Y_p \Delta p + Y_r \Delta r + Y_{\delta a}\Delta\delta a + Y_{\delta r}\Delta\delta r$$

   By substituting the obtained equation from part b, we have:

   $$\Delta\dot{\nu} = Y_\nu \Delta\nu + Y_p \Delta p + Y_r \Delta r + Y_{\delta a}\Delta\delta a + Y_{\delta r}\Delta\delta r + g\cos(\theta_0)\Delta\phi - \Delta r u_0 + \Delta p \omega_0$$

6.2.2

$$r = \dot{\psi}\cos(\theta)\cos(\phi) - \dot{\theta}\sin(\phi)$$

1. Equilibruim balance:

$$r = \dot{\psi}_0\cos(\theta_0)\cos(\phi_0) - \dot{\theta}_0\sin(\phi_0)$$
$$r = \dot{\psi}_0\cos(\theta_0)\cos(\phi_0) - \dot{\theta}_0\sin(\phi_0)$$

Substituting the values $r_0 = 0$, $\dot{\psi}_0 = 0$, $\phi_0 = 0$, and $\dot{\theta}_0 = 0$, we have:

$$0 = 0 \cdot \cos(0)\cos(0) - 0 \cdot \sin(0)$$

This simplifies to:

$$0 = 0$$

2. small disturbance and approximation

$$r_0 + \Delta r = (\dot{\psi}_0 + \Delta\dot{\psi})\cos(\theta_0 + \Delta\theta)\cos(\phi_0 + \Delta\phi) - (\dot{\theta}_0 + \Delta\dot{\theta})\sin(\phi_0 + \Delta\phi)$$

Given the conditions $r_0 = 0$, $\dot{\psi}_0 = 0$, $\phi_0 = 0$, and $\dot{\theta}_0 = 0$, we have:

$$\Delta r = \Delta\dot{\psi}\left[\cos(\theta_0)\cos(\Delta\theta) - \sin(\theta_0)\sin(\Delta\theta)\right]\cos(\Delta\phi) - \Delta\dot{\theta}\sin(\Delta\phi)$$

Using the approximations $\cos(\Delta\theta) \approx 1$ and $\sin(\Delta\theta) \approx \Delta\theta$, this becomes:

$$\Delta r = \Delta\dot{\psi}\cos(\theta_0) - \sin(\theta_0)\Delta\theta\Delta\dot{\psi}\cos(\Delta\phi) - \Delta\dot{\theta}\Delta\phi$$

By further approximating $\Delta\theta\Delta\dot{\psi} = 0$ and $\Delta\dot{\theta}\Delta\phi = 0$, we arrive at:

$$\Delta r = \Delta\dot{\psi}\cos(\theta_0)$$

3. Tylor series

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

.

$$p = \dot{\phi} - \dot{\psi}\sin(\theta)$$

1. Equilibruim balance
Starting with the equation:

$$p_0 = \dot{\phi}_0 - \dot{\psi}_0\sin(\theta_0)$$

Given the conditions $p_0 = 0$, $\dot{\phi}_0 = 0$, and $\dot{\psi}_0 = 0$, we get

$$0 = 0 - 0 \cdot \sin(\theta_0)$$

which is trivial

$$0 = 0$$

2. Small disturbance and approximation

$$p_0 + \Delta p = \dot{\phi}_0 + \Delta\dot{\phi} - (\dot{\psi}_0 + \Delta\dot{\psi})\sin(\theta_0 + \Delta\theta)$$

Substituting the values $p_0 = 0$, $\dot{\phi}_0 = 0$, and $\dot{\psi}_0 = 0$, we have:

$$\Delta p = \Delta\dot{\phi} - \Delta\dot{\psi}\left[\sin(\theta_0)\cos(\Delta\theta) + \cos(\theta_0)\sin(\Delta\theta)\right]$$

By approximating $\cos(\Delta\theta) \approx 1$ and $\sin(\Delta\theta) \approx \Delta\theta$, we get:

$$\Delta p = \Delta\dot{\phi} - \Delta\dot{\psi}\left(\sin(\theta_0) + \cos(\theta_0)\Delta\theta\right)\Delta\dot{\psi}$$

By further approximating $\Delta\theta\Delta\dot{\psi} = 0$, this simplifies to:

$$\Delta p = \Delta\dot{\phi} - \Delta\dot{\psi}\sin(\theta_0)$$

3. Taylor series
by referring to equation 2 where

$$\Delta r = \Delta\dot{\psi}\cos(\theta_0)$$

then we get:

$$\Delta\dot{\phi} = \Delta p + \Delta r\tan(\theta_0)$$

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

] The coupled equations

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

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

1. Equilibrium balance

$$L_0 = I_x\dot{p}_0 - I_{xz}\dot{r}_0 + qr(I_z - I_y) - I_{xz}pq_0$$
$$N_0 = -I_{xz}\dot{p}_0 + I_z\dot{r}_0 + pq(I_y - I_x) + I_{xz}qr_0$$

   with $L_0 = 0$, $N_0 = 0$, $\dot{p}_0 = 0$, $\dot{r}_0 = 0$, $r_0 = 0$, $q_0 = 0$, and $p_0 = 0$ bot equations give trivial solution

2. Small disturbance and approximation

$$L_0 + \Delta L = I_x(\dot{p}_0 + \Delta\dot{p}) - I_{xz}(\dot{r}_0 + \Delta\dot{r}) + (q_0 + \Delta q)(r_0 + \Delta r)(I_z - I_y) - I_{xz}(p_0 + \Delta p)(q_0 + \Delta q)$$

$$N_0 + \Delta N = -I_{xz}(\dot{p}_0 + \Delta\dot{p}) + I_z(\dot{r}_0 + \Delta\dot{r}) + (p_0 + \Delta p)(q_0 + \Delta q)(I_y - I_x) + I_{xz}(q_0 + \Delta q)(r_0 + \Delta r)$$

   With $L_0 = 0, N_0 = 0, \dot{p}_0 = 0, \dot{r}_0 = 0, r_0 = 0, q_0 = 0, p_0 = 0$, the equations reduce to:

$$\Delta L = \Delta\dot{p}I_x - I_{xz}\Delta r\Delta\dot{r}$$

$$\Delta N = -\Delta\dot{p}I_{xz} + I_z\Delta\dot{r}$$

   In matrix form, we can write:

$$\begin{bmatrix}\Delta L \\ \Delta N\end{bmatrix} = \begin{bmatrix} I_x & -I_{xz} \\ -I_{xz} & I_z\end{bmatrix}\begin{bmatrix}\Delta\dot{p} \\ \Delta\dot{r}\end{bmatrix}$$

   Thus, we get:

$$\begin{bmatrix}\Delta\dot{p} \\ \Delta\dot{r}\end{bmatrix} = \frac{1}{I_zI_x - I_{xz}^2}\begin{bmatrix}I_z + I_{xz} \\ I_{xz} + I_x\end{bmatrix}\begin{bmatrix}\Delta L \\ \Delta N\end{bmatrix}$$

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

   Then:

$$\Delta\dot{p} = G\left(I_z\Delta L + I_{xz}\Delta N\right)$$

$$\Delta\dot{r} = G\left(I_{xz}\Delta L + I_x\Delta N\right)$$

3. Taylor series

$$\frac{\Delta L}{I_x} = L_\nu\Delta\nu + L_p\Delta p + L_r\Delta r + L_{\delta a}\Delta\delta a + L_{\delta r}\Delta\delta r$$

$$\frac{\Delta N}{I_x} = N_\nu\Delta\nu + N_p\Delta p + N_r\Delta r + N_{\delta a}\Delta\delta a + N_{\delta r}\Delta\delta r$$

   Letting

$$L'_a = \frac{1}{G}L_a + \frac{I_{xz}}{I_x}N_a, \quad N'_a = \frac{I_{xz}}{I_z}L_a + \frac{1}{G}N_a$$

   Then we have the equations for $\dot{p}$ and $\dot{r}$:

$$\Delta\dot{p} = L'_\nu\Delta\nu + L'_p\Delta p + L'_r\Delta r + L'_{\delta a}\Delta\delta a + L'_{\delta r}\Delta\delta r$$

$$\Delta\dot{r} = N'_\nu\Delta\nu + N'_p\Delta p + N'_r\Delta r + N'_{\delta a}\Delta\delta a + N'_{\delta r}\Delta\delta r$$

Lateral State Space representation:

$$\begin{bmatrix} \Delta\dot{\nu} \\ \Delta\dot{p} \\ \Delta\dot{r} \\ \Delta\dot{\phi} \\ \Delta\dot{\psi} \end{bmatrix} = \begin{bmatrix} Y_\nu & Y_p + \omega_0 & Y_r - u_0 & g\cos(\theta_0) & 0 \\ L'_\nu & L'_p & L'_r & 0 & 0 \\ N'_\nu & N'_p & N'_r & 0 & 0 \\ 0 & 1 & \tan(\theta_0) & 0 & 0 \\ 0 & 0 & \sec(\theta_0) & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\nu \\ \Delta p \\ \Delta r \\ \Delta\phi \\ \Delta\psi \end{bmatrix} + \begin{bmatrix} Y_{\delta a} & Y_{\delta r} \\ L'_{\delta a} & L'_{\delta r} \\ N'_{\delta a} & N'_{\delta r} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta\delta a \\ \Delta\delta r \end{bmatrix}$$

6.3 Longitudnal linearization

6.3.1

$$X - mg\sin(\theta_0) = m(\dot{u}_0 + q_0 w_0 - r_0 \nu_0)$$

1. Equilibruim balance:

   $$X_0 - mg\sin(\theta_0) = m(\dot{u}_0 + q_0 w_0 - r_0 \nu_0)$$

   where $\dot{u}_0 = 0$, $q_0 = 0$, $r_0 = 0$, $\nu_0 = 0$. The equation reduces to:

   $$X_0 = mg\sin(\theta_0)$$

2. Small disturbance

   $$X_0 + \Delta X - mg\sin(\theta_0 + \Delta\theta) = m(\dot{u}_0 + \Delta\dot{u} + (q_0 + \Delta q)(w_0 + \Delta w))$$

   where $\dot{u}_0 = 0$, $q_0 = 0$, $r_0 = 0$, $\nu_0 = 0$.
   We get:

   $$X_0 + \Delta X - mg(\sin(\theta_0)\cos(\Delta\theta) + \cos(\theta_0)\sin(\Delta\theta)) = m(\Delta\dot{u} + \Delta q\,\Delta w + \Delta q\, w_0 - \Delta r\,\Delta\nu)$$

   Approximating $\cos(\Delta\theta) = 1$, $\sin(\Delta\theta) = \Delta\theta$, $\Delta r\,\Delta\nu = 0$, and $\Delta q\,\Delta w = 0$, we get:

   $$X_0 + \Delta X - mg(\sin(\theta_0) + \cos(\theta_0)\Delta\theta) = m(\Delta\dot{u} + \Delta q\, w_0)$$

   We know from step a that

   $$X_0 - mg\sin(\theta_0) = 0$$

   by substituting,

   $$\Delta X = m(g\cos(\theta_0)\Delta\theta + (\Delta\dot{u} + \Delta q\, w_0))$$

3. Taylor series

   $$\frac{\Delta X}{m} = X_u\Delta u + X_w\Delta w + X_{\delta e}\Delta\delta_e + X_{\delta t}\Delta\delta_t$$

   $$\Delta\dot{u} = X_u\Delta u + X_w\Delta w + X_{\delta e}\Delta\delta_e + X_{\delta t}\Delta\delta_t - g\Delta\theta\cos(\theta_0) - \Delta q\, w_0$$

6.3.2

$$Z + mg\cos(\theta)\cos(\phi) = m\left(\dot{w} + p\nu - qu\right)$$

1. Equilibruim banalnce

   $$Z_0 + mg\cos(\theta_0)\cos(\phi_0) = m\left(\dot{w}_0 + p_0\nu_0 - q_0 u_0\right)$$

   With $\phi_0 = 0,\ \dot{w}_0 = 0,\ p_0 = 0,\ \nu_0 = 0,\ q_0 = 0,$ the equation reduces to:

   $$Z_0 = -mg\cos(\theta_0)$$

2. Small disturbance

$Z_0+\Delta Z+mg\cos(\theta_0+\Delta\theta)\cos(\phi_0+\Delta\phi) = m\left(\dot{w}_0+\Delta\dot{w}_0+(p_0+\Delta p)(\nu_0+\Delta\nu)-(q_0+\Delta q)(u_0+\Delta u)\right)$

With $\phi_0 = 0$, $\dot{w}_0 = 0$, $p_0 = 0$, $\nu_0 = 0$, and $q_0 = 0$, the equation reduces to:

$$Z_0+\Delta Z+mg\left[\cos(\theta_0)\cos(\Delta\theta)-\sin(\theta_0)\sin(\Delta\theta)\right] = m\left(\Delta\dot{w}+\Delta p\Delta\nu-\Delta q u_0-\Delta q\Delta u\right)$$

After approximating $\cos(\Delta\theta) = 1$, $\sin(\Delta\theta) = \Delta\theta$, $\cos(\Delta\phi) = 1$, $\Delta p\Delta\nu = 0$, and $\Delta q\Delta u = 0$, we get:

$$Z_0+\Delta Z+mg\cos(\theta_0)-mg\Delta\theta\sin(\theta_0) = m\left(\Delta\dot{w}-u_0\Delta q\right)$$

from part a we know that:

$$Z_0+mg\cos(\theta_0) = 0$$

Then we have:

$$\Delta Z = mg\Delta\theta\sin(\theta_0)+m\left(\Delta\dot{w}-u_0\Delta q\right)$$

3. Taylor series

$$\frac{\Delta Z}{m} = Z_u\Delta u+Z_w\Delta w+Z_{\dot{w}}\Delta\dot{w}+Z_q\Delta q+Z_{\delta e}\delta e+Z_{\delta t}\delta t$$

Then we get:

$$\Delta\dot{w} = \frac{Z_u\Delta u}{1-Z_{\dot{w}}}+\frac{Z_w\Delta w}{1-Z_{\dot{w}}}+\frac{Z_q\Delta q}{1-Z_{\dot{w}}}+\frac{Z_{\delta e}\Delta\delta e}{1-Z_{\dot{w}}}+\frac{Z_{\delta t}\Delta\delta t}{1-Z_{\dot{w}}}-\frac{g\Delta\theta\sin(\theta_0)}{1-Z_{\dot{w}}}+\frac{u_0\Delta q}{1-Z_{\dot{w}}}$$

6.3.3

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

1. Equilibruim balance

$$M_0 = I_y\dot{q}_0+r_0q_0(I_x-I_z)+I_{xz}(p_0^2-r_0^2)$$

where $M_0 = 0, \quad \dot{q}_0 = 0, \quad r_0 = 0, \quad q_0 = 0, \quad p_0 = 0$
This leads to trivial soltion:

$$0 = 0 \quad \text{(trivial solution)}$$

2. Small disturbance

$$M_0+\Delta M = I_y(\dot{q}_0+\Delta\dot{q})+(r_0+\Delta r)(q_0+\Delta q)(I_x-I_z)+I_{xz}\left((p_0+\Delta p)^2-(r_0+\Delta r)^2\right)$$

Where:

$$M_0 = 0, \quad \dot{q}_0 = 0, \quad r_0 = 0, \quad q_0 = 0, \quad p_0 = 0$$

This reduces to:

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

With approximations:

$$\Delta r\Delta q = 0, \quad (\Delta p)^2 = 0, \quad \text{and} \quad (\Delta r)^2 = 0$$

Which gives:

$$\Delta M = I_y\Delta\dot{q}$$

3. Taylor series

$$\frac{\Delta M}{I_y} = M_u\Delta u+M_w\Delta w+M_{\dot{w}}\Delta\dot{w}+M_q\Delta q+M_{\delta e}\Delta\delta e+M_{\delta t}\Delta\delta t$$

from part 6.32, we can substitute with $\dot{w}$

$$\Delta\dot{q} = (M_u+M_{\dot{w}}\frac{Z_u}{1-Z_{\dot{w}}})\Delta u+(M_w+M_{\dot{w}}\frac{Z_w}{1-Z_{\dot{w}}})\Delta w-M_{\dot{w}}\frac{g\sin(\theta_0)}{1-Z_{\dot{w}}}\Delta\theta$$

+

$$(M_q+M_{\dot{w}}\frac{Z_q+u_0}{1-Z_{\dot{w}}})\Delta q+(M_{\delta e}+M_{\dot{w}}\frac{Z_{\delta e}}{1-Z_{\dot{w}}})\Delta\delta e+(M_{\delta t}+M_{\dot{w}}\frac{Z_{\delta t}}{1-Z_{\dot{w}}})\Delta\delta t$$

6.3.4

$$q = \dot{\theta}\cos\phi + \dot{\psi}\cos\theta\sin\phi$$

1. Equilibruim balance

$$q_0 + \Delta q = \dot{\theta}_0\cos\phi_0 + \dot{\psi}_0\cos\theta_0\sin\phi_0$$

Since $q_0 = 0$, $\dot{\theta}_0 = 0$, $\phi_0 = 0$, $\dot{\psi}_0 = 0$, the equation reduces to the trivial solution: $0 = 0$.

2. Small disturbance:

$$q_0 + \Delta q = (\dot{\theta}_0 + \Delta\dot{\theta})\cos(\phi_0 + \Delta\phi) + (\dot{\psi}_0 + \Delta\dot{\psi})\cos(\theta_0 + \Delta\theta)\sin(\phi_0 + \Delta\phi)$$

With $q_0 = 0, \dot{\theta}_0 = 0, \phi_0 = 0, \dot{\psi}_0 = 0$, the equation reduces to

$$\Delta q = \Delta\dot{\theta}\cos(\Delta\phi) + \Delta\psi\left(\cos(\theta_0)\cos(\Delta\theta) - \sin(\theta_0)\sin(\Delta\theta)\right)\sin(\Delta\phi)$$

With the approximations $\cos(\Delta\phi) = 1$, $\sin(\Delta\theta) = \Delta\theta$, $\cos(\Delta\theta) = 1$, and $\sin(\Delta\phi) = \Delta\phi$, we get

$$\Delta q = \Delta\dot{\theta} + \Delta\psi\left(\Delta\phi\cos(\theta_0) - \sin(\theta_0)\Delta\theta\Delta\phi\right)$$

With the approximations $\Delta\psi\Delta\phi = 0$ and $\Delta\theta\Delta\phi\Delta\psi = 0$, we obtain

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

3. Taylor series

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

Longitudinal State Space representation

$$\begin{bmatrix}\Delta\dot{u}\\ \Delta\dot{w}\\ \Delta\dot{q}\\ \Delta\dot{\theta}\end{bmatrix} = \begin{bmatrix} X_u & X_w & -w_0 & -g\cos(\theta_0)\\ \frac{Z_u}{1-Z_{\dot{w}}} & \frac{Z_w}{1-Z_{\dot{w}}} & \frac{Z_q+u_0}{1-Z_{\dot{w}}} & -\frac{g\sin(\theta_0)}{1-Z_{\dot{w}}}\\ M_u + M_{\dot{w}}\frac{Z_u}{1-Z_{\dot{w}}} & M_w + M_{\dot{w}}\frac{Z_w}{1-Z_{\dot{w}}} & M_q + M_{\dot{w}}\frac{Z_q+u_0}{1-Z_{\dot{w}}} & -M_{\dot{w}}\frac{g\sin(\theta_0)}{1-Z_{\dot{w}}}\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta w\\ \Delta q\\ \Delta\theta\end{bmatrix}$$

$+$

$$\begin{bmatrix} X_{\delta e} & X_{\delta t}\\ \frac{Z_{\delta e}}{1-Z_{\dot{w}}} & \frac{Z_{\delta t}}{1-Z_{\dot{w}}}\\ M_{\delta e} + M_{\dot{w}}\frac{Z_{\delta e}}{1-Z_{\dot{w}}} & M_{\delta t} + M_{\dot{w}}\frac{Z_{\delta t}}{1-Z_{\dot{w}}}\\ 0 & 0\end{bmatrix}\begin{bmatrix}\Delta\delta e\\ \Delta\delta t\end{bmatrix}$$

6.4 Longitudinal approximations

6.4.1 Short-period appoximation

An approximation to the short-period mode of motion can be obtained by assuming u = 0 and dropping the X-force equation.

$$\begin{bmatrix}\Delta\dot{w}\\ \Delta\dot{q}\end{bmatrix} = \begin{bmatrix}\frac{Z_w}{1-Z_{\dot{w}}} & \frac{Z_q+u_0}{1-Z_{\dot{w}}}\\ M_w + M_{\dot{w}}\frac{Z_w}{1-Z_{\dot{w}}} & M_q + M_{\dot{w}}\frac{Z_q+u_0}{1-Z_{\dot{w}}}\end{bmatrix}\begin{bmatrix}\Delta w\\ \Delta q\end{bmatrix} + \begin{bmatrix}\frac{Z_{\delta e}}{1-Z_{\dot{w}}} & \frac{Z_{\delta t}}{1-Z_{\dot{w}}}\\ M_{\delta e} + M_{\dot{w}}\frac{Z_{\delta e}}{1+Z_{\dot{w}}} & M_{\delta t} + M_{\dot{w}}\frac{Z_{\delta t}}{1+Z_{\dot{w}}}\end{bmatrix}\begin{bmatrix}\Delta\delta e\\ \Delta\delta t\end{bmatrix}$$

To calculate the frequencyω and damping ratioζ we need to get the charactaristique equation

$$|\lambda I - A| = 0$$

This would give

$$\lambda^2 - \left(\frac{Z_w + M_{\dot{w}}(Z_q + u_0)}{1 - Z_{\dot{w}}} + M_q\right)\lambda + \frac{Z_w M_q - M_w(Z_q + u_0)}{1 - Z_{\dot{w}}} = 0$$

and since $\lambda^2 - 2\zeta\omega_n\lambda + \omega_n^2 = 0$ Then:

$$\omega_n^2 = \sqrt{\frac{Z_w M_q - M_w(Z_q + u_0)}{1 - Z_{\dot{w}}}}$$

$$\zeta = \frac{\frac{Z_w + M_{\dot{w}}(Z_q + u_0)}{1 - Z_{\dot{w}}} + M_q}{2\sqrt{\frac{Z_w M_q - M_w(Z_q + u_0)}{1 - Z_{\dot{w}}}}}$$

6.4.2 Long-period approximation

An approximation to the long-period mode can be obtained by neglecting the pitching moment equation and assuming that the change in angle of attack is 0; that is
We set $\dot{W}$ equation equal to 0:

$$0 = Z_u u + Z_w W + (Z_q + u_0)q - g\sin\theta_0\theta + Z_{\delta e}\delta e + Z_{\delta t}\delta t$$

We know that $\dot{\theta} = q$ and we neglect the change in w which corresponds to the change in angle of attack then by some simplification:

$$\dot{\theta} = -\frac{Z_u}{Z_q + u_0}u + \frac{g\sin\theta_0}{Z_q + u_0}\theta - \frac{Z_{\delta e}}{Z_q + u_0}\delta e - \frac{Z_{\delta t}}{Z_q + u_0}\delta t$$

Then we get

$$\begin{bmatrix}\Delta\dot{u}\\ \Delta\dot{\theta}\end{bmatrix} = \begin{bmatrix} X_u & -g\cos\theta_0 \\ -\frac{Z_u}{Z_q+u_0} & \frac{g\sin\theta_0}{Z_q+u_0}\end{bmatrix}\begin{bmatrix}\Delta u\\ \Delta\theta\end{bmatrix} + \begin{bmatrix} X_{\delta e} & X_{\delta t} \\ -\frac{Z_{\delta e}}{Z_q+u_0} & -\frac{Z_{\delta t}}{Z_q+u_0}\end{bmatrix}\begin{bmatrix}\Delta\delta e\\ \Delta\delta t\end{bmatrix}$$

To calculate the frequencyω and damping ratioζ we need to get the characteristic equation

$$|\lambda I - A| = 0$$

This would give

$$\lambda^2 - \left(\frac{g\sin(\theta_0)}{Z_q + u_0} + X_u\right)\lambda + \frac{g(X_u\sin(\theta_0) - Z_u\cos(\theta_0))}{Z_q + u_0}$$

and since $\lambda^2 - 2\zeta\omega_n\lambda + \omega_n^2 = 0$ Then:

$$\omega_n = \sqrt{\frac{g(X_u\sin(\theta_0) - Z_u\cos(\theta_0))}{Z_q + u_0}}$$

$$\zeta = \frac{\frac{g\sin(\theta_0)}{Z_q + u_0} + X_u}{2\sqrt{\frac{g(X_u\sin(\theta_0) - Z_u\cos(\theta_0))}{Z_q + u_0}}}$$

6.5 Lateral-approximations

6.5.1 3-DOF spiral approximation

To make this approximation we need to make 3 major assumptions:

- Neglecting Y-force equation.
- Neglecting $\Delta\psi$ equation.
- Assuming $\Delta v = 0$.

$$\begin{bmatrix}\Delta\dot{p}\\ \Delta\dot{r}\\ \Delta\dot{\phi}\end{bmatrix} = \begin{bmatrix}L'_p & L'_r & 0\\ N'_p & N'_r & 0\\ 1 & \tan(\theta_0) & 0\end{bmatrix}\begin{bmatrix}\Delta p\\ \Delta r\\ \Delta\phi\end{bmatrix} + \begin{bmatrix}L'_{\delta_r}\\ N'_{\delta_r}\\ 0\end{bmatrix}\begin{bmatrix}\Delta\delta_r\end{bmatrix}$$

6.5.2 3-DOF Dutch-roll approximations

This approximation is made by neglecting rolling and yawing $L'_r = N'_p = 0$

$$\begin{bmatrix}\Delta\dot{v}\\ \Delta\dot{p}\\ \Delta\dot{r}\end{bmatrix} = \begin{bmatrix}Y_v & Y_p + w_0 & Y_r - u_0\\ L'_v & L'_p & 0\\ N'_v & 0 & N'_r\end{bmatrix}\begin{bmatrix}\Delta v\\ \Delta p\\ \Delta r\end{bmatrix} + \begin{bmatrix}Y_{\delta_a} & Y_{\delta_r}\\ L'_{\delta_a} & L'_{\delta_r}\\ N'_{\delta_a} & N'_{\delta_r}\end{bmatrix}\begin{bmatrix}\Delta\delta_a\\ \Delta\delta_r\end{bmatrix}$$

6.5.3 2-DOF Dutch-roll approximations

In this mode, the airplane is modeled with the following assumption:
Neglecting the L-moment equation $\Delta\theta = 0$

$$\begin{bmatrix}\Delta\dot{v}\\ \Delta\dot{r}\end{bmatrix} = \begin{bmatrix}Y_v & Y_r - u_0\\ N'_v & N'_r\end{bmatrix}\begin{bmatrix}\Delta v\\ \Delta r\end{bmatrix} + \begin{bmatrix}Y_{\delta_a} & Y_{\delta_r}\\ N'_{\delta_a} & N'_{\delta_r}\end{bmatrix}\begin{bmatrix}\Delta\delta_a\\ \Delta\delta_r\end{bmatrix}$$

6.5.4 1-DOF Dutch-roll

In this approximation, the airplane is modeled with the following assumption:

$$\dot{P} = L'_p\Delta P + L'_{\delta_a}\Delta\delta_a$$

6.6 applying on B747-FC3 Flight Condition

Longitudinal Mode State-Space Manual Linearization

$$A = \begin{bmatrix} -0.0050 & 0.0743 & 0 & -32.1270 \\ -0.0832 & -0.7585 & 505.8904 & -1.7932 \\ 0.0003 & -0.0026 & -1.8170 & 0.0040 \\ 0 & 0 & 1.0000 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1.1800 & 0.0001 \\ -22.4673 & -0.0000 \\ -1.3503 & 0.0000 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Longitudinal Mode Transfer Function Manual Linearization

· **Transfer Function for δ_e with output u:**

$$\frac{1.18s^3 + 1.37s^2 - 7.221s + 31.19}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

· **Transfer Function for δ_e with output w:**

$$\frac{-22.47s^3 - 724.2s^2 - 1.083s - 3.834}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

· **Transfer Function for δ_e with output q:**

$$\frac{-1.35s^3 - 0.9718s^2 - 0.01316s + 6.389 \times 10^{-19}}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

· **Transfer Function for δ_e with output θ:**

$$\frac{-1.35s^2 - 0.9718s - 0.01316}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

· **Transfer Function for δ_t with output u:**

$$\frac{5.05 \times 10^{-5}s^3 + 0.0001299s^2 + 0.0001377s - 8.102 \times 10^{-6}}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

· **Transfer Function for δ_t with output w:**

$$\frac{-2.267 \times 10^{-6}s^3 + 0.0000147s^2 + 1.007 \times 10^{-6}s + 7.804 \times 10^{-7}}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

· **Transfer Function for δ_t with output q:**

$$\frac{3.07 \times 10^{-7}s^3 + 2.57 \times 10^{-7}s^2 + 2.667 \times 10^{-8}s + 2.153 \times 10^{-26}}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

· **Transfer Function for δ_t with output θ:**

$$\frac{3.07 \times 10^{-7}s^2 + 2.57 \times 10^{-7}s + 2.667 \times 10^{-8}}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

Lateral Mode State-Space Manual Linearization

$$A = \begin{bmatrix} -0.1430 & 0.0541 & -0.9985 & 0.0640 & 0 \\ -0.0064 & -1.1162 & 0.3921 & 0 & 0 \\ 0.0017 & -0.0487 & -0.2534 & 0 & 0 \\ 0 & 1.0000 & 0.0542 & 0 & 0 \\ 0 & 0 & 1.0015 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 11.3452 \\ 0.2275 & 0.2867 \\ 0.0240 & -0.6190 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Lateral Mode Transfer Function Manual Linearization

· **Transfer Function for δ_a with output V:**

$$\frac{-0.01169s^2 + 0.002555s + 0.004347}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_a with output p:**

$$\frac{0.2275s^3 + 0.0996s^2 + 0.01014s - 1.906 \times 10^{-6}}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_a with output r:**

$$\frac{0.02403s^3 + 0.01917s^2 + 0.00228s + 3.52 \times 10^{-5}}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_a with output ϕ:**

$$\frac{0.2288s^2 + 0.1006s + 0.01026}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_a with output ψ:**

$$\frac{0.02407s^3 + 0.0192s^2 + 0.002283s + 3.525 \times 10^{-5}}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_r with output V:**

$$\frac{11.35s^3 + 16.17s^2 + 4.137s - 0.01333}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_r with output p:**

$$\frac{0.2867s^3 - 0.2021s^2 - 0.03859s + 1.209 \times 10^{-5}}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_r with output r:**

$$\frac{-0.619s^3 - 0.7737s^2 - 0.07542s - 0.0002232}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_r with output ϕ:**

$$\frac{0.2532s^2 - 0.244s - 0.04267}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

· **Transfer Function for δ_r with output ψ:**

$$\frac{-0.6199s^3 - 0.7748s^2 - 0.07553s - 0.0002236}{s^4 + 1.513s^3 + 0.4999s^2 + 0.04589s + 5.303 \times 10^{-5}}$$

6.6.1 Longitudinal Step Response

Elevator Input Step Responses
Each graph below shows the step response for Non-linear simulation, Linear full model, Short period, and Long period approximations for different elevator deflections δ_e.

Step Response with Elevator Deflection $\delta_e = 1°$

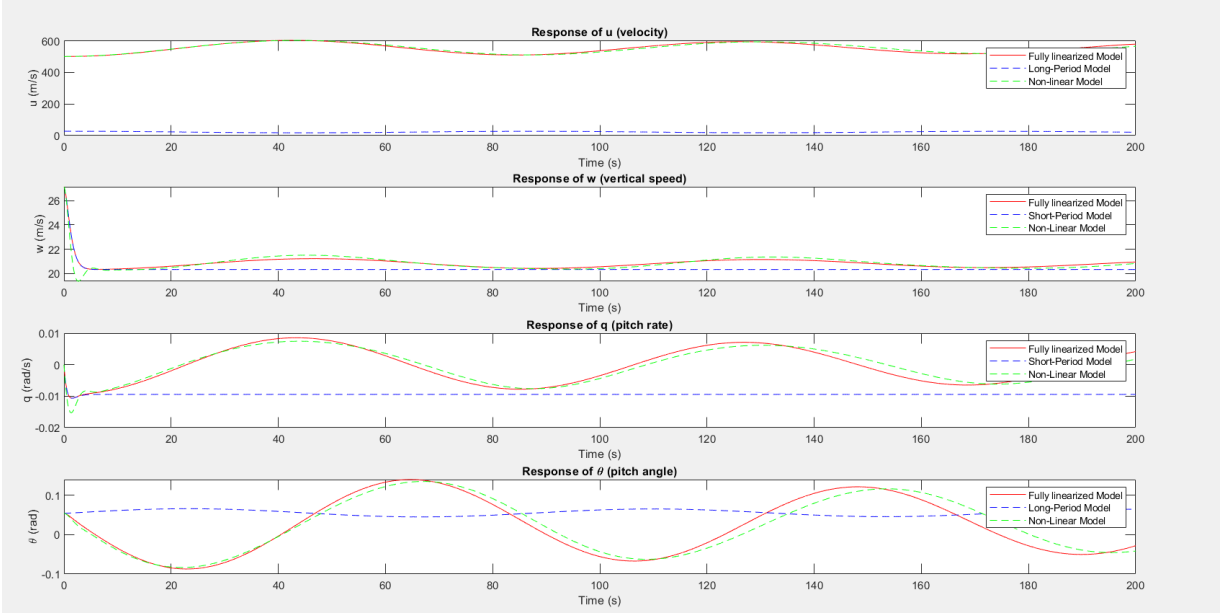


Figure 116: Step response with $\delta_e = 1°$

Comment: At this low deflection angle, the responses across Non-linear, Linear full model, Short period, and Long period approximations remain closely aligned. The Short period approximation captures the quick oscillations and returns to equilibrium without significant deviation, indicating stability in short-term pitch dynamics. The Linear and Non-linear models follow each other closely, suggesting the linearized model is accurate for small deflections.

Step Response with Elevator Deflection $\delta_e = 5°$

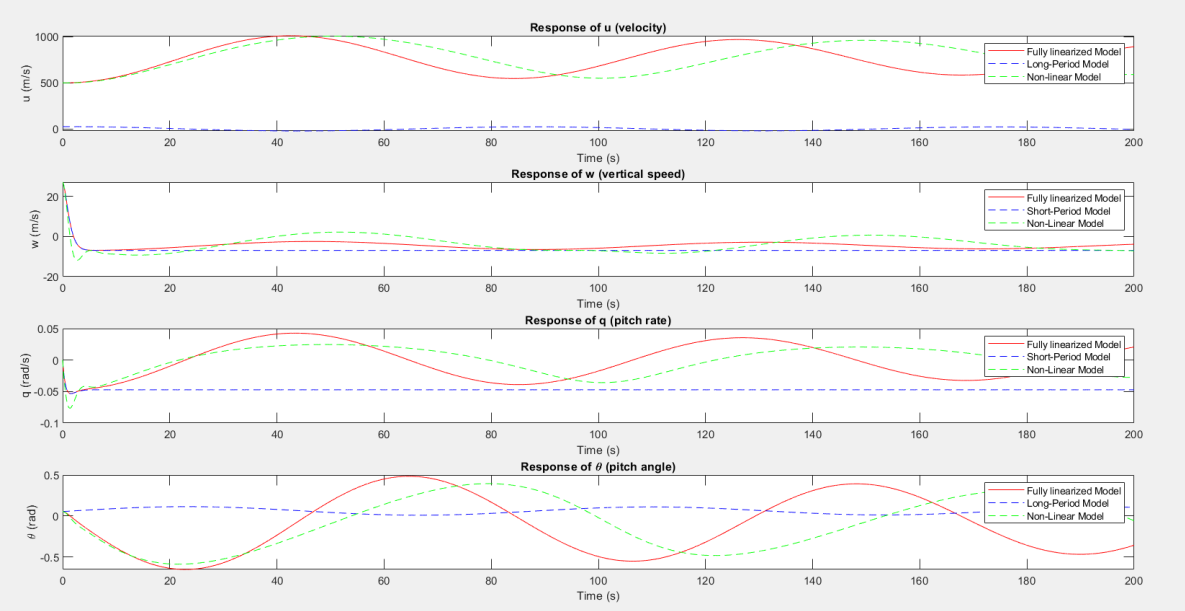


Figure 117: Step response with $\delta_e = 5°$

Comment: With a moderate increase in elevator deflection, the responses begin to differentiate slightly. The Non-linear and Linear full model responses still largely overlap, though the Long period approximation starts showing divergence in capturing the overall trend. Short period oscillations are still present and more pronounced here, highlighting the increased impact on pitch. Overall, this shows the system's sensitivity increases with elevator deflection, affecting model accuracy.

Step Response with Elevator Deflection $\delta_e = 10°$

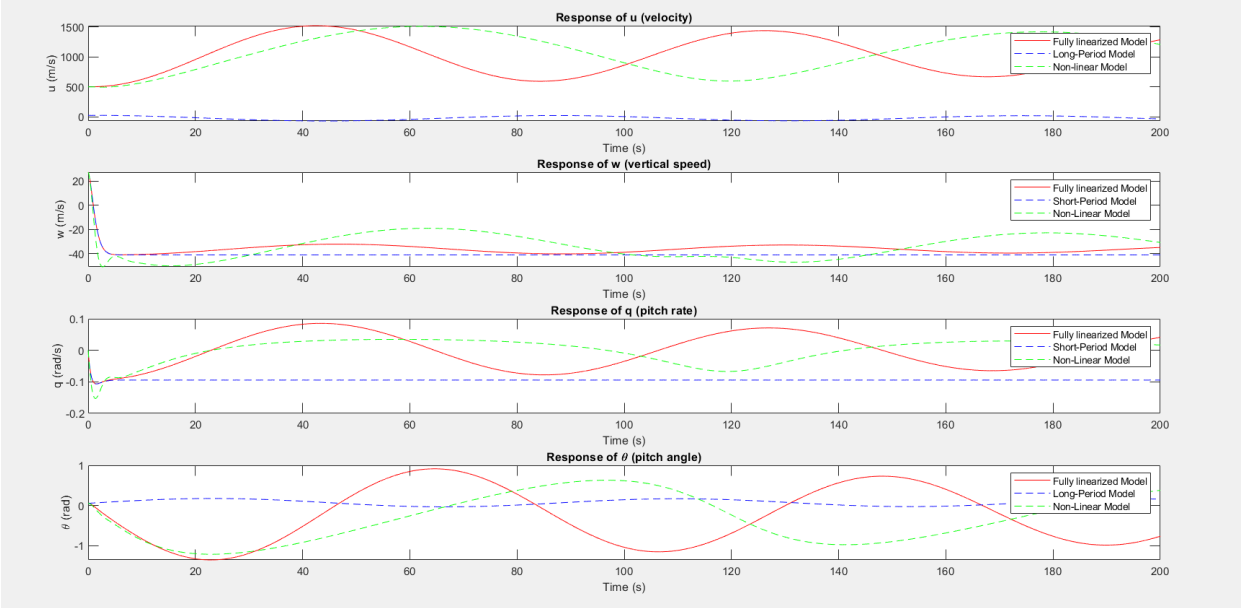


Figure 118: Step response with $\delta_e = 10°$

Comment : At 10° deflection, the Non-linear model demonstrates more noticeable divergence from the Linear full model and Long period approximations. The Short period approximation remains effective in capturing immediate oscillatory behavior but falls short in long-term response accuracy. This difference suggests that larger deflections push the system into regions where linear approximations become less reliable, reflecting greater influence from non-linear aerodynamic effects.

Step Response with Elevator Deflection $\delta_e = 25°$

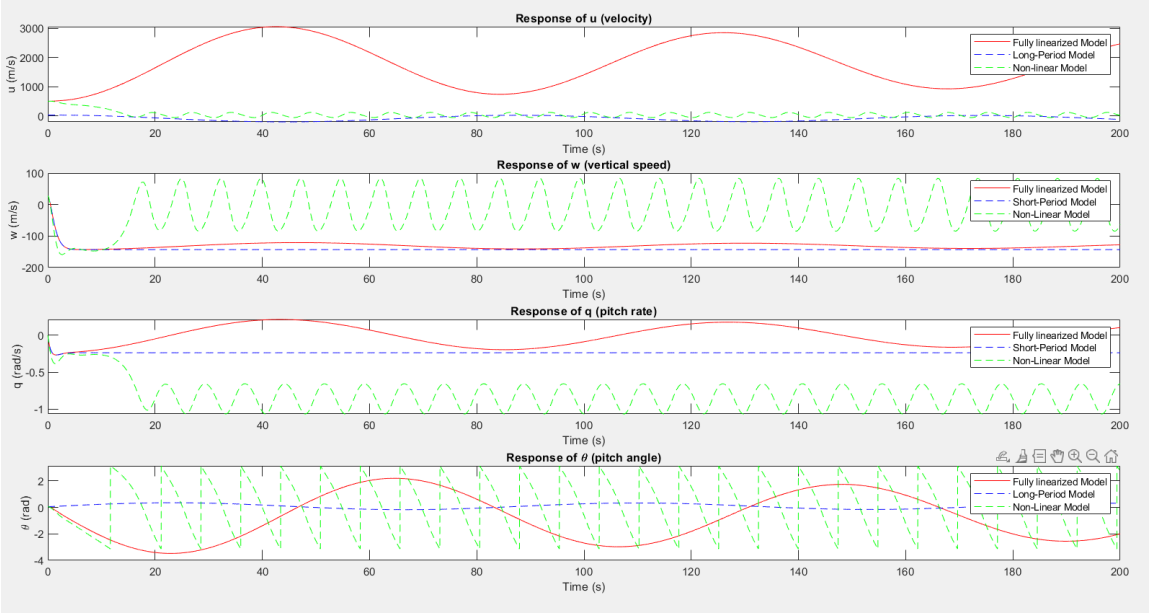


Figure 119: Step response with $\delta_e = 25°$

Comment : With the highest deflection, 25°, Non-linear behavior is prominent, as the Non-linear simulation significantly diverges from all linear approximations. The Linear full model partially captures the initial response but deviates over time. The Long period approximation fails to align with Non-linear trends, and the Short period loses accuracy after initial oscillations. This demonstrates that high elevator inputs induce significant non-linear aerodynamic effects that the linear approximations cannot represent accurately.

Thrust Input Step Responses

Each graph below shows the step response for Non-linear simulation, Linear full model, Short period, and Long period approximations for different thrust inputs.

Step Response with Thrust $T = 2000$

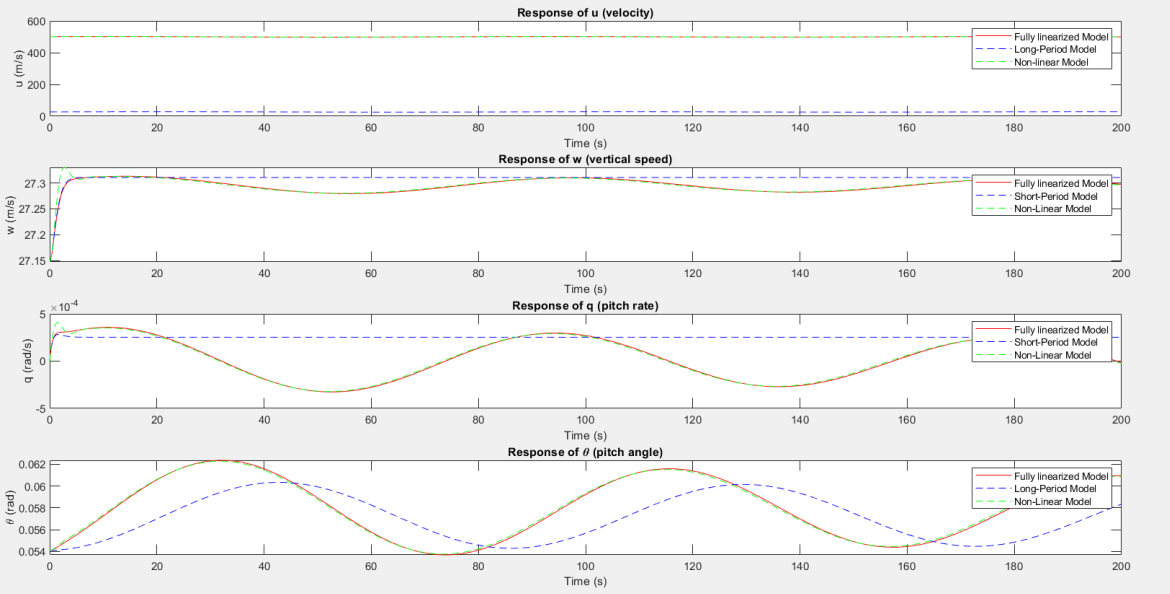


Figure 120: Step response with thrust $T = 2000$

Comment: At this low thrust level, all models, including Non-linear and Linear, show a comparable response, with minimal divergence among Short and Long period approximations. The system stabilizes quickly, reflecting that at lower thrust inputs, linear models are sufficient to represent the thrust-pitch response accurately. The oscillations are controlled, indicating minimal impact on aircraft stability.

Step Response with Thrust $T = 6000$

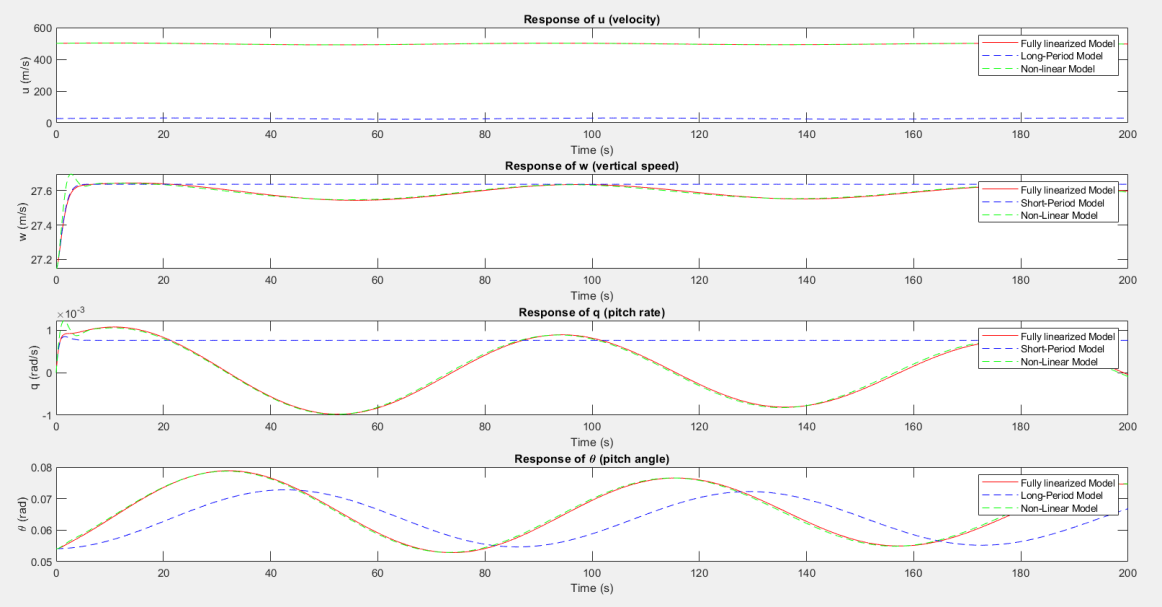


Figure 121: Step response with thrust $T = 6000$

Comment: With increased thrust, the Non-linear model begins to display slight deviation, particularly in the amplitude of oscillations. The Short period model captures initial transient dynamics, but longer-term behavior begins to differ from the Non-linear simulation. The Linear model holds closer to Non-linear than Short or Long period approximations, showing the importance of capturing both short and long-term dynamics as thrust increases.

Step Response with Thrust $T = 10000$

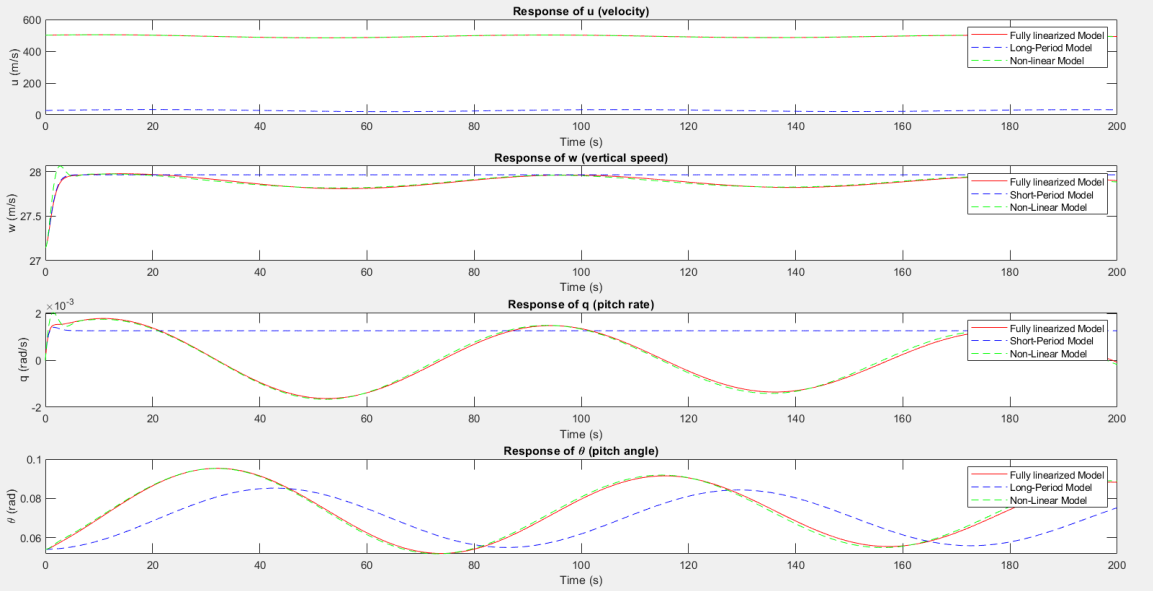


Figure 122: Step response with thrust $T = 10000$

Comment: At high thrust levels, the Non-linear simulation deviates significantly from the Linear model and both approximations. The Short period approximation captures only initial oscillations, while the Long period fails to follow the overall response. The Non-linear model reveals substantial dynamic changes, likely due to increased aerodynamic forces and stability margins affected by high thrust. This response emphasizes the non-linear complexity introduced at higher thrust inputs, where linear models lose validity.

6.6.2 Transfer Functions and Root Locus & Bode Plot for Each Input and Output State For Longitudinal Mode

The Delta Elevator (De) to Linear Velocity (u)

Full Mode Transfer Function:

$$\frac{1.18s^3 + 1.3698s^2 - 7.2213s + 31.1932}{s^4 + 2.5805s^3 + 2.7188s^2 + 0.015195s + 0.015019}$$

Long Mode Transfer Function:

$$\frac{1.18s - 1.431}{s^2 + 0.0014554s + 0.0052641}$$

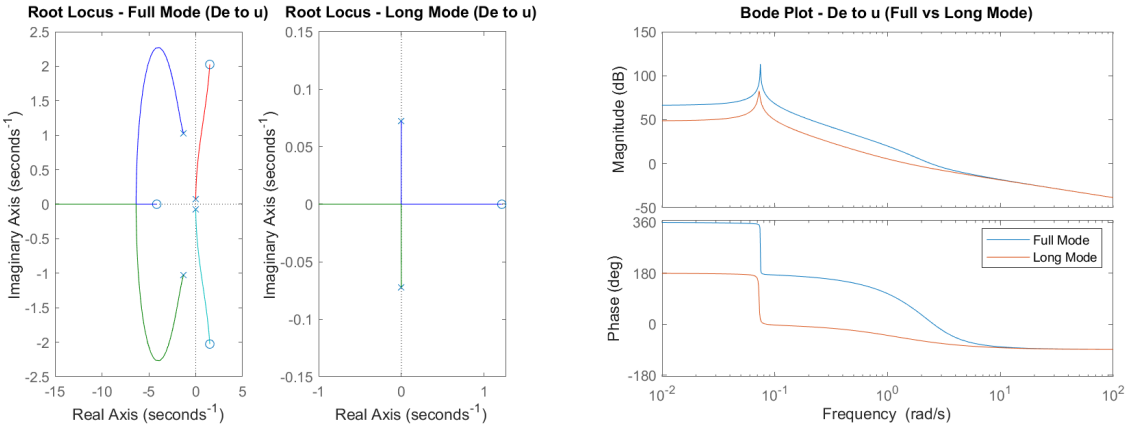


Figure 123: Root Locus and Bode plot for Δelevator (De) to linear velocity (u)

Comment:

The Root Locus for the two modes differs because the full mode has more poles than the long mode, leading to a more complex locus shape. However, the poles near the origin are at the same values in both modes, indicating similar low-frequency behavior.
In the Bode plot, both modes follow the same trend across frequencies but exhibit a consistent 180-degree phase shift. This phase shift implies that, despite similar magnitude responses, the two modes may respond oppositely at certain frequencies.

The Delta Elevator (De) to vertical velocity (w)

Full Mode Transfer Function:

$$\frac{-22.4673s^3 - 724.1614s^2 - 1.0835s - 3.834}{s^4 + 2.5805s^3 + 2.7188s^2 + 0.015195s + 0.015019}$$

Short Mode Transfer Function:

$$\frac{-22.4673s - 723.9511}{s^2 + 2.5755s + 1.8556}$$

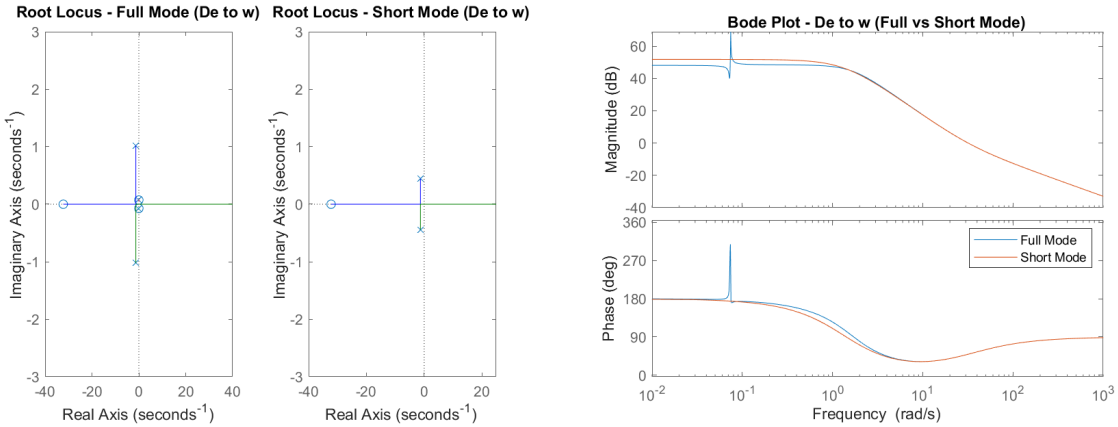


Figure 124: Root Locus and Bode plot for Δelevator (De) to vertical velocity (w)

Comment:

The Root Locus for the two modes has the same overall shape but differs in the number of poles; the full mode contains a larger number of poles compared to the long mode. However, the poles near the origin are positioned at the same values in both modes. In the Bode plot, both models align closely with each other except at a peak occurring at 10^{-1} rad/s in the full model. This peak is larger in the full model but quickly recovers, allowing the two plots to realign closely afterward.

The Delta Elevator (De) to Pitch Rate (q)

Full Mode Transfer Function:

$$\frac{-1.3503s^3 - 0.97176s^2 - 0.01316s + 6.3887 \times 10^{-19}}{s^4 + 2.5805s^3 + 2.7188s^2 + 0.015195s + 0.015019}$$

Short Mode Transfer Function:

$$\frac{-1.3503s - 1.0031}{s^2 + 2.5755s + 1.8556}$$

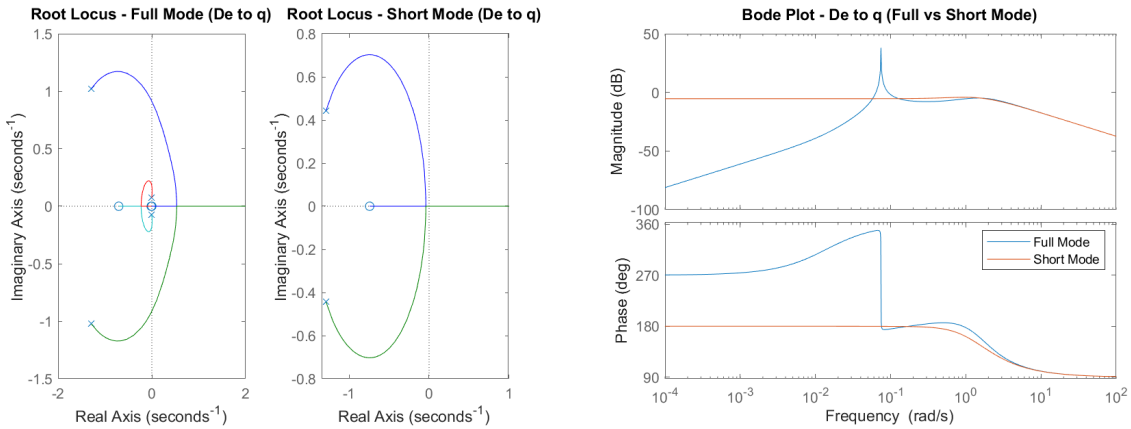


Figure 125: Root Locus and Bode plot for Δelevator (De) to pitch rate (q)

Comment:

In the Root Locus, the two modes differ due to the full mode having a greater number of poles than the short mode, which leads to a difference in locus shape. However, both modes share some of the same pole positions. In the Bode plot, the short model behaves differently from the full model at frequencies below 10^{-1} rad/s, but beyond this point, both plots follow a similar trend, aligning closely with each other.

The Delta Elevator (De) to Pitch Angle (θ)

Full Mode Transfer Function:

$$\frac{-1.3503s^2 - 0.97176s - 0.01316}{s^4 + 2.5805s^3 + 2.7188s^2 + 0.015195s + 0.015019}$$

Long Mode Transfer Function:

$$\frac{0.04411s + 0.00041561}{s^2 + 0.0014454s + 0.0052641}$$

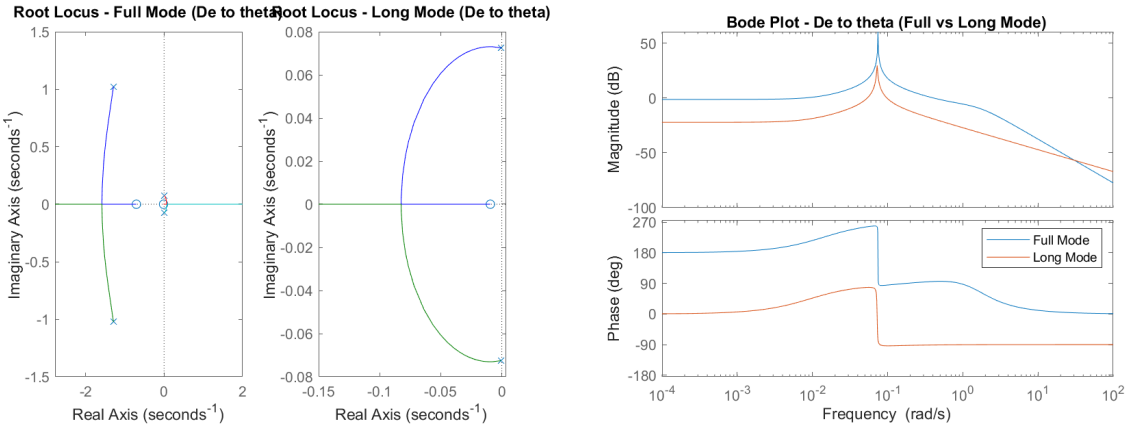


Figure 126: Root Locus and Bode plot for Δelevator (De) to pitch angle (θ)

Comment:

The Root Locus for the two modes differs because the full mode has a larger number of poles compared to the long mode. However, the poles near the origin are positioned at the same value in both modes. In the Bode plot, both modes follow the same trend but exhibit a 180-degree phase shift across all frequencies.

The Thrust (Th) to Linear Velocity (u)

Full Mode Transfer Function:

$$\frac{5.05 \times 10^{-5} s^3 + 0.0001299 s^2 + 0.00013771 s - 8.1018 \times 10^{-6}}{s^4 + 2.5805 s^3 + 2.7188 s^2 + 0.015195 s + 0.015019}$$

Long Mode Transfer Function:

$$\frac{5.05 \times 10^{-5} s - 3.2299 \times 10^{-7}}{s^2 + 0.0014454 s + 0.0052641}$$

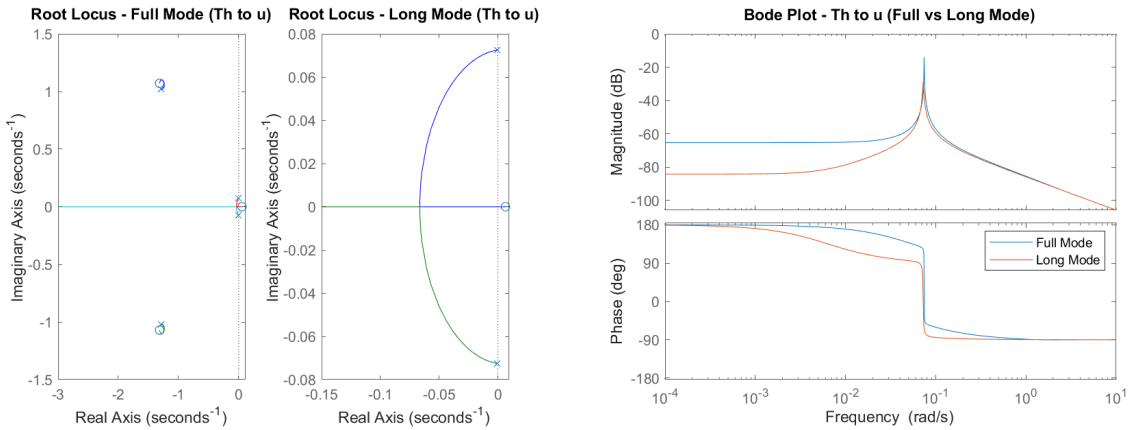


Figure 127: Root Locus and Bode plot for Thrust (Th) to linear velocity (u)

Comment:

The Root Locus for the two modes differs due to the full mode having a larger number of poles than the long mode. However, the poles near the origin are positioned at the same value in both modes. In the Bode plot, both modes follow a similar trend across all frequencies, except in the range from 10^{-3} to 10^{-1} rad/s, where there is a slight difference in phase and magnitude (dB), though it remains minor.

The Thrust (Th) to vertical velocity (w)

Full Mode Transfer Function:

$$\frac{-2.2673 \times 10^{-6} s^3 + 0.00014698 s^2 + 1.007 \times 10^{-6} s - 7.8039 \times 10^{-7}}{s^4 + 2.5805 s^3 + 2.7188 s^2 + 0.015195 s + 0.015019}$$

Short Mode Transfer Function:

$$\frac{-22.4673 s - 723.9511}{s^2 + 2.5755 s + 1.8556}$$

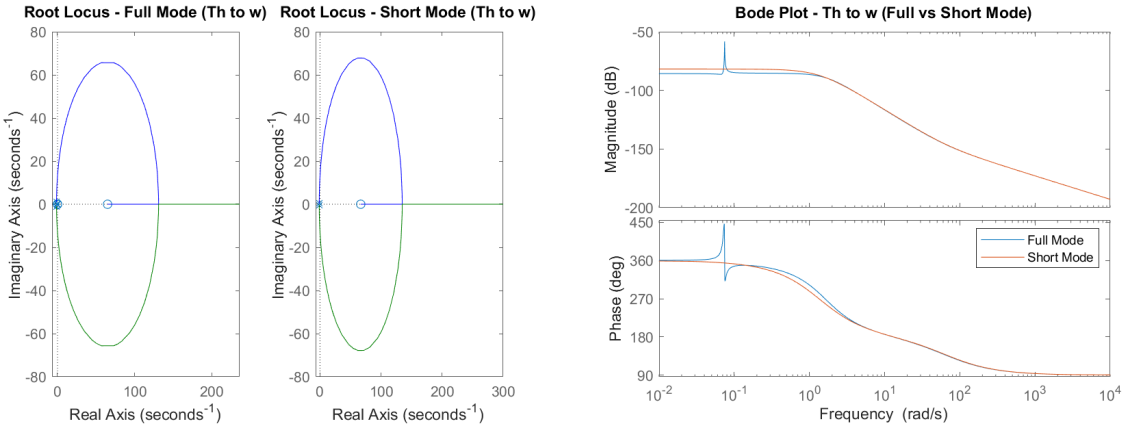


Figure 128: Root Locus and Bode plot for Thrust (Th) to vertical velocity (w)

Comment:

In the Root Locus, the short approximation positions poles nearly identically to the full model, resulting in similar system behavior. This similarity is also evident in the Bode plot, where both plots align closely with each other except at a peak occurring at 10^{-1} rad/s in the full model. In the full model, this peak is larger but quickly recovers, allowing the two plots to realign closely afterward.

The Thrust (Th) to Pitch Rate (q)

Full Mode Transfer Function:

$$\frac{3.0701 \times 10^{-7}s^3 + 2.57 \times 10^{-7}s^2 + 2.6671 \times 10^{-8}s + 2.1531 \times 10^{-26}}{s^4 + 2.5805s^3 + 2.7188s^2 + 0.015195s + 0.015019}$$

Short Mode Transfer Function:

$$\frac{-1.3503s - 1.0031}{s^2 + 2.5755s + 1.8556}$$

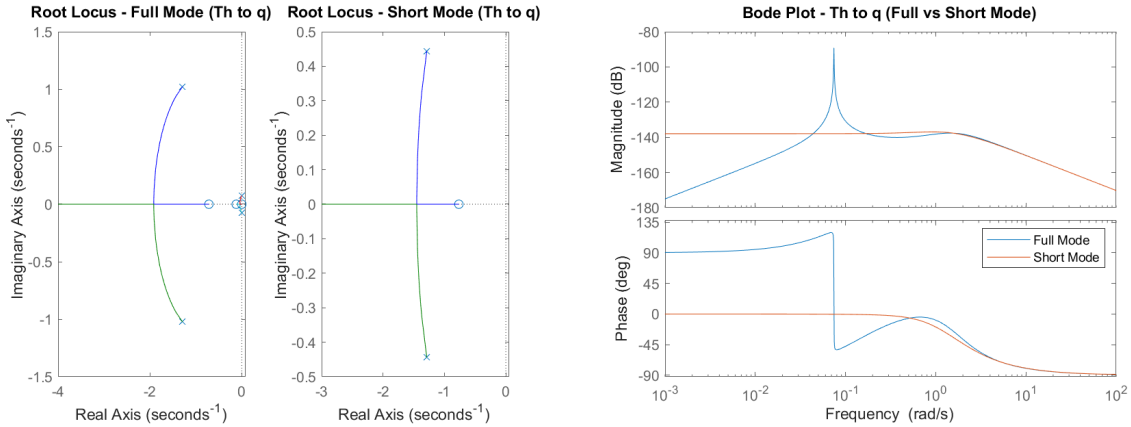


Figure 129: Root Locus and Bode plot for Thrust (Th) to pitch rate (q)

Comment:

In the Root Locus, both models exhibit nearly the same shape, but the poles are positioned at different values. Specifically, the pole in the short mode is approximately half the imaginary axis value of the full mode's pole. In the Bode plot, the short model behaves differently than the full model before a frequency of 10^{-1} rad/s, but beyond this point, both plots follow a similar trend, aligning closely with each other.

The Thrust (Th) to Pitch Angle (θ)

Full Mode Transfer Function:

$$\frac{3.0701 \times 10^{-7} s^2 + 2.57 \times 10^{-7} s + 2.6671 \times 10^{-8}}{s^4 + 2.5805 s^3 + 2.7188 s^2 + 0.015195 s + 0.015019}$$

Long Mode Transfer Function:

$$\frac{4.4819 \times 10^{-9} s + 8.3247 \times 10^{-9}}{s^2 + 0.0014454 s + 0.0052641}$$

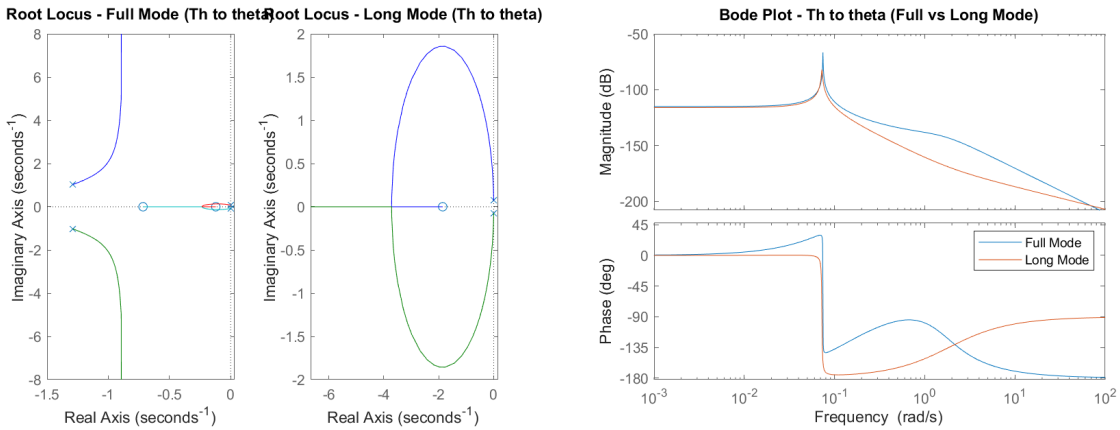


Figure 130: Root Locus and Bode plot for Thrust (Th) to pitch angle (θ)

Comment:

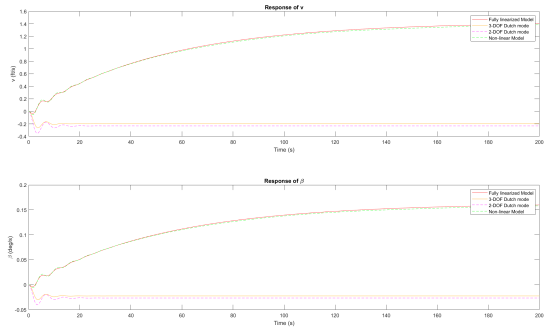
The Root Locus for the long approximation places poles closer to the origin, providing a slower, more stable response for pitch angle control. The full model's poles are further left, indicating higher responsiveness. In the Bode plot, both models behave similarly up to a frequency of 10^{-1} rad/s, after which noticeable differences in both phase and gain (dB) begin to appear.

6.6.3 Lateral Step Response

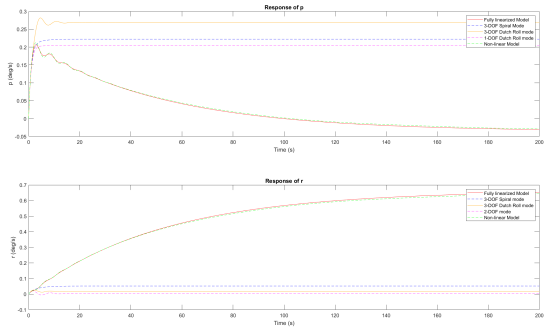
Aileron Input Step Responses

Each graph below shows the step response for Non-linear simulation, Linear full model, 3-DOF, 2-DOF, and 1-DOF approximations for different aileron deflections δ_a.

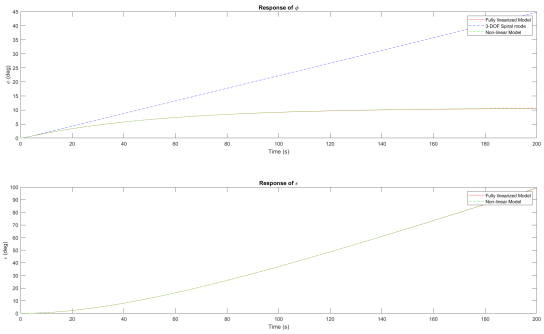
Step Response with Aileron Deflection $\delta_a = 1^\circ$



(a) Step Response for $\delta_a = 1^\circ$ (V-β)



(b) Step Response for $\delta_a = 1^\circ$ (p-r)



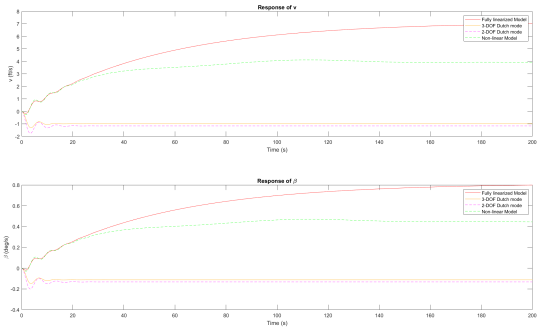
(c) Step Response for $\delta_a = 1^\circ$ (ϕ-ϵ)

Figure 131: **Step responses for $\delta_a = 1^\circ$**

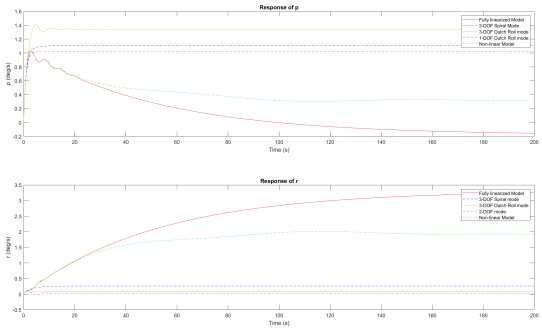
Comment:

- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

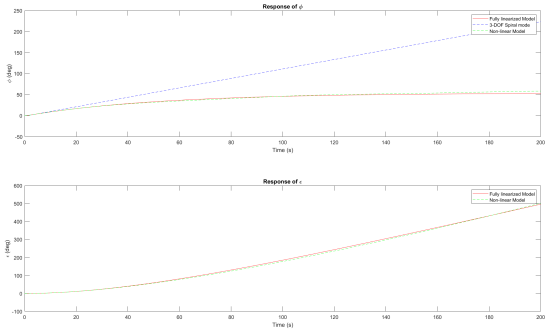
Step Response with Aileron Deflection $\delta_a = 5°$



(a) Step Response for $\delta_a = 5°$ (V-β)



(b) Step Response for $\delta_a = 5°$ (p-r)



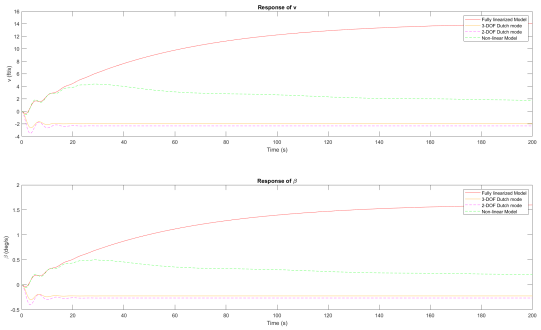
(c) Step Response for $\delta_a = 5°$ (ϕ-ϵ)

Figure 132: **Step responses for $\delta_a = 5°$**

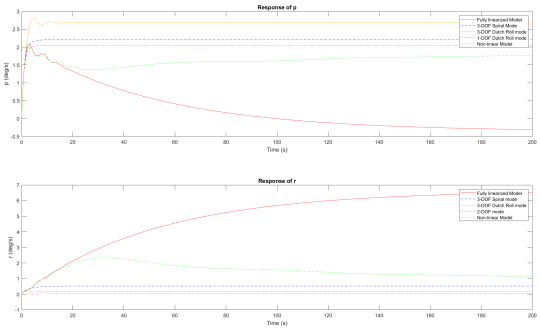
Comment:

- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

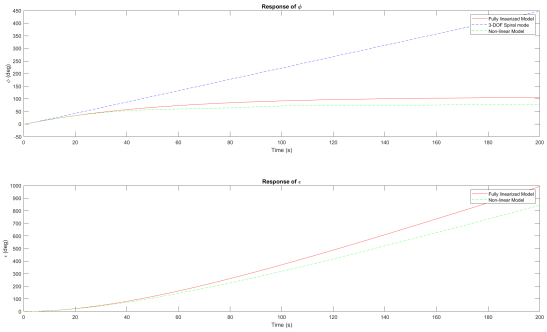
Step Response with Aileron Deflection $\delta_a = 10°$



(a) Step Response for $\delta_a = 10°$ (V-β)



(b) Step Response for $\delta_a = 10°$ (p-r)



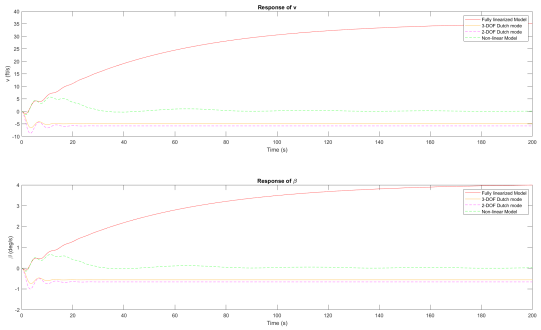
(c) Step Response for $\delta_a = 10°$ (ϕ-ϵ)

Figure 133: **Step responses for $\delta_a = 10°$**

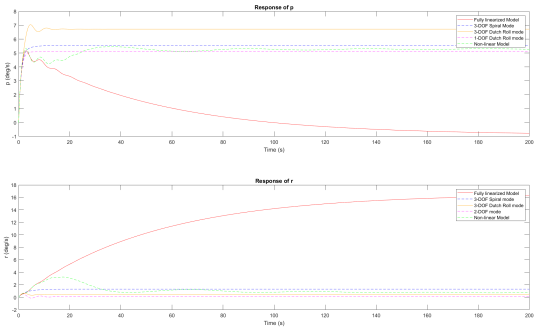
Comment:

- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

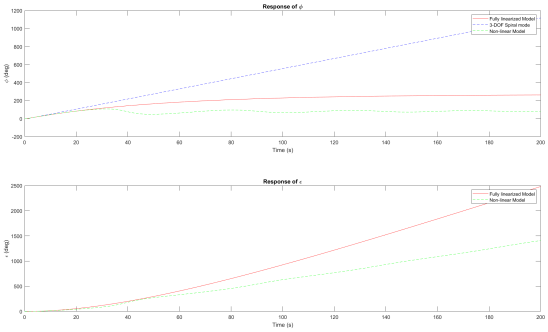
Step Response with Aileron Deflection $\delta_a = 25°$



(a) Step Response for $\delta_a = 25°$ (V-β)



(b) Step Response for $\delta_a = 25°$ (p-r)



(c) Step Response for $\delta_a = 25°$ (ϕ-ϵ)

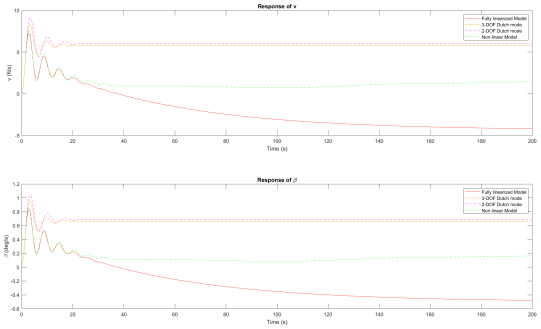
Figure 134: **Step responses for $\delta_a = 25°$**

Comment:

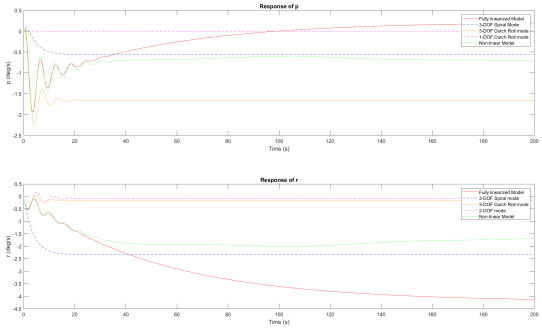
- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

Rudder Input Step Responses

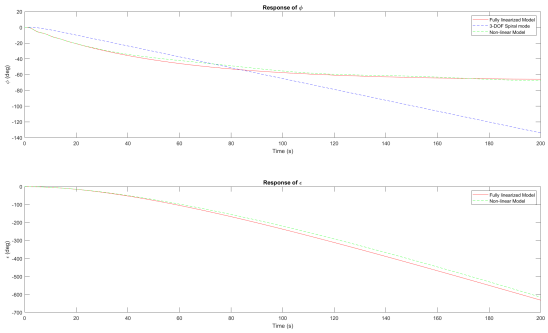
Each graph below shows the step response for Non-linear simulation, Linear full model, 3-DOF, 2-DOF, and 1-DOF approximations for different rudder inputs.



(a) Step Response for $\delta_r = 1^\circ$ (V-β)



(b) Step Response for $\delta_r = 1^\circ$ (p-r)



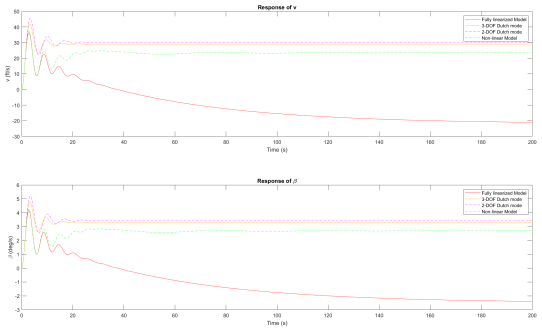
(c) Step Response for $\delta_r = 1^\circ$ (ϕ-ϵ)

Figure 135: **Step responses for $\delta_r = 1^\circ$**

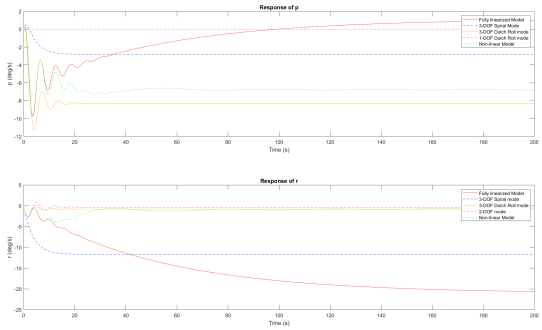
Comment:

- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

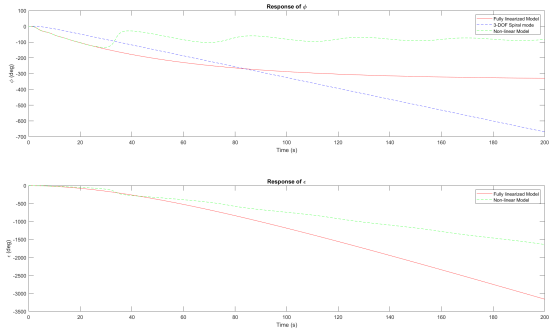
Step response with Rudder Deflection $\delta r = 5°$



(a) Step Response for $\delta_r = 5°$ (V-β)



(b) Step Response for $\delta_r = 5°$ (p-r)



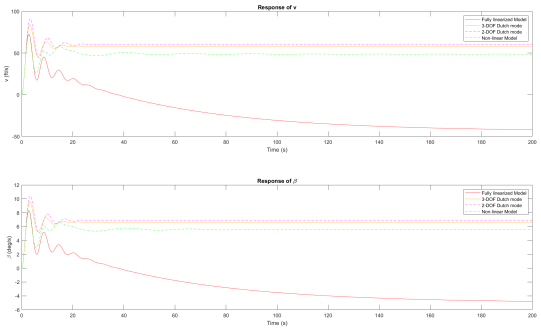
(c) Step Response for $\delta_r = 5°$ (ϕ-ϵ)

Figure 136: **Step responses for $\delta_r = 5°$**

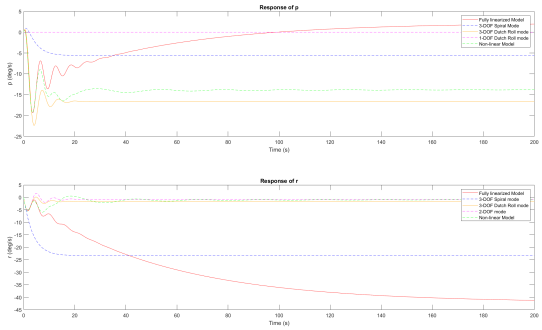
Comment:

- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

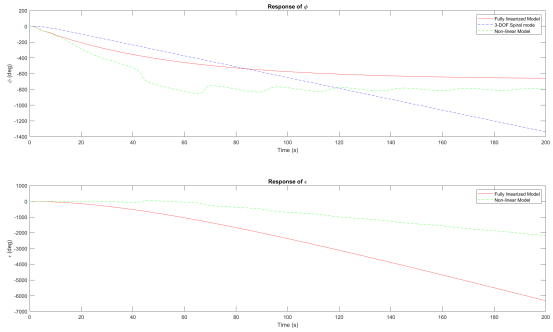
Step response with Rudder Deflection $\delta_r = 10°$



(a) Step Response for $\delta_r = 10°$ (V-β)



(b) Step Response for $\delta_r = 10°$ (p-r)



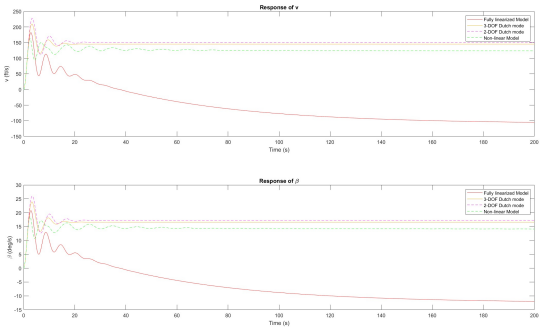
(c) Step Response for $\delta_r = 10°$ (ϕ-ϵ)

Figure 137: **Step responses for $\delta_r = 10°$**

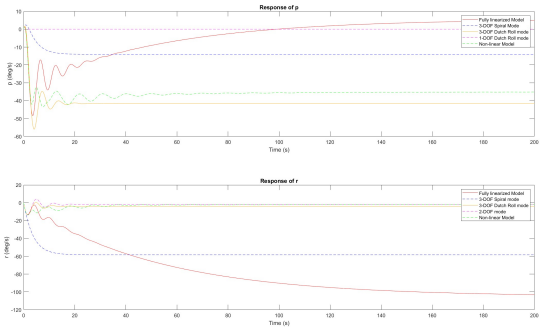
Comment:

- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

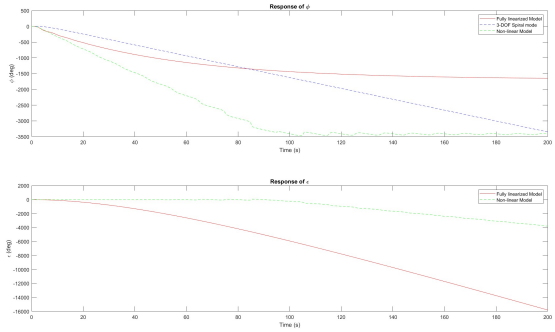
Step response with Rudder Deflection $\delta r = 25°$



(a) Step Response for $\delta_r = 25°$ (V-β)



(b) Step Response for $\delta_r = 25°$ (p-r)



(c) Step Response for $\delta_r = 25°$ (ϕ-ϵ)

Figure 138: **Step responses for $\delta_r = 25°$**

Comment:

- **The aileron deflection predominantly excites roll dynamics with coupled yaw and sideslip motions.**
- **The fully linearized model and the non-linear model show excellent agreement.**
- **The 3-DOF Spiral mode predicts roll instability over time, while the Dutch Roll mode underestimates lateral and yaw dynamics.**
- **The responses highlight the coupling between roll, yaw, and sideslip motion in lateral-directional dynamics.**

6.6.4 Transfer Functions and Root Locus & Bode Plot for Each Input and Output State For Lateral Mode

The Aileron Deflection (Da) to Side Velocity (v)

Full Mode Transfer Function:

$$\frac{-5.87s^2 + 1.2827s + 2.182}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

3-DOF Dutch Roll Mode Transfer Function:

$$\frac{-5.87s - 11.8808}{s^3 + 1.5126s^2 + 1.5245s + 1.057}$$

2-DOF Dutch Roll Mode Transfer Function:

$$\frac{-12.0456}{s^2 + 0.3964s + 0.90722}$$

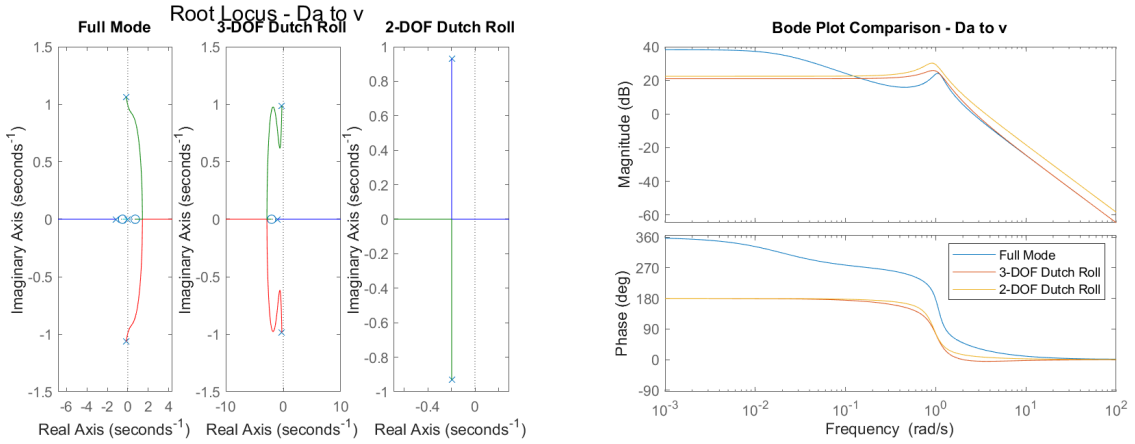


Figure 139: Root Locus and Bode plot for Δaileron (Da) to side velocity (v)

Comment:

The Root Locus for the full mode differs from that of the 3-DOF Dutch roll due to the presence of two zeros in the full mode, whereas the 3-DOF model has only one zero, and the 2-DOF model has none. This difference in zeros leads to distinct Root Locus patterns across the modes.

In the Bode plots, the full mode shows approximately double the magnitude and a 180-degree phase shift compared to the 3-DOF and 2-DOF modes over the lower frequency range (up to 10^0 rad/s). Beyond this point, all modes converge, aligning closely in both phase and magnitude.

The Aileron Deflection (Da) to Roll Rate (p)

Full Mode Transfer Function:

$$\frac{0.22748s^3 + 0.099595s^2 + 0.2853s - 0.00095702}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

3-DOF Dutch Roll Mode Transfer Function:

$$\frac{0.22748s^2 + 0.090173s + 0.28396}{s^3 + 1.5126s^2 + 1.5245s + 1.057}$$

1-DOF Roll Mode Transfer Function:

$$\frac{0.22748}{s + 1.1162}$$

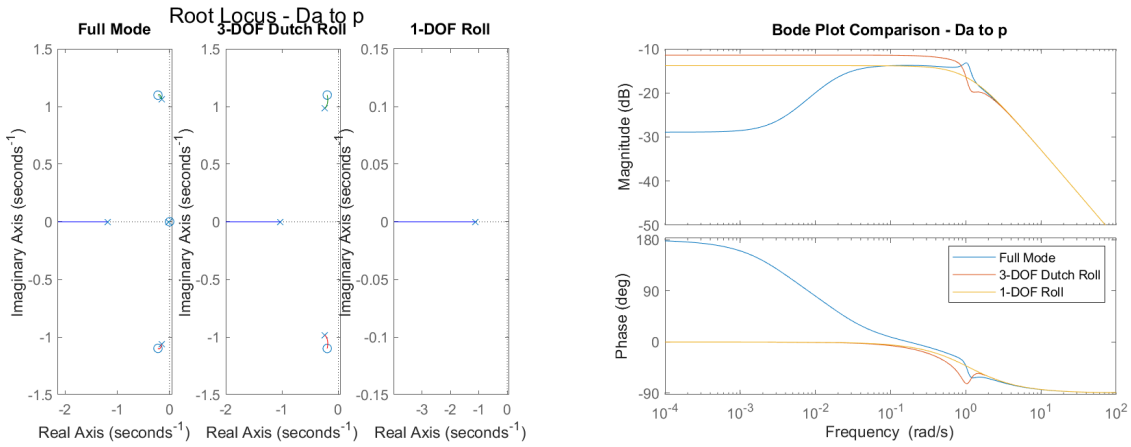


Figure 140: Root Locus and Bode plot for Δaileron (Da) to roll rate (p)

Comment:

The Root Locus for the full mode has 3 zeros and 4 poles, while the 3-DOF model has 2 zeros and 3 poles, excluding the zero and pole at the origin present in the full mode. The 1-DOF model, by contrast, has only one pole and no zeros, resulting in a simpler locus shape.

In the Bode plots, the full mode exhibits approximately three times the magnitude and a 180-degree phase shift compared to the 3-DOF and 1-DOF modes over the lower frequency range (up to 10^0 rad/s). Beyond this frequency, all modes closely align in both phase and magnitude.

The Aileron Deflection (Da) to Yaw Rate (r)

Full Mode Transfer Function:

$$\frac{0.02403s^3 + 0.019173s^2 + 0.017182s + 0.017671}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

3-DOF Dutch Roll Mode Transfer Function:

$$\frac{0.02403s^2 + 0.03026s + 0.018768}{s^3 + 1.5126s^2 + 1.5245s + 1.057}$$

2-DOF Dutch Roll Mode Transfer Function:

$$\frac{0.02403s + 0.0034363}{s^2 + 0.3964s + 0.90722}$$

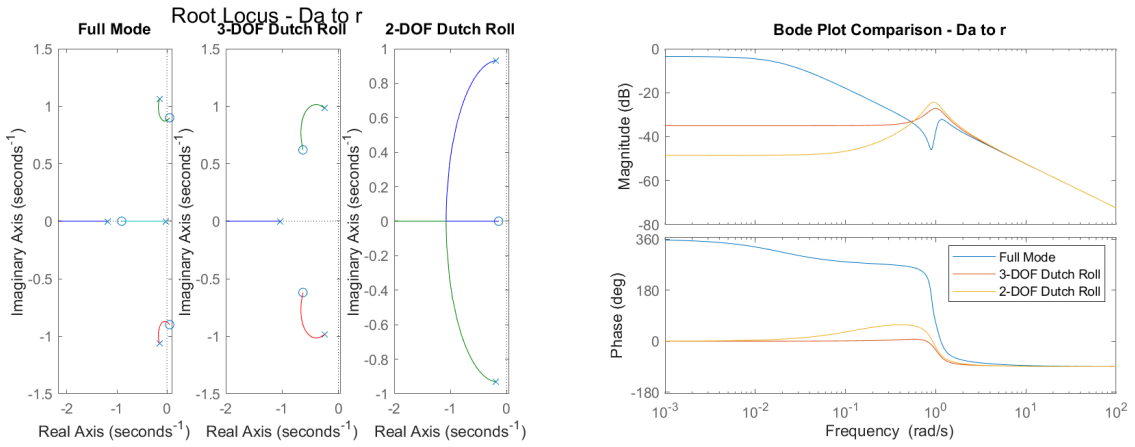


Figure 141: Root Locus and Bode plot for Δaileron (Da) to yaw rate (r)

Comment:

The Root Locus for the full mode has 3 zeros and 4 poles, while the 3-DOF model has 2 zeros and 3 poles, excluding the zero and pole located on the real axis. The 2-DOF model has only one zero and two poles, resulting in a simpler locus shape for each reduced mode.

In the Bode plots, the magnitude for the full mode starts at 0 dB, while the 3-DOF model begins at -38 dB and the 2-DOF model at -50 dB. Additionally, there is a 360-degree phase shift between the full mode and both the 3-DOF and 2-DOF modes over the lower frequency range (up to 10^0 rad/s). Beyond this frequency, all modes align closely in both phase and magnitude.

The Aileron Deflection (Da) to Roll Angle (phi)

Full Mode Transfer Function:

$$\frac{0.22878s^2 + 0.10063s + 0.28623}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

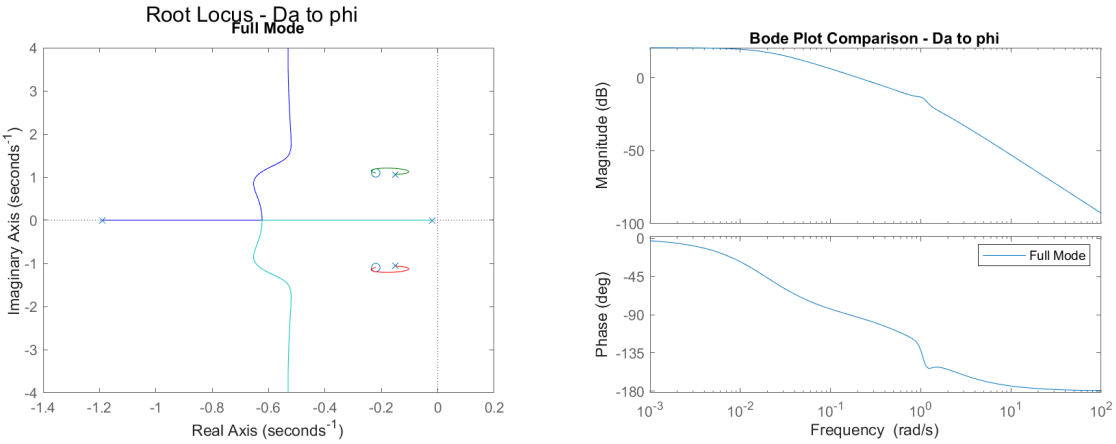


Figure 142: Root Locus and Bode plot for Δaileron (Da) to roll angle (phi)

Comment:

This transfer function appears only in the full mode, as shown in the Root Locus, where it has two zeros and four poles. In the Bode plot, both the phase and magnitude decrease over the entire frequency range, with the phase starting at 0° and dropping to -180°, and the magnitude starting at 20 dB and decreasing to -100 dB.

The Aileron Deflection (Da) to Yaw Angle (epsi)

Full Mode Transfer Function:

$$\frac{0.024066s^3 + 0.019201s^2 + 0.017208s + 0.017697}{s^5 + 1.5126s^4 + 1.5437s^3 + 1.4025s^2 + 0.026622s}$$

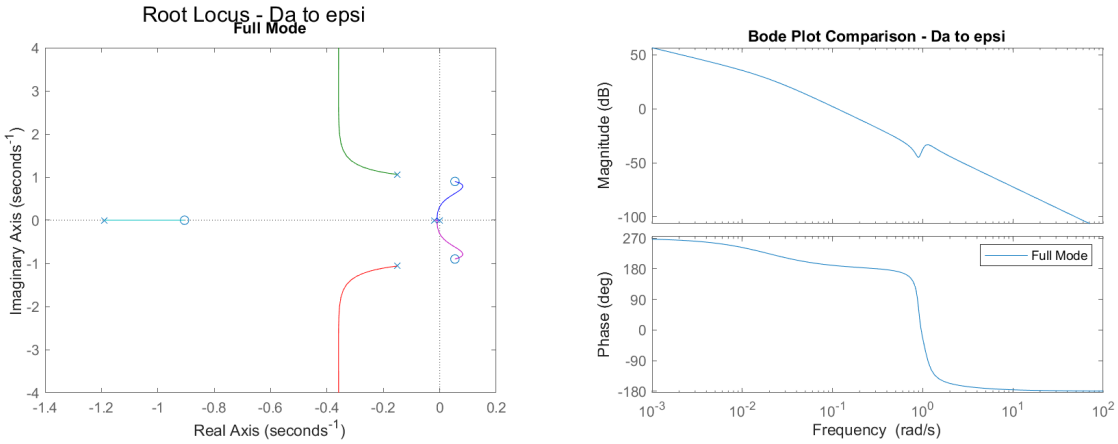


Figure 143: Root Locus and Bode plot for Δaileron (Da) to yaw angle (epsi)

Comment:

This transfer function appears only in the full mode, as shown in the Root Locus, where it has three zeros and five poles. Two of the zeros are located on the positive real axis, and there is a pole at the origin, making the system critically stable.
In the Bode plot, both the phase and magnitude decrease over the entire frequency range, with the phase starting at 270° and dropping to -180°, and the magnitude starting at 50 dB and decreasing to -100 dB.

The Rudder Deflection (Dr) to Side Velocity (v)

Full Mode Transfer Function:

$$\frac{11.3452s^3 + 333.5847s^2 + 360.2753s - 6.6895}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

3-DOF Dutch Roll Mode Transfer Function:

$$\frac{11.3452s^2 + 333.5847s + 351.5074}{s^3 + 1.5126s^2 + 1.5245s + 1.057}$$

2-DOF Dutch Roll Mode Transfer Function:

$$\frac{11.3452s + 313.1369}{s^2 + 0.3964s + 0.90722}$$

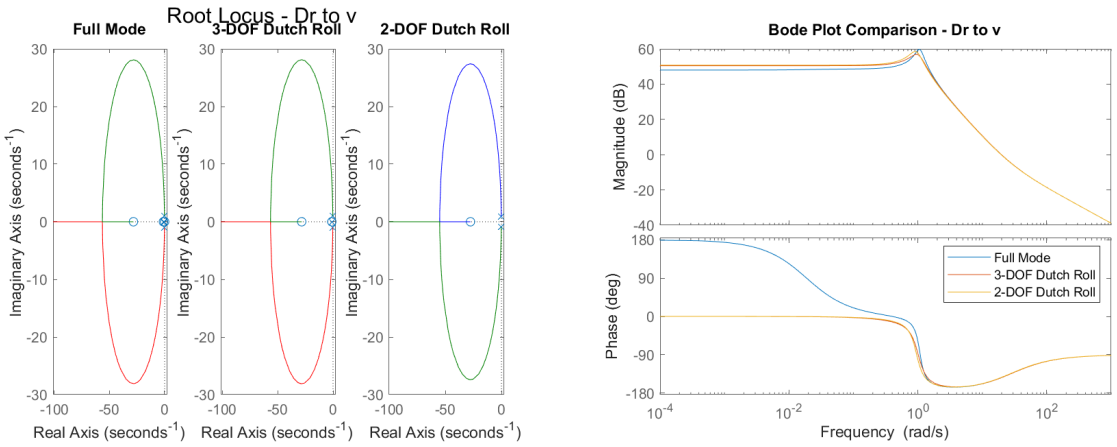


Figure 144: Root Locus and Bode plot for Δrudder (Dr) to side velocity (v)

Comment:

In the Root Locus, the full mode, 3-DOF, and 2-DOF models have different numbers of zeros and poles, but their Root Locus shapes are nearly identical. This similarity may be due to the fact that in each plot, the number of zeros is only one less than the number of poles.

In the Bode plot, all three models show identical magnitude responses throughout the frequency range, starting at 50 dB. However, there is an initial phase shift between the full mode and the 3-DOF and 2-DOF models up to a frequency of 10^0 rad/s, after which all three models align closely in phase.

The Rudder Deflection (Dr) to Roll Rate (p)

Full Mode Transfer Function:

$$\frac{0.28673s^3 - 0.20211s^2 - 1.7836s + 0.0060693}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

3-DOF Spiral Mode Transfer Function:

$$\frac{0.28673s - 0.17004}{s^2 + 1.3696s + 0.30196}$$

3-DOF Dutch Roll Mode Transfer Function:

$$\frac{0.28673s^2 + 0.040588s - 1.7567}{s^3 + 1.5126s^2 + 1.5245s + 1.057}$$

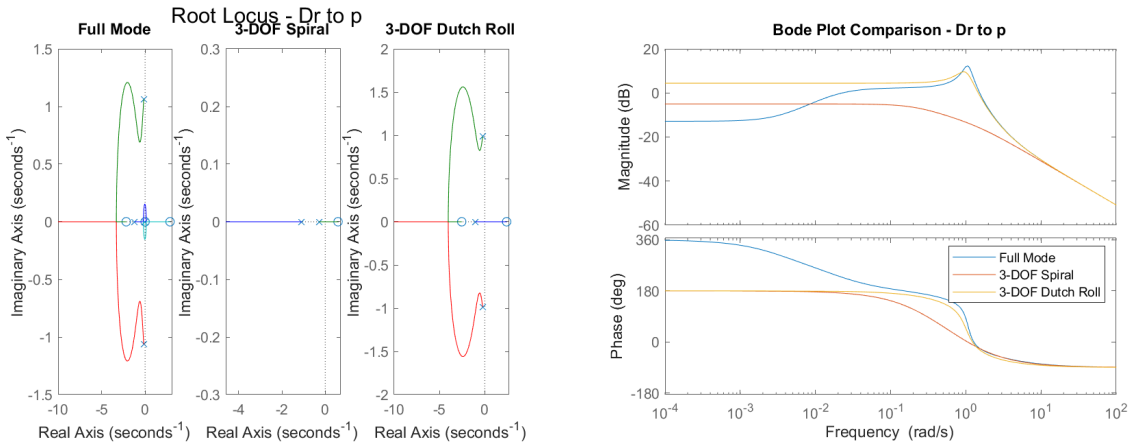


Figure 145: Root Locus and Bode plot for Δrudder (Dr) to roll rate (p)

Comment:

The Root Locus for the full mode has 3 zeros and 4 poles, while the 3-DOF spiral model has 2 poles and 1 zero, and the 3-DOF Dutch roll model has 3 poles and 2 zeros. Notably, the Root Locus shape for the 3-DOF Dutch roll model closely resembles that of the full mode.

In the Bode plots, the magnitude for the full mode starts at -10 dB, while the 3-DOF spiral model begins at -5 dB, and the 3-DOF Dutch roll model starts at 5 dB. Additionally, there is a 180-degree phase shift between the full mode and both the 3-DOF spiral and Dutch roll modes over the lower frequency range (up to 10^0 rad/s). Beyond this frequency, all modes closely align in both phase and magnitude. In the frequency range from 10^{-1} to 10^0 rad/s, there is a slight phase variation between the 3-DOF spiral and Dutch roll models, but they quickly realign for the remainder of the frequency range.

The Rudder Deflection (Dr) to Yaw Rate (r)

Full Mode Transfer Function:

$$\frac{-0.61896s^3 - 0.77368s^2 - 0.16993s - 0.11207}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

3-DOF Spiral Mode Transfer Function:

$$\frac{-0.61896s - 0.70488}{s^2 + 1.3696s + 0.30196}$$

3-DOF Dutch Mode Transfer Function:

$$\frac{-0.61896s^2 - 0.7597s - 0.17149}{s^3 + 1.5126s^2 + 1.5245s + 1.057}$$

2-DOF Roll Mode Transfer Function:

$$\frac{-0.61896s - 0.068798}{s^2 + 0.3964s + 0.90722}$$

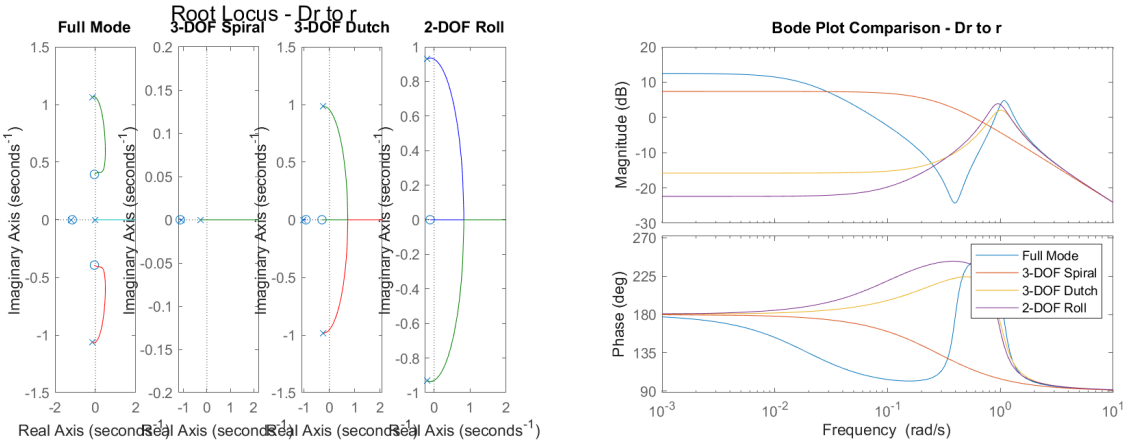


Figure 146: Root Locus and Bode plot for Δrudder (Dr) to yaw rate (r)

Comment:

In the Root Locus, the full mode has 3 zeros and 4 poles, while the 3-DOF spiral mode has 2 poles and 1 zero, with this zero positioned on the real axis in the full mode. The 3-DOF Dutch roll and 2-DOF roll modes share the same Root Locus shape. Although the 3-DOF model includes an additional pole and zero at the origin compared to the 2-DOF model, these do not affect the final Root Locus shape.

In the Bode plots, both the full mode and 3-DOF spiral mode start with similar magnitude trends, beginning at 12 dB and 9 dB, respectively, and following the same trend for the first half of the frequency range. The 3-DOF Dutch roll and 2-DOF roll modes start at -18 dB and -21 dB, respectively, and also share a similar magnitude trend in the first half. In terms of phase, all modes start at 180°, but each mode behaves differently throughout the frequency range, gradually converging to the same phase at the final frequency, 10^1 rad/s.

The Rudder Deflection (Dr) to Roll Angle (phi)

Full Mode Transfer Function:

$$\frac{0.2532s^2 - 0.24401s - 1.7928}{s^4 + 1.5126s^3 + 1.5437s^2 + 1.4025s + 0.026622}$$

3-DOF Spiral Mode Transfer Function:

$$\frac{0.2532s - 0.20822}{s^3 + 1.3696s^2 + 0.30196s}$$

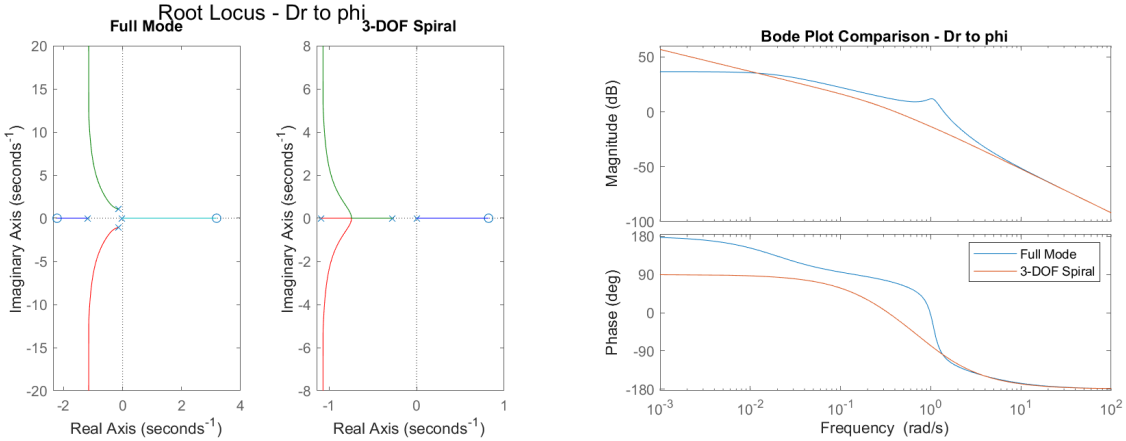


Figure 147: Root Locus and Bode plot for Δaileron (Dr) to roll angle (phi)

Comment:

In the Root Locus, the full mode has 4 poles and 2 zeros, while the 3-DOF spiral mode has only 1 zero and 3 poles. This difference in the number of zeros and poles results in distinct Root Locus shapes for the two modes.

In the Bode plot, both modes follow the same trend in both magnitude and phase over the entire frequency range. However, at the beginning of the range, up to 10^0 rad/s, there is a 90-degree phase shift. This phase difference gradually decreases with increasing frequency until it aligns at 10^0 rad/s.

The Rudder Deflection (Dr) to Yaw Angle (epsi)

Full Mode Transfer Function:

$$\frac{-0.61986s^3 - 0.77481s^2 - 0.17018s - 0.11223}{s^5 + 1.5126s^4 + 1.5437s^3 + 1.4025s^2 + 0.026622s}$$

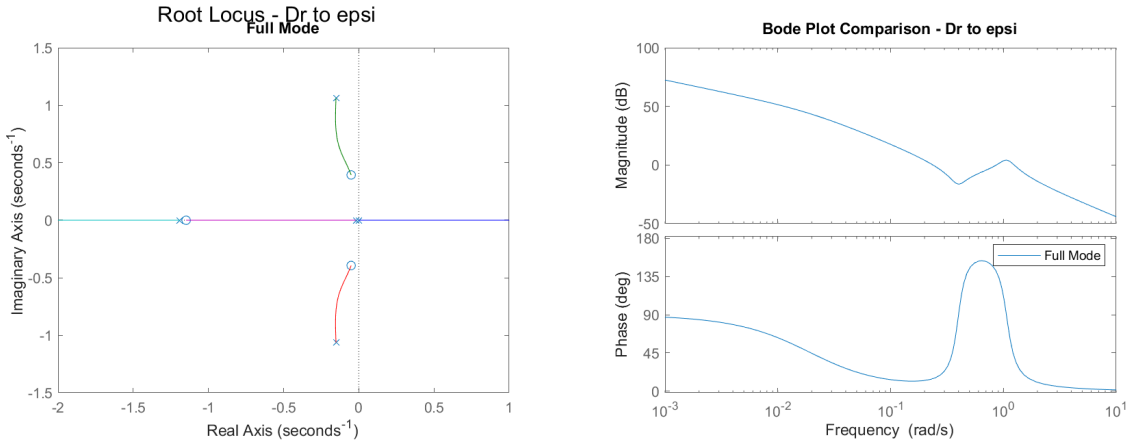


Figure 148: Root Locus and Bode plot for Δaileron (Dr) to yaw angle (epsi)

Comment:

This transfer function appears only in the full mode, as shown in the Root Locus, where it has three zeros and five poles. A pole is located at the origin, making the system critically stable.

In the Bode plot, both the phase and magnitude decrease over the entire frequency range. The phase starts at 90° and drops to -180°, with a significant peak occurring at 10^0 rad/s. The magnitude begins at 70 dB and decreases to -50 dB.

6.7 Longitudinal autopilot

6.7.1 Pitch control

The purpose of this section is to design the control loop to control the altitude using pitch, where the pitch is controlled using a elevator:
elevator deflection (δe)$\hookrightarrow$ pitch angle (θ) $\hookrightarrow$ altitude (H)

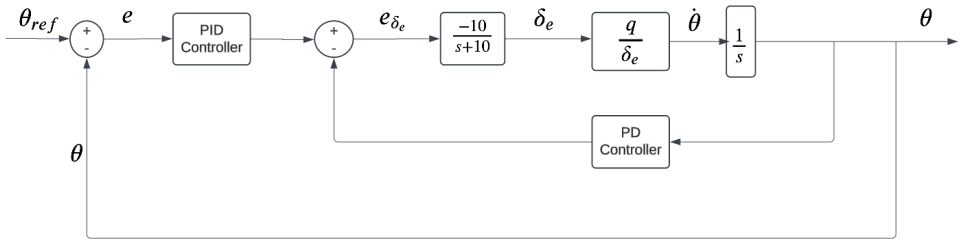


Figure 149: Control loop block diagram

Where:

$$\frac{q}{\delta_e} = \frac{-1.35s^3 - 0.9718s^2 - 0.01316s + 6.389 \times 10^{-19}}{s^4 + 2.581s^3 + 2.719s^2 + 0.01519s + 0.01502}$$

and $\frac{-10}{s+10}$ is the transfer function representing the servo actuator dynamics, The negative sign is introduced to counteract the negative sign in $\frac{q}{\delta_e}$, or alternatively, it can be achieved physically by installing the servo in reverse.

1-Design requirements
According to Cooper Harper flying qualities, we chose a $\zeta_{min} = 0.3$ and $\zeta_{max} = 2$ for the short period mode, and for the long-period we set $\zeta_{min} = 0.04$.

Phugoid mode	
Level 1	$\zeta > 0.04$
Level 2	$\zeta > 0$
Level 3	$T > 55$s

Short period mode	**Category A and C**		**Category B**	
	ζ_{min}	ζ_{max}	ζ_{min}	ζ_{max}
Level 1	0.35	1.30	0.3	2.0
Level 2	0.25	2.00	0.2	2.0
Level 3	0.15	-	0.15	-

2- Open Loop analysis:
As seen from the root locus after applying the boundaries of $\zeta_{min} = 0.04$ for long period and $\zeta_{min} = 0.3$ for short period, we can clearly see in the figures below that the short period's poles are achieving the design requirements, while the long-period's poles aren't.

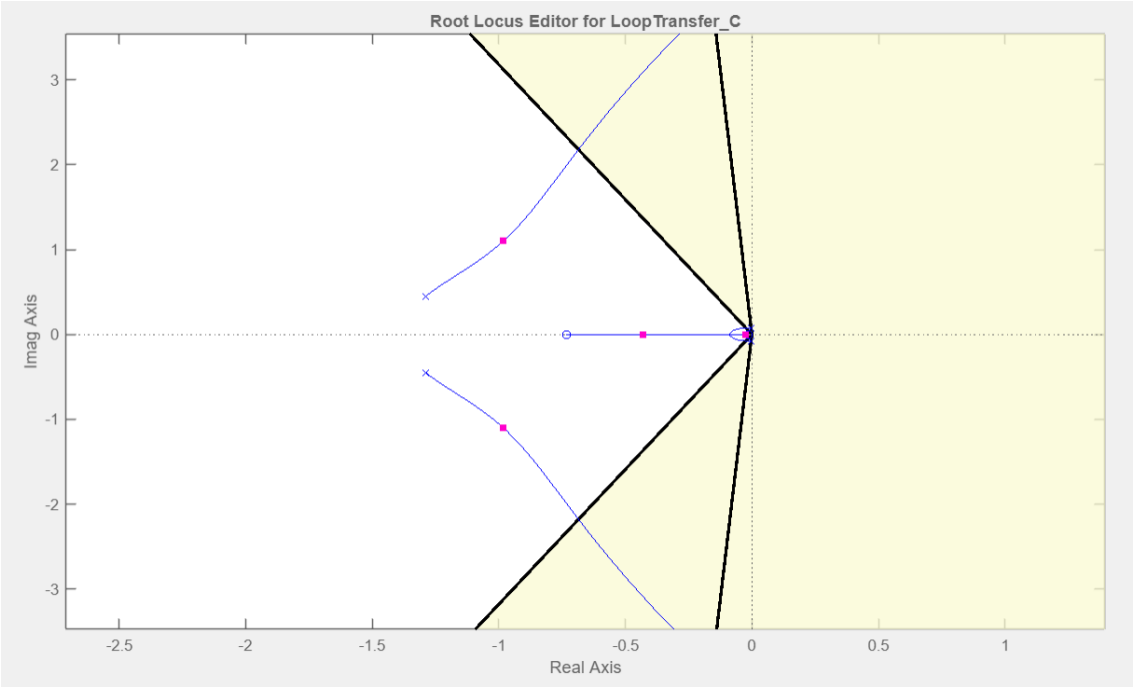


Figure 150: Open loop short period

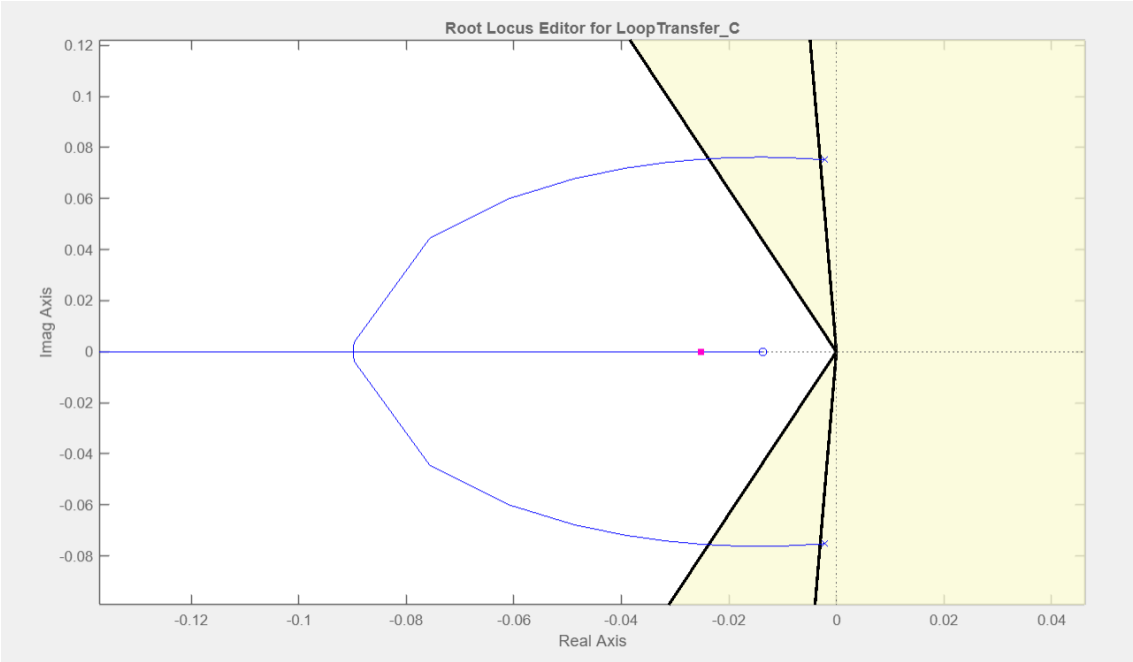


Figure 151: Open loop Long period

Also, we can notice in the step response shown below that the settling time is 151 seconds with an overshoot percentage of 69.4%,Therefore, pitch control is required.

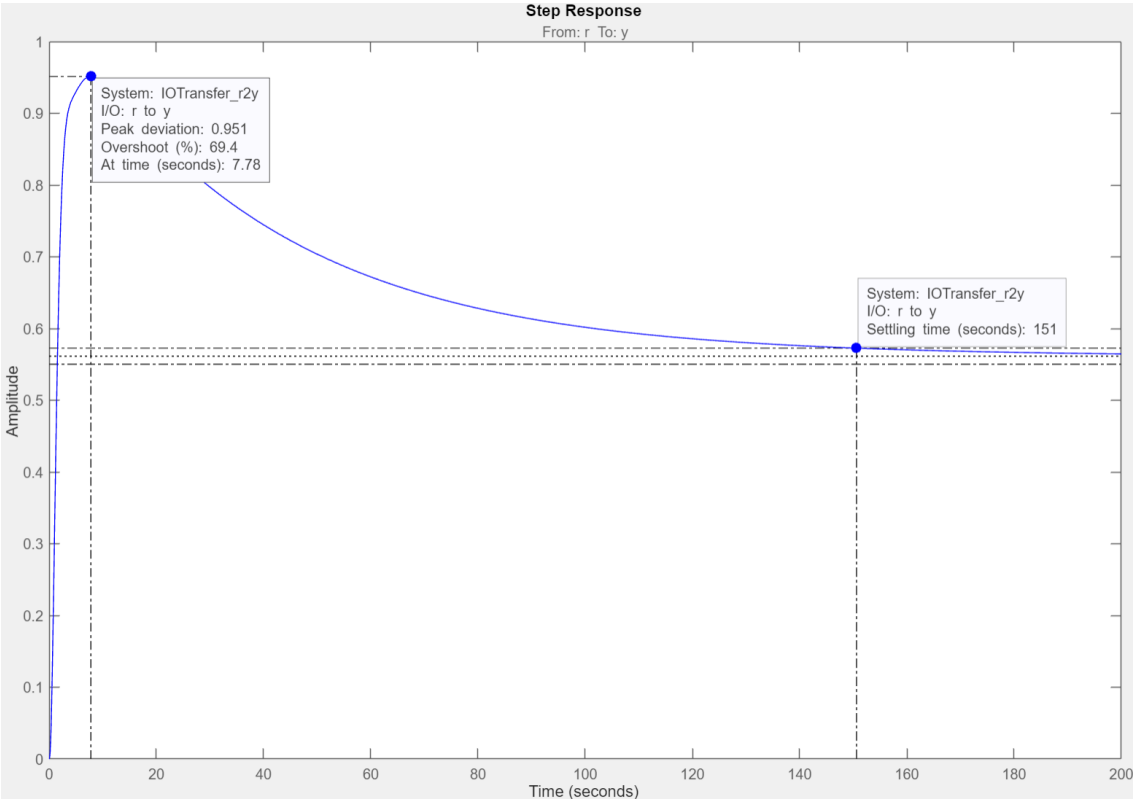


Figure 152: Open Loop Step Response

3- Closed Loop analysis:
To tune the model in Matlab we used the Control System Designer application. After trial and error, we reached this final pole position.

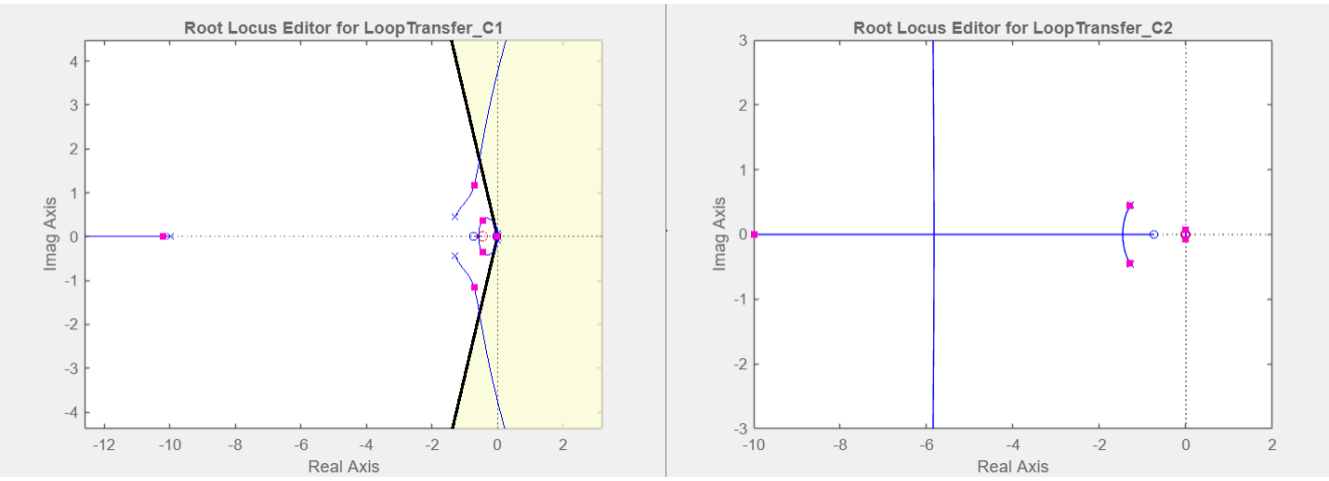


Figure 153: Closed loop root locus with control

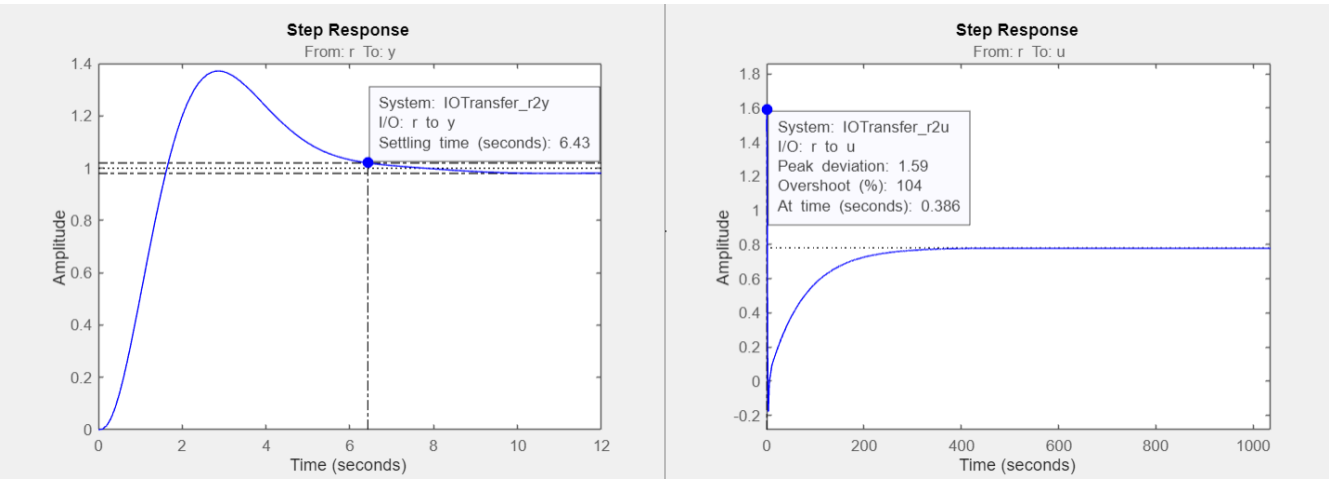


Figure 154: Closed loop step response

After tuning, the output values of the transfer functions of the controller turned out to be:

PI:

$$\frac{1.4618\,(s+0.4445)}{s}$$

PD:

$$0.005915s$$

The final settling time was 6.43 seconds, with a percentage overshoot of 37.1, and a peak deviation of 1.59.
The peak divination indicates that for each $1°\delta_e$), the pitch angle will change with 1.59°. Thus with the maximum deviation of 15° for elevator, the pitch angle will be $\theta = 15 \cdot 1.59 = 23.85°$.

4- Simulink Model:
After implementing the Milti Input Multi Output (MIMO) system in Simulink, these results were obtained when giving a command $\theta = 15\,\text{deg}$

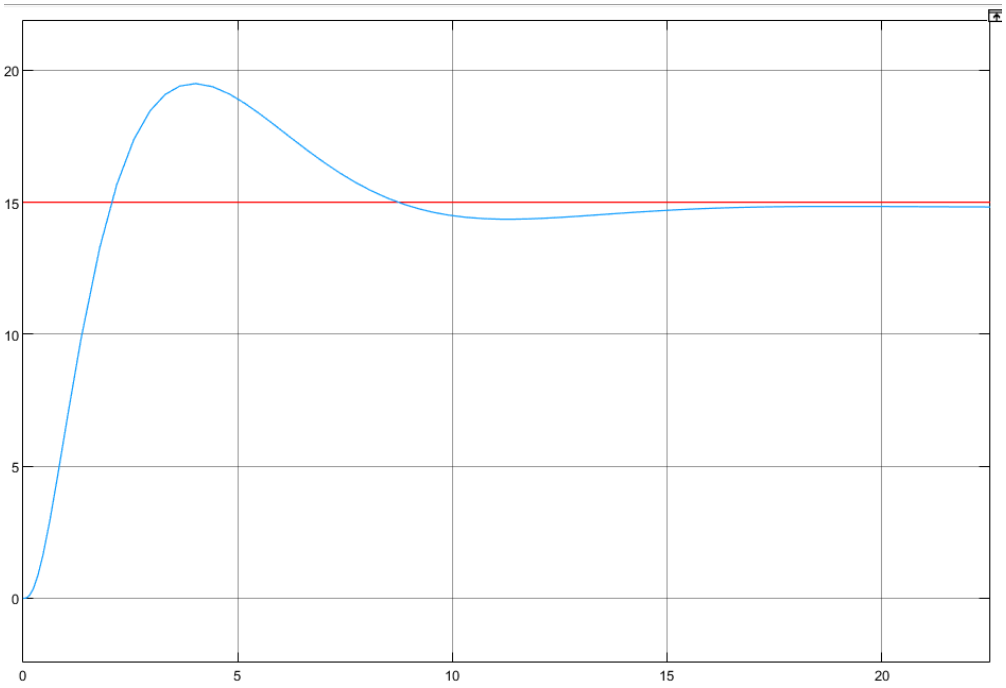


Figure 155: Step Response when $\theta = 15°$

To calculate the altitude, we used the following equations:

$$\dot{h} = V_{T0}\sin(\gamma)$$

Where

$$\gamma = \theta - \alpha$$
$$\theta = \theta_0 + \Delta\theta$$
$$\alpha = \tan^{-1}\left(\frac{w}{u}\right)$$
$$u = u_0 + \Delta u$$
$$w = w_0 + \Delta w$$
$$V_{T0} = \sqrt{u^2 + v^2 + w^2} = \sqrt{u^2 + w^2}$$

By differentiating $\dot{h}$, we can get $H = h + h_0$ where $h_0 = -z_0$

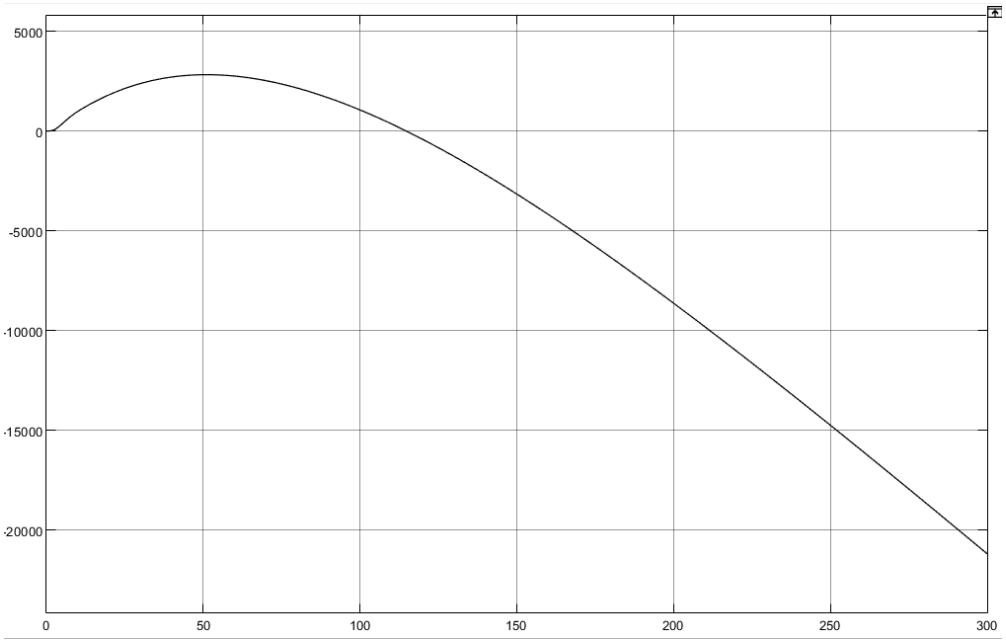


Figure 156: Altitude vs. time when $\theta = 15°$

We observe that the airplane initially ascends but then begins to descend. This behavior can be explained by the decrease in velocity; as the velocity decreases, the airplane lacks sufficient energy to sustain both a pitch angle of 15° and a continued increase in altitude. Consequently, it starts to descend while maintaining the pitch angle of 15°. A graph of the velocity is provided below to illustrate this behavior.

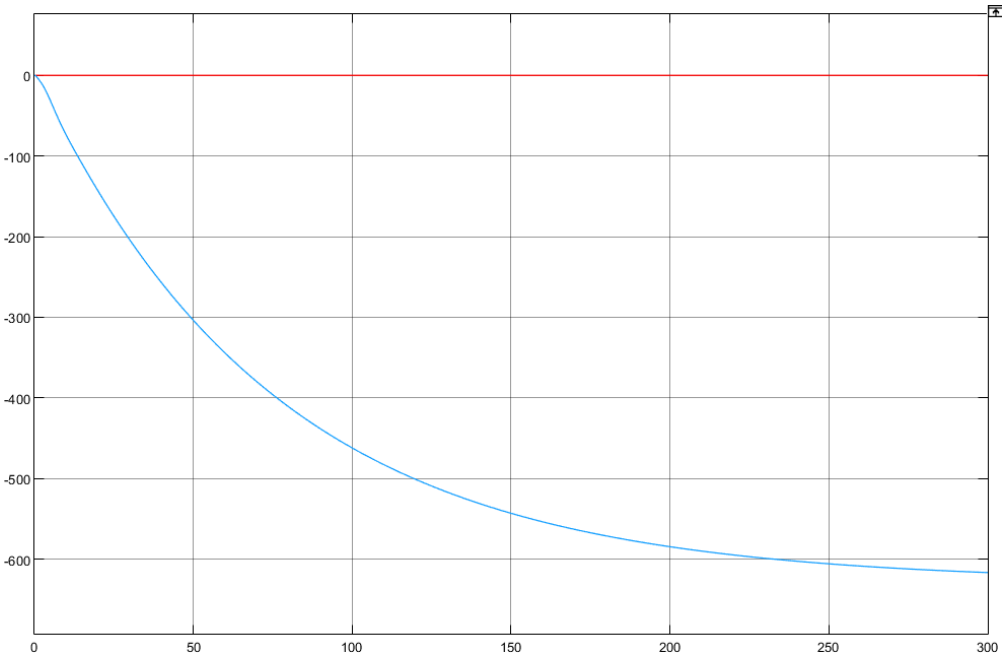


Figure 157: Velocity vs. time when $\theta = 15°$

6.7.2 Altitude Controller

The purpose of this section is to design the altitude control loop using pitch as the intermediate variable. The control process follows this sequence:

- **Input Command:** Commanded altitude (h_{com}) is compared to the actual altitude (h).
- **Error Correction:** A proportional or proportional-integral (P or PI) controller processes the altitude error to generate the commanded pitch angle (θ_{com}).
- **Pitch Loop Dynamics:** The commanded pitch angle (θ_{com}) is passed through the pitch control loop, resulting in the actual pitch angle (θ).
- **Altitude Dynamics:** The actual pitch angle (θ) affects the climb angle (γ) and subsequently the altitude (h).

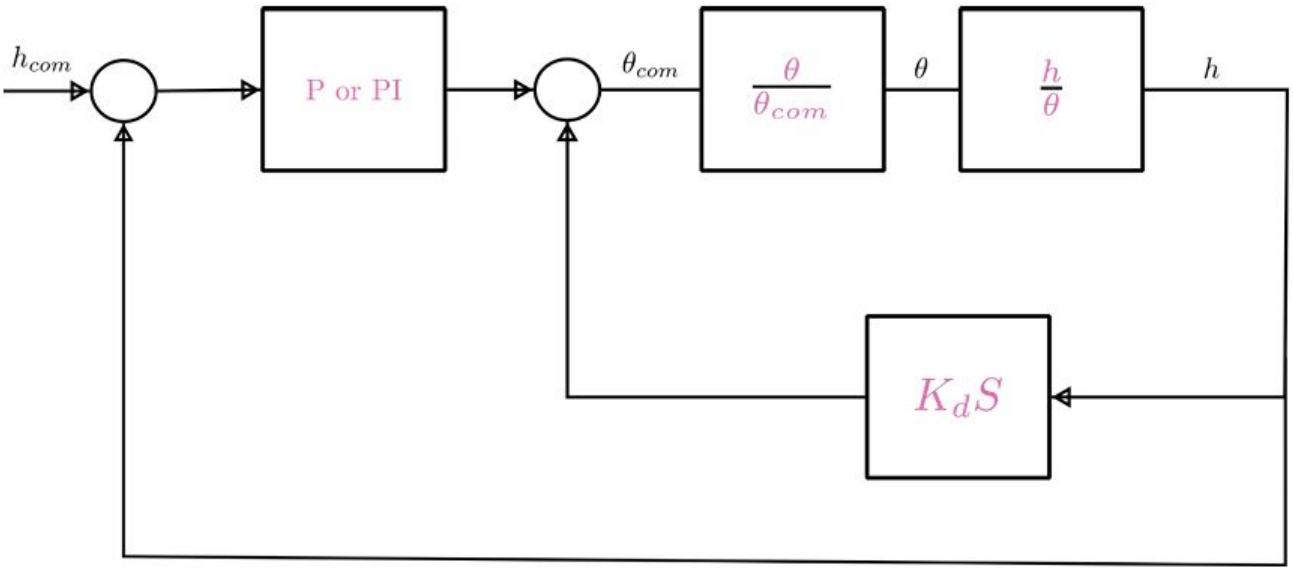


Figure 158: Control loop block diagram for the altitude controller.

The block diagram of the altitude control system is shown above. The pitch control loop is embedded as the inner loop, with its closed-loop transfer function $\frac{\theta}{\theta_{com}}$. The system also incorporates altitude dynamics represented by $\frac{h}{\theta}$.

—

1- Mathematical Formulation:
We derive the relationship between altitude (h) and pitch angle (θ) as follows:

1. **Vertical Acceleration:**
   The vertical acceleration a_z is defined as:

$$a_z = \dot{w} + pv - qu$$

   After linearization, assuming small perturbations about the equilibrium state:

$$a_z = \dot{w} - U_0 q$$

   Where:
2. U_0: Forward velocity at equilibrium.
3. q: Pitch rate.
4. $\dot{w}$: Derivative of vertical velocity.
5. **Altitude Dynamics:**
   Using the relation $\ddot{h} = -a_z$, the Laplace transform yields:

$$h(s) = -\frac{a_z}{s^2}$$

Substituting $a_z = \dot{w} - U_0 q$:

$$h(s) = -\frac{1}{s^2}\left(\dot{w} - U_0 q\right)$$

6. **Transfer Function:**
   Relating h to θ, we obtain:

$$\frac{h}{\theta} = -\frac{1}{s^2}\left(\frac{w}{\delta_e} - U_0 \frac{q}{\theta}\right)$$

Simplifying further using approximations for $\frac{q}{\delta_e}$ and $\frac{\theta}{\delta_e}$:

$$\frac{h}{\theta} = -\frac{U_0}{s}\left(\frac{\alpha/\delta_e}{\theta/\delta_e} - 1\right)$$

2- Open-Loop Analysis:
The open-loop transfer function from the commanded altitude (h_{com}) to the actual altitude (h) is given by:

$$\frac{h}{h_{com}} = \frac{-328.4s^4 - 837.1s^3 + 7026s^2 + 3302s + 19.1}{s^7 + 12.58s^6 + 27.76s^5 + 38.62s^4 + 23.77s^3 + 6.862s^2 + 0.0878s}$$

This transfer function represents the dynamics of the altitude control system and includes the inner pitch control loop.

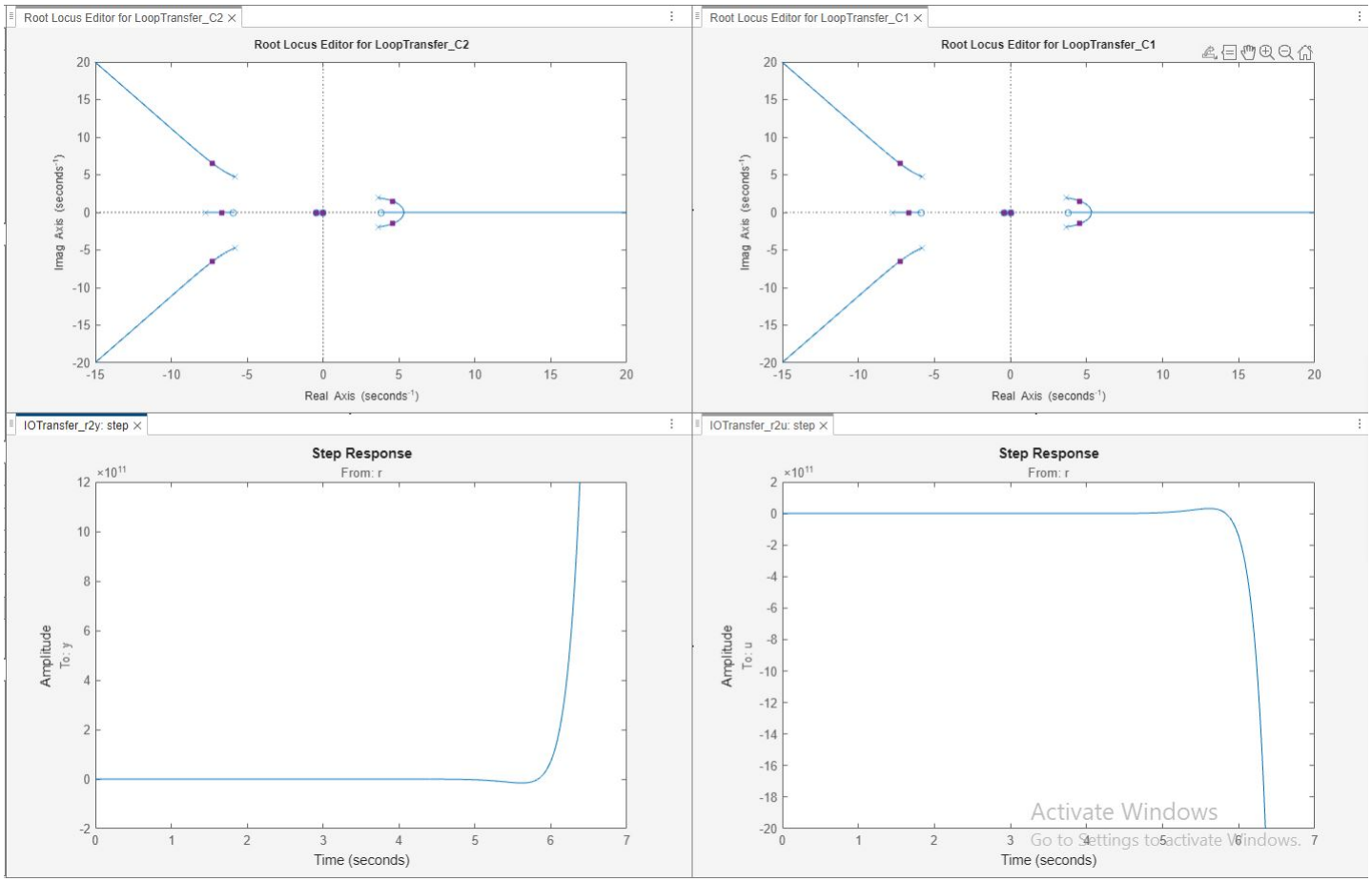


Figure 159: Open-loop step response and root locus analysis.

The figure above shows the open-loop response, including the step response and root locus. It indicates that the system's response does not meet the desired specifications, requiring closed-loop control for improvement.

3- Closed-Loop Analysis:
The closed-loop altitude control system incorporates a PI controller to process the altitude error. The transfer function of the PI controller is:

$$C(s) = \frac{0.00048163(s + 0.004667)}{s}$$

When the PI controller is applied, the resulting closed-loop transfer function becomes:

$$\frac{h}{h_{com}} = \frac{-0.1582s^5 - 0.4039s^4 + 3.382s^3 + 1.606s^2 + 0.01662s + 4.293 \times 10^{-5}}{s^8 + 12.58s^7 + 27.76s^6 + 38.46s^5 + 23.37s^4 + 10.24s^3 + 1.694s^2 + 0.01662s + 4.293 \times 10^{-5}}$$

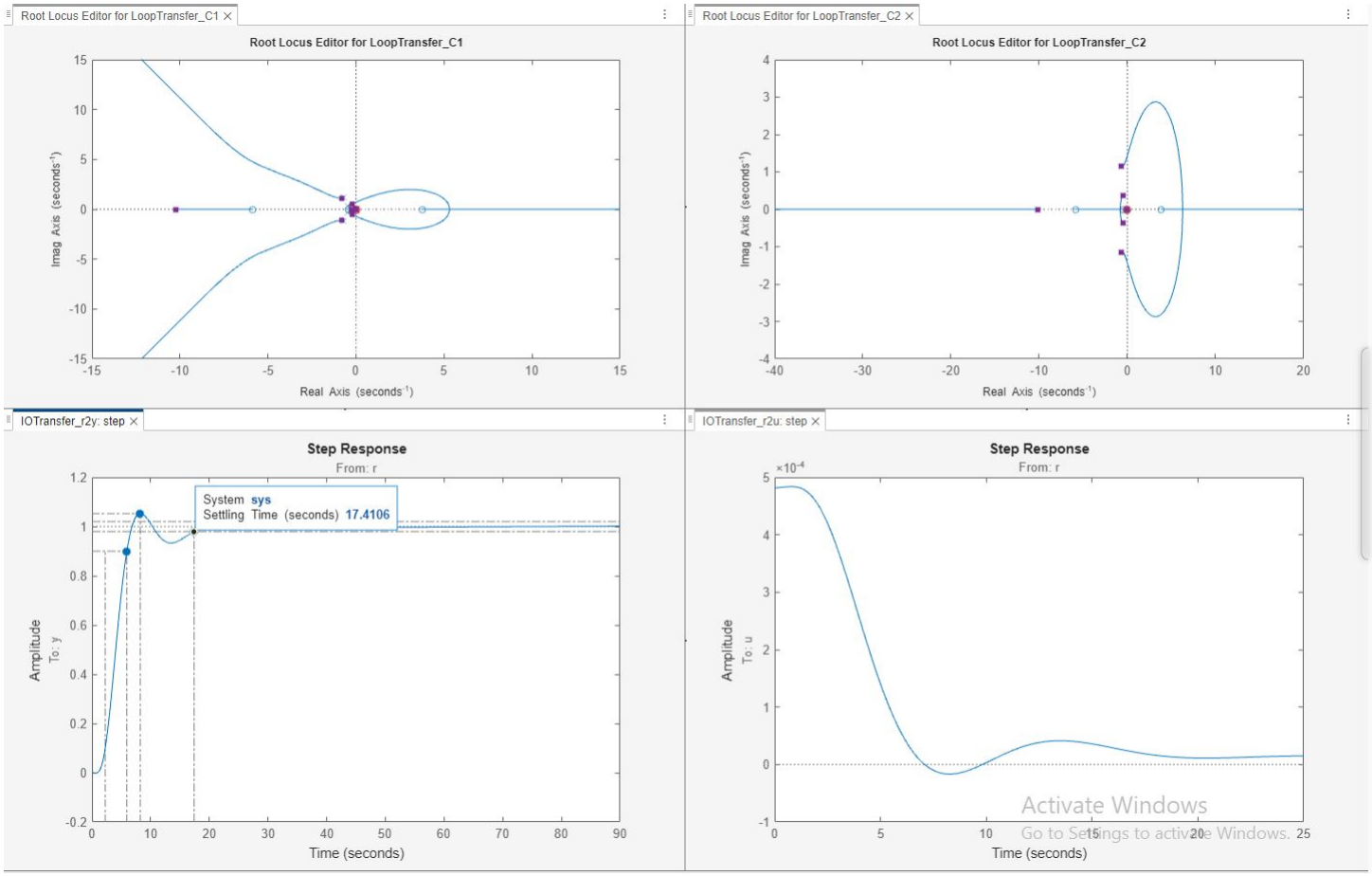


Figure 160: Closed-loop step response and root locus analysis.

The figure above shows the closed-loop step response and root locus. With the PI controller, the system achieves improved settling time and reduced overshoot, meeting the design requirements for altitude control.
The final settling time was 17.4106 seconds for each 1 step input for altitude (in feet), which corresponds approximately to 0.028 degrees of elevator deflection.

4- Simulink Model:
To validate the altitude control system, a Simulink model was created. The model includes the PI controller, pitch control dynamics, and altitude transfer function.
The system was tested with a 200-foot step input. The altitude reached the target with a settling time of approximately 17.41 seconds, matching MATLAB results. This corresponds to an elevator deflection of about 0.028 degrees.
The figures below show the simulation results: simulation:

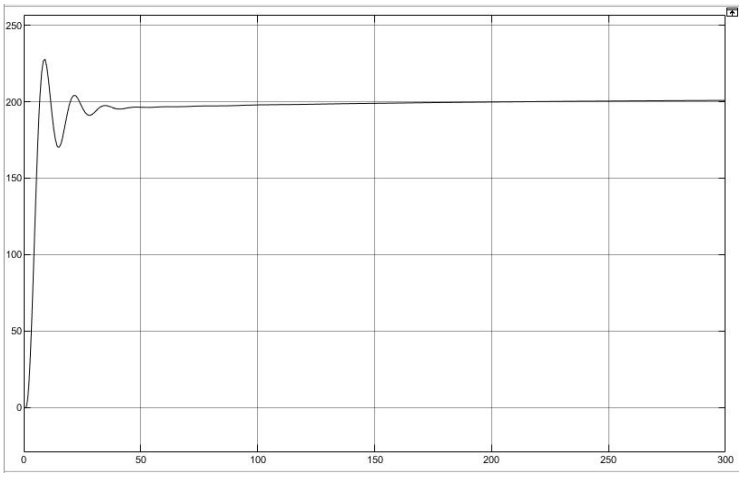


Figure 161: Simulink model output showing the altitude response to a step input of 200 feet.

The first figure above validates that the altitude successfully reaches the commanded value of 200 feet within the expected settling time.

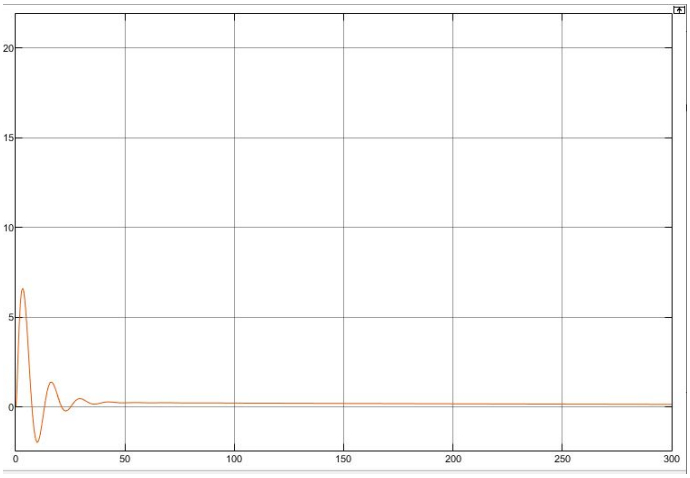


Figure 162: Simulink model output showing the pitch angle response (θ) to a step input of 200 feet.

The second figure shows that the pitch angle (θ) changes to nearly 5.6 degrees for the step input of 200 feet. This result is consistent with the altitude input and confirms the control system's stability and responsiveness.
These results validate the design objectives of the altitude control system, demonstrating that it performs as expected under simulation conditions.

6.8 Velocity Control

In this section, we discuss the velocity control mechanism for an aircraft achieved through throttle adjustments. The throttle serves as the primary actuator to regulate engine power,

which directly influences the velocity of the aircraft. The command velocity is computed using the following equation:

$$\mathrm{OL}_{u_{ucom}} = -\mathrm{servo} \cdot \mathrm{engine\_timelag} \cdot u_{dT} \tag{1}$$

Now we analyze the performance of the untuned velocity control system using the open-loop transfer function in the MATLAB `sisotool` environment. The plots in Figure 163 illustrate the root locus and step response of the system before tuning, providing insights into its behavior. The root locus plots (LoopTransfer_C1 and LoopTransfer_C2) in the upper portion of Figure 163 reveal several important characteristics. The poles of the system move along asymptotes towards infinity, indicating potential instability. In LoopTransfer_C1, some poles cross into the right-hand side of the complex plane, signifying an unstable open-loop system. The insufficient damping of poles in the left-hand plane suggests that the system would exhibit oscillatory behavior if closed-loop control were applied. Additionally, the lack of pole-zero proximity in the stable region indicates poor transient response characteristics. The step response plots (IOTransfer_r2u and IOTransfer_r2y) in the lower portion of Figure 163 also highlight significant issues in the system's dynamics. The step response of `IOTransfer_r2u` shows a very large transient amplitude, indicating excessive overshoot and instability. Similarly, the step response of `IOTransfer_r2y` exhibits unbounded behavior with oscillations and significant deviation from the desired steady-state value. These responses demonstrate the system's inability to control its dynamics effectively in the untuned state. From these observations, it is evident that the untuned system is unstable and exhibits poor transient response. To achieve stability and proper dynamic behavior, it is essential to tune the controller to place the poles of the closed-loop system in the left-hand plane with adequate damping.

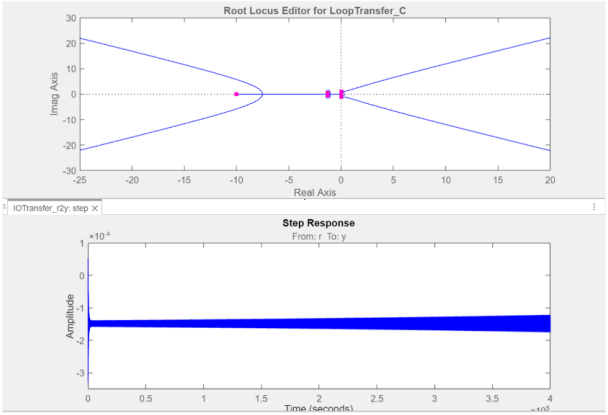


Figure 163: Root locus and step response plots for the untuned velocity control system.

The performance of the velocity control system was improved by tuning the open-loop transfer function using MATLAB `sisotool`. The tuning involved modifications to the compensators C_1 (PI Controller) and C_2 (Lead Compensator) as follows:

- For C_1 (PI Controller), the transfer function is given by:

$$C_1(s) = \frac{-1606.2\,(s + 0.2334)}{s} \tag{2}$$

This includes a real zero at -0.2334 with a damping ratio of 1, a frequency of 0.2334, and an integrator at $s = 0$.

- For C_2 (Lead Compensator), the transfer function is given by:

$$C_2(s) = \frac{-1.6408 \times 10^5\,(s - 0.0007846)}{(s + 1.307)} \tag{3}$$

This includes a real pole at -1.307 with a damping ratio of 1 and a frequency of 1.307, and a real zero at 0.0007846 with a damping ratio of -1 and a frequency of 0.0007846.

Figure 164 shows the root locus and step response plots of the tuned system.
The root locus plots (LoopTransfer_C1 and LoopTransfer_C2) in the upper portion of Figure 164 indicate significant improvements in the system's stability and dynamics. In LoopTransfer_C1, the addition of the real zero and integrator has ensured that the poles remain in the left-hand plane, resulting in a stable system. Similarly, the adjustments in LoopTransfer_C2 have ensured that the poles and zeros are appropriately distributed, contributing to a stable response with adequate damping. The overall root locus behavior shows a well-tuned system with reduced potential for oscillatory or unstable behavior.
The step response plots (IOTransfer_r2u and IOTransfer_r2y) in the lower portion of Figure 164 further confirm the improved performance. The step response of `IOTransfer_r2y` shows a smooth transient response with minimal overshoot and a settling time of approximately 52.1 seconds, indicating improved transient stability and reduced oscillations. On the other hand, the step response of `IOTransfer_r2u` demonstrates the system's ability to handle control input dynamics effectively, with a stable response achieved over time despite initial transient variations.

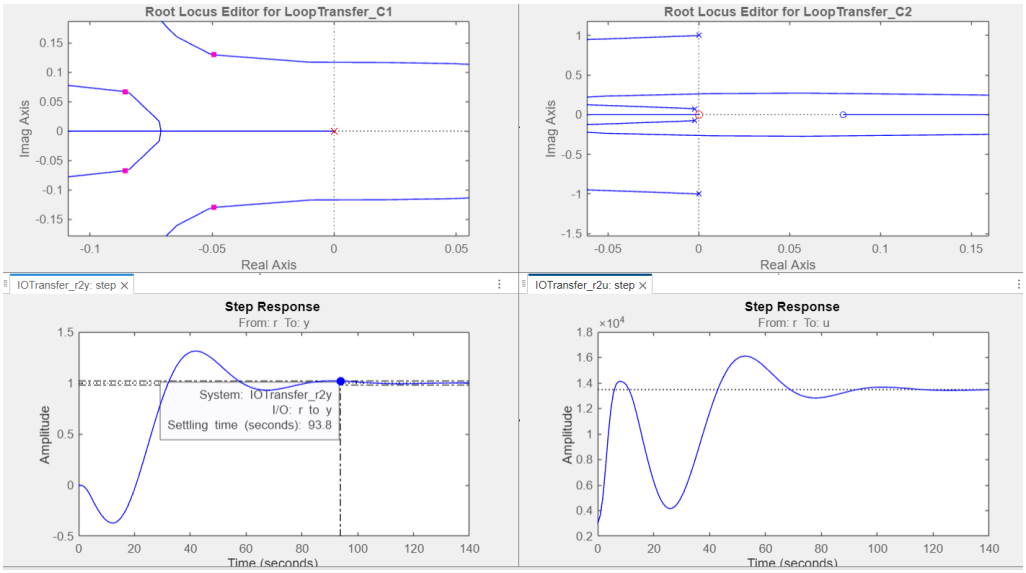


Figure 164: Root locus and step response plots for the untuned velocity control system.

To evaluate the performance of the control system, the tuned transfer functions were implemented in a Simulink model to design the control loop. The input command for the velocity (u) was set to 10, and the results are illustrated in the plots below.
The first plot (Figure 165) shows the red curve, which represents the desired velocity (u), and the brown curve, which represents the system's actual performance in reaching the desired velocity. The control system effectively tracks the desired velocity, demonstrating a smooth transient response. There is an initial overshoot, which peaks at approximately 15 before gradually settling around the desired velocity of 10. This indicates that the system achieves satisfactory stability and demonstrates an acceptable transient performance, with minimal oscillations in the settling phase. Such behavior is indicative of a well-tuned controller, capable of handling velocity commands effectively.
The second plot (Figure 166) illustrates the altitude gained by the aircraft during the simulation. The altitude increases steadily over time, following a near-linear trajectory. This response suggests that the system maintains consistent climb performance while executing the control input for velocity. The absence of fluctuations in altitude highlights the robustness of the control system in managing vertical dynamics in conjunction with velocity control.
In conclusion, the Simulink simulation confirms that the tuned control system performs effectively, achieving the desired velocity while ensuring smooth altitude transitions. These results validate the design of the control loop using the tuned transfer functions.
Figure 167 shows the throttle input used by the controller to achieve the desired velocity

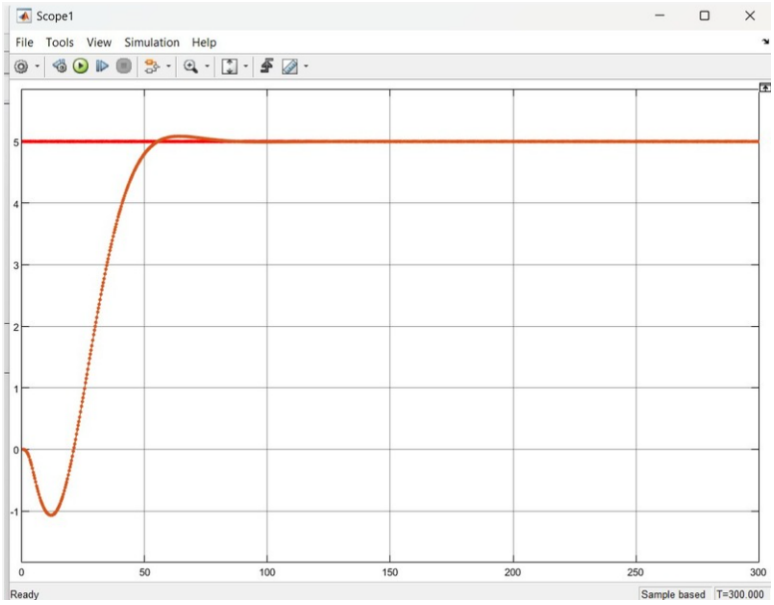


Figure 165: Velocity response: The red curve represents the desired velocity ($u = 5$), and the brown curve shows the system's performance in tracking the desired velocity.

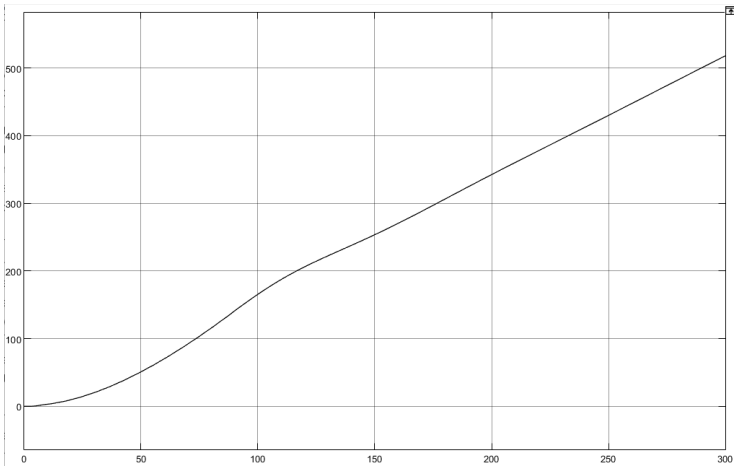


Figure 166: Altitude response: The altitude gained by the aircraft over time, showing steady and consistent climb performance.

($u = 10$). Initially, the throttle exhibits high-frequency oscillations, reflecting the controller's efforts to correct large velocity discrepancies during the transient phase. As the system stabilizes, the throttle reduces sharply to counteract overshoot and then gradually adjusts to maintain the desired velocity. By around 200 seconds, the throttle settles into a steady-state value, demonstrating effective control.

While the initial oscillations indicate room for further tuning to smooth the transient response, the system successfully achieves stability and consistent throttle control in the steady state.

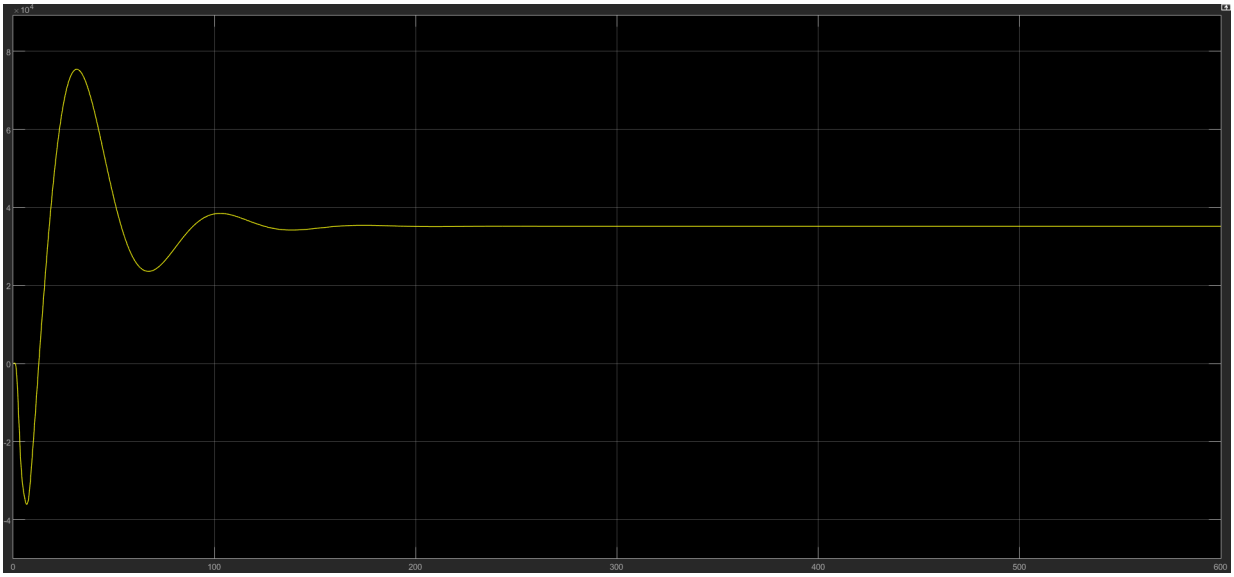


Figure 167: Throttle response: The throttle input used by the controller to achieve the desired velocity ($u = 5$).

6.9 Lateral Autopilot

The lateral autopilot plays a vital role in stabilizing and controlling an aircraft's motion in the lateral-directional plane. It ensures smooth, coordinated turns by managing the rudder and aileron. When combined with the longitudinal autopilot, the system achieves a coordinated level turn, maintaining both heading and altitude effectively.
Coordinated turns are necessary to ensure passenger comfort and minimize energy losses. Without coordination, lateral forces or sideslip may occur, resulting in discomfort and increased drag. In manually controlled flights, pilots adjust the rudder, aileron, and elevator to achieve coordination. In contrast, the lateral autopilot automates these tasks, improving efficiency and precision.
This section outlines the design of a lateral autopilot system, incorporating yaw damper and roll controllers, as well as an outer-loop heading control.

6.9.1 Yaw Damper Control

The role of the yaw damper is to enhance damping in the Dutch roll mode, a lightly damped oscillatory motion, to stabilize yaw dynamics.
To perform a coordinated turn, both the aileron and rudder are used to produce the required bank angle ϕ and yawing acceleration r to achieve no slip or zero lateral acceleration while turning, compensating for adverse yaw. The yaw damper autopilot design is shown in the block diagram below.

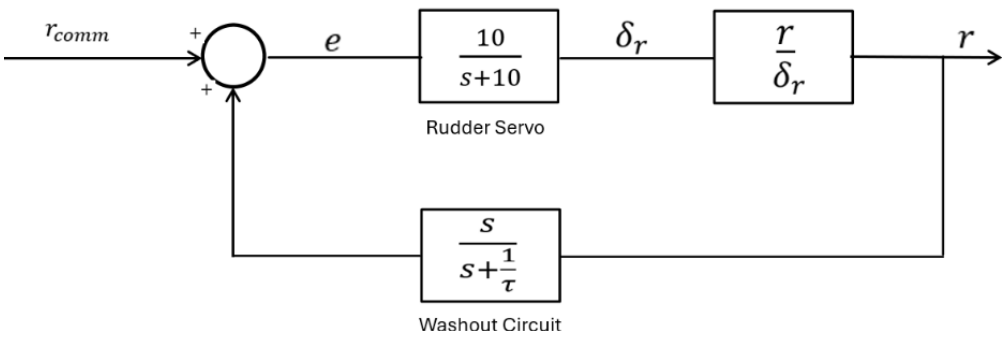


Figure 168: Yaw Damper controller

This block diagram includes the transfer function of yaw rate with respect to rudder deflection, $\frac{r}{\delta_r}$, in addition to the rudder servo dynamics transfer function. To achieve a turn with a yaw acceleration of non-zero steady-state value, a washout circuit (high-pass filter) is required, as illustrated in the block diagram.
The yaw rate transfer function is given as:

$$\frac{r}{\delta_r} = \frac{-0.6149s^3 - 0.7762s^2 - 0.1704s - 0.1123}{s^5 + 1.509s^4 + 1.479s^3 + 1.401s^2 + 0.02655s}$$

The rudder servo actuator dynamics transfer function is represented by:

$$\frac{10}{s + 10}$$

1 - Design Requirements:
According to Cooper-Harper flying qualities, for Category B, Level 1, we chose a minimum damping ratio $\zeta_{\min} = 0.08$.
2 - Closed-Loop Analysis:
Using the SISO tool in MATLAB for the high-pass filter (washout circuit) design, we first inserted the minimum design requirements on the open-loop root locus with a damping ratio of $\zeta_d = 0.08$. After trial and error, we determined the final pole position, as shown below:

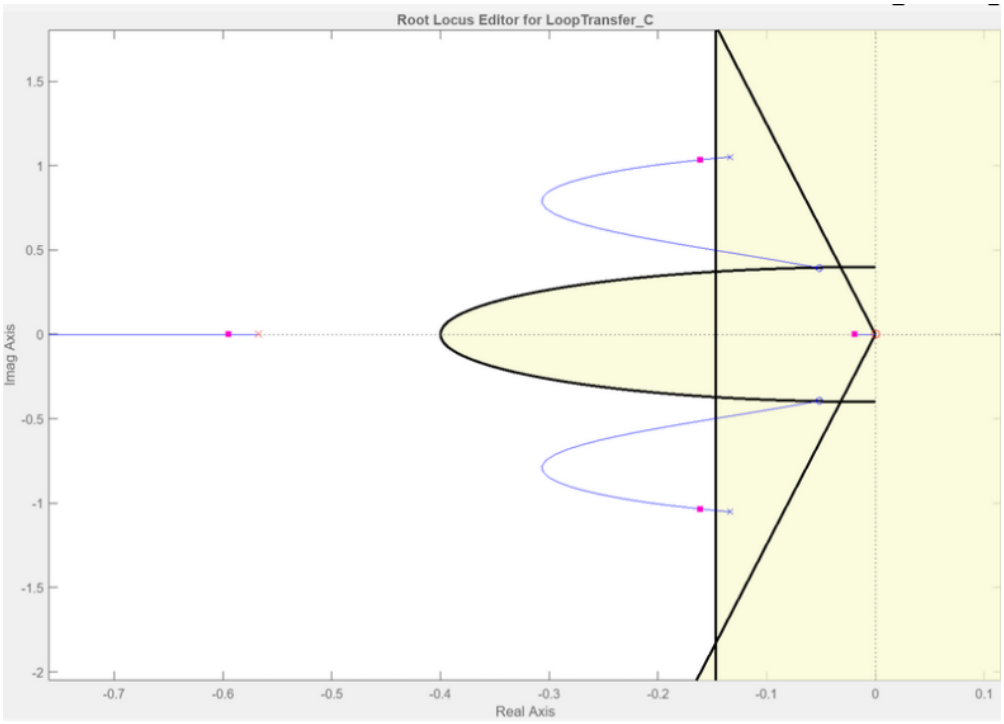


Figure 169: Root locus of closed-loop transfer function

After tuning and designing the high-pass filter, the impulse response for the output and δ_r control action are as follows:

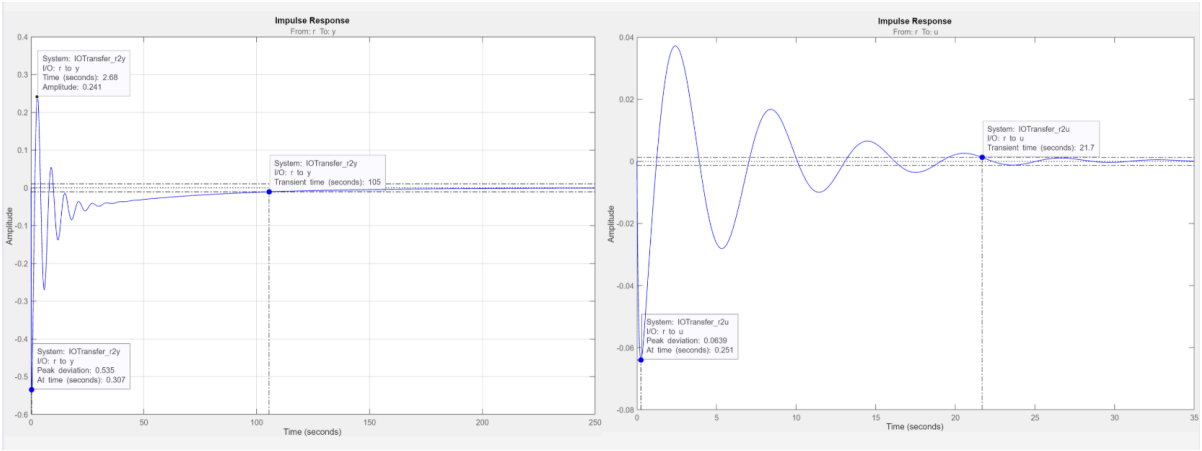


Figure 170: Yaw Damper Closed-Loop Impulse Response

The washout circuit results in a settling time of $T_s = 0.307$ seconds and a control action peak response of 0.0639 radians.
The transfer function of the washout circuit is:

$$\frac{0.13388\,s}{s + 0.5674}$$

3 - Simulink Model:
After implementing the Multi-Input Multi-Output (MIMO) system in Simulink, the following results were obtained when giving a command $r_{\text{comm}} = 0$. The control behavior, angles, and rate responses are shown in the figures below:

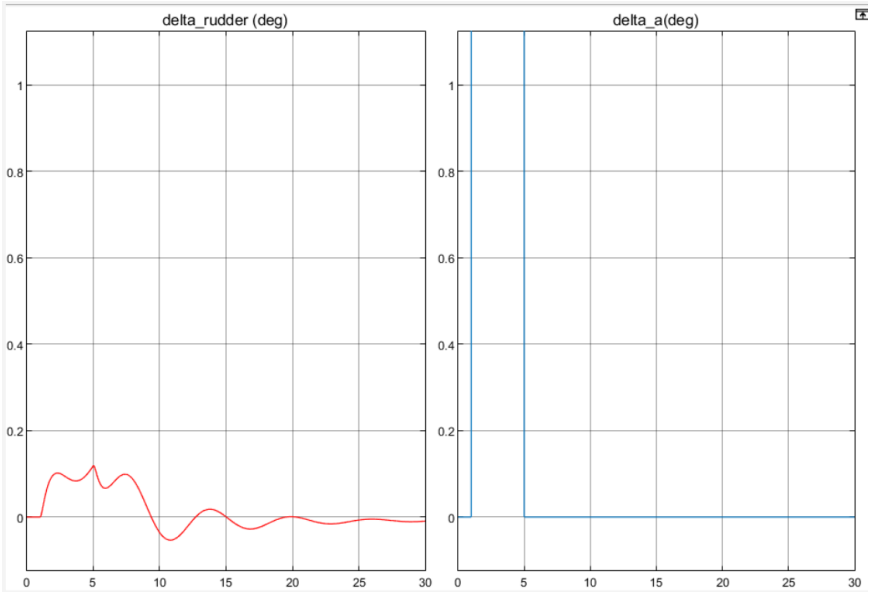


Figure 171: Control Surfaces Behavior

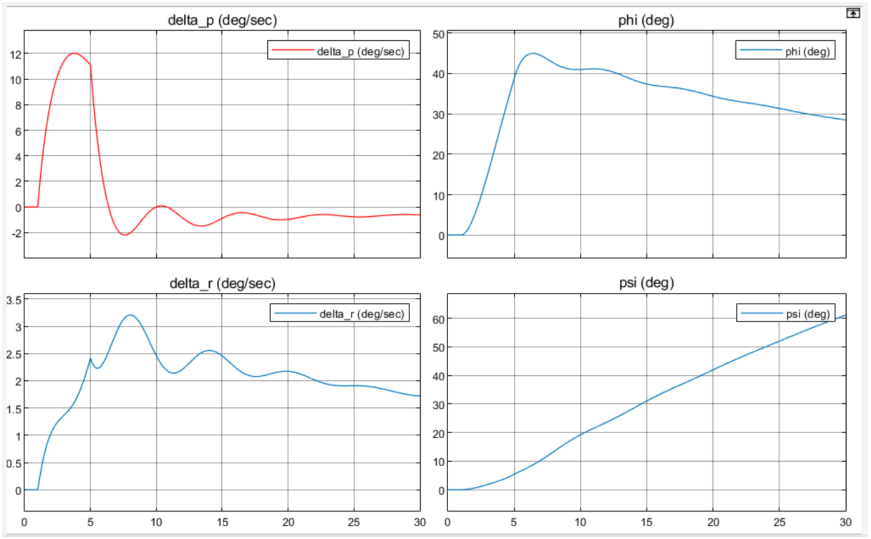


Figure 172: Control Surfaces Responses

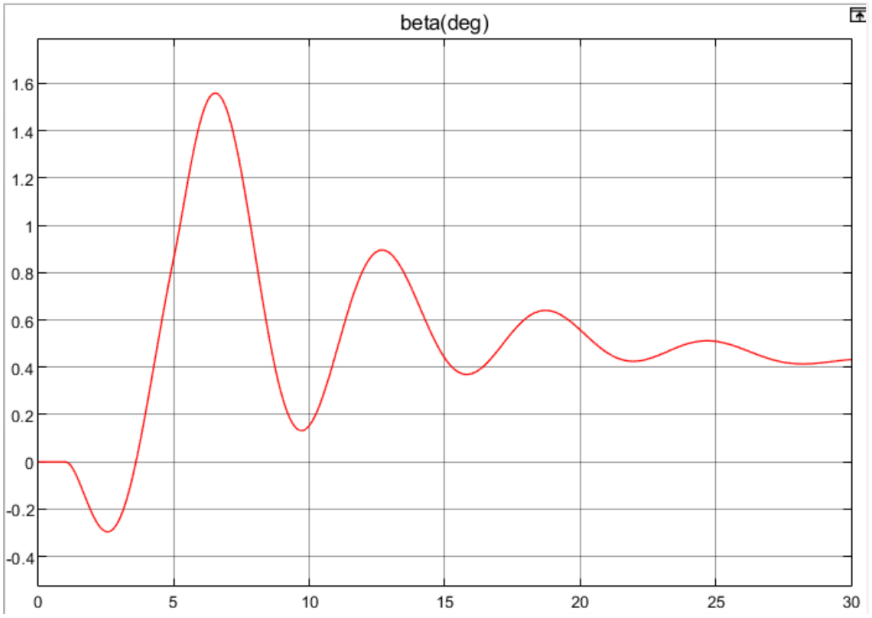


Figure 173: Angles and Rates Response

Comments over response plots

The response plots demonstrate that the lateral autopilot system, including the yaw damper, roll controller, and heading controller, functions effectively:

- The control surfaces exhibit stable behavior with well-damped oscillations.
- Bank angle and heading changes are achieved smoothly with minimal overshoot or instability.
- The sideslip angle (β) is well-controlled, ensuring coordinated turns.

These results validate the controllers' design and confirm their ability to meet performance requirements.

6.9.2 Roll control

To control the other state, we need to design a roll controller, but since the open loop tf was already stable, there was no control needed, as shown in the following block diagram.

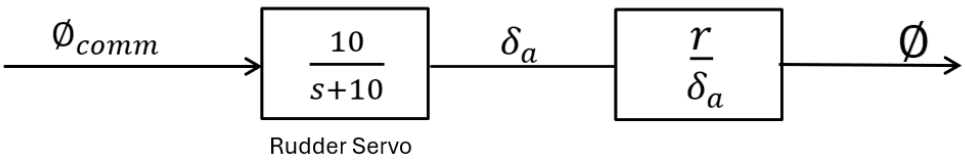


Figure 174: Roll Damper controller

As shown in the figure. It includes the transfer function of the roll angle w.r.t. aileron a/(δ_a) but for the coordinated plane. We add the servo dynamics transfer function to express ailerons servos.

1- Open Loop analysis:

As seen from the root locus after applying the boundaries of $\zeta_{min} = 0.08$. we can clearly see in the figures below that the poles are achieving the design requirements.

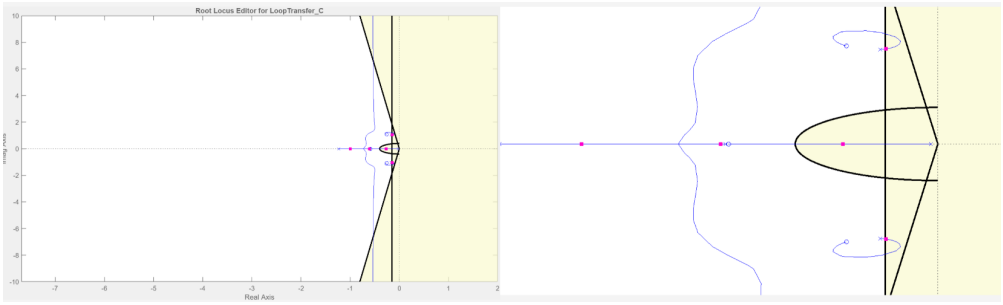


Figure 175: Root locus of open loop TF

Also, we can notice in the step response shown below that the Transient time is 16.3 seconds , and control action peak response of 0.173 rad.

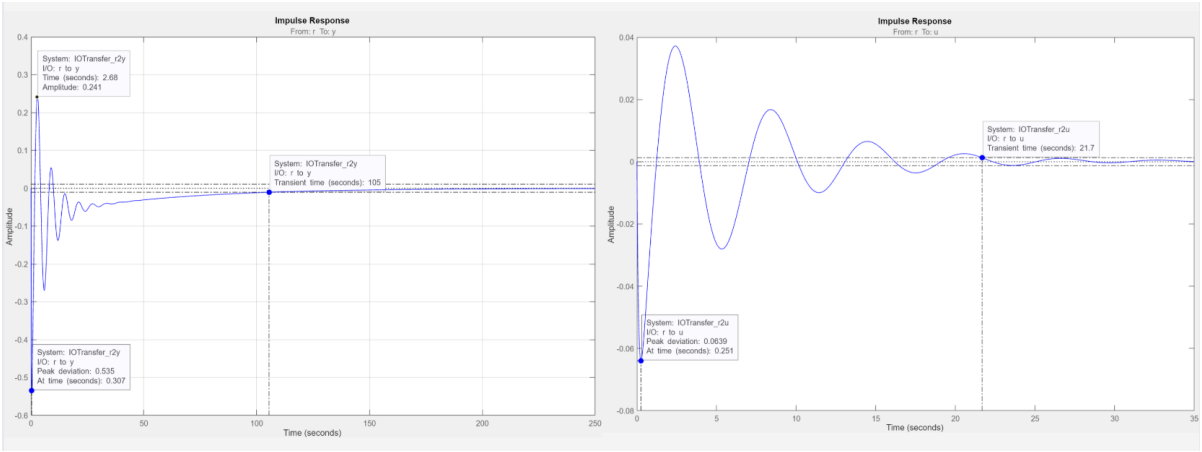


Figure 176: Impulse response of open loop TF $\phi/(\delta_a)$.

6.9.3 Heading controller design

To achieve coordination where the total acceleration is equal to zero, we will have to obtain a certain bank angle ϕ which corresponds for a certain speed U_0 and a certain turning rate ϵ_0, and this bank angle can be obtained from the free body diagram of the airplane during turn. As we have already obtained the yaw damper, we want to control the heading ϵ now.

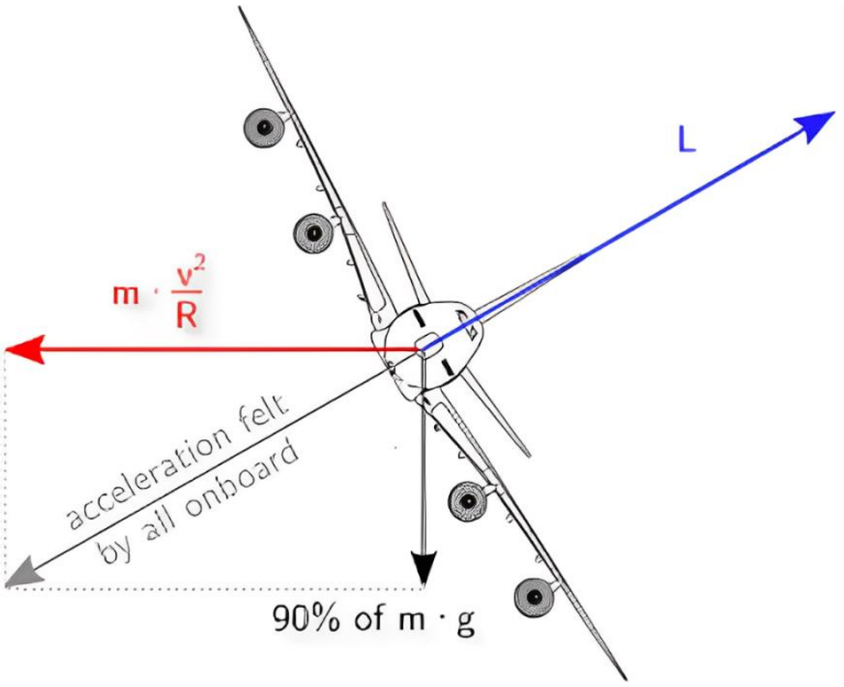


Figure 177: free body diagram of the airplane during turn.

Knowing that the tangential velocity is U_0 and that R is the radius of the turn:

$$U_0 = R \cdot \dot{\psi}$$

To control the heading (ψ), the aircraft must bank to achieve the desired directional change. This process results in a coordinated turn with an angular rate of $\dot{\psi}$.
When the aircraft is banked at an angle ϕ, the forces along the body y-axis are balanced. The relationship is expressed as:

$$mg \sin \phi = mu_0 \dot{\psi} \cos \phi$$

Since the bank angle ϕ is typically small ($\phi \ll 1$), the approximation $\tan \phi \approx \phi$ can be used. Therefore, the bank angle is calculated as:

$$\phi = \frac{u_0 \dot{\psi}}{g}$$

For a given desired heading ψ_d, the heading ψ must follow ψ_d gradually. The dynamics are expressed as:

$$\tau_\psi \dot{\psi} + \psi = \psi_{\text{com}}$$

Rearranging gives:

$$\frac{\dot{\psi}}{\psi_{\text{com}}} = \frac{1}{\tau_\psi s + 1}, \quad \frac{\dot{\psi}}{\psi_{\text{com}}} = \frac{\frac{1}{\tau_\psi}}{s + \frac{1}{\tau_\psi}}$$

This acts as a low-pass filter that suppresses higher frequency noise. The desired bank angle is then calculated as:

$$\phi_{\text{com}} = \frac{u_0 \dot{\psi}}{g} = \frac{u_0}{\tau_\psi g}(\psi_{\text{com}} - \psi)$$

For this system, τ_ψ is chosen between 15 s and 20 s, depending on the specific aircraft and its objectives.

Now integrating the controllers to operate simultaneously on the aircraft using the Simulink model. the following limits were applied during simulation:

- The command bank angle (ϕ_{com}) was limited to $\pm 30°$.
- The rudder deflection (δ_r) was limited to $\pm 15°$.
- The aileron deflection (δ_a), which is the sum of both left and right aileron deflections, was limited to $\pm 50°$.

A step input of 360° in heading was tested, and the settling time and coordination were verified by observing the slip angle (β).

The resulting simulation outputs are as follows:

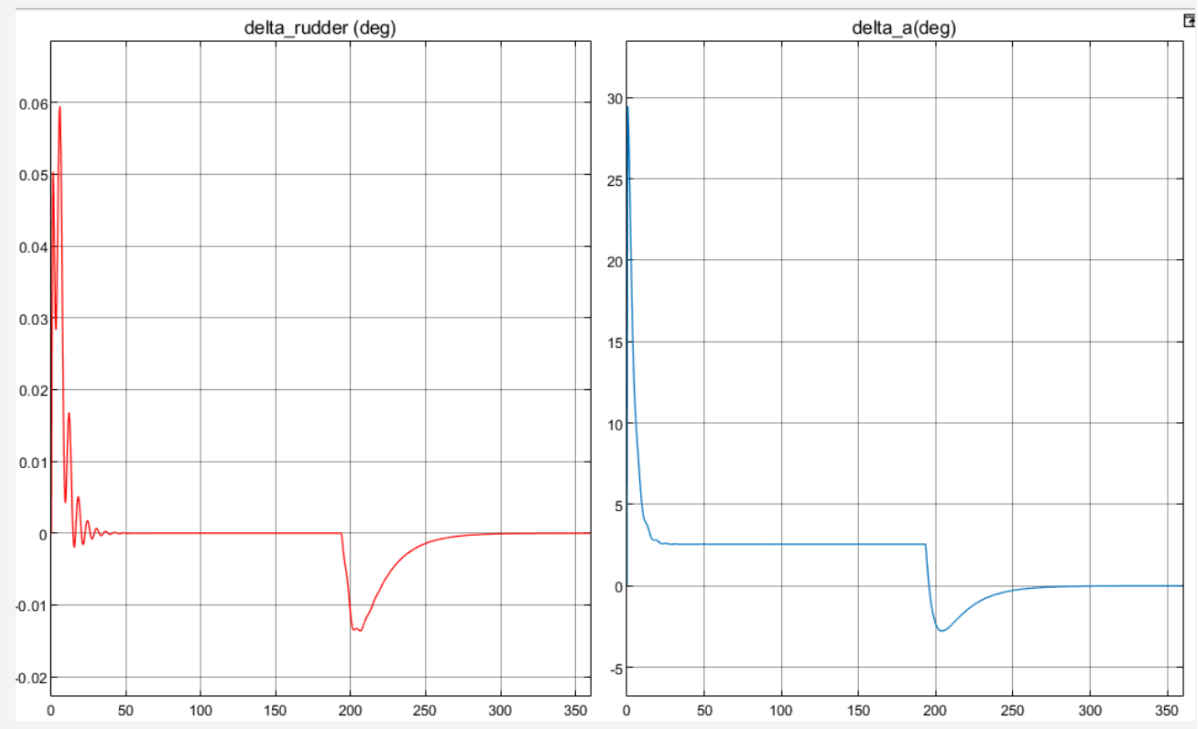
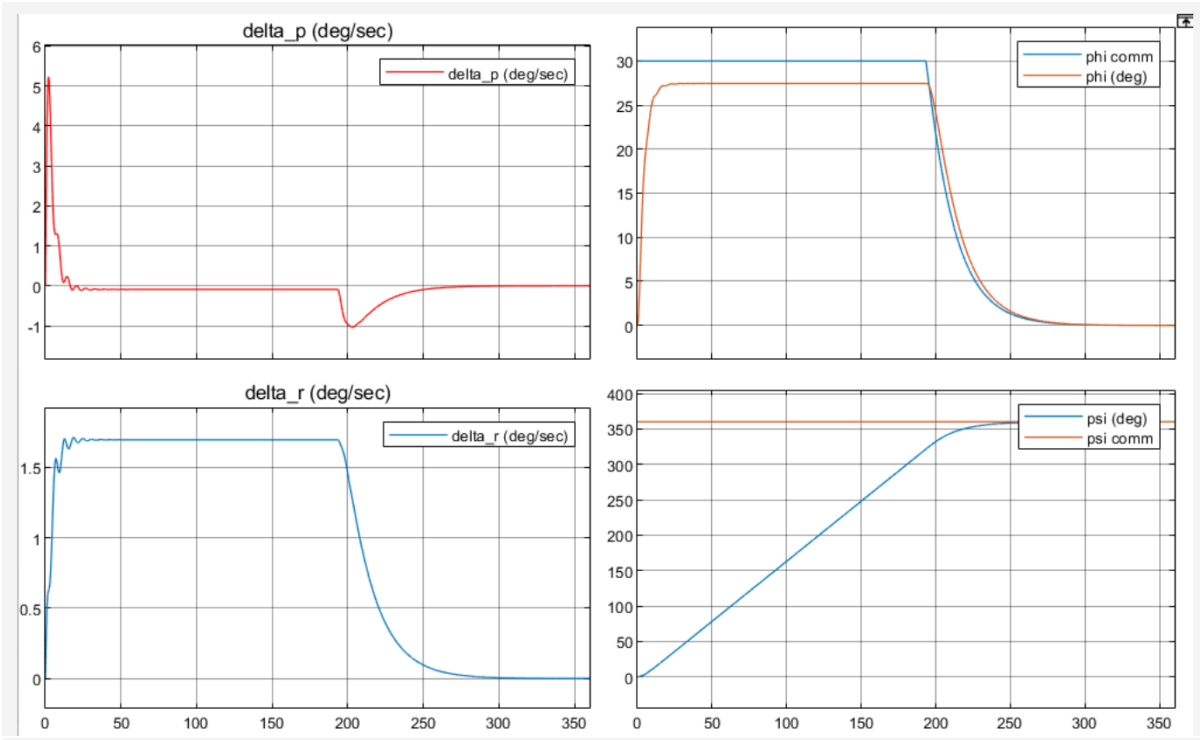


Figure 178: Control Surfaces Behavior from Lateral Autopilot



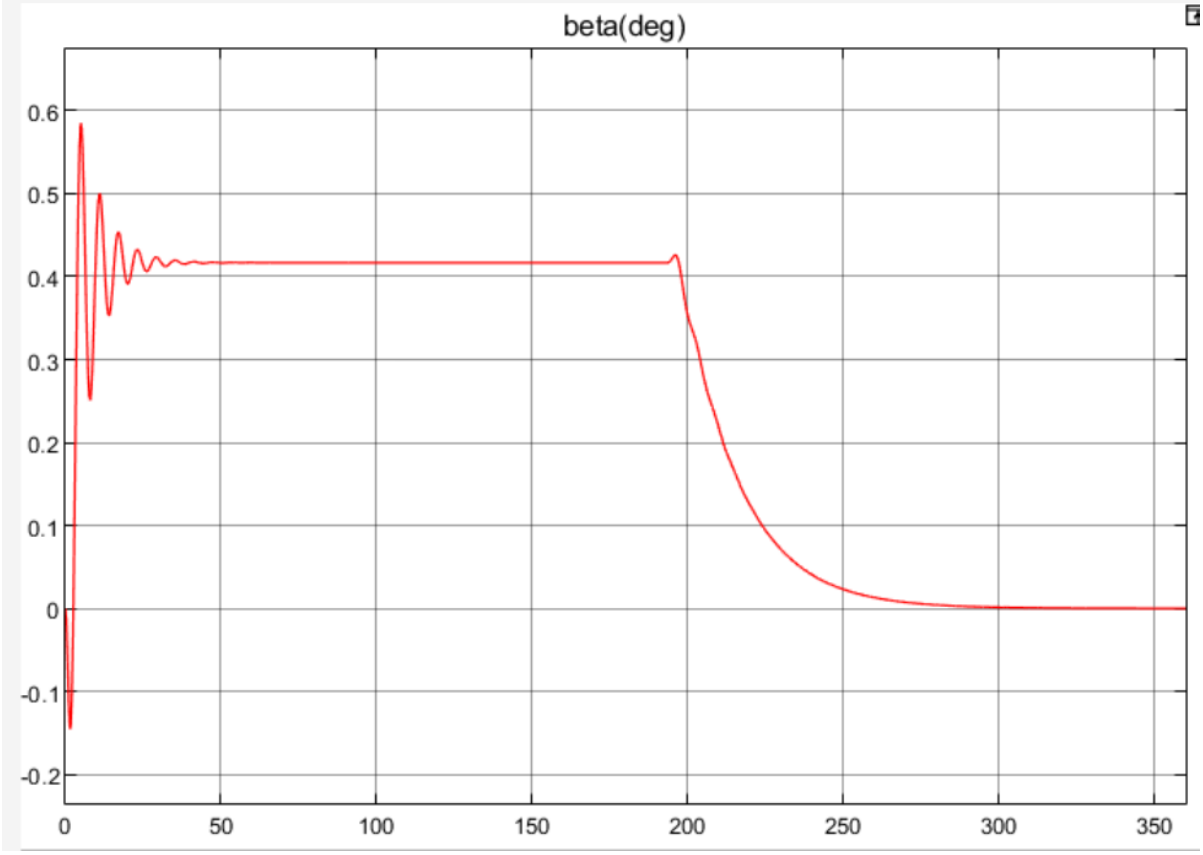


Figure 179: Lateral Autopilot's states responses.

Overall Performance

The plots validate the lateral autopilot's performance across multiple states:

· The rudder and aileron deflections effectively counteract disturbances and maintain stability.

- The roll and yaw dynamics are well-controlled, achieving rapid stabilization and smooth transitions.
- The heading and bank angles accurately follow commands, with minimal deviation.
- The sideslip angle response confirms the system's capability to maintain coordinated turns without lateral acceleration.
- The system demonstrates robust and efficient lateral control, meeting design expectations.

7 Nonlinear Simulation and Mission

In this section, we will evaluate the lateral and longitudinal autopilot controllers for the B747 airplane flight condition 3 using a realistic 6DOF nonlinear airplane dynamics model. Unlike the linearized state-space model, this nonlinear model incorporates coupling between longitudinal and lateral dynamics, offering a more accurate representation of the airplane's behavior and enabling simultaneous testing of all controllers.

The focus will be on testing the autopilot's ability to manage four critical control actions—aileron, rudder, elevator, and thrust—to maintain the required attitude and altitude during complex maneuvers, such as coordinated-level turns, climbs, and descents. The evaluation aims to validate both individual controller components and their combined performance in executing a complete autonomous mission.

This section will include:

- Development of a testing loop to integrate the controllers with the simulator, providing real-time feedback for continuous adjustment of control actions.
- Independent testing of pitch control, velocity control, altitude hold, and lateral control to verify their specific functionalities.
- Combined testing of lateral and longitudinal controllers to assess their coordinated operation during turns and altitude maintenance.
- Execution of a full autonomous mission to demonstrate the integration and reliability of the autopilot system in realistic flight scenarios.

The autonomous mission will simulate sequential phases of climbing, cruising, turning, and descending, ensuring the autopilot system's ability to handle dynamic flight conditions and transitions seamlessly.

7.1 Developing the Test Loop

A closed-loop system was developed to integrate the autopilot with the non-linear simulator:

- The autopilot generated control actions (aileron, rudder, elevator, thrust).
- These actions were fed into the non-linear simulator, which calculated the updated airplane states.
- The updated states were fed back as sensor signals to the autopilot for further action.

This feedback mechanism ensured a dynamic and realistic testing process as shown in the below flow digram.

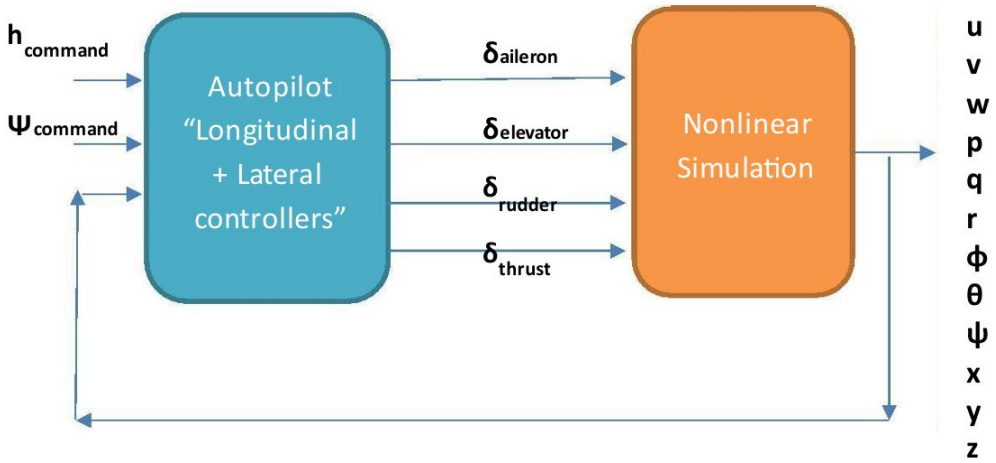


Figure 180: Flowchart of flight Controller closed loop

7.2 Simulation of Linear and Non-Linear Models

7.2.1 Pitch Controller Test

- **Commanded input:** delta pitch angle of 5°.
- **Observed States:** Pitch angle (θ), velocity (u), flight path angle (γ), altitude (H), and elevator deflection (δ_e) for linear and non-linear Model.

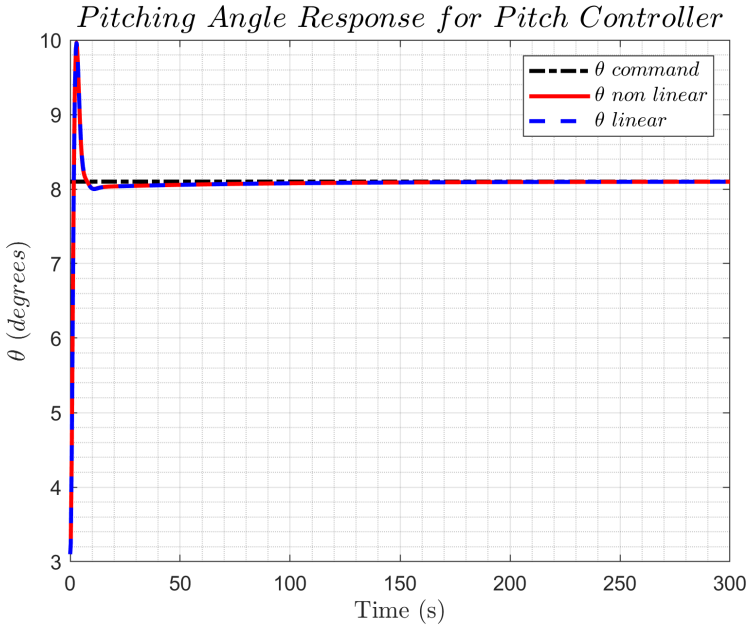


Figure 181: θ Response for Pitch Controller for linear and nonlinear models

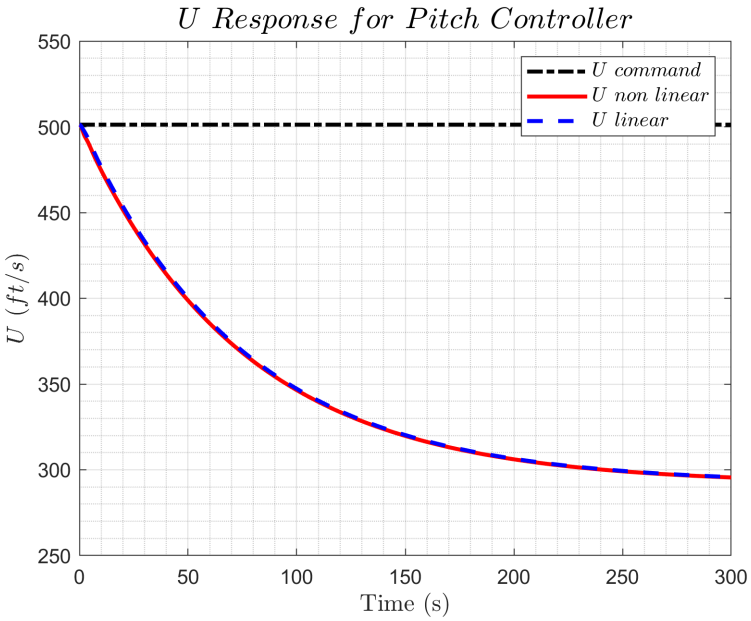


Figure 182: U Response for Pitch Controller for linear and nonlinear models

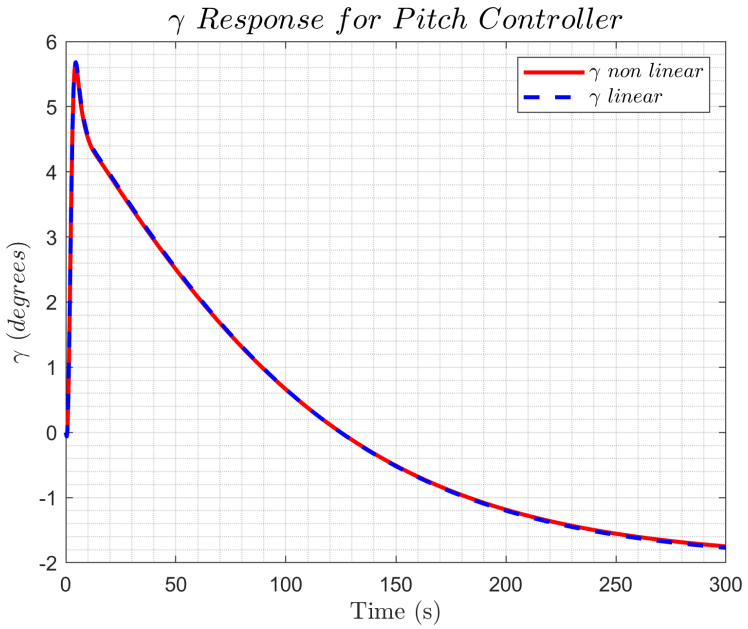


Figure 183: γ Response for Pitch Controller for linear and nonlinear models

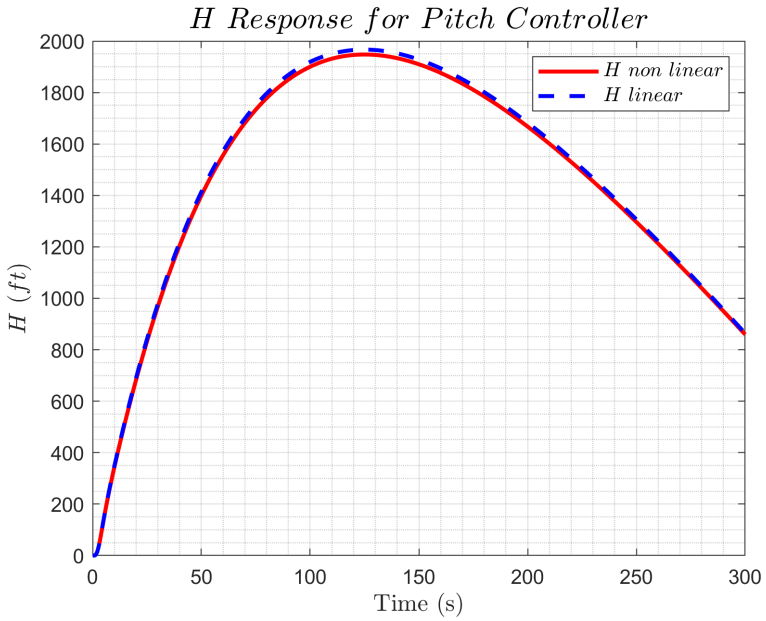


Figure 184: H Response for Pitch Controller for linear and nonlinear models

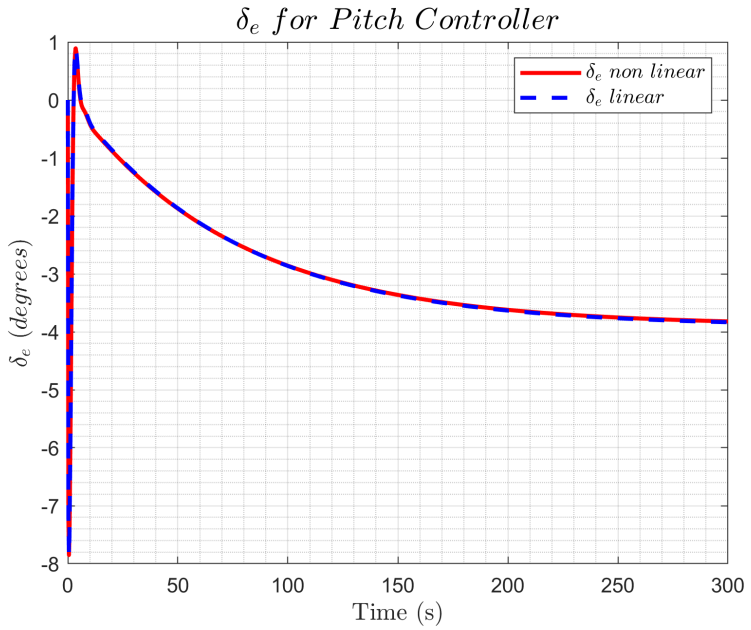


Figure 185: δ_e Response for Pitch Controller for linear and nonlinear models

- **Result Comments :**The responses of the linear and nonlinear models exhibit an almost identical match, aligning closely with the pitch controller output from the longitudinal autopilot section.

7.2.2 Pitch Controller and Velocity Controller Test

- **Commanded input:** delta velocity of 20 ft/s.
- **Observed States:** Pitch angle (θ), velocity (u), flight path angle (γ), altitude (H), elevator deflection (δ_e), and throttle input (δ_{th}) for linear and nonlinear models.

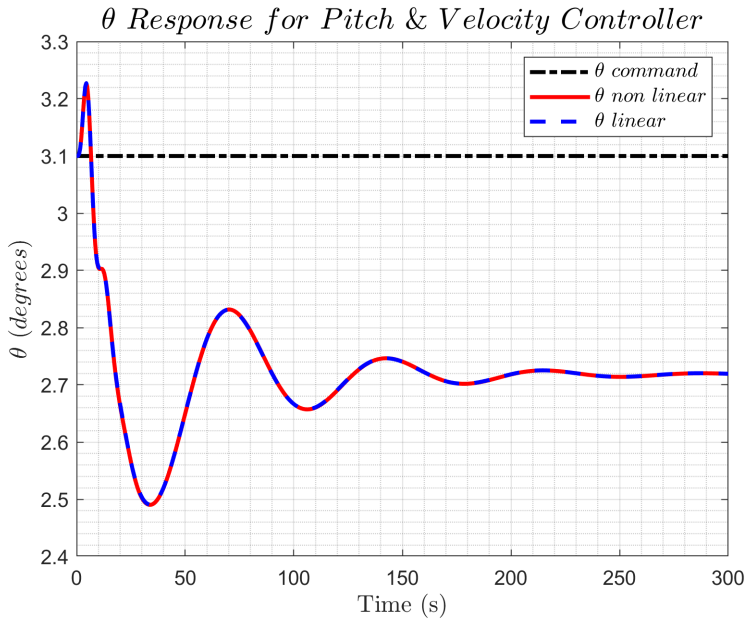


Figure 186: θ Response for Pitch and Velocity Controller for linear and nonlinear models

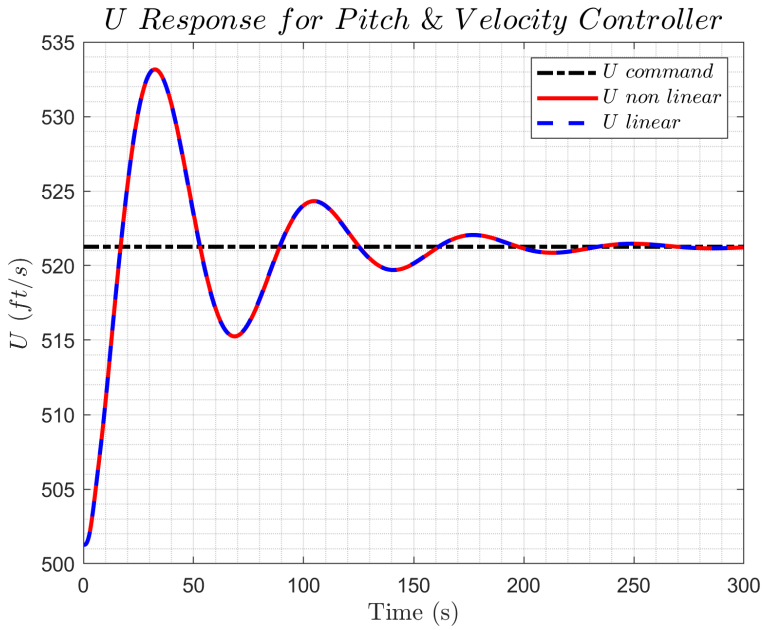


Figure 187: u Response for Pitch and Velocity Controller for linear and nonlinear models

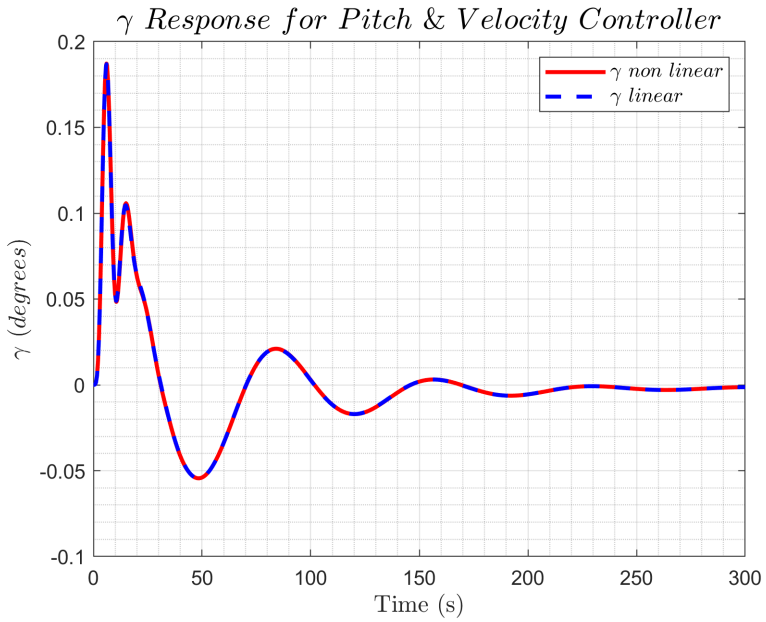


Figure 188: γ Response for Pitch and Velocity Controller for linear and nonlinear models

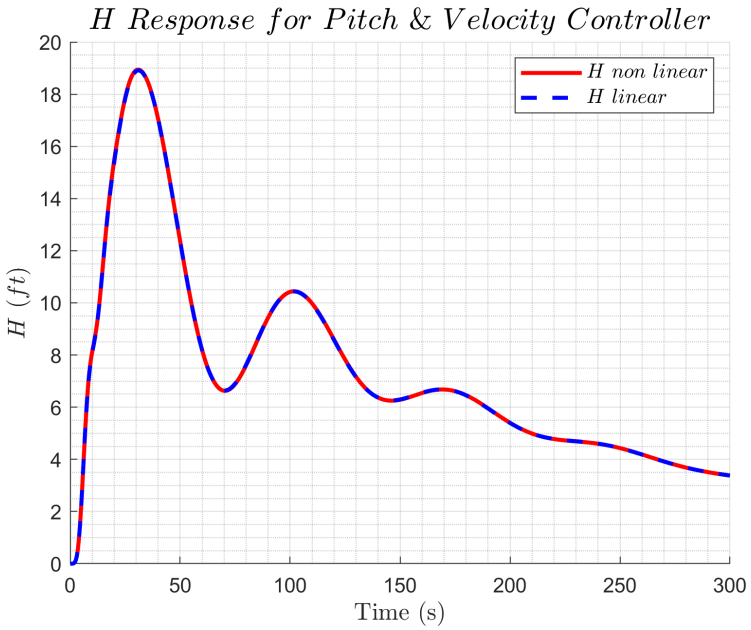


Figure 189: H Response for Pitch and Velocity Controller for linear and nonlinear models

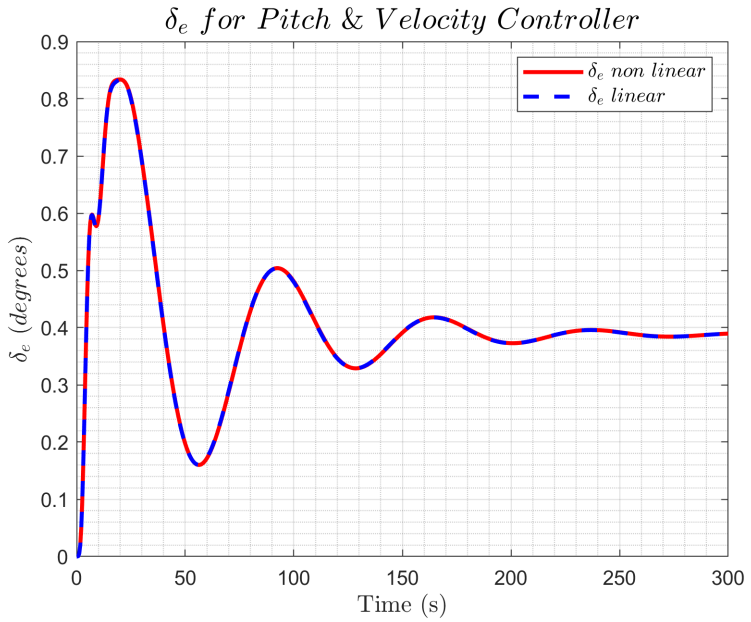


Figure 190: (δ_e) Response for Pitch and Velocity Controller for linear and nonlinear models

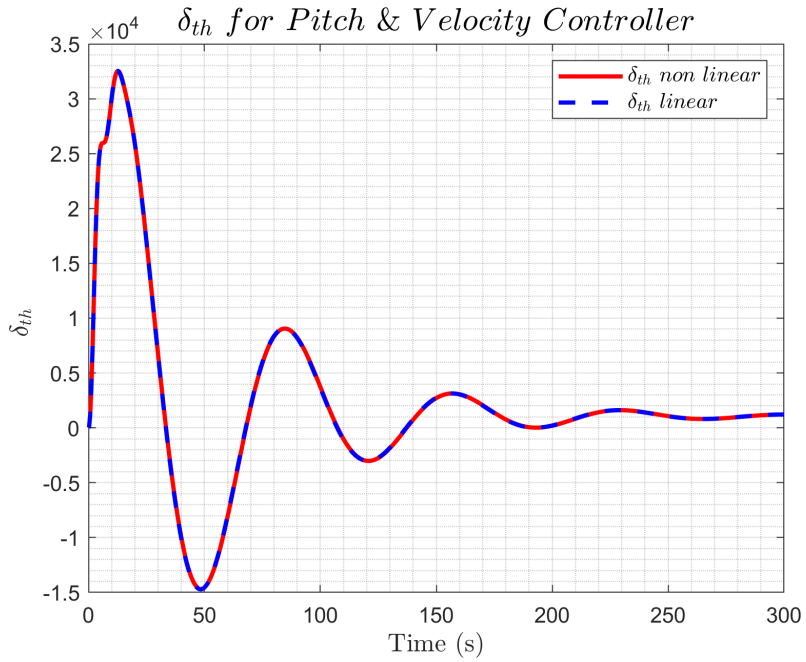


Figure 191: (δ_{th}) Response for Pitch and Velocity Controller for linear and nonlinear models

- **Result Comments:** The responses of the linear and nonlinear models exhibit a high degree of similarity. Both controllers effectively coordinated to achieve the desired velocity and maintain stable flight, validating their combined functionality.

7.2.3 Altitude Hold Controller Test

- **Commanded input:** delta Altitude 200 ft with velocity of 20 ft/s.
- **Observed States:** Pitch angle (θ), velocity (u), flight path angle (γ), altitude (H), elevator deflection (δ_e), and throttle input (δ_{th}) for linear and nonlinear models.

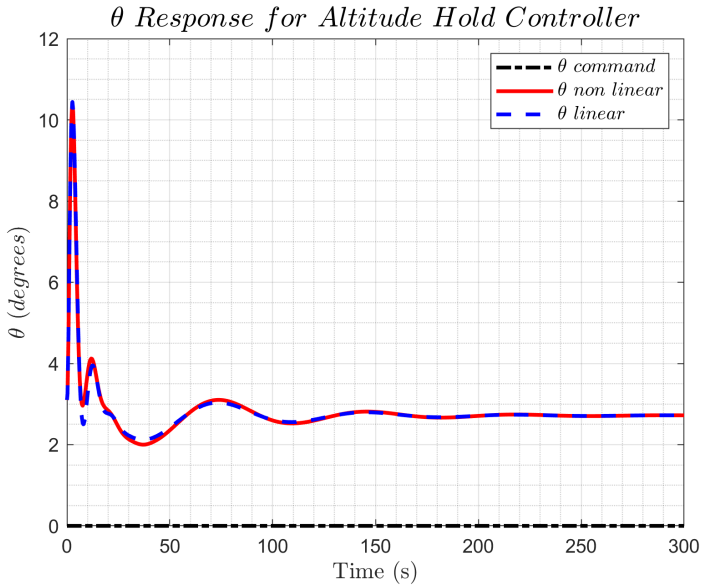


Figure 192: θ Response for Altitude Hold Controller for linear and nonlinear models

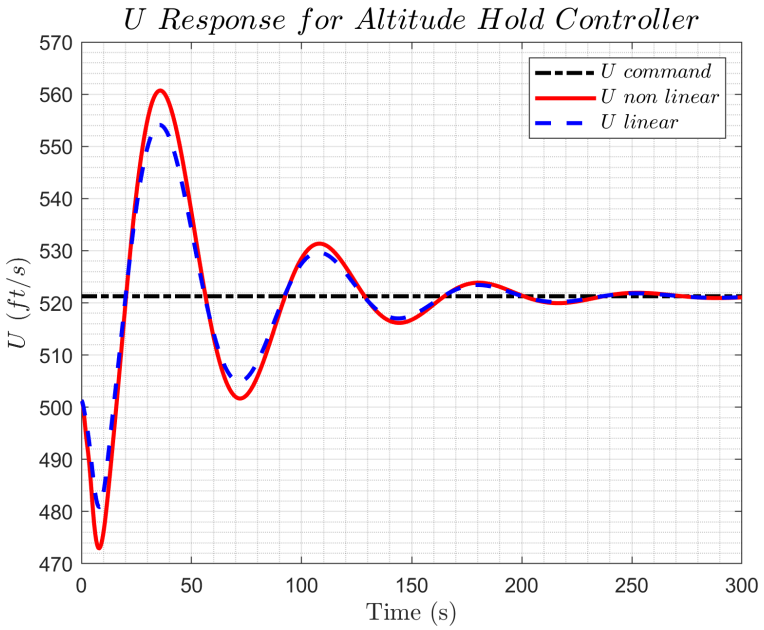


Figure 193: u Response for Altitude Hold Controller for linear and nonlinear models

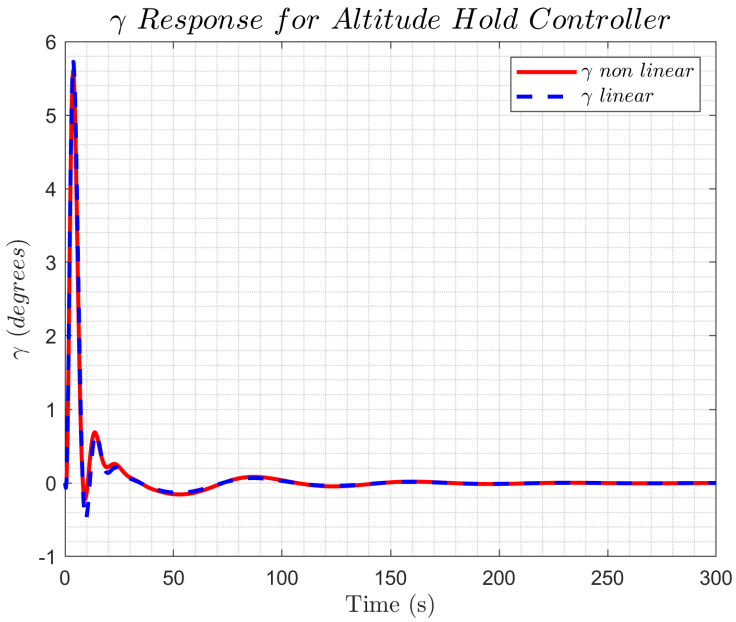


Figure 194: γ Response for Altitude Hold Controller for linear and nonlinear models

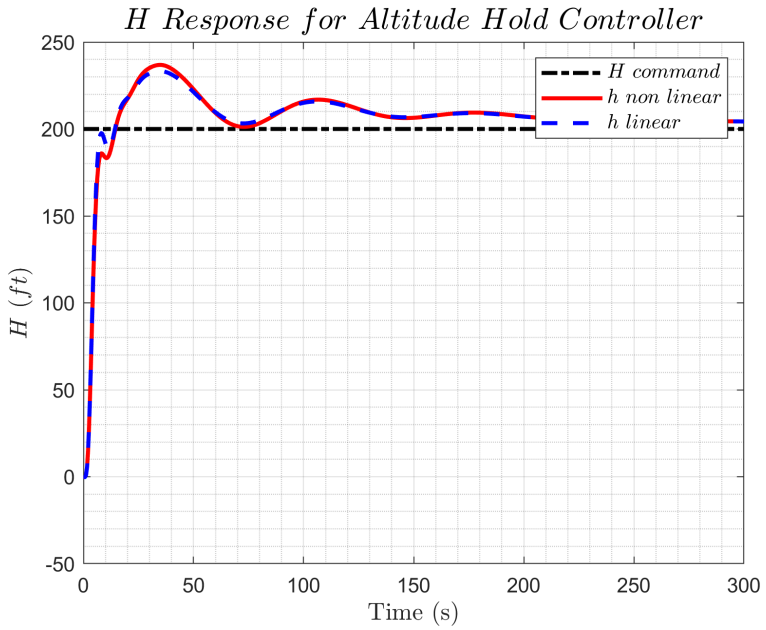


Figure 195: H Response for Altitude Hold Controller for linear and nonlinear models

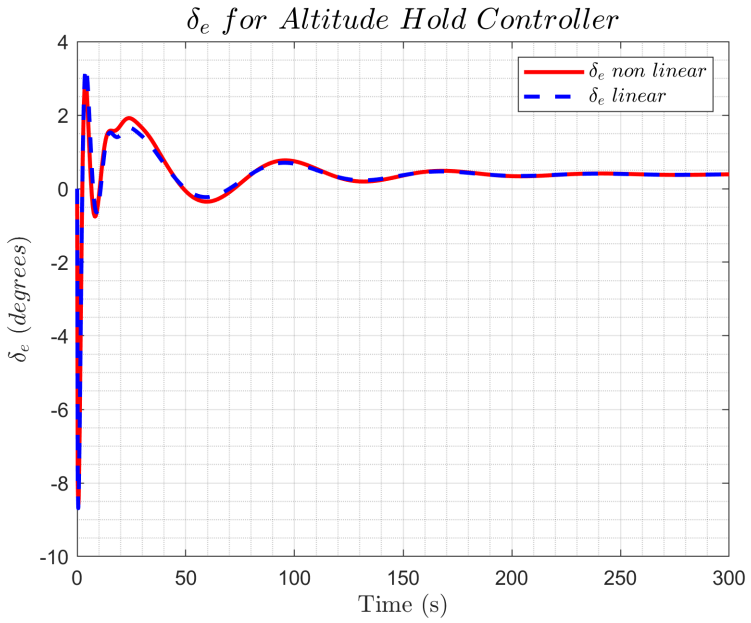


Figure 196: (δ_e) Response for Altitude Hold Controller for linear and nonlinear models

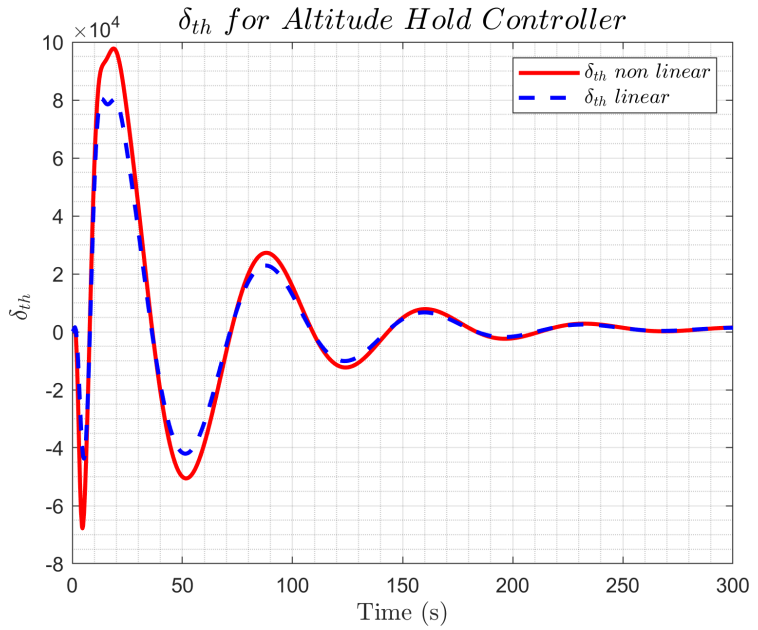


Figure 197: (δ_{th}) Response for Altitude Hold Controller for linear and nonlinear models

- **Result Comments:** The responses of the linear and non-linear models closely align, validating the functionality of the altitude hold controller. Minor differences in throttle input (δ_{th}) and Velocity U were observed, reflecting the increased complexity of the non-linear model.

7.2.4 Lateral Controller Test

- **Commanded input:** Heading change of 360°.
- **Observed States:** Sideslip angle (β), roll angle (ϕ), heading angle (ψ), altitude (H), rudder deflection (δ_r), and aileron deflection (δ_a) for linear and nonlinear models.

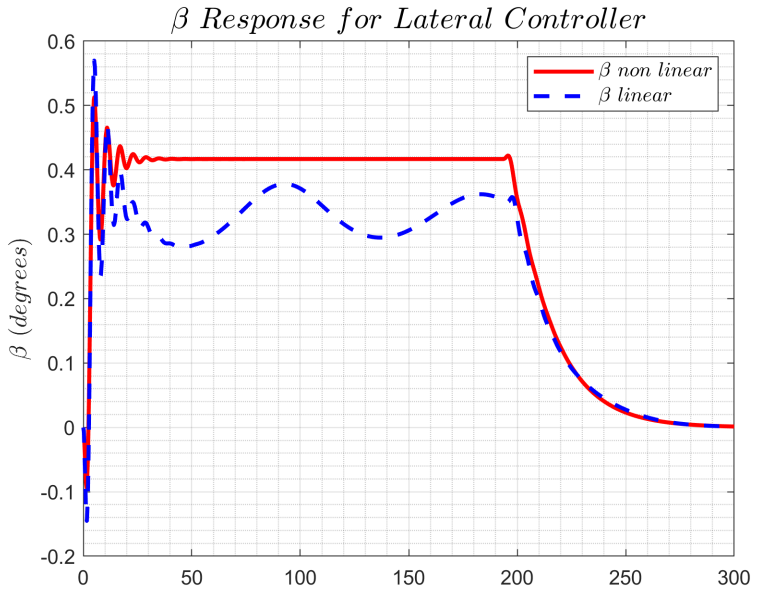


Figure 198: β Response for Lateral Controller for linear and nonlinear models

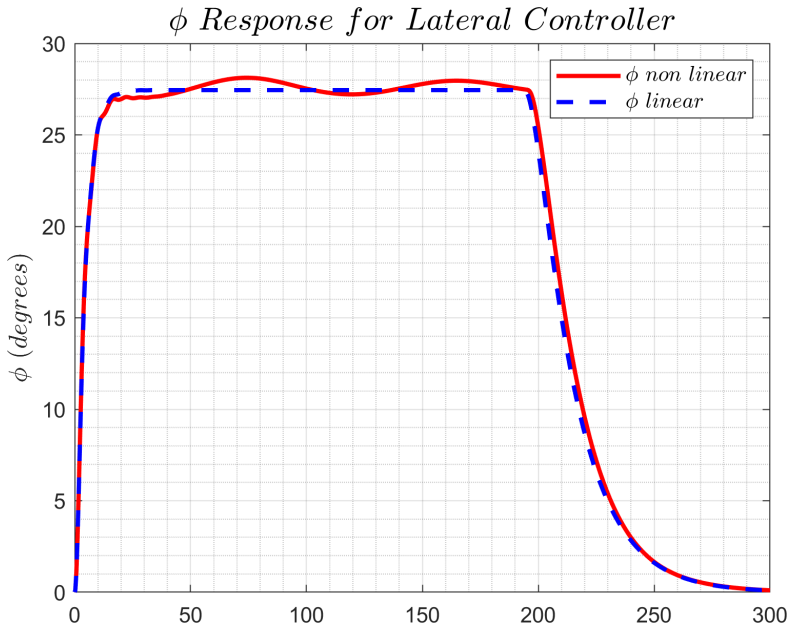


Figure 199: ϕ Response for Lateral Controller for linear and nonlinear models

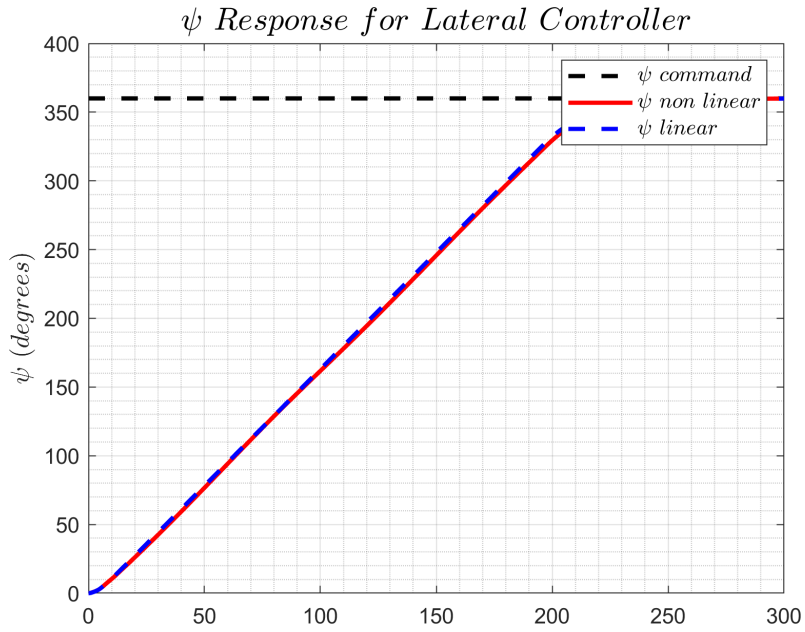


Figure 200: ψ Response for Lateral Controller for linear and nonlinear models

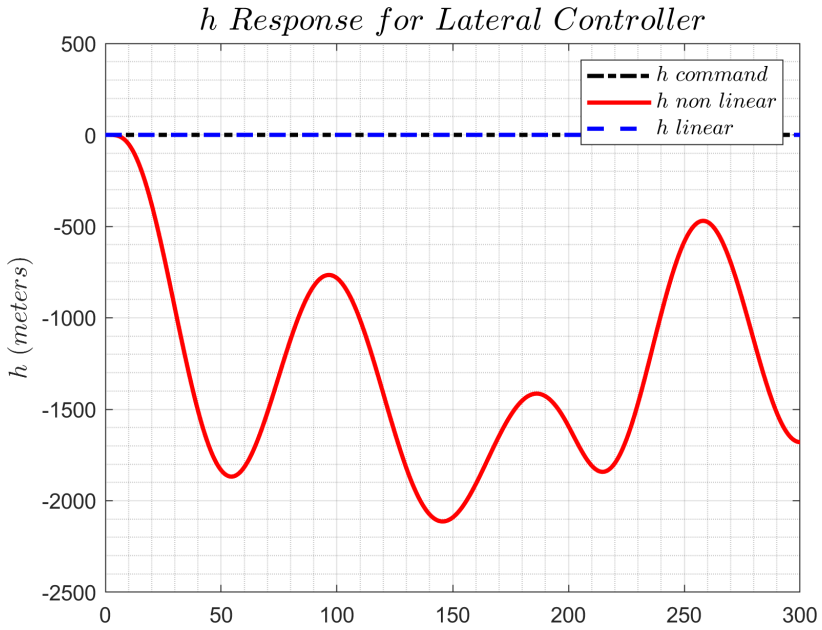


Figure 201: H Response for Lateral Controller for linear and nonlinear models

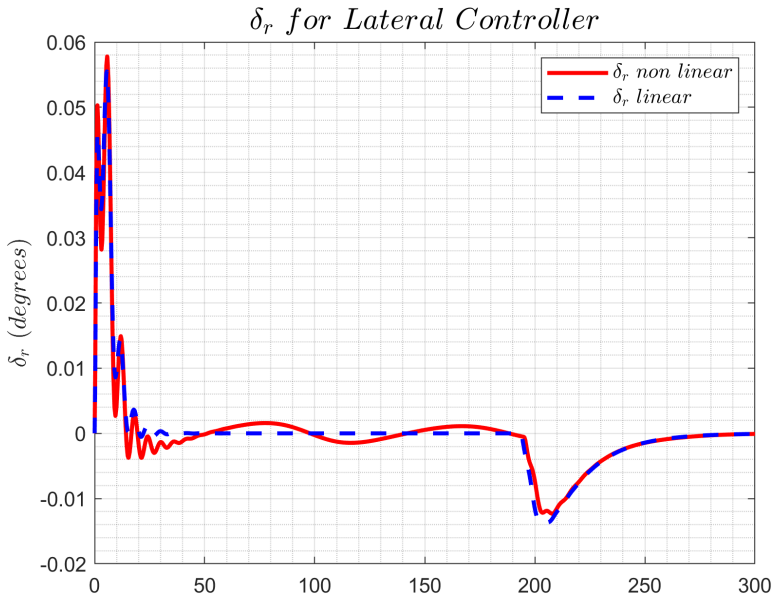


Figure 202: (δ_r) Response for Lateral Controller for linear and nonlinear models

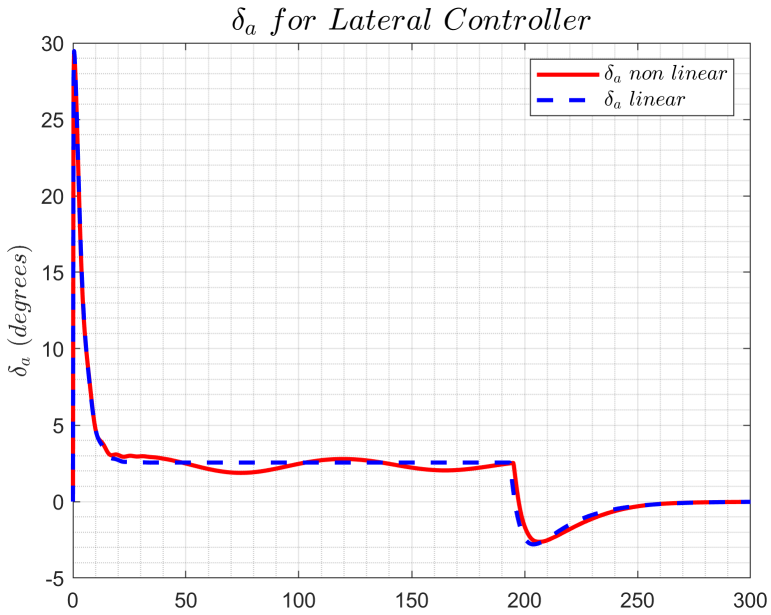


Figure 203: (δ_a) Response for Lateral Controller for linear and nonlinear models

- **Result Comments:** The responses from the linear and nonlinear models show consistency across most states. Minor differences in roll angle (ϕ) and sideslip angle (β) were observed in the nonlinear model due to its increased complexity. The altitude response in the nonlinear model exhibited significant nonlinearity, affecting the lateral controller's performance, while the linear model showed no change in altitude.

7.2.5 Lateral Controller and Altitude Hold Controller Test

- **Commanded input:** Heading change of 360° with altitude hold with 0 ft.
- **Observed States:** Sideslip angle (β), roll angle (ϕ), heading angle (ψ), altitude (H), rudder deflection (δ_r), aileron deflection (δ_a), elevator deflection (δ_e), and throttle input (δ_{th}) for linear and nonlinear models.

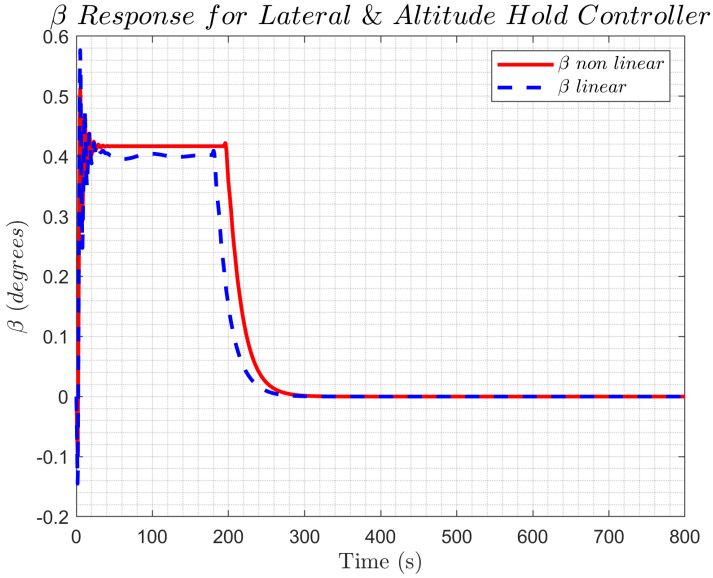


Figure 204: β Response for Lateral and Altitude Hold Controller for linear and nonlinear models

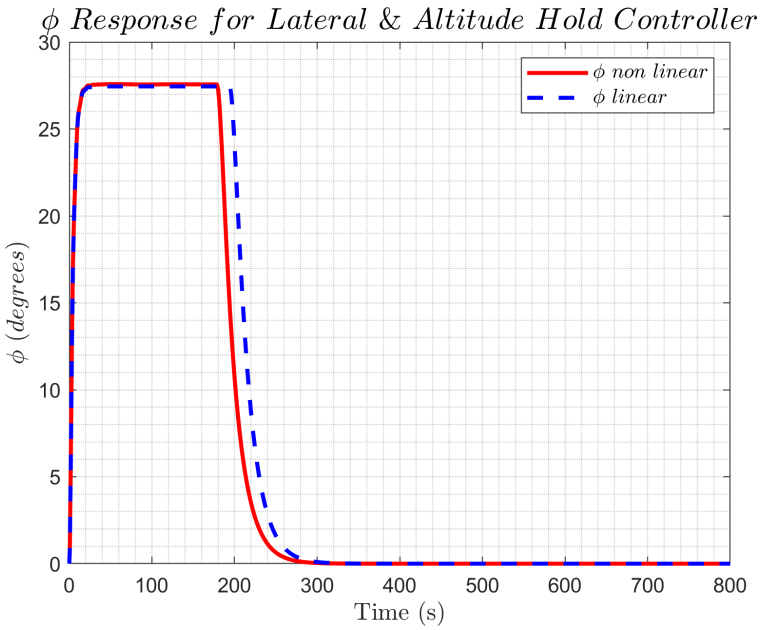


Figure 205: ϕ Response for Lateral and Altitude Hold Controller for linear and nonlinear models

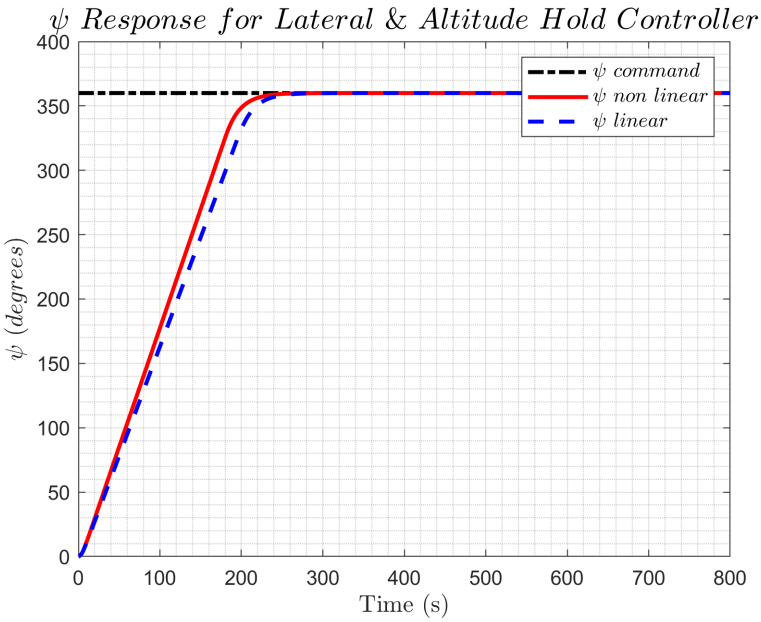


Figure 206: ψ Response for Lateral and Altitude Hold Controller for linear and nonlinear models

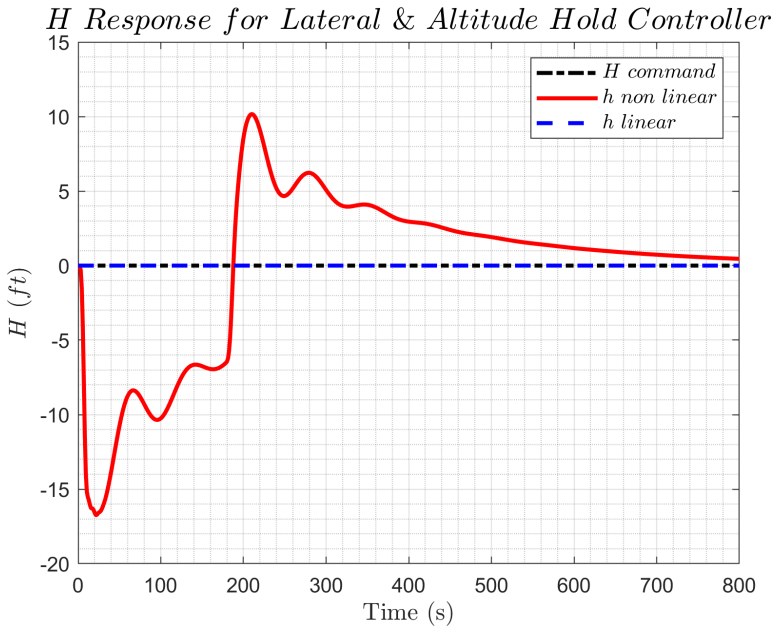


Figure 207: H Response for Lateral and Altitude Hold Controller for linear and nonlinear models

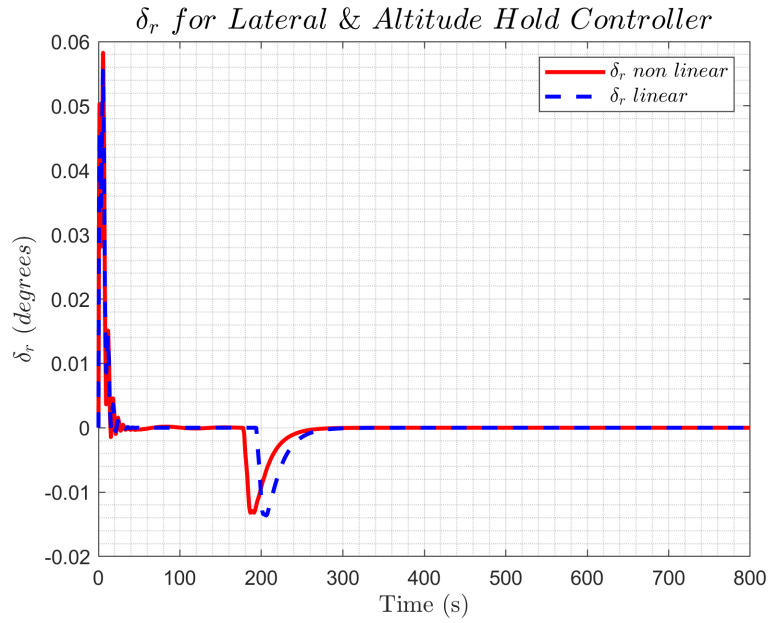


Figure 208: (δ_r) Response for Lateral and Altitude Hold Controller for linear and nonlinear models

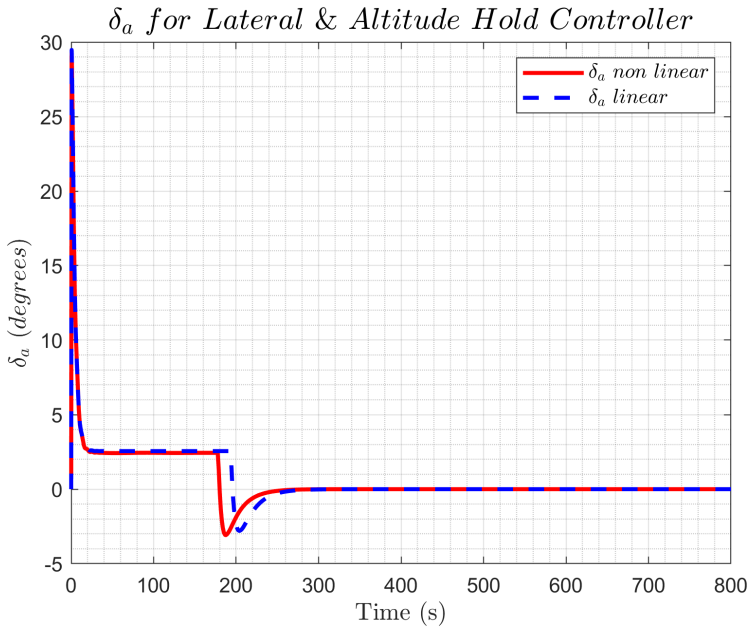


Figure 209: (δ_a) Response for Lateral and Altitude Hold Controller for linear and nonlinear models

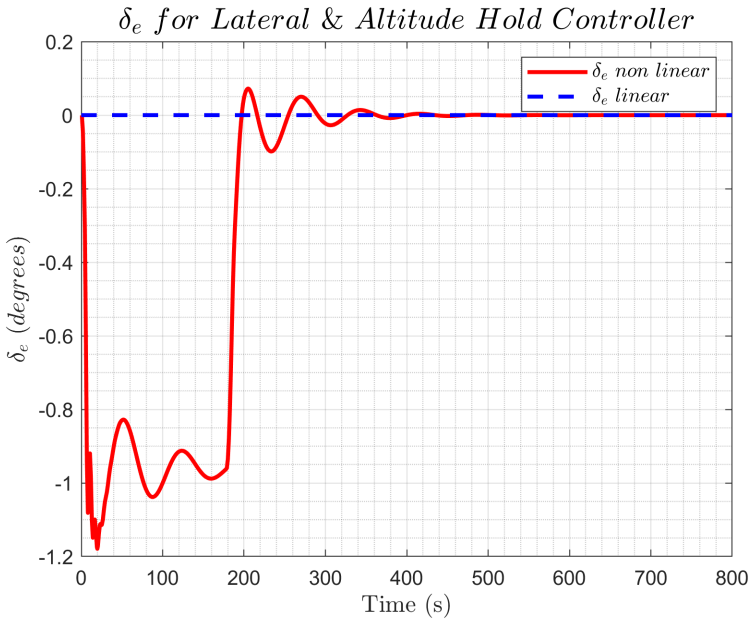


Figure 210: (δ_e) Response for Lateral and Altitude Hold Controller for linear and nonlinear models

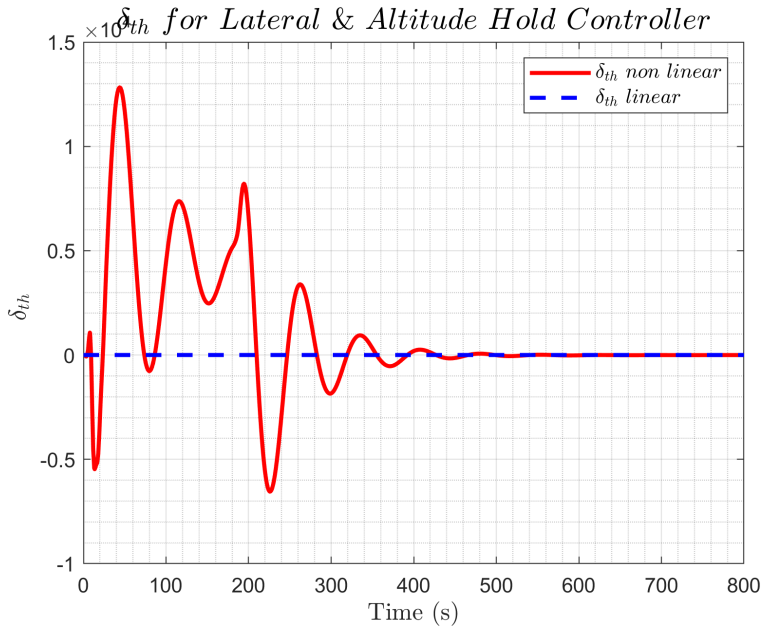


Figure 211: (δ_{th}) Response for Lateral and Altitude Hold Controller for linear and nonlinear models

- **Result Comments:** The responses of the linear and nonlinear models were consistent across observed states. The lateral controller effectively managed heading changes, and the altitude hold controller maintained the desired altitude, demonstrating successful coordination between lateral and longitudinal controls, resulting in a successfully achieved coordinated level turn.

7.2.6 Complete Autonomous Mission

After validating the functionality of the longitudinal and lateral controllers, a complete mission was performed to evaluate the integrated autopilot system. The mission included a sequence of maneuvers: climb, cruise, turn, and descent. The detailed mission phases are as follows:

- **Mission Phases:**
  1. **Climb:** Ascend to an altitude of 10,000 ft at a climb rate of 1500 ft/min.
  2. **Cruise:** Maintain constant velocity and heading for 2 minutes.
  3. **Turn:** Execute a coordinated 360-degree turn.
  4. **Cruise:** Continue at constant velocity and heading for another 2 minutes.
  5. **Climbing Turn:** Execute a 30-degree heading change with a step altitude increase of 100 ft.
  6. **Descent:** Descend to the starting altitude at a descent rate of 1500 ft/min.

To execute this mission, the autopilot was provided with input commands for altitude (H) and heading angle (ψ). These inputs, representing the mission phases, are detailed in Table 4.

Table 4: Mission segments points.

Altitude (H) input signal		**Heading angle (ψ) input signal**	
Time (s)	**Altitude (ft)**	**Time (s)**	**Heading (°)**
0	0	0	0
400	10000	400	0
520	10000	520	0
760	10000	760	360
880	10000	880	360
880	10100	1000	360
1000	10100	1000	390
1400	0	1400	390
1800	0	1800	0

After providing the input signals to the autopilot, the following figures display the corresponding responses for the altitude (H) and the heading angle (ψ):

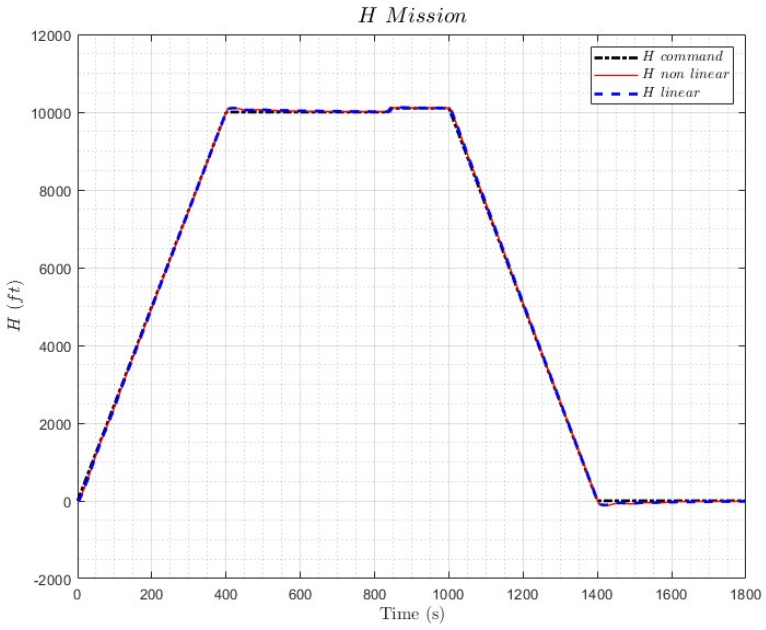


Figure 212: Altitude (H) Response for Mission

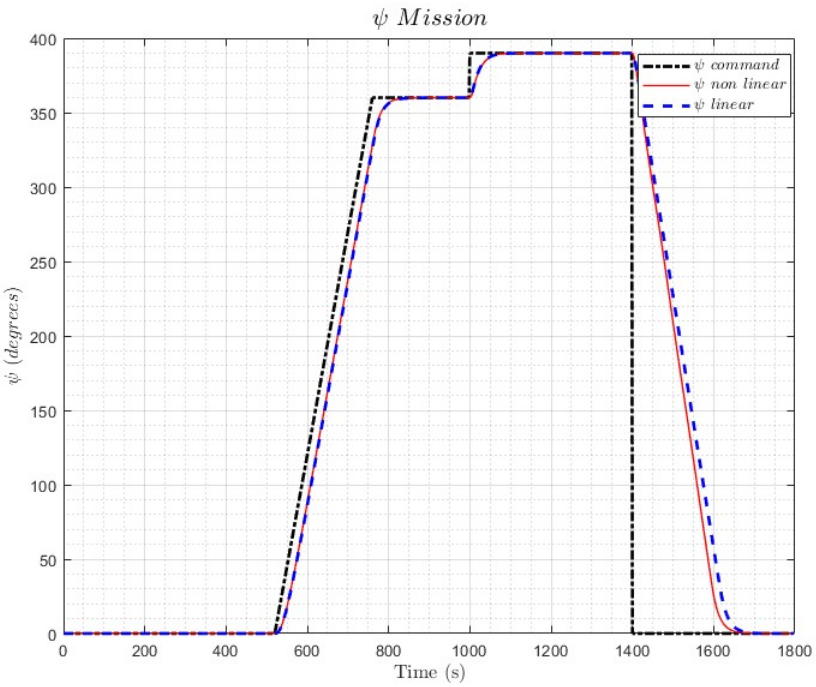


Figure 213: Heading Angle (ψ) Response for Mission

- **Result Comments:** The altitude (H) and heading angle (ψ) responses confirmed the seamless integration and flawless operation of all controllers. The autopilot demonstrated exceptional performance, achieving precise control and successfully accomplishing all mission objectives.

7.3 Flight Gear Simulation

The mission was conducted using the Flight Gear simulator, linked to Simulink. The Boeing 747-400 model was installed, and the autopilot was successfully operated to execute the required mission. A video animation of the simulation can be accessed here:Flight Gear Mission

8 References

[**1**]R. C. Nelson, Flight stability and automatic control. Boston, Mass.: Mcgraw Hill, 1998.
[**2**]D. Barry, "How long do pilots really spend on autopilot?," Cranfield.ac.uk, 2018. https://insights.cranfield.ac.uk/blog/how-long-do-pilots-really-spend-on-autopilot
[**3**]Wikipedia Contributors, "Autopilot," Wikipedia, Dec. 23, 2018. https://en.wikipedia.org/wiki/Autopilot
[**4**]"SAS, Autopilots and Flight Directors - What's in a Name? — helicoptermaintenancemagazine.com," helicoptermaintenancemagazine.com. https://helicoptermaintenancemagazine.com/article/sas-autopilots-and-flight-directors-what
[**5**]V. Sazdovski, T. Kolemishevska-Gugulovska, and M. Stankovski, "KALMAN FILTER IMPLEMENTATION FOR UNMANNED AERIAL VEHICLES NAVIGATION DEVELOPED WITHIN A GRADUATE COURSE," IFAC Proceedings Volumes, vol. 38, no. 1, pp. 12–17, 2005, doi: https://doi.org/10.3182/20050703-6-cz-1902.02265.
[**6**]Airbus, "Safety innovation 1: Fly-by-wire (FBW)," Airbus.com, 2023. https://www.airbus.com/en/newsroom/stories/2022-06-safety-innovation-1-fly-by-wire-fbw
[**7**]Bae Systems, "What are fly-by-wire systems?," BAE Systems — United States, 2023. https://www.baesystems.com/en-us/definition/what-are-fly-by-wire-systems
[**8**]"PX4 Open Source Autopilot," PX4 Open Source Autopilot, 2019. https://px4.io/
[**9**]"Open Source Drone Software. Versatile, Trusted, Open. ArduPilot.," ardupilot.org. https://ardupilot.org/
[**10**]"CubePilot — Autopilot-on-Module — Blue Manufactured in USA — Blue Assembled in USA — Pixhawk Original Team," Cubepilot.org, 2024. https://cubepilot.org//cube/features?utm$_source = unmannedsystemstechnology.comutm_medium = referral$

9 Appendix

9.1 Appendix A Matlab Codes

9.1.1 MATLAB Code for Solving the System of ODEs Using the RK4 Method

```
function [t, y] = RK4_solver(ode_func, y0, t0, tf, n)
% RK4_SOLVER Solves a system of ODEs using the 4th-order Runge-Kutta method.
%   Inputs:
%     ode_func - function handle of the system of ODEs (dy/dt = f(t, y))
%     y0       - vector of initial conditions (column vector)
%     t0       - initial time
%     tf       - final time
%     n        - number of steps
%
%   Outputs:
%     t        - time vector
%     y        - solution matrix (each row corresponds to time steps)

    h = (tf - t0) / n;          % Step size
    t = linspace(t0, tf, n); % Time vector
    y = zeros(length(y0), n);% Solution matrix
    y(:,1) = y0;                % Set initial condition

    % RK4 Loop
    for i = 1:n
        k1 = h * ode_func(t(i), y(:,i));
        k2 = h * ode_func(t(i) + 0.5*h, y(:,i) + 0.5*k1);
        k3 = h * ode_func(t(i) + 0.5*h, y(:,i) + 0.5*k2);
        k4 = h * ode_func(t(i) + h, y(:,i) + k3);
        y(:,i+1) = y(:,i) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
    end
    y=y(:,1:n); % to remove the last row
end
```

```
% Define the system of ODEs as a function handle
ode_system = @(t, y) [sin(t) + cos(y(1)) + sin(y(2));
                      cos(t) + sin(y(2))];

% Initial conditions
y0 = [-1; 1];
t0 = 0;
tf = 20;
n = 100; % Number of steps

% Call the RK4 solver function
[t, y] = RK4_solver(ode_system, y0, t0, tf, n);

% Plot the results
plot(t, y(1,:), t, y(2,:));
legend('y_1', 'y_2');
xlabel('Time t');
ylabel('Solutions y_1 and y_2');
title('Solutions of the ODE system using RK4');

% Validate against ode45
[t_ode45, y_ode45] = ode45(ode_system, [t0 tf], y0);
hold on;
plot(t_ode45, y_ode45(:,1), '--', t_ode45, y_ode45(:,2), '--');
legend('y_1 RK4', 'y_2 RK4', 'y_1 ode45', 'y_2 ode45');
```

9.1.2 MATLAB Code for Solving Airplane EOM

```
function diff=getstates(t, y)
    global F
    global m
    global I
    global M
    %% Allocating the States Vector
    velocities=y(1:3); % Allocating velocity positions
```

```
    angular_velocities=y(4:6); % Allocating angular velocity positions
    angles=y(7:9); % Allocating angles positions
    position=y(10:12); % Allocating xyz positions
    %% Differential Equations Setup
    velocities_dot=(1/m)*F - cross(angular_velocities ,velocities); % Velocity
    derivative equations
    angular_velocities_dot=I\(M-cross(angular_velocities,I*angular_velocities));
    % Angular Velocity derivative equations
    angles_dot=[1, sin(angles(1)) * tan(angles(2)), cos(angles(1)) * tan(angles
    (2));
         0, cos(angles(1)), -sin(angles(1));
        0, sin(angles(1)) / cos(angles(2)), cos(angles(1)) / cos(angles(2))] *
    angular_velocities; % Angles derivative equations
    positions_dot=eul2rotm([angles(3) angles(2) angles(1)])*velocities; %
    Position derivatives equations
    diff=[velocities_dot; angular_velocities_dot; angles_dot; positions_dot]; %
    Creating the differential equation matrix
end
```

```
%% Setup Parameters
t0 = 0; % Initial time
tf = 25; % Final Time
n = 10000; % Number of Steps
%% Identifying Variables
global F
global m
global I
global M
F=[10;5;3]; % Forces Input
M=[10;15;20]; % Moments Input
m=30; % Mass Input
I=[1,-2,-1;-2,5,-4;-1,-4,0.2]; % Inertia Input

initial=[10;2;0;2*pi/180;pi/180;0;20*pi/180;15*pi/180;30*pi/180;2;4;7]; % Initial
     Parameters
[t2, y2] = RK4_solver(@getstates, initial, t0, tf, n); % Solving the airplane EOM
     equations
% Modified plotting with 4 figures, each having 3 plots (Simulink vs. MATLAB
    comparison)
%
simOut = sim('simulinktask2', 'ReturnWorkspaceOutputs', 'on');

% Extracting the outputs from the Simulink model
t = simOut.tout; % Time vector
out = simOut;    % Store Simulink outputs (make sure 'out' is a structured output
     in the model)

% Figure 1: Velocities (u, v, w)
figure;
vel_names = {'u', 'v', 'w'};
for i = 1:3
    subplot(3,1,i)
    plot(t,out.velocities.Data(:,i), 'r', t2, y2(i,:), 'b--')
    title(['Velocity: ', vel_names{i}])
    xlabel("Time (Sec)")
    ylabel([vel_names{i}, ' (m/sec)'])
    legend('Simulink', 'MATLAB')
end
sgtitle("Velocities Comparison")

% Figure 2: Angular Velocities (p, q, r)
figure;
ang_vel_names = {'p', 'q', 'r'};
for i = 1:3
    subplot(3,1,i)
    plot(t,out.angular_velocities.Data(:,i), 'r', t2, y2(i+3,:), 'b--')
    title(['Angular Velocity: ', ang_vel_names{i}])
    xlabel("Time (Sec)")
    ylabel([ang_vel_names{i}, ' (rad/sec)'])
    legend('Simulink', 'MATLAB')
end
```

```
sgtitle("Angular Velocities Comparison")

% Figure 3: Angles (phi, theta, psi)
figure;
angle_names = {'\phi (phi)', '\theta (theta)', '\psi (psi)'};
for i = 1:3
    subplot(3,1,i)
    plot(t,wrapToPi(out.angles.Data(:,i)), 'r', t2, wrapToPi(y2(i+6,:)), 'b--')
    title(['Angle: ', angle_names{i}])
    xlabel("Time (Sec)")
    ylabel([angle_names{i}, ' (rad)'])
    legend('Simulink', 'MATLAB')
end
sgtitle("Angles Comparison")

% Figure 4: Positions (x, y, z)
figure;
pos_names = {'x', 'y', 'z'};
for i = 1:3
    subplot(3,1,i)
    plot(t,out.position.Data(:,i), 'r', t2, y2(i+9,:), 'b--')
    title(['Position: ', pos_names{i}])
    xlabel("Time (Sec)")
    ylabel([pos_names{i}, ' (m)'])
    legend('Simulink', 'MATLAB')
end
sgtitle("Positions Comparison")
```

9.1.3 MATLAB Code for calculating forces and moments

```
function diff=F_M_Cal(SD,Delta,Initial_F_M, phi,theta,psi)
  global m g I
    % Force and moment matrix, based on stability derivatives
    F_M_Matrix =  [SD(1) 0 SD(4) 0 0 0 0 0 0 SD(11) SD(14);
                   0 SD(17) 0 0 0 0 0 0 SD(26) 0 0;
                   SD(2) 0 SD(5) SD(7) 0 SD(8) 0 0 0 SD(12) SD(15);
                   0 0 0 0 SD(21) 0 SD(23) SD(27) SD(29) 0 0;
                   SD(3) 0 SD(6) SD(9) 0 SD(10) 0 0 0 SD(13) SD(16);
                   0 0 0 0 SD(22) 0 SD(24) SD(28) SD(30) 0 0];

    % Calculate the linear change in forces and moments
    Delta_F_M = F_M_Matrix * Delta;  % Perturbations in forces and moments


    F_M = Delta_F_M + Initial_F_M + [-m * g * sin(theta); m * g * cos(theta) *
    sin(phi); m * g * cos(theta) * cos(phi);0;0;0];

    diff=F_M;
end
```

9.1.4 MATLAB Code for transforming lateral derivatives

```
function SD_Lat=LateralFunc(SD_Lat_dash,Ixz,Izz,Ixx,V)

    G=1/(1-Ixz^2/(Ixx*Izz));
    A=Ixz/Ixx;
    B=Ixz/Izz;
    dashed=[SD_Lat_dash(3);SD_Lat_dash(5);SD_Lat_dash(7);SD_Lat_dash(13);
    SD_Lat_dash(11)
            SD_Lat_dash(4);SD_Lat_dash(6);SD_Lat_dash(8);SD_Lat_dash(14);
    SD_Lat_dash(12)];
    Matrix=[G 0 0 0 0 G*A 0 0 0 0;
            0 G 0 0 0 0 G*A 0 0 0;
            0 0 G 0 0 0 0 G*A 0 0;
            0 0 0 G 0 0 0 0 G*A 0;
            0 0 0 0 G 0 0 0 0 G*A;
            G*B 0 0 0 0 G 0 0 0 0;
            0 G*B 0 0 0 0 G 0 0 0;
            0 0 G*B 0 0 0 0 G 0 0;
            0 0 0 G*B 0 0 0 0 G 0;
            0 0 0 0 G*B 0 0 0 0 G];
```

```

    star=[SD_Lat_dash(9);SD_Lat_dash(10)];
    non_dashed=inv(Matrix)*dashed;
    non_stared=star*V;
    SD_Lat=[SD_Lat_dash(1);SD_Lat_dash(2);non_dashed(1);non_dashed(6);
            non_dashed(2);non_dashed(7);non_dashed(3);non_dashed(8);
            non_stared(1);non_stared(2);non_dashed(5);non_dashed(10);
            non_dashed(4);non_dashed(9)];
end
```

9.1.5 MATLAB Code for RK4 modified

```
function [t, y] = RK4_solver(ode, y0, t0, tf, n,SD, Initial_F_M,dc,dt)
    global M m
    global F I g
    h = (tf - t0) / n;          % Step size
    t = linspace(t0, tf, n+1); % Time vector
    y = zeros(length(y0), n+1);% Solution matrix
    y(:,1) = y0;                % Set initial condition
    wdot_0 = 0;
    % RK4 Loop
    for i = 1:n
        k1 =  h*(ode(t(i), y(:,i)));
        k2 =  h*(ode(t(i) + 0.5*h, y(:,i) + 0.5*k1));
        k3 =  h*(ode(t(i) + 0.5*h, y(:,i) + 0.5*k2));
        k4 =  h*(ode(t(i) + h, y(:,i) + k3));
        y(:,i+1) = y(:,i) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);

        u = y(1, i);
        v = y(2, i);
        w = y(3, i);
        p = y(4, i);
        q = y(5, i);
        r = y(6, i);
        phi = y(7, i);
        theta = y(8, i);
        psi = y(9, i);
        xx = y(10, i);
        yy = y(11, i);
        zz = y(12, i);




        if i == 1
            wdot = (w-y0(3))/dt;
        else
            wdot = (w-y(3, i-1))/dt;
        end

        Delta = [u-y0(1); v-y0(2); w-y0(3);wdot-wdot_0; p-y0(4); q-y0(5); r-y0(6)
    ;dc(1);dc(2);dc(3);dc(4)];
        F_M=F_M_Cal(SD,Delta,Initial_F_M, phi,theta,psi);
        F=F_M(1:3);
        M=F_M(4:6);

    end
    y=y(:,1:n); % to remove the last row
    t=t(:,1:n);
end
```

9.1.6 MATLAB Code for calculating the states of airplane

```
% Main Script: Airplane Nonlinear Simulator with RBD Solver

% Clear workspace
clc;
clear all;
close all;
```

```
global m
global I
global F M
global g
%% Load Aircraft Data from Excel
filename_density_L = 'excelsheet_data_modified';  % Path to your excel sheet
aircraft_data = xlsread(filename_density_L, 'B2:B61');  % Read from Excel

% Time vector parameters
dt = aircraft_data(1);
tfinal = aircraft_data(2);
time_V = (0:dt:tfinal)';

% Initial conditions for states
s0 = aircraft_data(4:15);
sdot0 = zeros(12,1);  % Initial derivative of states
Vto = sqrt(s0(1)^2 + s0(2)^2 + s0(3)^2);  % Compute initial total velocity (Vto)

% Control actions values
dc = [ aircraft_data(57:59) * pi/180 ; aircraft_data(60) ];  % Control inputs in
    radians

% Gravity, mass, and inertia values from aircraft data

m = 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;

% Inertia matrix and inverse inertia matrix

I = [Ixx, -Ixy, -Ixz; -Ixy, Iyy, -Iyz; -Ixz, -Iyz, Izz];
invI = inv(I);

% Stability derivatives for longitudinal motion
SD_Long = aircraft_data(21:36);
SD_Long_final = SD_Long;

% Stability derivatives for lateral motion
SD_Lat_dash = aircraft_data(37:50);
SD_Lat = LateralFunc(SD_Lat_dash, Ixz, Izz, Ixx, Vto);
SD_Lat_final = SD_Lat;

% Initial gravity force in body frame
mg0 = m * g * [sin(s0(8)); -cos(s0(8)) * sin(s0(7)); -cos(s0(8)) * cos(s0(7))];


%% Calculate initial forces and moments
phi_0=s0(7);
theta_0=s0(8);
psi_0=s0(9);
Initial_F_M = [m * g * sin(theta_0); -m * g * cos(theta_0) * sin(phi_0); -m * g *
     cos(theta_0) * cos(phi_0); 0; 0; 0];


% Combine stability derivatives into a single array
for i = 1:16
    SD(i) = SD_Long_final(i);
end
for j = 1:14
    SD(j+16) = SD_Lat_final(j);  % All stability derivatives in one array
end

% Delta represents perturbations from trim conditions
Delta = [0; 0; 0; 0; 0; 0; 0; dc(1); dc(2); dc(3); dc(4)];  % Initial delta u, v,
     w,wdot, p, q, r, da, dr, de, dth


```

```
F_M=F_M_Cal(SD,Delta, Initial_F_M,phi_0,theta_0,psi_0);

F=F_M(1:3);
M=F_M(4:6);


%% Call RK4 solver with RBD Solver to simulate the airplane's motion
n = 1000;  % Number of steps for simulation
velocity_dot_0=s0(1:3);
angular_v_dot_0=s0(4:6);
angles_dot_0=s0(7:9);
position_dot_0=s0(10:12);
Initial_state=[velocity_dot_0;angular_v_dot_0;angles_dot_0;position_dot_0];
[t2, y2] = RK4_solver(@getstates,Initial_state , 0, tfinal, n,SD, Initial_F_M,dc,
    dt);



%% Plotting
rad_to_deg = 180 / pi;  % Conversion from radians to degrees

% Plot results (e.g., velocities, angles, positions)
figure;

% Plot forward, side, and vertical velocities (u, v, w)
subplot(4,3,1);
plot(t2, y2(1,:), 'b');  % u (forward velocity)
title('Forward Velocity (u)');
xlabel('Time (s)');
ylabel('Velocity (ft/s)');

subplot(4,3,2);
plot(t2, y2(2,:), 'r');  % v (side velocity)
title('Side Velocity (v)');
xlabel('Time (s)');
ylabel('Velocity (ft/s)');

subplot(4,3,3);
plot(t2, y2(3,:), 'g');  % w (vertical velocity)
title('Vertical Velocity (w)');
xlabel('Time (s)');
ylabel('Velocity (ft/s)');

% Angular velocities (p, q, r)

subplot(4,3,4);
plot(t2, y2(4,:), 'b');  % p (roll rate)
title('Roll Rate (p)');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');

subplot(4,3,5);
plot(t2, y2(5,:), 'r');  % q (pitch rate)
title('Pitch Rate (q)');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');

subplot(4,3,6);
plot(t2, y2(6,:), 'g');  % r (yaw rate)
title('Yaw Rate (r)');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');

% Euler angles (phi, theta, psi) in degrees

subplot(4,3,7);
plot(t2, y2(7,:) * rad_to_deg, 'b');  % phi (roll angle in degrees)
title('Roll Angle (phi) in degrees');
xlabel('Time (s)');
ylabel('Angle (deg)');

```

```
subplot(4,3,8);
plot(t2, y2(8,:) * rad_to_deg, 'r');  % theta (pitch angle in degrees)
title('Pitch Angle (theta) in degrees');
xlabel('Time (s)');
ylabel('Angle (deg)');

subplot(4,3,9);
plot(t2, y2(9,:) * rad_to_deg, 'g');  % psi (yaw angle in degrees)
title('Yaw Angle (psi) in degrees');
xlabel('Time (s)');
ylabel('Angle (deg)');

% Position (x, y, z) in feet

subplot(4,3,10);
plot(t2, y2(10,:), 'b');
title('Position X (ft)');
xlabel('Time (s)');
ylabel('Position (ft)');

subplot(4,3,11);
plot(t2, y2(11,:), 'r');
title('Position Y (ft)');
xlabel('Time (s)');
ylabel('Position (ft)');

subplot(4,3,12);
plot(t2, y2(12,:), 'g');
title('Position Z (ft)');
xlabel('Time (s)');
ylabel('Position (ft)');

% Trajectory (x vs y vs z in feet)
figure;
plot3(y2(10,:), y2(11,:), y2(12,:), 'b');  % 3D trajectory plot
title('Trajectory of the Aircraft in Feet');
xlabel('X (ft)');
ylabel('Y (ft)');
zlabel('Z (ft)');
grid on;

% Additional plots for angular velocities, angles, and positions can be added
    here

% RBD Solver Function with m, g, I passed as parameters
```

9.1.7 MATLAB Code Longitudinal Lateral stability analysis

```
% Main Script: Airplane Nonlinear Simulator with RBD Solver

% Clear workspace
clc;
clear all;
close all;
global m
global I
global F M
global g
%% Load Aircraft Data
%read from Execel File
% filename_density_L = 'ourflightcondation';  % Path to your excel sheet
% aircraft_data = xlsread(filename_density_L, 'B2:B61');  % Read from Excel

% read Data from CSV
filename_density_L = 'excelsheet_data_modified2';  % Path to your CSV file
aircraft_data = readmatrix(filename_density_L, 'Range', 'B2:B61');  % Read from
    CSV

% Time vector parameters
dt = aircraft_data(1);
tfinal = aircraft_data(2);
```

```
t0 = aircraft_data(3);


% Initial conditions for states
s0 = aircraft_data(4:15);
sdot0 = zeros(12,1);  % Initial derivative of states
Vto = sqrt(s0(1)^2 + s0(2)^2 + s0(3)^2);  % Compute initial total velocity (Vto)
initialstatemodified=[s0(1:3);0;s0(4:6)];
% Control actions values
dc = [ aircraft_data(57:59) * pi/180 ; aircraft_data(60) ];  % Control inputs in
    radians

% Gravity, mass, and inertia values from aircraft data

m = 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;

% Inertia matrix and inverse inertia matrix

I = [Ixx, -Ixy, -Ixz; -Ixy, Iyy, -Iyz; -Ixz, -Iyz, Izz];
invI = inv(I);

%Initial velocities and rates
u_0 = s0(1);
v_0 = s0(2);
w_0 = s0(3);
p_0 = s0(4);
q_0 = s0(5);
r_0 = s0(6);
phi_0 = s0(7);
theta_0 = s0(8);
psi_0 = s0(9);
x_0 = s0(10);
y_0 = s0(11);
z_0 = s0(12);
wdot_0 = 0;
wdot = 0;
mass = m;

% Stability derivatives for longitudinal motion
SD_Long = aircraft_data(21:36);
SD_Long_final = SD_Long;
templong  =num2cell(SD_Long);
[Xu,Zu,Mu,Xw,Zw,Mw,Zwd,Zq,Mwd,Mq,Xde,Zde,Mde,Xth,Zdth,Mdth] = deal(templong{:});
clear templong;


% Stability derivatives for lateral motion
SD_Lat_dash = aircraft_data(37:50);
SD_Lat = LateralFunc(SD_Lat_dash, Ixz, Izz, Ixx, Vto);
SD_Lat_final = SD_Lat;
templatrel = num2cell(SD_Lat);
[Yv,Yb,Lb,Nb,Lp,Np,Lr,Nr,Yda,Ydr,Lda,Nda,Ldr,Ndr] = deal(templatrel{:});
clear templatrel;

Lv=Lb/Vto;
Nv=Nb/Vto;
Yp=0;
Yr=0;

% Initial gravity force in body frame
mg0 = m * g * [sin(s0(8)); -cos(s0(8)) * sin(s0(7)); -cos(s0(8)) * cos(s0(7))];
initial_F_M = m * g * [sin(s0(8)); -cos(s0(8)) * sin(s0(7)); -cos(s0(8)) * cos(s0
    (7));0;0;0]; % Simulink Input
M_0 = [0;0;0];

```

```
%% Matrix used in simulink
State_Matrix = [ Xu   0    Xw   0    0    0    0;
                 0    Yv   0    Yp   0    Yr   0;
                 Zu   0    Zw   0    Zq   Lr   Zwd;
                 0    Lv   0    Lp   0    Lr   0;
                 Mu   0    Mw   0    Mq   0   Mwd;
                 0    Nv   0    Np   0    Nr  0];

Control_Matrix = [0 Xde Xth 0
                  Yda 0 0 Ydr
                  0  Zde Zdth 0
                  Lda 0  0  Ldr
                  0  Mde Mdth 0
                  Nda 0  0 Ndr];


%% intial struc state
so_struct.x = s0(10);
so_struct.y = s0(11);
so_struct.z = s0(12);
so_struct.phi = s0(7);
so_struct.theta = s0(8);
so_struct.epsi = s0(9);
so_struct.p = s0(4);
so_struct.q = s0(5);
so_struct.r = s0(6);
so_struct.u = s0(1);
so_struct.v = s0(2);
so_struct.w = s0(3);
so_struct.alpha = s0(8);
so_struct.Beta = 0;
so_struct.V_total = Vto;
so_struct.W_dot = 0;

%% Combine stability derivatives into a single array
for i = 1:16
    SD(i) = SD_Long_final(i);
end
for j = 1:14
    SD(j+16) = SD_Lat_final(j);  % All stability derivatives in one array
end

F_M_Matrix_dv =  [SD(1) 0 SD(4) 0 0 0 0 0 0 SD(11) SD(14);
                  0 SD(17) 0 0 0 0 0 0 SD(26) 0 0;
                  SD(2) 0 SD(5) SD(7) 0 SD(8) 0 0 0 SD(12) SD(15);
                  0 SD(19)/Vto 0 0 SD(21) 0 SD(23) SD(27) SD(29) 0 0;
                  SD(3) 0 SD(6) SD(9) 0 SD(10) 0 0 0 SD(13) SD(16);
                  0 SD(20)/Vto 0 0 SD(22) 0 SD(24) SD(28) SD(30) 0 0]; %
    Simulink Input
%% Calculating forces and moments
n = (tfinal - t0) / dt;
velocity_0=s0(1:3);
angular_v0=s0(4:6);
angles_0=s0(7:9);
position_0=s0(10:12);
t = (0:dt:tfinal)';

Initial_state=[velocity_0;angular_v0;angles_0;position_0];
[F,M]=F_M_Cal(SD,s0,Vto,dc,Initial_state,0);
[t, states] = RK4_solver(@getstates,Initial_state , 0, tfinal,n, SD,Vto, dc);

%% Initiating simulink
modelName = 'Final_Task3_Simulink_23b';
load_system(modelName);
set_param(modelName, 'StartTime', '0', 'StopTime', num2str(tfinal));
simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');

t2 = simOut.tout;   % Time vector from simout
y2 = simOut.simout.Data;
y2 = y2';  % Now y2 is [12 x 200001]

```

```
%% Task 4
%% Full Longitudinal System
% Define constants
theta_0 = s0(8); wo = s0(3);u_0 = s0(1);wdot=0;

% Define constants and matrices for the longitudinal state-space model
u = [dc(3)* ones(size(t)); 1*ones(size(t))];

A_longitudinal = [Xu, Xw,0, -g*cos(theta_0);
                  Zu/(1-Zwd), Zw/(1-Zwd), (Zq + u_0)/(1-Zwd), -g*sin(theta_0)/(1-Zwd);
                  Mu + Mwd*Zu/(1-Zwd), Mw + wdot*Zw/(1-Zwd), Mq + Mwd*(Zq+u_0)/(1-Zwd), -Mwd*g*sin(theta_0)/(1-Zwd);
                  0, 0, 1, 0];

B_longitudinal = [Xde, Xth;
                  Zde/(1-Zwd), Zdth/(1-Zwd);
                  Mde + Mwd*Zde/(1-Zwd), Mdth + Mwd*Zdth/(1-Zwd);
                  0, 0];

C_longitudinal = eye(4); % Observing all states
D_longitudinal = zeros(4, 2); % No direct feedthrough
% Create the state-space model

sys_longitudinal = ss(A_longitudinal, B_longitudinal, C_longitudinal , D_longitudinal );
tf_full_longitudinal = tf(sys_longitudinal);
x0=[u_0;w_0;q_0;theta_0];
[y_full, t, x] = initial(sys_longitudinal, x0, t);

% Transfer function from elevator deflection (De) to each state output
tf_full_longitudinal_De_with_stateu = minreal(tf_full_longitudinal(1,1)); % u state
tf_full_longitudinal_De_with_statew = minreal(tf_full_longitudinal(2,1)); % w state
tf_full_longitudinal_De_with_stateq = minreal(tf_full_longitudinal(3,1)); % q state
tf_full_longitudinal_De_with_state_theta = minreal(tf_full_longitudinal(4,1)); % theta state
% Transfer function from throttle (Th) to each state output
tf_full_longitudinal_Th_with_stateu = minreal(tf_full_longitudinal(1,2)); % u state
tf_full_longitudinal_Th_with_statew = minreal(tf_full_longitudinal(2,2)); % w state
tf_full_longitudinal_Th_with_stateq = minreal(tf_full_longitudinal(3,2)); % q state
tf_full_longitudinal_Th_with_state_theta = minreal(tf_full_longitudinal(4,2)); % theta state

%% Long Mode For Longitudinal
A_longitudinal_long = [Xu, -g*cos(theta_0); -Zu/(Zq+u_0), g*sin(theta_0)/(Zq+u_0)];
B_longitudinal_long = [Xde, Xth; -Zde/(Zq+u_0), -Zdth/(Zq+u_0)];

% Assuming the output directly maps to the state (no C and D matrices provided, use identity matrix for direct mapping)
C_longitudinal_long  = eye(2); % Direct mapping of state to output
D_longitudinal_long  = zeros(2, 2); % No direct feedthrough

% Define state-space model for the long-period longitudinal system
sys_longitudinal_long = ss(A_longitudinal_long, B_longitudinal_long, C_longitudinal_long, D_longitudinal_long);
tf_long_period = tf(sys_longitudinal_long);

% Initial state vector for long-period mode (u and theta components)
x0_long = [u_0; theta_0];

% Run the simulation for the long-period mode with external input 'u'
% 'u' should be defined as the input vector for the simulation
[y_long, t, x_long] = lsim(sys_longitudinal_long, u, t, x0_long);

```

```
% Transfer function from elevator deflection (De) to each state output
tf_long_period_De_with_state_u = minreal(tf_long_period(1,1)); % u state
tf_long_period_De_with_state_theta = minreal(tf_long_period(2,1)); % theta state

% Transfer function from throttle (Th) to each state output
tf_long_period_Th_with_state_u = minreal(tf_long_period(1,2)); % u state
tf_long_period_Th_with_state_theta = minreal(tf_long_period(2,2)); % theta state

%% Short Mode For Longitudinal
A_longitudinal_short = [
    Zw / (1 - Zwd), (Zq + u_0) / (1 - Zwd);
    Mw + (Mwd * (Zw / (1 - Zwd))), Mq + (Mwd * ((Zq + u_0) / (1 - Zwd)))
];
B_longitudinal_short = [
    Zde / (1 - Zwd), Zdth / (1 - Zwd);
    Mde + (Mwd * Zde / (1 - Zwd)), Mdth + (Mwd*Zdth / (1 - Zwd))
];
C_longitudinal_short = eye(2); % Direct mapping of state to output
D_longitudinal_short = zeros(2, 2); % No direct feedthrough

% Define state-space model for the short-period longitudinal system
sys_longitudinal_short = ss(A_longitudinal_short, B_longitudinal_short,
    C_longitudinal_short, D_longitudinal_short);
tf_short_period = tf(sys_longitudinal_short);

% Initial state vector for short-period mode ()
x0_short = [w_0; q_0]; % Corrected to match the 2-dimensional system

% Run the simulation for the short-period mode
[y_short, t, x_short] = initial(sys_longitudinal_short, x0_short, t);


% Extract and simplify transfer functions with corresponding labels
% Transfer function from elevator deflection (De) to each state output
tf_short_period_De_with_state_w = minreal(tf_short_period(1,1)); % w state
tf_short_period_De_with_state_q = minreal(tf_short_period(2,1)); % q state

% Transfer function from throttle (Th) to each state output
tf_short_period_Th_with_state_w = minreal(tf_short_period(1,2)); % w state
tf_short_period_Th_with_state_q = minreal(tf_short_period(2,2)); % q state

%% Full Lateral Mode System
A_lateral = [Yv, Yp+wo, Yr-u_0, g*cos(theta_0), 0;
             Lv, Lp, Lr, 0, 0;
             Nv, Np, Nr, 0, 0;
             0, 1, tan(theta_0), 0, 0;
             0, 0, 1/cos(theta_0), 0, 0];

B_lateral = [Yda, Ydr;
             Lda, Ldr;
             Nda, Ndr;
             0, 0;
             0, 0];

C_lateral = eye(5); % Observing all states
D_lateral = zeros(5, 2); % No direct feedthrough

u_lat = [dc(1)* ones(size(t)),1*ones(size(t))]';


% Create the state-space model for the lateral dynamics
sys_lateral = ss(A_lateral, B_lateral, C_lateral, D_lateral);
tf_full_lateral = tf(sys_lateral);

% Initial state vector for the lateral dynamics
x0_lateral = [v_0; p_0; r_0; phi_0; psi_0];  % Define initial conditions for
    lateral states


% Run the simulation for the lateral dynamics with the specified input
[y_lateral, t, x_lateral] = lsim(sys_lateral, u_lat, t, x0_lateral);
```

```
288
289 % Transfer function from aileron deflection (Da) to each state output
290 tf_full_lateral_Da_with_state_v = minreal(tf_full_lateral(1,1));
291 tf_full_lateral_Da_with_state_p = minreal(tf_full_lateral(2,1));
292 tf_full_lateral_Da_with_state_r = minreal(tf_full_lateral(3,1));
293 tf_full_lateral_Da_with_state_phi = minreal(tf_full_lateral(4,1));
294 tf_full_lateral_Da_with_state_epsi = minreal(tf_full_lateral(5,1));
295
296 % Transfer function from rudder deflection (Dr) to each state output
297 tf_full_lateral_Dr_with_state_v = minreal(tf_full_lateral(1,2));
298 tf_full_lateral_Dr_with_state_p = minreal(tf_full_lateral(2,2));
299 tf_full_lateral_Dr_with_state_r = minreal(tf_full_lateral(3,2));
300 tf_full_lateral_Dr_with_state_phi = minreal(tf_full_lateral(4,2));
301 tf_full_lateral_Dr_with_state_epsi = minreal(tf_full_lateral(5,2));
302
303 %% 3-DOF Approximations for Spiral lateral Mode
304 A_lateral_3DOF_spiral = [Lp, Lr, 0;
305                          Np, Nr, 0;
306                          1, tan(theta_0), 0];
307 % B_lateral_3DOF_spiral = [Ldr;Ndr;0];
308 B_lateral_3DOF_spiral = [Lda, Ldr;
309                          Nda, Ndr;
310                          0, 0];
311
312 % Define the output matrix to observe all states (p, r, and v)
313 C_lateral_3DOF_spiral = eye(3);  % Observing all three states
314 D_lateral_3DOF_spiral = zeros(3, 2);  % No direct feedthrough from input to
        output
315
316 % Create the state-space model for the spiral mode approximation
317 sys_lateral_3DOF_spiral = ss(A_lateral_3DOF_spiral, B_lateral_3DOF_spiral,
        C_lateral_3DOF_spiral, D_lateral_3DOF_spiral);
318 tf_3DOF_spiral = tf(sys_lateral_3DOF_spiral);
319
320 % Initial state vector for the spiral mode with 3 DOF (initial conditions for p,
        r, and v)
321 x0_spiral_3DOF = [p_0; r_0; phi_0];  % Assuming zero initial conditions
322
323 % Run the simulation for the spiral mode with 3 DOF using the specified input
324 [y_spiral_3DOF, t, x_spiral_3DOF] = lsim(sys_lateral_3DOF_spiral, u_lat, t,
        x0_spiral_3DOF);
325
326 % Transfer functions from rudder deflection (Dr) to each state output
327 tf_3DOF_spiral_Dr_with_state_p = minreal(tf_3DOF_spiral(1,1)); % Roll Rate (p)
328 tf_3DOF_spiral_Dr_with_state_r = minreal(tf_3DOF_spiral(2,1)); % Yaw Rate (r)
329 tf_3DOF_spiral_Dr_with_state_phi = minreal(tf_3DOF_spiral(3,1)); % Roll Angle (
        phi)
330
331 %% 3-DOF Approximations for Dutch roll lateral Mode
332 A_lateral_3DOF_dutch = [Yv,  Yp+wo, Yr-u_0;
333                         Lv, Lp, 0;
334                         Nv, 0, Nr];
335 B_lateral_3DOF_dutch = [Yda, Ydr;
336                         Lda, Ldr;
337                         Nda, Ndr];
338 % Output matrix to observe all states (v, p, and r)
339 C_lateral_3DOF_dutch = eye(3);  % Observing all three states
340 D_lateral_3DOF_dutch = zeros(3, 2);  % No direct feedthrough from input to output
341
342 % Create the state-space model for the Dutch Roll mode approximation
343 sys_lateral_3DOF_dutch = ss(A_lateral_3DOF_dutch, B_lateral_3DOF_dutch,
        C_lateral_3DOF_dutch, D_lateral_3DOF_dutch);
344 tf_3DOF_dutch = tf(sys_lateral_3DOF_dutch);
345
346 % Initial state vector for the Dutch Roll mode with 3 DOF (initial conditions for
         v, p, and r)
347 x0_dutch_3DOF = [v_0; p_0; r_0];  % Assuming zero initial conditions
348
349 % Run the simulation for the Dutch Roll mode with 3 DOF using the specified input
350 [y_dutch_3DOF, t, x_dutch_3DOF] = lsim(sys_lateral_3DOF_dutch, u_lat, t,
        x0_dutch_3DOF);
```

```
351
352 % Transfer functions from aileron deflection (Da) to each state output
353 tf_3DOF_dutch_Da_with_state_v = minreal(tf_3DOF_dutch(1,1)); % Side velocity (v)
354 tf_3DOF_dutch_Da_with_state_p = minreal(tf_3DOF_dutch(2,1)); % Roll angle (phi)
355 tf_3DOF_dutch_Da_with_state_r = minreal(tf_3DOF_dutch(3,1)); % Yaw rate (r)
356
357 % Transfer functions from rudder deflection (Dr) to each state output
358 tf_3DOF_dutch_Dr_with_state_v = minreal(tf_3DOF_dutch(1,2));
359 tf_3DOF_dutch_Dr_with_state_p = minreal(tf_3DOF_dutch(2,2));
360 tf_3DOF_dutch_Dr_with_state_r = minreal(tf_3DOF_dutch(3,2));
361
362 %% 2-DOF Approximations for the Dutch roll lateral Mode
363 A_lateral_2DOF_dutch = [Yv, Yr-u_0;
364                         Nv, Nr];
365 B_lateral_2DOF_dutch = [Yda, Ydr;
366                         Nda, Ndr];
367
368 % Output matrix to observe both states (v and r)
369 C_lateral_2DOF_dutch = eye(2);  % Observing both states
370 D_lateral_2DOF_dutch = zeros(2, 2);  % No direct feedthrough from input to output
371
372 % Create the state-space model for the 2-DOF Dutch Roll mode
373 sys_lateral_2DOF_dutch = ss(A_lateral_2DOF_dutch, B_lateral_2DOF_dutch,
        C_lateral_2DOF_dutch, D_lateral_2DOF_dutch);
374 tf_2DOF_dutch = tf(sys_lateral_2DOF_dutch);
375
376 % Initial state vector for the 2-DOF Dutch Roll mode (initial conditions for v
        and r)
377 x0_dutch_2DOF = [v_0;r_0];  % Assuming zero initial conditions
378
379 % Run the simulation for the Dutch Roll mode with 2 DOF using the specified input
380 [y_dutch_2DOF, t, x_dutch_2DOF] = lsim(sys_lateral_2DOF_dutch, u_lat, t,
        x0_dutch_2DOF);
381
382
383
384 % Transfer functions from aileron deflection (Da) to each state output
385 tf_2DOF_dutch_Da_with_state_v = minreal(tf_2DOF_dutch(1,1)); % Side velocity (v)
386 tf_2DOF_dutch_Da_with_state_r = minreal(tf_2DOF_dutch(2,1)); % Yaw rate (r)
387
388 % Transfer functions from rudder deflection (Dr) to each state output
389 tf_2DOF_dutch_Dr_with_state_v = minreal(tf_2DOF_dutch(1,2));
390 tf_2DOF_dutch_Dr_with_state_r = minreal(tf_2DOF_dutch(2,2));
391
392 %% 1-DOF Approximations for the Roll lateral Mode
393 A_lateral_1DOF = Lp; % Only the roll damping term Lp
394 B_lateral_1DOF = Lda; % Aileron control input affecting roll rate
395 C_lateral_1DOF = 1; % Observe roll rate (p) directly
396 D_lateral_1DOF = 0; % No direct feedthrough
397
398 sys_lateral_1DOF = ss(A_lateral_1DOF, B_lateral_1DOF, C_lateral_1DOF,
        D_lateral_1DOF);
399 % Calculate the transfer function from aileron deflection (Da) to roll rate (p)
400 tf_1DOF_roll_Da_with_state_p = minreal(tf(sys_lateral_1DOF));
401
402 % Define the input signal (u), representing aileron deflection
403 u_1dof = ones(size(t));  % Constant aileron deflection for demonstration
404
405 % Initial state vector for roll rate
406 x0_roll_1DOF = p_0;  % Assuming zero initial roll rate
407
408 % Run the simulation for the roll mode with the specified input
409 [y_roll_1DOF, t, x_roll_1DOF] = lsim(sys_lateral_1DOF, u_1dof, t, x0_roll_1DOF);
410
411
412 %% Ploting
413 % MATLAB Plotting Task 3
414 rad_to_deg = 180 / pi;  % Conversion from radians to degrees
415
416 % Plot results (e.g., velocities, angles, positions)
417 figure;
```

```
sgtitle('Matlab Code Plots');

% Plot forward, side, and vertical velocities (u, v, w)
subplot(4,3,1);
plot(t, states(1,:), 'b');  % u (forward velocity)
title('Forward Velocity (u)');
xlabel('Time (s)');
ylabel('Velocity (ft/s)');

% subplot(4,3,2);
% plot(t2, y2(2,:), 'r');  % v (side velocity)
% title('Side Velocity (v)');
% xlabel('Time (s)');
% ylabel('Velocity (ft/s)');
%
% subplot(4,3,3);
% plot(t2, y2(3,:), 'g');  % w (vertical velocity)
% title('Vertical Velocity (w)');
% xlabel('Time (s)');
% ylabel('Velocity (ft/s)');

subplot(4,3,2);
plot(t, asin(states(2,:)/Vto)*rad_to_deg, 'r');  % bita (side velocity)
title('bita');
xlabel('Time (s)');
ylabel('Velocity (ft/s)');

subplot(4,3,3);
plot(t, atan(states(3,:)./states(1,:))*rad_to_deg, 'g');  % alpha (vertical
    velocity)
title('Alpa');
xlabel('Time (s)');
ylabel('Velocity (ft/s)');

% Angular velocities (p, q, r)

subplot(4,3,4);
plot(t, states(4,:)*rad_to_deg, 'b');  % p (roll rate)
title('Roll Rate (p)');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');

subplot(4,3,5);
plot(t, states(5,:)*rad_to_deg, 'r');  % q (pitch rate)
title('Pitch Rate (q)');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');

subplot(4,3,6);
plot(t, states(6,:)*rad_to_deg, 'g');  % r (yaw rate)
title('Yaw Rate (r)');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');

% Euler angles (phi, theta, psi) in degrees

subplot(4,3,7);
plot(t, states(7,:) * rad_to_deg, 'b');  % phi (roll angle in degrees)
title('Roll Angle (phi) in degrees');
xlabel('Time (s)');
ylabel('Angle (deg)');

subplot(4,3,8);
plot(t, states(8,:) * rad_to_deg, 'r');  % theta (pitch angle in degrees)
title('Pitch Angle (theta) in degrees');
xlabel('Time (s)');
ylabel('Angle (deg)');

subplot(4,3,9);
plot(t, states(9,:) * rad_to_deg, 'g');  % psi (yaw angle in degrees)
title('Yaw Angle (psi) in degrees');
```

```
xlabel('Time (s)');
ylabel('Angle (deg)');

% Position (x, y, z) in feet

subplot(4,3,10);
plot(t, states(10,:), 'b');
title('Position X (ft)');
xlabel('Time (s)');
ylabel('Position (ft)');

subplot(4,3,11);
plot(t, states(11,:), 'r');
title('Position Y (ft)');
xlabel('Time (s)');
ylabel('Position (ft)');

subplot(4,3,12);
plot(t, round(states(12,:),3), 'g');
title('Position Z (ft)');
xlabel('Time (s)');
ylabel('Position (ft)');


% Trajectory (x vs y vs z in feet)
figure;
plot3(states(10,:), states(11,:),  round(states(12,:),3), 'b');  % 3D trajectory
    plot
title('Matlab Code Trajectory (ft)', 'FontWeight', 'bold', 'FontSize', 14);
xlabel('X (ft)', 'FontWeight', 'bold', 'FontSize', 12);
ylabel('Y (ft)', 'FontWeight', 'bold', 'FontSize', 12);
zlabel('Z (ft)', 'FontWeight', 'bold', 'FontSize', 12);
grid on;

%% Simulink Plot Task 3
% Plot results (e.g., velocities, angles, positions)
figure;
sgtitle('Simulink Code Plots');
% Plot forward, side, and vertical velocities (u, v, w)
subplot(4,3,1);
plot(t2, y2(1,:), 'b');  % u (forward velocity)
title('Forward Velocity (u)', 'FontWeight', 'bold', 'FontSize', 12);
xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
ylabel('Velocity (ft/s)', 'FontWeight', 'bold', 'FontSize', 10);

subplot(4,3,2);
plot(t2, y2(2,:), 'r');  % v (side velocity)
title('Side Velocity (v)', 'FontWeight', 'bold', 'FontSize', 12);
xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
ylabel('Velocity (ft/s)', 'FontWeight', 'bold', 'FontSize', 10);

subplot(4,3,3);
plot(t2, y2(3,:), 'g');  % w (vertical velocity)
title('Vertical Velocity (w)', 'FontWeight', 'bold', 'FontSize', 12);
xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
ylabel('Velocity (ft/s)', 'FontWeight', 'bold', 'FontSize', 10);

%
% plots of alpha and beta
% subplot(4,3,2);
% plot(t2, y2(13,:)*rad_to_deg, 'r');  % v (side angle beta)
% title('Side angle (Beta)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Beta (deg)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,3);
% plot(t2, y2(14,:)*rad_to_deg, 'g');  % w (alpha )
% title('angle of attach (Alpha)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('alpha (deg)', 'FontWeight', 'bold', 'FontSize', 10);

```

```
558
559 % Angular velocities (p, q, r)
560 subplot(4,3,4);
561 plot(t2, y2(4,:)* rad_to_deg, 'b');  % p (roll rate)
562 title('Roll Rate (p)', 'FontWeight', 'bold', 'FontSize', 12);
563 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
564 ylabel('Angular Velocity (rad/s)', 'FontWeight', 'bold', 'FontSize', 10);
565
566 subplot(4,3,5);
567 plot(t2, y2(5,:)*rad_to_deg, 'r');  % q (pitch rate)
568 title('Pitch Rate (q)', 'FontWeight', 'bold', 'FontSize', 12);
569 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
570 ylabel('Angular Velocity (rad/s)', 'FontWeight', 'bold', 'FontSize', 10);
571
572 subplot(4,3,6);
573 plot(t2, y2(6,:)*rad_to_deg, 'g');  % r (yaw rate)
574 title('Yaw Rate (r)', 'FontWeight', 'bold', 'FontSize', 12);
575 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
576 ylabel('Angular Velocity (rad/s)', 'FontWeight', 'bold', 'FontSize', 10);
577
578 % Euler angles (phi, theta, psi) in degrees
579 subplot(4,3,7);
580 plot(t2, y2(7,:) * rad_to_deg, 'b');  % phi (roll angle in degrees)
581 title('Roll Angle (phi) in degrees', 'FontWeight', 'bold', 'FontSize', 12);
582 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
583 ylabel('Angle (deg)', 'FontWeight', 'bold', 'FontSize', 10);
584
585 subplot(4,3,8);
586 plot(t2, y2(8,:) * rad_to_deg, 'r');  % theta (pitch angle in degrees)
587 title('Pitch Angle (theta) in degrees', 'FontWeight', 'bold', 'FontSize', 12);
588 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
589 ylabel('Angle (deg)', 'FontWeight', 'bold', 'FontSize', 10);
590
591 subplot(4,3,9);
592 plot(t2, unwrap(y2(9,:)) * rad_to_deg, 'g');  % psi (yaw angle in degrees)
593 title('Yaw Angle (psi) in degrees', 'FontWeight', 'bold', 'FontSize', 12);
594 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
595 ylabel('Angle (deg)', 'FontWeight', 'bold', 'FontSize', 10);
596
597 % Position (x, y, z) in feet
598 subplot(4,3,10);
599 plot(t2, y2(10,:), 'b');
600 title('Position X (ft)', 'FontWeight', 'bold', 'FontSize', 12);
601 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
602 ylabel('Position (ft)', 'FontWeight', 'bold', 'FontSize', 10);
603
604 subplot(4,3,11);
605 plot(t2, y2(11,:), 'r');
606 title('Position Y (ft)', 'FontWeight', 'bold', 'FontSize', 12);
607 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
608 ylabel('Position (ft)', 'FontWeight', 'bold', 'FontSize', 10);
609
610 subplot(4,3,12);
611 plot(t2, round(y2(12,:), 3), 'g');
612 title('Position Z (ft)', 'FontWeight', 'bold', 'FontSize', 12);
613 xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
614 ylabel('Position (ft)', 'FontWeight', 'bold', 'FontSize', 10);
615
616 % Trajectory (x vs y vs z in feet)
617 figure;
618 plot3(y2(10,:), y2(11,:), round(y2(12,:), 3), 'b');  % 3D trajectory plot
619 title('Simulink Code Trajectory (ft)', 'FontWeight', 'bold', 'FontSize', 14);
620 xlabel('X (ft)', 'FontWeight', 'bold', 'FontSize', 12);
621 ylabel('Y (ft)', 'FontWeight', 'bold', 'FontSize', 12);
622 zlabel('Z (ft)', 'FontWeight', 'bold', 'FontSize', 12);
623 grid on;
624
625 % Step response plots
626
627 %Longitudinal
628
```

```
%Compare 'u' (velocity) from full and long-period models
subplot(4,1,1);
plot(t, unwrap(y_full(:,1)+x0(1)), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
plot(t, unwrap(y_long(:,1)+x0(1)), 'b--', 'DisplayName', 'Long-Period Model');
plot(t2, unwrap(y2(1,:)), 'g--', 'DisplayName', 'Non-linear Model');
title('Response of u (velocity)'); xlabel('Time (s)'); ylabel('u (m/s)');
legend;
hold off;

%Compare 'w' (vertical speed) from full and short-period models
subplot(4,1,2);
plot(t, unwrap(y_full(:,2)+x0(2)), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
plot(t, unwrap(y_short(:,1)+x0(2)), 'b--', 'DisplayName', 'Short-Period Model');
plot(t, unwrap(states(3,:)), 'g--', 'DisplayName', 'Non-Linear Model');
title('Response of w (vertical speed)'); xlabel('Time (s)'); ylabel('w (m/s)');
legend;
hold off;

%Compare 'q' (pitch rate) from full and short-period models
subplot(4,1,3);
plot(t, unwrap(y_full(:,3)+x0(3))*(180/pi), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
plot(t, unwrap(y_short(:,2)+x0(3))*(180/pi), 'b--', 'DisplayName', 'Short-Period Model');
plot(t2, unwrap(y2(5,:))*(180/pi), 'g--', 'DisplayName', 'Non-Linear Model');
title('Response of q (pitch rate)'); xlabel('Time (s)'); ylabel('q (deg/s)');
legend;
hold off;

%Compare 'theta' (pitch angle) from full and long-period models
subplot(4,1,4);
plot(t,(unwrap(y_full(:,4)+x0(4)))*(180/pi), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
plot(t, (unwrap(y_long(:,2)+x0(4)))*(180/pi), 'b--', 'DisplayName', 'Long-Period Model');
plot(t2, (unwrap(y2(8,:)))*(180/pi), 'g--', 'DisplayName', 'Non-Linear Model');
title('Response of \theta (pitch angle)'); xlabel('Time (s)'); ylabel('\theta (deg)');
legend;
hold off;


% Compare 'Beta' sideaslip velocity from 3DOF Dutch,2DOF dutch
subplot(4,1,1);
plot(t, (y_lateral(:,1)+x0_lateral(1))*(1/Vto), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
plot(t, (y_dutch_3DOF(:,1)+ x0_dutch_3DOF(1))*(1/Vto), 'Color', '#FFA500', 'DisplayName', '3-DOF Dutch mode');
plot(t, (y_dutch_2DOF(:,1)+ x0_dutch_2DOF(1))*(1/Vto), 'm--', 'DisplayName', '2-DOF Dutch mode');
plot(t2, y2(2,:)*(1/Vto), 'g--', 'DisplayName', 'Non-linear Model');
title('Response of v '); xlabel('Time (s)'); ylabel('Beta (ft/s)');
legend;
hold off;

% Compare 'p'
subplot(4,1,2);
plot(t, y_lateral(:,2)+x0_lateral(2), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
plot(t, y_spiral_3DOF(:,1)+x0_spiral_3DOF(1), 'b--', 'DisplayName', '3-DOF Spiral Mode');
plot(t, y_dutch_3DOF(:,2)+x0_dutch_3DOF(2), 'Color', '#FFA500', 'DisplayName', '3-DOF Dutch Roll mode');
plot(t, y_roll_1DOF(:,1)+x0_roll_1DOF, 'm--', 'DisplayName', '1-DOF Dutch Roll mode');
plot(t2, y2(4,:), 'g--', 'DisplayName', 'Non-linear Model');
title('Response of p'); xlabel('Time (s)'); ylabel('p (rad/s)');
legend;
hold off;
```

```
686
687 % Compare 'r'
688 subplot(4,1,3);
689 plot(t, y_lateral(:,3)+x0_lateral(3), 'r', 'DisplayName', 'Fully linearized Model
        '); hold on;
690 plot(t, y_spiral_3DOF(:,2)+x0_spiral_3DOF(2), 'b--', 'DisplayName', '3-DOF
        Spiralmode');
691 plot(t, y_dutch_3DOF(:,3)+x0_dutch_3DOF(3), 'Color', '#FFA500', 'DisplayName', '
        3-DOF Dutch Roll mode');
692 plot(t, y_dutch_2DOF(:,2)+ x0_dutch_2DOF(2), 'm--', 'DisplayName', '2-DOF mode');
693 plot(t2, y2(6,:), 'g--', 'DisplayName', 'Non-linear Model');
694 title('Response of r'); xlabel('Time (s)'); ylabel('r (rad/s)');
695 legend;
696 hold off;
697
698 % Compare 'phi'
699 subplot(4,1,4);
700 plot(t, y_lateral(:,4)+x0_lateral(4), 'r', 'DisplayName', 'Fully linearized Model
        '); hold on;
701 plot(t, y_spiral_3DOF(:,3)+x0_spiral_3DOF(3), 'b--', 'DisplayName', '3-DOF Spiral
         mode');
702 plot(t2, y2(7,:), 'g--', 'DisplayName', 'Non-linear Model');
703 title('Response of phi'); xlabel('Time (s)'); ylabel('\phi (rad)');
704 legend;
705 hold off;
706
707
708 % Compare 'epsi'
709 figure;
710 plot(t, y_lateral(:,5)++x0_lateral(5), 'r', 'DisplayName', 'Fully linearized
        Model'); hold on;
711 plot(t2, y2(9,:), 'g--', 'DisplayName', 'Non-linear Model');
712 title('Response of epsi'); xlabel('Time (s)'); ylabel('epsi (rad)');
713 legend;
714 hold off;
715
716
717 %% Ploting Rotloucs and Bodeplot For Longitudinal
718 % Call the function for each transfer function comparison between full and short
        modes
719
720 % Full longitudinal vs short-period
721 plotAndSaveSubplots(tf_full_longitudinal_De_with_statew,
        tf_short_period_De_with_state_w, 'Short', 'De', 'w');
722 plotAndSaveSubplots(tf_full_longitudinal_De_with_stateq,
        tf_short_period_De_with_state_q, 'Short', 'De', 'q');
723 plotAndSaveSubplots(tf_full_longitudinal_Th_with_statew,
        tf_short_period_Th_with_state_w, 'Short', 'Th', 'w');
724 plotAndSaveSubplots(tf_full_longitudinal_Th_with_stateq,
        tf_short_period_Th_with_state_q, 'Short', 'Th', 'q');
725
726 % Full longitudinal vs long-period
727 plotAndSaveSubplots(tf_full_longitudinal_De_with_stateu,
        tf_long_period_De_with_state_u, 'Long', 'De', 'u');
728 plotAndSaveSubplots(tf_full_longitudinal_De_with_state_theta,
        tf_long_period_De_with_state_theta, 'Long', 'De', 'theta');
729 plotAndSaveSubplots(tf_full_longitudinal_Th_with_stateu,
        tf_long_period_Th_with_state_u, 'Long', 'Th', 'u');
730 plotAndSaveSubplots(tf_full_longitudinal_Th_with_state_theta,
        tf_long_period_Th_with_state_theta, 'Long', 'Th', 'theta');
731
732
733
734 %% Plot and save Root Locus and Bode plots for each mode
735
736 % Define the transfer functions and labels for comparisons
737
738 % Aileron Deflection (Da) Comparisons
739 Da_v = {tf_full_lateral_Da_with_state_v, tf_3DOF_dutch_Da_with_state_v,
        tf_2DOF_dutch_Da_with_state_v};
740 Da_p = {tf_full_lateral_Da_with_state_p, tf_3DOF_dutch_Da_with_state_p,
```

```
    tf_1DOF_roll_Da_with_state_p};
Da_r = {tf_full_lateral_Da_with_state_r, tf_3DOF_dutch_Da_with_state_r,
    tf_2DOF_dutch_Da_with_state_r};
Da_phi = {tf_full_lateral_Da_with_state_phi};
Da_epsi = {tf_full_lateral_Da_with_state_epsi};

% Rudder Deflection (Dr) Comparisons
Dr_v = {tf_full_lateral_Dr_with_state_v, tf_3DOF_dutch_Dr_with_state_v,
    tf_2DOF_dutch_Dr_with_state_v};
Dr_p = {tf_full_lateral_Dr_with_state_p, tf_3DOF_spiral_Dr_with_state_p,
    tf_3DOF_dutch_Dr_with_state_p};
Dr_r = {tf_full_lateral_Dr_with_state_r, tf_3DOF_spiral_Dr_with_state_r,
    tf_3DOF_dutch_Dr_with_state_r, tf_2DOF_dutch_Dr_with_state_r};
Dr_phi = {tf_full_lateral_Dr_with_state_phi,tf_3DOF_spiral_Dr_with_state_phi};
Dr_epsi = {tf_full_lateral_Dr_with_state_epsi};


% Mode labels for Aileron Deflection (Da) Comparisons
mode_labels_Da_v = {'Full Mode', '3-DOF Dutch Roll', '2-DOF Dutch Roll'};
mode_labels_Da_p = {'Full Mode','3-DOF Dutch Roll','1-DOF Roll'};
mode_labels_Da_r = {'Full Mode', '3-DOF Dutch Roll', '2-DOF Dutch Roll'};
mode_labels_Da_phi = {'Full Mode'};
mode_labels_Da_epsi = {'Full Mode'};

% Mode labels for Rudder Deflection (Dr) Comparisons
mode_labels_Dr_v = {'Full Mode', '3-DOF Dutch Roll', '2-DOF Dutch Roll'};
mode_labels_Dr_p = {'Full Mode', '3-DOF Spiral', '3-DOF Dutch Roll'};
mode_labels_Dr_r = {'Full Mode', '3-DOF Spiral', '3-DOF Dutch Roll', '2-DOF Dutch
     Roll'};
mode_labels_Dr_phi = {'Full Mode', '3-DOF Spiral'};
mode_labels_Dr_epsi = {'Full Mode'};

%Call the function to compare each mode for specific input-output pairs

% Aileron Deflection (Da) Comparisons
plotAndCompareModesWithSubplots(Da_v, mode_labels_Da_v, 'Da', 'v');
plotAndCompareModesWithSubplots(Da_p, mode_labels_Da_p, 'Da', 'p');
plotAndCompareModesWithSubplots(Da_r, mode_labels_Da_r, 'Da', 'r');
plotAndCompareModesWithSubplots(Da_phi, mode_labels_Da_phi, 'Da', 'phi');
plotAndCompareModesWithSubplots(Da_epsi, mode_labels_Da_epsi, 'Da', 'epsi');

% Rudder Deflection (Dr) Comparisons
plotAndCompareModesWithSubplots(Dr_v, mode_labels_Dr_v, 'Dr', 'v');
plotAndCompareModesWithSubplots(Dr_p, mode_labels_Dr_p, 'Dr', 'p');
plotAndCompareModesWithSubplots(Dr_r, mode_labels_Dr_r, 'Dr', 'r');
plotAndCompareModesWithSubplots(Dr_phi, mode_labels_Dr_phi, 'Dr', 'phi');
plotAndCompareModesWithSubplots(Dr_epsi, mode_labels_Dr_epsi, 'Dr', 'epsi');

%% Display Tranfer Function For Longitudinal
% Call the function below for each input-output pair comparison

% Full longitudinal vs short-period
displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_statew,
    tf_short_period_De_with_state_w, 'Short', 'De', 'w');
displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_stateq,
    tf_short_period_De_with_state_q, 'Short', 'De', 'q');
displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_statew,
    tf_short_period_De_with_state_w, 'Short', 'Th', 'w');
displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_stateq,
    tf_short_period_De_with_state_q, 'Short', 'Th', 'q');

% Full longitudinal vs long-period
displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_stateu,
    tf_long_period_De_with_state_u, 'Long', 'De', 'u');
displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_state_theta,
    tf_long_period_De_with_state_theta, 'Long', 'De', 'theta');
displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_stateu,
    tf_long_period_Th_with_state_u, 'Long', 'Th', 'u');
displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_state_theta,
    tf_long_period_Th_with_state_theta, 'Long', 'Th', 'theta');

```

```
%% Display the transfer functions in fraction format
% Call the function below for each group of transfer functions

% Aileron Deflection (Da) Comparisons
displayTransferFunctionGroup(Da_v, mode_labels_Da_v, 'Da', 'v');
displayTransferFunctionGroup(Da_p, mode_labels_Da_p, 'Da', 'p');
displayTransferFunctionGroup(Da_r, mode_labels_Da_r, 'Da', 'r');
displayTransferFunctionGroup(Da_phi, mode_labels_Da_phi, 'Da', 'phi');
displayTransferFunctionGroup(Da_epsi, mode_labels_Da_epsi, 'Da', 'epsi');

% Rudder Deflection (Dr) Comparisons
displayTransferFunctionGroup(Dr_v, mode_labels_Dr_v, 'Dr', 'v');
displayTransferFunctionGroup(Dr_p, mode_labels_Dr_p, 'Dr', 'p');
displayTransferFunctionGroup(Dr_r, mode_labels_Dr_r, 'Dr', 'r');
displayTransferFunctionGroup(Dr_phi, mode_labels_Dr_phi, 'Dr', 'phi');
displayTransferFunctionGroup(Dr_epsi, mode_labels_Dr_epsi, 'Dr', 'epsi');

%% Functions

% Define a function to display the transfer function expressions as a large
    fraction
function displayTransferFunctionAsFraction(tf_full, tf_mode, mode_name,
    input_name, output_name)
    % Extract numerator and denominator for the full mode transfer function
    [num_full, den_full] = tfdata(tf_full, 'v'); % Get numerator and denominator
    as vectors
    % Convert numerator and denominator to strings
    num_full_str = poly2str(num_full, 's');
    den_full_str = poly2str(den_full, 's');

    % Extract numerator and denominator for the mode transfer function (short/
    long)
    [num_mode, den_mode] = tfdata(tf_mode, 'v'); % Get numerator and denominator
    as vectors
    % Convert numerator and denominator to strings
    num_mode_str = poly2str(num_mode, 's');
    den_mode_str = poly2str(den_mode, 's');

    % Display the formatted transfer functions as a large fraction
    fprintf('Comparison for Transfer Function (%s to %s):\n', input_name,
    output_name);
    fprintf('-------------------------------------------------------\n');

    % Display the full mode transfer function in S domain format as a large
    fraction
    fprintf('Full Mode Transfer Function:\n');
    fprintf('    %s\n', num_full_str);  % Numerator
    fprintf('    %s\n', repmat('-', max(length(num_full_str), length(den_full_str
    )), 1)); % Divider line
    fprintf('    %s\n\n', den_full_str); % Denominator

    % Display the alternative mode (short or long) transfer function as a large
    fraction
    fprintf('%s Mode Transfer Function:\n', mode_name);
    fprintf('    %s\n', num_mode_str);  % Numerator
    fprintf('    %s\n', repmat('-', max(length(num_mode_str), length(den_mode_str
    )), 1)); % Divider line
    fprintf('    %s\n\n', den_mode_str); % Denominator

    % Add a line break for readability between comparisons
    fprintf('\n\n');
end

% Define function to display the transfer functions for a group in fraction
    format
function displayTransferFunctionGroup(tf_list, mode_labels, input_name,
    output_name)
    num_modes = length(tf_list);
    fprintf('Transfer Function Comparisons for %s to %s:\n', input_name,
    output_name);
    fprintf('-------------------------------------------------------\n');
```

```
856
857     for k = 1:num_modes
858         % Extract numerator and denominator
859         [num, den] = tfdata(tf_list{k}, 'v');
860         num_str = poly2str(num, 's');
861         den_str = poly2str(den, 's');
862
863         % Display transfer function for each mode
864         fprintf('%s Mode Transfer Function:\n', mode_labels{k});
865         fprintf('    %s\n', num_str);  % Numerator
866         fprintf('    %s\n', repmat('-', max(length(num_str), length(den_str)), 1)
    ); % Divider line
867         fprintf('    %s\n\n', den_str); % Denominator
868     end
869     fprintf('\n\n'); % Add extra line spacing between groups for readability
870 end
871
872 % Define function to plot and save Root Locus and Bode plots with subplots for
    each mode comparison
873 function plotAndCompareModesWithSubplots(tf_list, mode_labels, input_name,
    output_name)
874     num_modes = length(tf_list);
875
876     % Plot Root Locus with each mode in its own subplot
877     figure;
878     set(gcf, 'WindowState', 'maximized'); % Maximize the figure window
879     for k = 1:num_modes
880         subplot(1, num_modes, k); % Create a subplot for each mode
881         rlocus(tf_list{k});
882         title(['Root Locus Comparison - ', mode_labels{k}, ' (', input_name, ' to
     ', output_name, ')']);
883         xlabel('Real Axis');
884         ylabel('Imaginary Axis');
885     end
886     sgtitle(['Root Locus - ', input_name, ' to ', output_name]);
887     saveas(gcf, ['RootLocus_Comparison_', input_name, '_to_', output_name, '.png'
    ]);
888
889     % Plot Bode Plot with each mode in the same plot for overlay comparison
890     figure;
891     set(gcf, 'WindowState', 'maximized'); % Maximize the figure window
892     hold on;
893     for k = 1:num_modes
894         bode(tf_list{k});
895     end
896     hold off;
897     title(['Bode Plot Comparison - ', input_name, ' to ', output_name]);
898     legend(mode_labels);
899     saveas(gcf, ['BodePlot_Comparison_', input_name, '_to_', output_name, '.png'
    ]);
900 end
901
902 % Define a function to plot and save root locus and Bode plots with subplots for
    each transfer function
903 function plotAndSaveSubplots(tf_full, tf_short, mode_name, input_name,
    output_name)
904     % Create figure for Root Locus with two subplots
905     figure;
906
907     % Full Mode Root Locus in the first subplot
908     subplot(1, 2, 1);
909     rlocus(tf_full);
910     title(['Root Locus - Full Mode (', input_name, ' to ', output_name, ')']);
911     xlabel('Real Axis');
912     ylabel('Imaginary Axis');
913
914     % Short Mode Root Locus in the second subplot
915     subplot(1, 2, 2);
916     rlocus(tf_short);
917     title(['Root Locus - ', mode_name, ' Mode (', input_name, ' to ', output_name
    , ')']);
```

```
    xlabel('Real Axis');
    ylabel('Imaginary Axis');

    % Save the combined root locus figure
    saveas(gcf, ['RootLocus_', input_name, '_to_', output_name, '_Full_vs_',
    mode_name, '.png']);

     % Bode plot (Separate for Full and Short Mode)
    figure;
    bode(tf_full);
    hold on;
    bode(tf_short);
    hold off;
    title(['Bode Plot - ', input_name, ' to ', output_name, ' (Full vs ',
    mode_name, ' Mode)']);
    legend('Full Mode', [mode_name, ' Mode']);
    saveas(gcf, ['BodePlot_', input_name, '_to_', output_name, '_', mode_name, '.
    png']);
end
```

9.1.8 MATLAB Code for Autopilot Longitudinal Control

```
% Main Script: Airplane Nonlinear Simulator with RBD Solver

% Clear workspace
clc;
clear all;
close all;
global m
global I
global F M
global g
%% Load Aircraft Data
%read from Execel File
% filename_density_L = 'ourflightcondation';  % Path to your excel sheet
% aircraft_data = xlsread(filename_density_L, 'B2:B61');  % Read from Excel

% read Data from CSV
filename_density_L = 'ourflightcondation';  % Path to your CSV file
aircraft_data = readmatrix(filename_density_L, 'Range', 'B2:B61');  % Read from
    CSV

% Time vector parameters
dt = aircraft_data(1);
tfinal = aircraft_data(2);
t0 = aircraft_data(3);


% Initial conditions for states
s0 = aircraft_data(4:15);
sdot0 = zeros(12,1);  % Initial derivative of states
Vto = sqrt(s0(1)^2 + s0(2)^2 + s0(3)^2);  % Compute initial total velocity (Vto)
initialstatemodified=[s0(1:3);0;s0(4:6)];
% Control actions values
dc = [ aircraft_data(57:59) * pi/180 ; aircraft_data(60) ];  % Control inputs in
    radians

% Gravity, mass, and inertia values from aircraft data

m = 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;

% Inertia matrix and inverse inertia matrix

I = [Ixx, -Ixy, -Ixz; -Ixy, Iyy, -Iyz; -Ixz, -Iyz, Izz];
invI = inv(I);
```

```
48
49 D2R=pi/180;
50 R2D=180/pi;
51
52 %Initial velocities and rates
53 u_0 = s0(1);
54 v_0 = s0(2);
55 w_0 = s0(3);
56 p_0 = s0(4);
57 q_0 = s0(5);
58 r_0 = s0(6);
59 phi_0 = s0(7);
60 theta_0 = s0(8);
61 psi_0 = s0(9);
62 x_0 = s0(10);
63 y_0 = s0(11);
64 z_0 = s0(12);
65 wdot_0 = 0;
66 wdot = 0;
67 mass = m;
68
69 % Stability derivatives for longitudinal motion
70 SD_Long = aircraft_data(21:36);
71 SD_Long_final = SD_Long;
72 templong  =num2cell(SD_Long);
73 [Xu,Zu,Mu,Xw,Zw,Mw,Zwd,Zq,Mwd,Mq,Xde,Zde,Mde,Xth,Zdth,Mdth] = deal(templong{:});
74 clear templong;
75
76
77 % Stability derivatives for lateral motion
78 SD_Lat_dash = aircraft_data(37:50);
79 SD_Lat = LateralFunc(SD_Lat_dash, Ixz, Izz, Ixx, Vto);
80 SD_Lat_final = SD_Lat;
81 templatrel = num2cell(SD_Lat_dash);
82 [Yv,Yb,Lb,Nb,Lp,Np,Lr,Nr,Yda,Ydr,Lda,Nda,Ldr,Ndr] = deal(templatrel{:});
83 clear templatrel;
84
85 Lv=Lb/Vto;
86 Nv=Nb/Vto;
87 Yp=0;
88 Ydr=Ydr*Vto;
89
90 servo= tf(10,[1 10]);
91 integrator=tf(1,[1 0]);
92 diffrentiator=tf([1 0],1);
93 engine_timelag=tf(0.1,[1 0 1]);
94
95
96 %% Full Longitudinal System
97
98 % Define constants
99 theta_0 = s0(8); wo = s0(3);u_0 = s0(1);wdot=0;
100 long0=[u_0;w_0;q_0;theta_0];
101
102
103 A_longitudinal = [Xu, Xw,-w_0, -g*cos(theta_0);
104                   Zu/(1-Zwd), Zw/(1-Zwd), (Zq + u_0)/(1-Zwd), -g*sin(theta_0)/(1-
    Zwd);
105                    Mu + (Zu * Mwd) / (1 - Zwd), Mw + (Zw * Mwd) / (1 - Zwd), Mq +
    ((Zq + u_0) * Mwd) / (1 - Zwd), - (g * Mwd * sin(theta_0)) / (1 - Zwd);
106                   0, 0, 1, 0];
107 B_longitudinal = [Xde, Xth;
108                   Zde/(1-Zwd), Zdth/(1-Zwd);
109                   Mde + Mwd*Zde/(1-Zwd), Mdth + Mwd*Zdth/(1-Zwd);
110                   0, 0];
111
112 C_longitudinal = eye(4); % Observing all states
113 D_longitudinal = zeros(4, 2); % No direct feedthrough
114 sys_longitudinal = ss(A_longitudinal, B_longitudinal, C_longitudinal ,
    D_longitudinal );
115 tf_full_longitudinal = tf(sys_longitudinal);
```

```

% Transfer function from elevator deflection (De) to each state output
De_u = minreal(tf_full_longitudinal(1,1)); % u state
De_w = minreal(tf_full_longitudinal(2,1)); % w state
De_q = minreal(tf_full_longitudinal(3,1)); % q state
De_theta = minreal(tf_full_longitudinal(4,1)); % theta state
De_Wd=De_w*diffrentiator;
De_az=De_Wd-u_0*De_q;
Theta_az=De_az/De_theta;
De_alpha=De_w/u_0;
De_gamma=De_theta-De_alpha;

% Transfer function from throttle (Th) to each state output
Th_u = minreal(tf_full_longitudinal(1,2)); % u state
Th_w = minreal(tf_full_longitudinal(2,2)); % w state
Th_q = minreal(tf_full_longitudinal(3,2)); % q state
Th_theta = minreal(tf_full_longitudinal(4,2)); % theta state
u_dT=tf_full_longitudinal(1,2);


%pitch control

ol_theta_thetacomm=-servo*De_theta;

% Velocity Control
OL_u_ucom=-servo*engine_timelag*u_dT;
```

9.1.9 MATLAB Code for Autopilot Lateral Control

```
% Main Script: Airplane Nonlinear Simulator with RBD Solver

% Clear workspace
clc;
close all;
global m
global I
global F M
global g
%% Load Aircraft Data
%read from Execel File
% filename_density_L = 'ourflightcondation';  % Path to your excel sheet
% aircraft_data = xlsread(filename_density_L, 'B2:B61');  % Read from Excel

% read Data from CSV
filename_density_L = 'ourflightcondation';  % Path to your CSV file
aircraft_data = readmatrix(filename_density_L, 'Range', 'B2:B61');  % Read from
    CSV

% Time vector parameters
dt = aircraft_data(1);
tfinal = aircraft_data(2);
t0 = aircraft_data(3);


% Initial conditions for states
s0 = aircraft_data(4:15);
sdot0 = zeros(12,1);  % Initial derivative of states
Vto = sqrt(s0(1)^2 + s0(2)^2 + s0(3)^2);  % Compute initial total velocity (Vto)
initialstatemodified=[s0(1:3);0;s0(4:6)];
% Control actions values
dc = [ aircraft_data(57:59) * pi/180 ; aircraft_data(60) ];  % Control inputs in
    radians

% Gravity, mass, and inertia values from aircraft data

m = 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;

% Inertia matrix and inverse inertia matrix

I = [Ixx, -Ixy, -Ixz; -Ixy, Iyy, -Iyz; -Ixz, -Iyz, Izz];
invI = inv(I);

D2R=pi/180;
R2D=180/pi;

%Initial velocities and rates
u_0 = s0(1);
v_0 = s0(2);
w_0 = s0(3);
p_0 = s0(4);
q_0 = s0(5);
r_0 = s0(6);
phi_0 = s0(7);
theta_0 = s0(8);
psi_0 = s0(9);
x_0 = s0(10);
y_0 = s0(11);
z_0 = s0(12);
wdot_0 = 0;
wdot = 0;
mass = m;

% Stability derivatives for longitudinal motion
SD_Long = aircraft_data(21:36);
SD_Long_final = SD_Long;
templong  =num2cell(SD_Long);
[Xu,Zu,Mu,Xw,Zw,Mw,Zwd,Zq,Mwd,Mq,Xde,Zde,Mde,Xth,Zdth,Mdth] = deal(templong{:});
clear templong;


% Stability derivatives for lateral motion
SD_Lat_dash = aircraft_data(37:50);
templatrel = num2cell(SD_Lat_dash);
[Yv,Yb,Lb,Nb,Lp,Np,Lr,Nr,Yda,Ydr,Lda,Nda,Ldr,Ndr] = deal(templatrel{:});
clear templatrel;

Lv=Lb/Vto;
Nv=Nb/Vto;
Yp=0;
Yr=0;
Ydr=Ydr*Vto;

servo= tf(10,[1 10]);
integrator=tf(1,[1 0]);
diffrentiator=tf([1 0],1);
engine_timelag=tf(0.1,[1 0 1]);


%% Full Longitudinal System

% Define constants
theta_0 = s0(8); wo = s0(3);u_0 = s0(1);wdot=0;


A_longitudinal = [Xu, Xw,-w_0, -g*cos(theta_0);
                  Zu/(1-Zwd), Zw/(1-Zwd), (Zq + u_0)/(1-Zwd), -g*sin(theta_0)/(1-
    Zwd);
                   Mu + (Zu * Mwd) / (1 - Zwd), Mw + (Zw * Mwd) / (1 - Zwd), Mq +
    ((Zq + u_0) * Mwd) / (1 - Zwd), - (g * Mwd * sin(theta_0)) / (1 - Zwd);
                  0, 0, 1, 0];
B_longitudinal = [Xde, Xth;
                  Zde/(1-Zwd), Zdth/(1-Zwd);
                  Mde + Mwd*Zde/(1-Zwd), Mdth + Mwd*Zdth/(1-Zwd);
                  0, 0];

```

```
C_longitudinal = eye(4); % Observing all states
D_longitudinal = zeros(4, 2); % No direct feedthrough
sys_longitudinal = ss(A_longitudinal, B_longitudinal, C_longitudinal ,
    D_longitudinal );
tf_full_longitudinal = tf(sys_longitudinal);


% Transfer function from elevator deflection (De) to each state output
De_u = minreal(tf_full_longitudinal(1,1)); % u state
De_w = minreal(tf_full_longitudinal(2,1)); % w state
De_q = minreal(tf_full_longitudinal(3,1)); % q state
De_theta = minreal(tf_full_longitudinal(4,1)); % theta state
De_Wd=De_w*diffrentiator;
De_az=De_Wd-u_0*De_q;
Theta_az=De_az/De_theta;
De_alpha=De_w/u_0;
De_gamma=De_theta-De_alpha;

% Transfer function from throttle (Th) to each state output
Th_u = minreal(tf_full_longitudinal(1,2)); % u state
Th_w = minreal(tf_full_longitudinal(2,2)); % w state
Th_q = minreal(tf_full_longitudinal(3,2)); % q state
Th_theta = minreal(tf_full_longitudinal(4,2)); % theta state
u_dT=tf_full_longitudinal(1,2);


%pitch control

ol_theta_thetacomm=-servo*De_theta;

% Velocity Control
OL_u_ucom=-servo*engine_timelag*u_dT;

%% Full Lateral Mode System
x0_lateral = [v_0; p_0; r_0; phi_0; psi_0];


A_lateral = [Yv, Yp+wo, Yr-u_0, g*cos(theta_0), 0;
             Lv, Lp, Lr, 0, 0;
             Nv, Np, Nr, 0, 0;
             0, 1, tan(theta_0), 0, 0;
             0, 0, 1/cos(theta_0), 0, 0];

B_lateral = [Yda, Ydr;
             Lda, Ldr;
             Nda, Ndr;
             0, 0;
             0, 0];

C_lateral = eye(5); % Observing all states
D_lateral = zeros(5, 2); % No direct feedthrough

% Create the state-space model for the lateral dynamics
sys_lateral = ss(A_lateral, B_lateral, C_lateral, D_lateral);
tf_full_lateral = tf(sys_lateral);

v_da=tf_full_lateral(1,1);
p_da=tf_full_lateral(2,1);
r_da=tf_full_lateral(3,1);
phi_da=tf_full_lateral(4,1);
psi_da=tf_full_lateral(5,1);
beta_da=v_da/u_0;



v_dr=tf_full_lateral(1,2);
p_dr=tf_full_lateral(2,2);
r_dr=tf_full_lateral(3,2);
phi_dr=tf_full_lateral(4,2);
psi_dr=tf_full_lateral(5,2);
beta_dr=v_da/u_0;
```

```
179
180 ol_r_rcomm=servo*r_dr;
181 load('Yaw_Damper.mat')
182
183 Lat_YawDamper_ss_series=feedback(series(append(1,servo),sys_lateral,[1 2],[1 2]),
    HPF_YawDamper,2,3,1);
184 Lat_YawDamper_tf_series=tf(Lat_YawDamper_ss_series);
185 phi_da_AfterYawDamper=Lat_YawDamper_tf_series(4,1);
186 ol_phi_phicomm=minreal(servo*phi_da_AfterYawDamper);
```

9.1.10 MATLAB Code for Simulation to Complete Linear and Non-Linear model

```
1  % Main Script
2  clc;
3  %clear all;
4  %close all;
5  global m
6  global I
7  global F M
8  global g
9
10 %% Load Aircraft Data from CSV and assigning them to variables
11 filename_density_L = 'ourflightcondation.csv';
12 aircraft_data = readmatrix(filename_density_L, 'Range', 'B2:B61');
13
14 % Time vector parameters
15 dt = aircraft_data(1);
16 tfinal = aircraft_data(2);
17 time_V = (0:dt:tfinal)';
18
19 % Initial conditions for states
20 s0 = aircraft_data(4:15);
21 sdot0 = zeros(12,1);  % Initial derivative of states
22 Vto = sqrt(s0(1)^2 + s0(2)^2 + s0(3)^2);  % Compute initial total velocity (Vto)
23 initialstatemodified=[s0(1:3);0;s0(4:6)];
24
25 % Control actions values
26 dc = [ aircraft_data(57:59) * pi/180 ; aircraft_data(60) ];  % Control inputs in
    radians
27
28 % Gravity, mass, and inertia values from aircraft data
29 m = aircraft_data(51);
30 g = aircraft_data(52);
31 Ixx = aircraft_data(53);
32 Iyy = aircraft_data(54);
33 Izz = aircraft_data(55);
34 Ixz = aircraft_data(56);
35 Ixy = 0;  Iyz = 0;
36
37 % Inertia matrix and inverse inertia matrix
38 I = [Ixx, -Ixy, -Ixz; -Ixy, Iyy, -Iyz; -Ixz, -Iyz, Izz];
39 invI = inv(I);
40
41 % Angle rad to deg and vice versa
42 D2R=pi/180;
43 R2D=180/pi;
44
45 % Stability derivatives for longitudinal motion
46 SD_Long = aircraft_data(21:36);
47 SD_Long_final = SD_Long;
48 templong  =num2cell(SD_Long);
49 [Xu,Zu,Mu,Xw,Zw,Mw,Zwd,Zq,Mwd,Mq,Xde,Zde,Mde,Xth,Zdth,Mdth] = deal(templong{:});
50 clear templong;
51
52 % Stability derivatives for lateral motion
53 SD_Lat_dash = aircraft_data(37:50);
54 SD_Lat = LateralFunc(SD_Lat_dash, Ixz, Izz, Ixx, Vto);
55 SD_Lat_final = SD_Lat;
56 templatrel = num2cell(SD_Lat);
57 [Yv,Yb,Lb,Nb,Lp,Np,Lr,Nr,Yda,Ydr,Lda,Nda,Ldr,Ndr] = deal(templatrel{:});
58 clear templatrel;
```

```
59
60 Lv=Lb/Vto;
61 Nv=Nb/Vto;
62 Yp=0; % not found in Nasa data
63 Yr=0; % not found in nasa data
64
65 % Initial gravity force in body frame
66 mg0 = m * g * [sin(s0(8)); -cos(s0(8)) * sin(s0(7)); -cos(s0(8)) * cos(s0(7))];
67 initial_F_M = m * g * [sin(s0(8)); -cos(s0(8)) * sin(s0(7)); -cos(s0(8)) * cos(s0
       (7));0;0;0]; % Simulink Input
68 M_0 = [0;0;0];
69
70 %Initial velocities and rates
71 u_0 = s0(1);
72 v_0 = s0(2);
73 w_0 = s0(3);
74 p_0 = s0(4);
75 q_0 = s0(5);
76 r_0 = s0(6);
77 phi_0 = s0(7);
78 theta_0 = s0(8);
79 psi_0 = s0(9);
80 x_0 = s0(10);
81 y_0 = s0(11);
82 z_0 = s0(12);
83 wdot_0 = 0;
84 wdot = 0;
85 mass = m;
86
87 %% Matlab Nonlinear Model
88 for i = 1:16
89     SD(i) = SD_Long_final(i);
90 end
91 for j = 1:14
92     SD(j+16) = SD_Lat_final(j);  % All stability derivatives in one array
93 end
94
95 n = 1000;
96 velocity_0=s0(1:3);
97 angular_v0=s0(4:6);
98 angles_0=s0(7:9);
99 position_0=s0(10:12);
100 Initial_state=[velocity_0;angular_v0;angles_0;position_0];
101
102 % solve and get the state and data
103 [F,M]=F_M_Cal(SD,s0,Vto,dc,Initial_state,0);
104 [t, states] = RK4_solver(@getstates,Initial_state , 0, tfinal,n, SD,Vto, dc);
105
106 %% Simulink Nonlinear Model
107
108 State_Matrix = [ Xu  0   Xw  0   0   0   0;
109                  0   Yv  0   Yp  0   Yr  0;
110                  Zu  0   Zw  0   Zq  Lr  Zwd;
111                  0   Lv  0   Lp  0   Lr  0;
112                  Mu  0   Mw  0   Mq  0  Mwd;
113                  0   Nv  0   Np  0   Nr 0];
114
115 Control_Matrix = [0 Xde Xth 0
116                   Yda 0 0 Ydr
117                   0  Zde Zdth 0
118                   Lda 0  0  Ldr
119                   0  Mde Mdth 0
120                   Nda 0  0 Ndr];
121
122 % intial struc state
123 so_struct.x = s0(10);
124 so_struct.y = s0(11);
125 so_struct.H = -1*s0(12);
126 so_struct.phi = s0(7);
127 so_struct.theta = s0(8);
128 so_struct.psi = s0(9);
```

```
so_struct.p = s0(4);
so_struct.q = s0(5);
so_struct.r = s0(6);
so_struct.u = s0(1);
so_struct.v = s0(2);
so_struct.w = s0(3);
so_struct.alpha = s0(8);
so_struct.Beta = 0;
so_struct.V_total = Vto;
so_struct.W_dot = 0;
so_struct.gamma=0;

% % run the model and get data
% modelName = 'Final_Task3_Simulink';
% load_system(modelName);
% set_param(modelName, 'StartTime', '0', 'StopTime', num2str(tfinal));
% simOut = sim(modelName, 'ReturnWorkspaceOutputs', 'on');
%
% t2 = simOut.tout;
% y2 = simOut.simout.Data;
% y2 = y2';

%% Linear Model
x0=[u_0;w_0;q_0;theta_0]; %input
%% Full Longitudinal System
% Define constants
theta_0 = s0(8); wo = s0(3);u_0 = s0(1);wdot=0;

% Define constants and matrices for the longitudinal state-space model
u = [dc(3)* ones(size(t)); (dc(4)*ones(size(t)))];

A_longitudinal = [Xu, Xw,-w_0, -g*cos(theta_0);
                  Zu/(1-Zwd), Zw/(1-Zwd), (Zq + u_0)/(1-Zwd), -g*sin(theta_0)/(1-
    Zwd);
                   Mu + (Zu * Mwd) / (1 - Zwd), Mw + (Zw * Mwd) / (1 - Zwd), Mq +
    ((Zq + u_0) * Mwd) / (1 - Zwd), - (g * Mwd * sin(theta_0)) / (1 - Zwd);
                  0, 0, 1, 0];
B_longitudinal = [Xde, Xth;
                  Zde/(1-Zwd), Zdth/(1-Zwd);
                  Mde + Mwd*Zde/(1-Zwd), Mdth + Mwd*Zdth/(1-Zwd);
                  0, 0];
C_longitudinal = eye(4); % Observing all states
D_longitudinal = zeros(4, 2); % No direct feedthrough
sys_longitudinal = ss(A_longitudinal, B_longitudinal, C_longitudinal ,
    D_longitudinal );
tf_full_longitudinal = tf(sys_longitudinal);

[y_full, t, x] = lsim(sys_longitudinal,u, t);

% Transfer function from elevator deflection (De) to each state output
tf_full_longitudinal_De_with_stateu = minreal(tf_full_longitudinal(1,1)); % u
    state
tf_full_longitudinal_De_with_statew = minreal(tf_full_longitudinal(2,1)); % w
    state
tf_full_longitudinal_De_with_stateq = minreal(tf_full_longitudinal(3,1)); % q
    state
tf_full_longitudinal_De_with_state_theta = minreal(tf_full_longitudinal(4,1)); %
    theta state
% Transfer function from throttle (Th) to each state output
tf_full_longitudinal_Th_with_stateu = minreal(tf_full_longitudinal(1,2)); % u
    state
tf_full_longitudinal_Th_with_statew = minreal(tf_full_longitudinal(2,2)); % w
    state
tf_full_longitudinal_Th_with_stateq = minreal(tf_full_longitudinal(3,2)); % q
    state
tf_full_longitudinal_Th_with_state_theta = minreal(tf_full_longitudinal(4,2)); %
    theta state

%% Long Mode For Longitudinal
A_longitudinal_long = [Xu, -g*cos(theta_0); -Zu/(Zq+u_0), g*sin(theta_0)/(Zq+u_0)
    ];
```

```
B_longitudinal_long = [Xde, Xth; -Zde/(Zq+u_0), -Zdth/(Zq+u_0)];

% Assuming the output directly maps to the state (no C and D matrices provided, use identity matrix for direct mapping)
C_longitudinal_long  = eye(2); % Direct mapping of state to output
D_longitudinal_long  = zeros(2, 2); % No direct feedthrough

% Define state-space model for the long-period longitudinal system
sys_longitudinal_long = ss(A_longitudinal_long, B_longitudinal_long, C_longitudinal_long, D_longitudinal_long);
tf_long_period = tf(sys_longitudinal_long);

% Run the simulation for the long-period mode with external input 'u'
[y_long, t, x] = lsim(sys_longitudinal_long, u, t);

% Transfer function from elevator deflection (De) to each state output
tf_long_period_De_with_state_u = minreal(tf_long_period(1,1)); % u state
tf_long_period_De_with_state_theta = minreal(tf_long_period(2,1)); % theta state

% Transfer function from throttle (Th) to each state output
tf_long_period_Th_with_state_u = minreal(tf_long_period(1,2)); % u state
tf_long_period_Th_with_state_theta = minreal(tf_long_period(2,2)); % theta state

%% Short Mode For Longitudinal
A_longitudinal_short = [
    Zw / (1 - Zwd), (Zq + u_0) / (1 - Zwd);
    Mw + (Mwd * (Zw / (1 - Zwd))), Mq + (Mwd * ((Zq + u_0) / (1 - Zwd)))
];
B_longitudinal_short = [
    Zde / (1 - Zwd), Zdth / (1 - Zwd);
    Mde + (Mwd * Zde / (1 - Zwd)), Mdth + (Mwd*Zdth / (1 - Zwd))
];
C_longitudinal_short = eye(2); % Direct mapping of state to output
D_longitudinal_short = zeros(2, 2); % No direct feedthrough

% Define state-space model for the short-period longitudinal system
sys_longitudinal_short = ss(A_longitudinal_short, B_longitudinal_short, C_longitudinal_short, D_longitudinal_short);
tf_short_period = tf(sys_longitudinal_short);

% Initial state vector for short-period mode ()
x0_short = [w_0; q_0]; % Corrected to match the 2-dimensional system

% Run the simulation for the short-period mode
[y_short, t, x] = lsim(sys_longitudinal_short, u, t);

% Extract and simplify transfer functions with corresponding labels
% Transfer function from elevator deflection (De) to each state output
tf_short_period_De_with_state_w = minreal(tf_short_period(1,1)); % w state
tf_short_period_De_with_state_q = minreal(tf_short_period(2,1)); % q state

% Transfer function from throttle (Th) to each state output
tf_short_period_Th_with_state_w = minreal(tf_short_period(1,2)); % w state
tf_short_period_Th_with_state_q = minreal(tf_short_period(2,2)); % q state

%% Full Lateral Mode System
x0_lateral = [v_0; p_0; r_0; phi_0; psi_0];


A_lateral = [Yv, Yp+wo, Yr-u_0, g*cos(theta_0), 0;
             Lv, Lp, Lr, 0, 0;
             Nv, Np, Nr, 0, 0;
             0, 1, tan(theta_0), 0, 0;
             0, 0, 1/cos(theta_0), 0, 0];

B_lateral = [Yda, Ydr;
             Lda, Ldr;
             Nda, Ndr;
             0, 0;
             0, 0];

```

```
C_lateral = eye(5); % Observing all states
D_lateral = zeros(5, 2); % No direct feedthrough

u_lat = [dc(1)* ones(size(t)),1*ones(size(t))]';

% Create the state-space model for the lateral dynamics
sys_lateral = ss(A_lateral, B_lateral, C_lateral, D_lateral);
tf_full_lateral = tf(sys_lateral);

% Initial state vector for the lateral dynamics

% Run the simulation for the lateral dynamics with the specified input
[y_lateral, t, x_lateral] = lsim(sys_lateral, u_lat, t, x0_lateral);

% Transfer function from aileron deflection (Da) to each state output
tf_full_lateral_Da_with_state_v = minreal(tf_full_lateral(1,1));
tf_full_lateral_Da_with_state_p = minreal(tf_full_lateral(2,1));
tf_full_lateral_Da_with_state_r = minreal(tf_full_lateral(3,1));
tf_full_lateral_Da_with_state_phi = minreal(tf_full_lateral(4,1));
tf_full_lateral_Da_with_state_epsi = minreal(tf_full_lateral(5,1));

% Transfer function from rudder deflection (Dr) to each state output
tf_full_lateral_Dr_with_state_v = minreal(tf_full_lateral(1,2));
tf_full_lateral_Dr_with_state_p = minreal(tf_full_lateral(2,2));
tf_full_lateral_Dr_with_state_r = minreal(tf_full_lateral(3,2));
tf_full_lateral_Dr_with_state_phi = minreal(tf_full_lateral(4,2));
tf_full_lateral_Dr_with_state_epsi = minreal(tf_full_lateral(5,2));

%% 3-DOF Approximations for Spiral lateral Mode
A_lateral_3DOF_spiral = [Lp, Lr, 0;
                         Np, Nr, 0;
                         1, tan(theta_0), 0];
% B_lateral_3DOF_spiral = [Ldr;Ndr;0];
B_lateral_3DOF_spiral = [Lda, Ldr;
                         Nda, Ndr;
                         0, 0];

% Define the output matrix to observe all states (p, r, and v)
C_lateral_3DOF_spiral = eye(3);  % Observing all three states
D_lateral_3DOF_spiral = zeros(3, 2);  % No direct feedthrough from input to
    output

% Create the state-space model for the spiral mode approximation
sys_lateral_3DOF_spiral = ss(A_lateral_3DOF_spiral, B_lateral_3DOF_spiral,
    C_lateral_3DOF_spiral, D_lateral_3DOF_spiral);
tf_3DOF_spiral = tf(sys_lateral_3DOF_spiral);

% Initial state vector for the spiral mode with 3 DOF (initial conditions for p,
    r, and v)
x0_spiral_3DOF = [p_0; r_0; phi_0];  % Assuming zero initial conditions

% Run the simulation for the spiral mode with 3 DOF using the specified input
[y_spiral_3DOF, t, x_spiral_3DOF] = lsim(sys_lateral_3DOF_spiral, u_lat, t,
    x0_spiral_3DOF);

% Transfer functions from rudder deflection (Dr) to each state output
tf_3DOF_spiral_Dr_with_state_p = minreal(tf_3DOF_spiral(1,1)); % Roll Rate (p)
tf_3DOF_spiral_Dr_with_state_r = minreal(tf_3DOF_spiral(2,1)); % Yaw Rate (r)
tf_3DOF_spiral_Dr_with_state_phi = minreal(tf_3DOF_spiral(3,1)); % Roll Angle (
    phi)

%% 3-DOF Approximations for Dutch roll lateral Mode
A_lateral_3DOF_dutch = [Yv,  Yp+wo, Yr-u_0;
                        Lv, Lp, 0;
                        Nv, 0, Nr];
B_lateral_3DOF_dutch = [Yda, Ydr;
                        Lda, Ldr;
                        Nda, Ndr];
% Output matrix to observe all states (v, p, and r)
C_lateral_3DOF_dutch = eye(3);  % Observing all three states
D_lateral_3DOF_dutch = zeros(3, 2);  % No direct feedthrough from input to output
```

```
322
323 % Create the state-space model for the Dutch Roll mode approximation
324 sys_lateral_3DOF_dutch = ss(A_lateral_3DOF_dutch, B_lateral_3DOF_dutch,
        C_lateral_3DOF_dutch, D_lateral_3DOF_dutch);
325 tf_3DOF_dutch = tf(sys_lateral_3DOF_dutch);
326
327 % Initial state vector for the Dutch Roll mode with 3 DOF (initial conditions for
        v, p, and r)
328 x0_dutch_3DOF = [v_0; p_0; r_0];  % Assuming zero initial conditions
329
330 % Run the simulation for the Dutch Roll mode with 3 DOF using the specified input
331 [y_dutch_3DOF, t, x_dutch_3DOF] = lsim(sys_lateral_3DOF_dutch, u_lat, t,
        x0_dutch_3DOF);
332
333 % Transfer functions from aileron deflection (Da) to each state output
334 tf_3DOF_dutch_Da_with_state_v = minreal(tf_3DOF_dutch(1,1)); % Side velocity (v)
335 tf_3DOF_dutch_Da_with_state_p = minreal(tf_3DOF_dutch(2,1)); % Roll angle (phi)
336 tf_3DOF_dutch_Da_with_state_r = minreal(tf_3DOF_dutch(3,1)); % Yaw rate (r)
337
338 % Transfer functions from rudder deflection (Dr) to each state output
339 tf_3DOF_dutch_Dr_with_state_v = minreal(tf_3DOF_dutch(1,2));
340 tf_3DOF_dutch_Dr_with_state_p = minreal(tf_3DOF_dutch(2,2));
341 tf_3DOF_dutch_Dr_with_state_r = minreal(tf_3DOF_dutch(3,2));
342
343 %% 2-DOF Approximations for the Dutch roll lateral Mode
344 A_lateral_2DOF_dutch = [Yv, Yr-u_0;
345                         Nv, Nr];
346 B_lateral_2DOF_dutch = [Yda, Ydr;
347                         Nda, Ndr];
348
349 % Output matrix to observe both states (v and r)
350 C_lateral_2DOF_dutch = eye(2);  % Observing both states
351 D_lateral_2DOF_dutch = zeros(2, 2);  % No direct feedthrough from input to output
352
353 % Create the state-space model for the 2-DOF Dutch Roll mode
354 sys_lateral_2DOF_dutch = ss(A_lateral_2DOF_dutch, B_lateral_2DOF_dutch,
        C_lateral_2DOF_dutch, D_lateral_2DOF_dutch);
355 tf_2DOF_dutch = tf(sys_lateral_2DOF_dutch);
356
357 % Initial state vector for the 2-DOF Dutch Roll mode (initial conditions for v
        and r)
358 x0_dutch_2DOF = [v_0;r_0];  % Assuming zero initial conditions
359
360 % Run the simulation for the Dutch Roll mode with 2 DOF using the specified input
361 [y_dutch_2DOF, t, x_dutch_2DOF] = lsim(sys_lateral_2DOF_dutch, u_lat, t,
        x0_dutch_2DOF);
362
363 % Transfer functions from aileron deflection (Da) to each state output
364 tf_2DOF_dutch_Da_with_state_v = minreal(tf_2DOF_dutch(1,1)); % Side velocity (v)
365 tf_2DOF_dutch_Da_with_state_r = minreal(tf_2DOF_dutch(2,1)); % Yaw rate (r)
366
367 % Transfer functions from rudder deflection (Dr) to each state output
368 tf_2DOF_dutch_Dr_with_state_v = minreal(tf_2DOF_dutch(1,2));
369 tf_2DOF_dutch_Dr_with_state_r = minreal(tf_2DOF_dutch(2,2));
370
371 %% 1-DOF Approximations for the Roll lateral Mode
372 A_lateral_1DOF = Lp; % Only the roll damping term Lp
373 B_lateral_1DOF = Lda; % Aileron control input affecting roll rate
374 C_lateral_1DOF = 1; % Observe roll rate (p) directly
375 D_lateral_1DOF = 0; % No direct feedthrough
376
377 sys_lateral_1DOF = ss(A_lateral_1DOF, B_lateral_1DOF, C_lateral_1DOF,
        D_lateral_1DOF);
378 % Calculate the transfer function from aileron deflection (Da) to roll rate (p)
379 tf_1DOF_roll_Da_with_state_p = minreal(tf(sys_lateral_1DOF));
380
381 % Define the input signal (u), representing aileron deflection
382 u_1dof = ones(size(t));  % Constant aileron deflection for demonstration
383
384 % Initial state vector for roll rate
385 x0_roll_1DOF = p_0;  % Assuming zero initial roll rate
```

```
386
387 % Run the simulation for the roll mode with the specified input
388 [y_roll_1DOF, t, x_roll_1DOF] = lsim(sys_lateral_1DOF, u_1dof, t, x0_roll_1DOF);
389
390
391 %% Longitudinal Controller Design
392
393 % Transfer function from elevator deflection (De) to each state output
394 De_u = minreal(tf_full_longitudinal(1,1)); % u state
395 De_w = minreal(tf_full_longitudinal(2,1)); % w state
396 De_q = minreal(tf_full_longitudinal(3,1)); % q state
397 De_theta = minreal(tf_full_longitudinal(4,1)); % theta state
398
399 % Transfer function from throttle (Th) to each state output
400 Th_u = minreal(tf_full_longitudinal(1,2)); % u state
401 Th_w = minreal(tf_full_longitudinal(2,2)); % w state
402 Th_q = minreal(tf_full_longitudinal(3,2)); % q state
403 Th_theta = minreal(tf_full_longitudinal(4,2)); % theta state
404 u_dT= tf_full_longitudinal(1,2);
405
406 % Transfer function used in building the controller
407 servo= tf(10,[1 10]);
408 integrator=tf(1,[1 0]);
409 diffrentiator=tf([1 0],1);
410 engine_timelag=tf(0.1,[1 0 1]);
411
412 De_Wd=De_w*diffrentiator;
413 De_az=De_Wd-u_0*De_q;
414 Theta_az=De_az/De_theta;
415 De_alpha=De_w/u_0;
416 De_gamma=De_theta-De_alpha;
417
418 %pitch control
419 ol_theta_thetacomm=-servo*De_theta;
420
421 % Velocity Control
422 OL_u_ucom=servo*engine_timelag*u_dT;
423
424 load('Velocity_Controller_tf.mat')
425 load('eltaw2am_c.mat')
426 load('Pitch_Design2.mat')
427 load('Altitude_Design.mat')
428
429 %% Lateral Controller Design
430
431 % Transfer function from aileron deflection
432 v_da=minreal(tf_full_lateral(1,1));
433 p_da=minreal(tf_full_lateral(2,1));
434 r_da=minreal(tf_full_lateral(3,1));
435 phi_da=minreal(tf_full_lateral(4,1));
436 psi_da=minreal(tf_full_lateral(5,1));
437 beta_da=v_da/u_0;
438
439 % Transfer function from rudder deflection
440 v_dr=minreal(tf_full_lateral(1,2));
441 p_dr=minreal(tf_full_lateral(2,2));
442 r_dr=minreal(tf_full_lateral(3,2));
443 phi_dr=minreal(tf_full_lateral(4,2));
444 psi_dr=minreal(tf_full_lateral(5,2));
445 beta_dr=v_da/u_0;
446
447 % yaw controller
448 ol_r_rcomm=servo*r_dr;
449 load('Yaw_Damper.mat')
450
451 Lat_YawDamper_ss_series=feedback(series(append(1,servo),sys_lateral,[1 2],[1 2]),
      HPF_YawDamper,2,3,1);
452 Lat_YawDamper_tf_series=tf(Lat_YawDamper_ss_series);
453 phi_da_AfterYawDamper=Lat_YawDamper_tf_series(4,1);
454 % phi controller
455 ol_phi_phicomm=minreal(servo*phi_da_AfterYawDamper);
```

```
456
457
458 % %% Ploting
459 % %% MATLAB Plotting
460 % rad_to_deg = 180 / pi;  % Conversion from radians to degrees
461 %
462 % % Plot results (e.g., velocities, angles, positions)
463 % figure;
464 % sgtitle('Matlab Code Plots');
465 %
466 % % Plot forward, side, and vertical velocities (u, v, w)
467 % subplot(4,3,1);
468 % plot(t, states(1,:), 'b');  % u (forward velocity)
469 % title('Forward Velocity (u)');
470 % xlabel('Time (s)');
471 % ylabel('Velocity (ft/s)');
472 %
473 % % subplot(4,3,2);
474 % % plot(t2, y2(2,:), 'r');  % v (side velocity)
475 % % title('Side Velocity (v)');
476 % % xlabel('Time (s)');
477 % % ylabel('Velocity (ft/s)');
478 % %
479 % % subplot(4,3,3);
480 % % plot(t2, y2(3,:), 'g');  % w (vertical velocity)
481 % % title('Vertical Velocity (w)');
482 % % xlabel('Time (s)');
483 % % ylabel('Velocity (ft/s)');
484 %
485 % subplot(4,3,2);
486 % plot(t, asin(states(2,:)/Vto)*rad_to_deg, 'r');  % bita (side velocity)
487 % title('bita');
488 % xlabel('Time (s)');
489 % ylabel('Velocity (ft/s)');
490 %
491 % subplot(4,3,3);
492 % plot(t, atan(states(3,:)./states(1,:))*rad_to_deg, 'g');  % alpha (vertical
      velocity)
493 % title('Alpa');
494 % xlabel('Time (s)');
495 % ylabel('Velocity (ft/s)');
496 %
497 % % Angular velocities (p, q, r)
498 %
499 % subplot(4,3,4);
500 % plot(t, states(4,:), 'b');  % p (roll rate)
501 % title('Roll Rate (p)');
502 % xlabel('Time (s)');
503 % ylabel('Angular Velocity (rad/s)');
504 %
505 % subplot(4,3,5);
506 % plot(t, states(5,:), 'r');  % q (pitch rate)
507 % title('Pitch Rate (q)');
508 % xlabel('Time (s)');
509 % ylabel('Angular Velocity (rad/s)');
510 %
511 % subplot(4,3,6);
512 % plot(t, states(6,:), 'g');  % r (yaw rate)
513 % title('Yaw Rate (r)');
514 % xlabel('Time (s)');
515 % ylabel('Angular Velocity (rad/s)');
516 %
517 % % Euler angles (phi, theta, psi) in degrees
518 %
519 % subplot(4,3,7);
520 % plot(t, states(7,:) * rad_to_deg, 'b');  % phi (roll angle in degrees)
521 % title('Roll Angle (phi) in degrees');
522 % xlabel('Time (s)');
523 % ylabel('Angle (deg)');
524 %
525 % subplot(4,3,8);
```

```
% plot(t, states(8,:) * rad_to_deg, 'r');  % theta (pitch angle in degrees)
% title('Pitch Angle (theta) in degrees');
% xlabel('Time (s)');
% ylabel('Angle (deg)');
%
% subplot(4,3,9);
% plot(t, states(9,:) * rad_to_deg, 'g');  % psi (yaw angle in degrees)
% title('Yaw Angle (psi) in degrees');
% xlabel('Time (s)');
% ylabel('Angle (deg)');
%
% % Position (x, y, z) in feet
%
% subplot(4,3,10);
% plot(t, states(10,:), 'b');
% title('Position X (ft)');
% xlabel('Time (s)');
% ylabel('Position (ft)');
%
% subplot(4,3,11);
% plot(t, states(11,:), 'r');
% title('Position Y (ft)');
% xlabel('Time (s)');
% ylabel('Position (ft)');
%
% subplot(4,3,12);
% plot(t, round(states(12,:),3), 'g');
% title('Position Z (ft)');
% xlabel('Time (s)');
% ylabel('Position (ft)');
%
%
% % Trajectory (x vs y vs z in feet)
% figure;
% plot3(states(10,:), states(11,:),  round(states(12,:),3), 'b');  % 3D
    trajectory plot
% title('Matlab Code Trajectory (ft)', 'FontWeight', 'bold', 'FontSize', 14);
% xlabel('X (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% ylabel('Y (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% zlabel('Z (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% grid on;
%
%
% %% Simulink Plot
% % Plot results (e.g., velocities, angles, positions)
% figure;
% sgtitle('Simulink Code Plots');
% % Plot forward, side, and vertical velocities (u, v, w)
% subplot(4,3,1);
% plot(t2, y2(1,:), 'b');  % u (forward velocity)
% title('Forward Velocity (u)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Velocity (ft/s)', 'FontWeight', 'bold', 'FontSize', 10);
%
% % subplot(4,3,2);
% % plot(t2, y2(2,:), 'r');  % v (side velocity)
% % title('Side Velocity (v)', 'FontWeight', 'bold', 'FontSize', 12);
% % xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% % ylabel('Velocity (ft/s)', 'FontWeight', 'bold', 'FontSize', 10);
% %
% % subplot(4,3,3);
% % plot(t2, y2(3,:), 'g');  % w (vertical velocity)
% % title('Vertical Velocity (w)', 'FontWeight', 'bold', 'FontSize', 12);
% % xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% % ylabel('Velocity (ft/s)', 'FontWeight', 'bold', 'FontSize', 10);
% %
%
% % plots of alpha and beta
% subplot(4,3,2);
% plot(t2, y2(13,:)*rad_to_deg, 'r');  % v (side angle beta)
% title('Side angle (Beta)', 'FontWeight', 'bold', 'FontSize', 12);
```

```
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Beta (deg)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,3);
% plot(t2, y2(14,:)*rad_to_deg, 'g');  % w (alpha )
% title('angle of attach (Alpha)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('alpha (deg)', 'FontWeight', 'bold', 'FontSize', 10);
%
%
% % Angular velocities (p, q, r)
% subplot(4,3,4);
% plot(t2, y2(4,:)* rad_to_deg, 'b');  % p (roll rate)
% title('Roll Rate (p)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Angular Velocity (rad/s)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,5);
% plot(t2, y2(5,:)*rad_to_deg, 'r');  % q (pitch rate)
% title('Pitch Rate (q)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Angular Velocity (rad/s)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,6);
% plot(t2, y2(6,:)*rad_to_deg, 'g');  % r (yaw rate)
% title('Yaw Rate (r)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Angular Velocity (rad/s)', 'FontWeight', 'bold', 'FontSize', 10);
%
% % Euler angles (phi, theta, psi) in degrees
% subplot(4,3,7);
% plot(t2, y2(7,:) * rad_to_deg, 'b');  % phi (roll angle in degrees)
% title('Roll Angle (phi) in degrees', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Angle (deg)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,8);
% plot(t2, y2(8,:) * rad_to_deg, 'r');  % theta (pitch angle in degrees)
% title('Pitch Angle (theta) in degrees', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Angle (deg)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,9);
% plot(t2, unwrap(y2(9,:)) * rad_to_deg, 'g');  % psi (yaw angle in degrees)
% title('Yaw Angle (psi) in degrees', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Angle (deg)', 'FontWeight', 'bold', 'FontSize', 10);
%
% % Position (x, y, z) in feet
% subplot(4,3,10);
% plot(t2, y2(10,:), 'b');
% title('Position X (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Position (ft)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,11);
% plot(t2, y2(11,:), 'r');
% title('Position Y (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Position (ft)', 'FontWeight', 'bold', 'FontSize', 10);
%
% subplot(4,3,12);
% plot(t2, round(y2(12,:), 3), 'g');
% title('Position Z (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% xlabel('Time (s)', 'FontWeight', 'bold', 'FontSize', 10);
% ylabel('Position (ft)', 'FontWeight', 'bold', 'FontSize', 10);
%
% % Trajectory (x vs y vs z in feet)
% figure;
% plot3(y2(10,:), y2(11,:), round(y2(12,:), 3), 'b');  % 3D trajectory plot
% title('Simulink Code Trajectory (ft)', 'FontWeight', 'bold', 'FontSize', 14);
```

```
% xlabel('X (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% ylabel('Y (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% zlabel('Z (ft)', 'FontWeight', 'bold', 'FontSize', 12);
% grid on;
%
% %% Step response plots
%
% %Longitudinal
%
% % Compare 'u' (velocity) from full and long-period models
% subplot(4,1,1);
% plot(t, y_full(:,1)+x0(1), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
% plot(t, y_long(:,1)+x0(1), 'b--', 'DisplayName', 'Long-Period Model');
% plot(t2, y2(1,:), 'g--', 'DisplayName', 'Non-linear Model');
% title('Response of u (velocity)'); xlabel('Time (s)'); ylabel('u (m/s)');
% legend;
% hold off;
%
% % Compare 'w' (vertical speed) from full and short-period models
% subplot(4,1,2);
% plot(t, y_full(:,2)+x0(2), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
% plot(t, y_short(:,1)+x0(2), 'b--', 'DisplayName', 'Short-Period Model');
% plot(t, states(3,:), 'g--', 'DisplayName', 'Non-Linear Model');
% title('Response of w (vertical speed)'); xlabel('Time (s)'); ylabel('w (m/s)');
% legend;
% hold off;
%
% % Compare 'q' (pitch rate) from full and short-period models
% subplot(4,1,3);
% plot(t, y_full(:,3)+x0(3), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
% plot(t, y_short(:,2)+x0(3), 'b--', 'DisplayName', 'Short-Period Model');
% plot(t2, y2(5,:), 'g--', 'DisplayName', 'Non-Linear Model');
% title('Response of q (pitch rate)'); xlabel('Time (s)'); ylabel('q (rad/s)');
% legend;
% hold off;
%
% % Compare 'theta' (pitch angle) from full and long-period models
% subplot(4,1,4);
% plot(t,y_full(:,4)+x0(4), 'r', 'DisplayName', 'Fully linearized Model'); hold on;
% plot(t, y_long(:,2)+x0(4), 'b--', 'DisplayName', 'Long-Period Model');
% plot(t2, y2(8,:), 'g--', 'DisplayName', 'Non-Linear Model');
% title('Response of \theta (pitch angle)'); xlabel('Time (s)'); ylabel('\theta (rad)');
% legend;
% hold off;
%
%
%
% %% Ploting Rotloucs and Bodeplot For Longitudinal
% % Call the function for each transfer function comparison between full and short modes
%
% % Full longitudinal vs short-period
% plotAndSaveSubplots(tf_full_longitudinal_De_with_statew, tf_short_period_De_with_state_w, 'Short', 'De', 'w');
% plotAndSaveSubplots(tf_full_longitudinal_De_with_stateq, tf_short_period_De_with_state_q, 'Short', 'De', 'q');
% plotAndSaveSubplots(tf_full_longitudinal_Th_with_statew, tf_short_period_Th_with_state_w, 'Short', 'Th', 'w');
% plotAndSaveSubplots(tf_full_longitudinal_Th_with_stateq, tf_short_period_Th_with_state_q, 'Short', 'Th', 'q');
%
% % Full longitudinal vs long-period
% plotAndSaveSubplots(tf_full_longitudinal_De_with_stateu, tf_long_period_De_with_state_u, 'Long', 'De', 'u');
% plotAndSaveSubplots(tf_full_longitudinal_De_with_state_theta, tf_long_period_De_with_state_theta, 'Long', 'De', 'theta');
```

```
% plotAndSaveSubplots(tf_full_longitudinal_Th_with_stateu,
    tf_long_period_Th_with_state_u, 'Long', 'Th', 'u');
% plotAndSaveSubplots(tf_full_longitudinal_Th_with_state_theta,
    tf_long_period_Th_with_state_theta, 'Long', 'Th', 'theta');
%
%
% %% Plot and save Root Locus and Bode plots for each mode
% % Define the transfer functions and labels for comparisons
%
% % Aileron Deflection (Da) Comparisons
% Da_v = {tf_full_lateral_Da_with_state_v, tf_3DOF_dutch_Da_with_state_v,
    tf_2DOF_dutch_Da_with_state_v};
% Da_p = {tf_full_lateral_Da_with_state_p, tf_3DOF_dutch_Da_with_state_p,
    tf_1DOF_roll_Da_with_state_p};
% Da_r = {tf_full_lateral_Da_with_state_r, tf_3DOF_dutch_Da_with_state_r,
    tf_2DOF_dutch_Da_with_state_r};
% Da_phi = {tf_full_lateral_Da_with_state_phi};
% Da_epsi = {tf_full_lateral_Da_with_state_epsi};
%
% % Rudder Deflection (Dr) Comparisons
% Dr_v = {tf_full_lateral_Dr_with_state_v, tf_3DOF_dutch_Dr_with_state_v,
    tf_2DOF_dutch_Dr_with_state_v};
% Dr_p = {tf_full_lateral_Dr_with_state_p, tf_3DOF_spiral_Dr_with_state_p,
    tf_3DOF_dutch_Dr_with_state_p};
% Dr_r = {tf_full_lateral_Dr_with_state_r, tf_3DOF_spiral_Dr_with_state_r,
    tf_3DOF_dutch_Dr_with_state_r, tf_2DOF_dutch_Dr_with_state_r};
% Dr_phi = {tf_full_lateral_Dr_with_state_phi,tf_3DOF_spiral_Dr_with_state_phi};
% Dr_epsi = {tf_full_lateral_Dr_with_state_epsi};
%
% % Mode labels for Aileron Deflection (Da) Comparisons
% mode_labels_Da_v = {'Full Mode', '3-DOF Dutch Roll', '2-DOF Dutch Roll'};
% mode_labels_Da_p = {'Full Mode','3-DOF Dutch Roll','1-DOF Roll'};
% mode_labels_Da_r = {'Full Mode', '3-DOF Dutch Roll', '2-DOF Dutch Roll'};
% mode_labels_Da_phi = {'Full Mode'};
% mode_labels_Da_epsi = {'Full Mode'};
%
% % Mode labels for Rudder Deflection (Dr) Comparisons
% mode_labels_Dr_v = {'Full Mode', '3-DOF Dutch Roll', '2-DOF Dutch Roll'};
% mode_labels_Dr_p = {'Full Mode', '3-DOF Spiral', '3-DOF Dutch Roll'};
% mode_labels_Dr_r = {'Full Mode', '3-DOF Spiral', '3-DOF Dutch Roll', '2-DOF
    Dutch Roll'};
% mode_labels_Dr_phi = {'Full Mode', '3-DOF Spiral'};
% mode_labels_Dr_epsi = {'Full Mode'};
%
% %Call the function to compare each mode for specific input-output pairs
%
% % Aileron Deflection (Da) Comparisons
% plotAndCompareModesWithSubplots(Da_v, mode_labels_Da_v, 'Da', 'v');
% plotAndCompareModesWithSubplots(Da_p, mode_labels_Da_p, 'Da', 'p');
% plotAndCompareModesWithSubplots(Da_r, mode_labels_Da_r, 'Da', 'r');
% plotAndCompareModesWithSubplots(Da_phi, mode_labels_Da_phi, 'Da', 'phi');
% plotAndCompareModesWithSubplots(Da_epsi, mode_labels_Da_epsi, 'Da', 'epsi');
%
% % Rudder Deflection (Dr) Comparisons
% plotAndCompareModesWithSubplots(Dr_v, mode_labels_Dr_v, 'Dr', 'v');
% plotAndCompareModesWithSubplots(Dr_p, mode_labels_Dr_p, 'Dr', 'p');
% plotAndCompareModesWithSubplots(Dr_r, mode_labels_Dr_r, 'Dr', 'r');
% plotAndCompareModesWithSubplots(Dr_phi, mode_labels_Dr_phi, 'Dr', 'phi');
% plotAndCompareModesWithSubplots(Dr_epsi, mode_labels_Dr_epsi, 'Dr', 'epsi');
%
% %% Display Tranfer Function For Longitudinal
% % Call the function below for each input-output pair comparison
%
% % Full longitudinal vs short-period
% displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_statew,
    tf_short_period_De_with_state_w, 'Short', 'De', 'w');
% displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_stateq,
    tf_short_period_De_with_state_q, 'Short', 'De', 'q');
% displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_statew,
    tf_short_period_De_with_state_w, 'Short', 'Th', 'w');
% displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_stateq,
```

```
    tf_short_period_De_with_state_q, 'Short', 'Th', 'q');
%
% % Full longitudinal vs long-period
% displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_stateu,
    tf_long_period_De_with_state_u, 'Long', 'De', 'u');
% displayTransferFunctionAsFraction(tf_full_longitudinal_De_with_state_theta,
    tf_long_period_De_with_state_theta, 'Long', 'De', 'theta');
% displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_stateu,
    tf_long_period_Th_with_state_u, 'Long', 'Th', 'u');
% displayTransferFunctionAsFraction(tf_full_longitudinal_Th_with_state_theta,
    tf_long_period_Th_with_state_theta, 'Long', 'Th', 'theta');
%
% %% Display the transfer functions in fraction format
% % Call the function below for each group of transfer functions
%
% % Aileron Deflection (Da) Comparisons
% displayTransferFunctionGroup(Da_v, mode_labels_Da_v, 'Da', 'v');
% displayTransferFunctionGroup(Da_p, mode_labels_Da_p, 'Da', 'p');
% displayTransferFunctionGroup(Da_r, mode_labels_Da_r, 'Da', 'r');
% displayTransferFunctionGroup(Da_phi, mode_labels_Da_phi, 'Da', 'phi');
% displayTransferFunctionGroup(Da_epsi, mode_labels_Da_epsi, 'Da', 'epsi');
%
% % Rudder Deflection (Dr) Comparisons
% displayTransferFunctionGroup(Dr_v, mode_labels_Dr_v, 'Dr', 'v');
% displayTransferFunctionGroup(Dr_p, mode_labels_Dr_p, 'Dr', 'p');
% displayTransferFunctionGroup(Dr_r, mode_labels_Dr_r, 'Dr', 'r');
% displayTransferFunctionGroup(Dr_phi, mode_labels_Dr_phi, 'Dr', 'phi');
% displayTransferFunctionGroup(Dr_epsi, mode_labels_Dr_epsi, 'Dr', 'epsi');
%
% %% Functions
%
% % Define a function to display the transfer function expressions as a large
    fraction
% function displayTransferFunctionAsFraction(tf_full, tf_mode, mode_name,
    input_name, output_name)
%     % Extract numerator and denominator for the full mode transfer function
%     [num_full, den_full] = tfdata(tf_full, 'v'); % Get numerator and
    denominator as vectors
%     % Convert numerator and denominator to strings
%     num_full_str = poly2str(num_full, 's');
%     den_full_str = poly2str(den_full, 's');
%
%     % Extract numerator and denominator for the mode transfer function (short/
    long)
%     [num_mode, den_mode] = tfdata(tf_mode, 'v'); % Get numerator and
    denominator as vectors
%     % Convert numerator and denominator to strings
%     num_mode_str = poly2str(num_mode, 's');
%     den_mode_str = poly2str(den_mode, 's');
%
%     % Display the formatted transfer functions as a large fraction
%     fprintf('Comparison for Transfer Function (%s to %s):\n', input_name,
    output_name);
%     fprintf('-------------------------------------------------------\n');
%
%     % Display the full mode transfer function in S domain format as a large
    fraction
%     fprintf('Full Mode Transfer Function:\n');
%     fprintf('    %s\n', num_full_str);  % Numerator
%     fprintf('    %s\n', repmat('-', max(length(num_full_str), length(
    den_full_str)), 1)); % Divider line
%     fprintf('    %s\n\n', den_full_str); % Denominator
%
%     % Display the alternative mode (short or long) transfer function as a large
     fraction
%     fprintf('%s Mode Transfer Function:\n', mode_name);
%     fprintf('    %s\n', num_mode_str);  % Numerator
%     fprintf('    %s\n', repmat('-', max(length(num_mode_str), length(
    den_mode_str)), 1)); % Divider line
%     fprintf('    %s\n\n', den_mode_str); % Denominator
%
```

```
%     % Add a line break for readability between comparisons
%     fprintf('\n\n');
% end
%
% % Define function to display the transfer functions for a group in fraction format
% function displayTransferFunctionGroup(tf_list, mode_labels, input_name, output_name)
%     num_modes = length(tf_list);
%     fprintf('Transfer Function Comparisons for %s to %s:\n', input_name, output_name);
%     fprintf('------------------------------------------------------\n');
%
%     for k = 1:num_modes
%         % Extract numerator and denominator
%         [num, den] = tfdata(tf_list{k}, 'v');
%         num_str = poly2str(num, 's');
%         den_str = poly2str(den, 's');
%
%         % Display transfer function for each mode
%         fprintf('%s Mode Transfer Function:\n', mode_labels{k});
%         fprintf('    %s\n', num_str);  % Numerator
%         fprintf('    %s\n', repmat('-', max(length(num_str), length(den_str)), 1)); % Divider line
%         fprintf('    %s\n\n', den_str); % Denominator
%     end
%     fprintf('\n\n'); % Add extra line spacing between groups for readability
% end
%
% % Define function to plot and save Root Locus and Bode plots with subplots for each mode comparison
% function plotAndCompareModesWithSubplots(tf_list, mode_labels, input_name, output_name)
%     % Ensure 'graphs' directory exists
%     if ~exist('graphs', 'dir')
%         mkdir('graphs');
%     end
%
%     num_modes = length(tf_list);
%
%     % Plot Root Locus with each mode in its own subplot
%     figure;
%     for k = 1:num_modes
%         subplot(1, num_modes, k); % Create a subplot for each mode
%         rlocus(tf_list{k});
%         title([mode_labels{k}]);
%         xlabel('Real Axis');
%         ylabel('Imaginary Axis');
%     end
%     sgtitle(['Root Locus - ', input_name, ' to ', output_name]);
%     saveas(gcf, fullfile('graphs', ['RootLocus_Comparison_', input_name, '_to_', output_name, '.png']));
%
%     % Plot Bode Plot with each mode in the same plot for overlay comparison
%     figure;
%     hold on;
%     for k = 1:num_modes
%         bode(tf_list{k});
%     end
%     hold off;
%     title(['Bode Plot Comparison - ', input_name, ' to ', output_name]);
%     legend(mode_labels);
%     saveas(gcf, fullfile('graphs', ['BodePlot_Comparison_', input_name, '_to_', output_name, '.png']));
% end
%
% % Define a function to plot and save root locus and Bode plots with subplots for each transfer function
% function plotAndSaveSubplots(tf_full, tf_short, mode_name, input_name, output_name)
%     % Ensure 'graphs' directory exists
```

```
%     if ~exist('graphs', 'dir')
%         mkdir('graphs');
%     end
%
%     % Create figure for Root Locus with two subplots
%     figure;
%
%     % Full Mode Root Locus in the first subplot
%     subplot(1, 2, 1);
%     rlocus(tf_full);
%     title(['Root Locus - Full Mode (', input_name, ' to ', output_name, ')']);
%     xlabel('Real Axis');
%     ylabel('Imaginary Axis');
%
%     % Short Mode Root Locus in the second subplot
%     subplot(1, 2, 2);
%     rlocus(tf_short);
%     title(['Root Locus - ', mode_name, ' Mode (', input_name, ' to ',
    output_name, ')']);
%     xlabel('Real Axis');
%     ylabel('Imaginary Axis');
%
%     % Save the combined root locus figure
%     saveas(gcf, fullfile('graphs', ['RootLocus_', input_name, '_to_',
    output_name, '_Full_vs_', mode_name, '.png']));
%
%     % Bode plot (Separate for Full and Short Mode)
%     figure;
%     bode(tf_full);
%     hold on;
%     bode(tf_short);
%     hold off;
%     title(['Bode Plot - ', input_name, ' to ', output_name, ' (Full vs ',
    mode_name, ' Mode)']);
%     legend('Full Mode', [mode_name, ' Mode']);
%     saveas(gcf, fullfile('graphs', ['BodePlot_', input_name, '_to_',
    output_name, '_', mode_name, '.png']));
% end
airport_name ="Hilo International (PHTO)";


% %% Test_1
% %
% % Get the folder of the current MATLAB script
% current_folder = fileparts(mfilename('fullpath'));
% output_folder = fullfile(current_folder, 'Task7_result_graphs'); % Define the
    folder as 'Task7_result_graphs'
%
% % Create the folder if it does not exist
% if ~exist(output_folder, 'dir')
%     mkdir(output_folder);
% end
%
% % Figure 1: Pitching Angle Output for Pitch Controller
% figure(1)
% plot(out.tout, out.theta_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.theta_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.theta_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$Pitching$ $Angle$ $Response$ $for$ $Pitch$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\theta$ $command$', '$\theta$ $non$ $linear$', '$\theta$ $linear$',
    ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\theta$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
```

```
% save_path = fullfile(output_folder, '
    test1_Pitching_Angle_Response_for_Pitch_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Figure 2: U Output for Pitch Controller
% figure(2)
% plot(out.tout, out.u_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.u_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.u_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$U$ $Response$ $for$ $Pitch$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$U$ $command$', '$U$ $non$ $linear$', '$U$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$U$ $(ft/s)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'test1_U_Response_for_Pitch_Controller.png
    ');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Figure 3: Flight Path Angle for Pitch Controller
% figure(3)
% plot(out.tout, out.gamma_states.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.gamma_states.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\gamma$ $Response$ $for$ $Pitch$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\gamma$ $non$ $linear$', '$\gamma$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\gamma$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, '
    test1_Flight_Path_Angle_for_Pitch_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Figure 4: H Output for Pitch Controller
% figure(4)
% plot(out.tout, out.H_states.Data(:, 2), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.H_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$H$ $Response$ $for$ $Pitch$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$H$ $non$ $linear$', '$H$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$H$ $(ft)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'test1_H_Response_for_Pitch_Controller.png
    ');
% exportgraphics(gcf, save_path, 'Resolution', 300);
% % Plot  e  (elevator control action)
% figure(5)
% plot(out.tout, out.elevator_controller.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.elevator_controller.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\delta_e$ $for$ $Pitch$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_e$ $non$ $linear$', '$\delta_e$ $linear$', ...
```

```
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\delta_e$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'Test1_delta_e_for_Pitch_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);


%
% % %% Test 2
%%
% % Plot    (pitch angle)
% figure(6)
% plot(out.tout, out.theta_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.theta_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.theta_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$\theta$ $Response$ $for$ $Pitch$ $\&$ $Velocity$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\theta$ $command$', '$\theta$ $non$ $linear$', '$\theta$ $linear$',
    ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\theta$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, '
    Test2_theta_Response_for_Pitch_Velocity_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot u (forward velocity)
% figure(7)
% plot(out.tout, out.u_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.u_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.u_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$U$ $Response$ $for$ $Pitch$ $\&$ $Velocity$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$U$ $command$', '$U$ $non$ $linear$', '$U$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$U$ $(ft/s)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, '
    Test2_u_Response_for_Pitch_Velocity_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot    (flight path angle)
% figure(8)
% plot(out.tout, out.gamma_states.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.gamma_states.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\gamma$ $Response$ $for$ $Pitch$ $\&$ $Velocity$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\gamma$ $non$ $linear$', '$\gamma$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\gamma$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, '
```

```
    Test2_gamma_Response_for_Pitch_Velocity_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot h (altitude)
% figure(9)
% %plot(out.tout, out.H_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.H_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.H_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$H$ $Response$ $for$ $Pitch$ $\&$ $Velocity$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$H$ $non$ $linear$', '$H$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$H$ $(ft)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, '
    Test2_h_Response_for_Pitch_Velocity_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot  e  (elevator control action)
% figure(10)
% plot(out.tout, out.elevator_controller.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.elevator_controller.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\delta_e$ $for$ $Pitch$ $\&$ $Velocity$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_e$ $non$ $linear$', '$\delta_e$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\delta_e$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, '
    Test2_delta_e_for_Pitch_Velocity_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot  th  (throttle control action)
% figure(11)
% plot(out.tout, out.Trust_controller.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.Trust_controller.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\delta_{th}$ $for$ $Pitch$ $\&$ $Velocity$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_{th}$ $non$ $linear$', '$\delta_{th}$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\delta_{th}$ ', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, '
    Test2_delta_th_for_Pitch_Velocity_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);


% Test 3

% figure(12)
% plot(out.tout, out.theta_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.theta_states.Data(:, 2), 'r', 'LineWidth', 2);
```

```
% hold on
% plot(out.tout, out.theta_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$\theta$ $Response$ $for$ $Altitude$ $Hold$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\theta$ $command$', '$\theta$ $non$ $linear$', '$\theta$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$\theta$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test3_theta_Response_for_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% figure(13)
% plot(out.tout, out.u_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.u_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.u_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$U$ $Response$ $for$ $Altitude$ $Hold$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$U$ $command$', '$U$ $non$ $linear$', '$U$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$U$ $(ft/s)$', 'interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test3_u_Response_for_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% figure(14)
% plot(out.tout, out.gamma_states.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.gamma_states.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\gamma$ $Response$ $for$ $Altitude$ $Hold$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\gamma$ $non$ $linear$', '$\gamma$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$\gamma$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test3_gamma_Response_for_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% figure(15)
% plot(out.tout, out.H_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.H_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.H_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$H$ $Response$ $for$ $Altitude$ $Hold$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$H$ $command$', '$h$ $non$ $linear$', '$h$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$H$ $(ft)$', 'interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test3_h_Response_for_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% figure(16)
% plot(out.tout, out.elevator_controller.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.elevator_controller.Data(:, 2), '--b', 'LineWidth', 2);
```

```
% title('$\delta_e$ $for$ $Altitude$ $Hold$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_e$ $non$ $linear$', '$\delta_e$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$\delta_e$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test3_delta_e_for_Altitude_Hold_Controller
    .png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% figure(17)
% plot(out.tout, out.Trust_controller.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.Trust_controller.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\delta_{th}$ $for$ $Altitude$ $Hold$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_{th}$ $non$ $linear$', '$\delta_{th}$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$\delta_{th}$ ', 'interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, '
    Test3_delta_th_for_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);

% %% Test 4
%
% % Plot    (sideslip angle)
% figure(18)
% plot(out.tout, out.beta_states.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.beta_states.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\beta$ $Response$ $for$ $Lateral$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\beta$ $non$ $linear$', '$\beta$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$\beta$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'Test4_beta_Response_for_Lateral_Controller
    .png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot    (bank angle)
% figure(19)
% plot(out.tout, out.phi_states.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.phi_states.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\phi$ $Response$ $for$ $Lateral$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\phi$ $non$ $linear$', '$\phi$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12);
% ylabel('$\phi$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'Test4_phi_Response_for_Lateral_Controller.
    png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot    (heading angle)
% figure(20)
% plot(out.tout, out.psi_states.Data(:, 1), '--k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.psi_states.Data(:, 2), 'r', 'LineWidth', 2);
```

```
% hold on
% plot(out.tout, out.psi_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$\psi$ $Response$ $for$ $Lateral$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\psi$ $command$', '$\psi$ $non$ $linear$', '$\psi$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\psi$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'Test4_psi_Response_for_Lateral_Controller.
    png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot h (altitude)
% figure(21)
% plot(out.tout, out.H_states.Data(:, 1), '-.k', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.H_states.Data(:, 2), 'r', 'LineWidth', 2);
% hold on
% plot(out.tout, out.H_states.Data(:, 3), '--b', 'LineWidth', 2);
% title('$H$ $Response$ $for$ $Lateral$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$H$ $command$', '$h$ $non$ $linear$', '$h$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$H$ $(ft)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'Test4_h_Response_for_Lateral_Controller.
    png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot  r  (rudder control action)
% figure(22)
% plot(out.tout, out.rudder_controller.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.rudder_controller.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\delta_r$ $for$ $Lateral$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_r$ $non$ $linear$', '$\delta_r$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\delta_r$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'Test4_delta_r_for_Lateral_Controller.png')
    ;
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot  a  (aileron control action)
% figure(23)
% plot(out.tout, out.aileron_controller.Data(:, 1), 'r', 'LineWidth', 2);
% grid on
% grid minor
% hold on
% plot(out.tout, out.aileron_controller.Data(:, 2), '--b', 'LineWidth', 2);
% title('$\delta_a$ $for$ $Lateral$ $Controller$', ...
%     'interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_a$ $non$ $linear$', '$\delta_a$ $linear$', ...
%     'interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\delta_a$ $(degrees)$', 'interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% save_path = fullfile(output_folder, 'Test4_delta_a_for_Lateral_Controller.png')
    ;
```

```
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% %% Test 5
% % Plot    (sideslip angle)
% figure(24)
% plot(out.tout, out.beta_states.Data(:, 1), 'r', 'LineWidth', 2); % Nonlinear
% grid on
% grid minor
% hold on
% plot(out.tout, out.beta_states.Data(:, 2), '--b', 'LineWidth', 2); % Linear
% title('$\beta$ $Response$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$',
    ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\beta$ $non$ $linear$', '$\beta$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\beta$ $(degrees)$', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, '
    Test5_beta_Response_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot    (bank angle)
% figure(25)
% plot(out.tout, out.phi_states.Data(:, 1), 'r', 'LineWidth', 2); % Nonlinear
% grid on
% grid minor
% hold on
% plot(out.tout, out.phi_states.Data(:, 2), '--b', 'LineWidth', 2); % Linear
% title('$\phi$ $Response$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$',
    ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\phi$ $non$ $linear$', '$\phi$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\phi$ $(degrees)$', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, '
    Test5_phi_Response_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot    (heading angle)
% figure(26)
% plot(out.tout, out.psi_states.Data(:, 1), '-.k', 'LineWidth', 2); % Command
% grid on
% grid minor
% hold on
% plot(out.tout, out.psi_states.Data(:, 2), 'r', 'LineWidth', 2); % Nonlinear
% hold on
% plot(out.tout, out.psi_states.Data(:, 3), '--b', 'LineWidth', 2); % Linear
% title('$\psi$ $Response$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$',
    ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\psi$ $command$', '$\psi$ $non$ $linear$', '$\psi$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\psi$ $(degrees)$', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, '
    Test5_psi_Response_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot h (altitude)
% figure(27)
% plot(out.tout, out.H_states.Data(:, 1), '-.k', 'LineWidth', 2); % Command
% grid on
% grid minor
% hold on
% plot(out.tout, out.H_states.Data(:, 2), 'r', 'LineWidth', 2); % Nonlinear
% hold on
% plot(out.tout, out.H_states.Data(:, 3), '--b', 'LineWidth', 2); % Linear
```

```
% title('$H$ $Response$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$H$ $command$', '$h$ $non$ $linear$', '$h$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis label
% ylabel('$H$ $(ft)$', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test5_h_Response_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot  r  (rudder control action)
% figure(28)
% plot(out.tout, out.rudder_controller.Data(:, 1), 'r', 'LineWidth', 2); % Nonlinear
% grid on
% grid minor
% hold on
% plot(out.tout, out.rudder_controller.Data(:, 2), '--b', 'LineWidth', 2); % Linear
% title('$\delta_r$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_r$ $non$ $linear$', '$\delta_r$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis label
% ylabel('$\delta_r$ $(degrees)$', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test5_delta_r_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot  a  (aileron control action)
% figure(29)
% plot(out.tout, out.aileron_controller.Data(:, 1), 'r', 'LineWidth', 2); % Nonlinear
% grid on
% grid minor
% hold on
% plot(out.tout, out.aileron_controller.Data(:, 2), '--b', 'LineWidth', 2); % Linear
% title('$\delta_a$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_a$ $non$ $linear$', '$\delta_a$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis label
% ylabel('$\delta_a$ $(degrees)$', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test5_delta_a_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
% % Plot  e  (elevator control action)
% figure(30)
% plot(out.tout, out.elevator_controller.Data(:, 1), 'r', 'LineWidth', 2); % Nonlinear
% grid on
% grid minor
% hold on
% plot(out.tout, out.elevator_controller.Data(:, 2), '--b', 'LineWidth', 2); % Linear
% title('$\delta_e$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_e$ $non$ $linear$', '$\delta_e$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis label
% ylabel('$\delta_e$ $(degrees)$', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, 'Test5_delta_e_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
%
```

```
% % Plot  th  (throttle control action)
% figure(31)
% plot(out.tout, out.Trust_controller.Data(:, 1), 'r', 'LineWidth', 2); %
    Nonlinear
% grid on
% grid minor
% hold on
% plot(out.tout, out.Trust_controller.Data(:, 2), '--b', 'LineWidth', 2); %
    Linear
% title('$\delta_{th}$ $for$ $Lateral$ $\&$ $Altitude$ $Hold$ $Controller$', ...
%     'Interpreter', 'latex', 'FontSize', 15);
% legend('$\delta_{th}$ $non$ $linear$', '$\delta_{th}$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\delta_{th}$ ', 'Interpreter', 'latex', 'FontSize', 12);
% save_path = fullfile(output_folder, '
    Test5_delta_th_for_Lateral_Altitude_Hold_Controller.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);

% % Test 6
% %%
% % Plot    (heading angle) for Mission
% figure(32)
% plot(out.tout, out.psi_states.Data(:, 1), '-.k', 'LineWidth', 2); % Command
% grid on
% grid minor
% hold on
% plot(out.tout, out.psi_states.Data(:, 2), 'r', 'LineWidth', 1); % Nonlinear
% hold on
% plot(out.tout, out.psi_states.Data(:, 3), '--b', 'LineWidth', 2); % Linear
% title('$\psi$ $Mission$', 'Interpreter', 'latex', 'FontSize', 15);
% legend('$\psi$ $command$', '$\psi$ $non$ $linear$', '$\psi$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$\psi$ $(degrees)$', 'Interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% % % save_path = fullfile(output_folder, 'Test6_Mission_psi.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
% % Plot H (altitude) for Mission
% figure(33)
% plot(out.tout, out.H_states.Data(:, 1), '-.k', 'LineWidth', 2); % Command
% grid on
% grid minor
% hold on
% plot(out.tout, out.H_states.Data(:, 2), 'r', 'LineWidth', 1); % Nonlinear
% hold on
% plot(out.tout, out.H_states.Data(:, 3), '--b', 'LineWidth', 2); % Linear
% title('$H$ $Mission$', 'Interpreter', 'latex', 'FontSize', 15);
% legend('$H$ $command$', '$H$ $non$ $linear$', '$H$ $linear$', ...
%     'Interpreter', 'latex', 'FontSize', 10);
% xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 12); % Added x-axis
    label
% ylabel('$H$ $(ft)$', 'Interpreter', 'latex', 'FontSize', 12);
% % Save Figure
% % save_path = fullfile(output_folder, 'Test6_Mission_H.png');
% exportgraphics(gcf, save_path, 'Resolution', 300);
```

9.2 Appendix B Simulink Models

9.2.1 6DOF Euler Angles Simulink Model

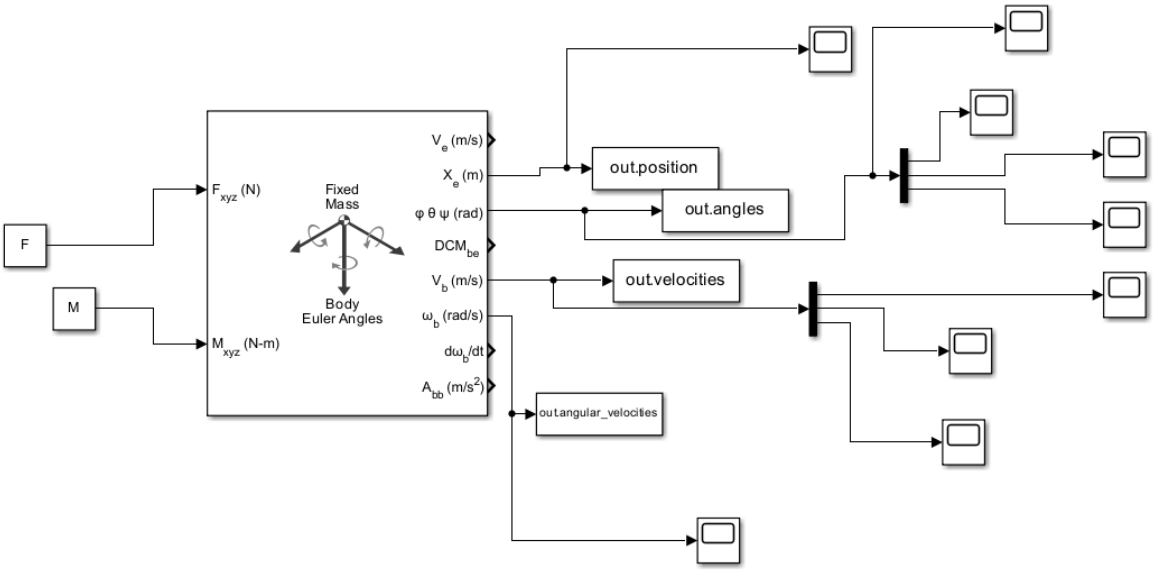


Figure 214: Simulink Model with 6DOF Euler Angles Block for Solving Airplane EOM

9.2.2 Calculation of states Simulink Model

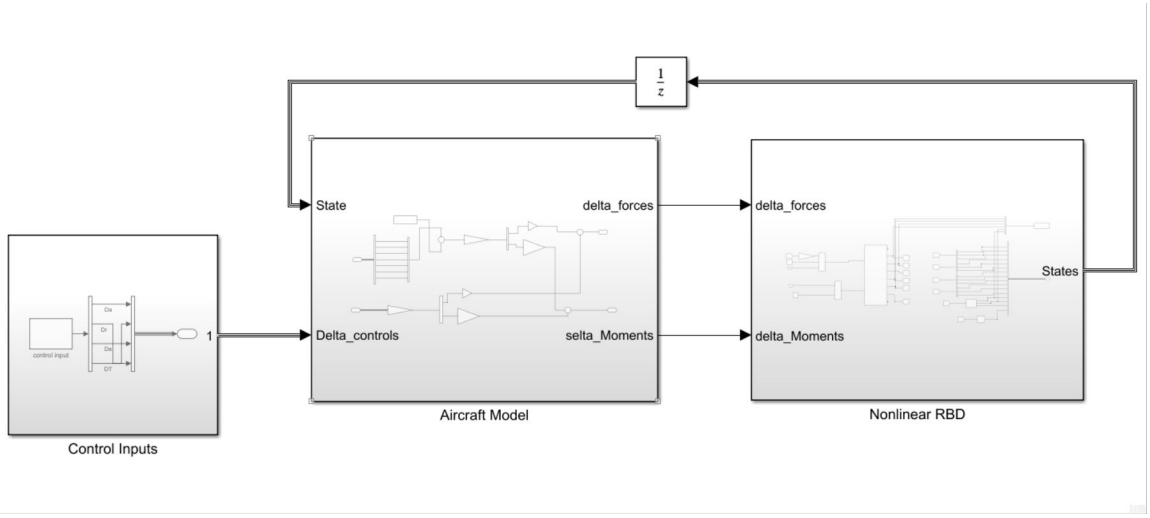


Figure 215: Main Simulink Model with for calculating the states of airplane

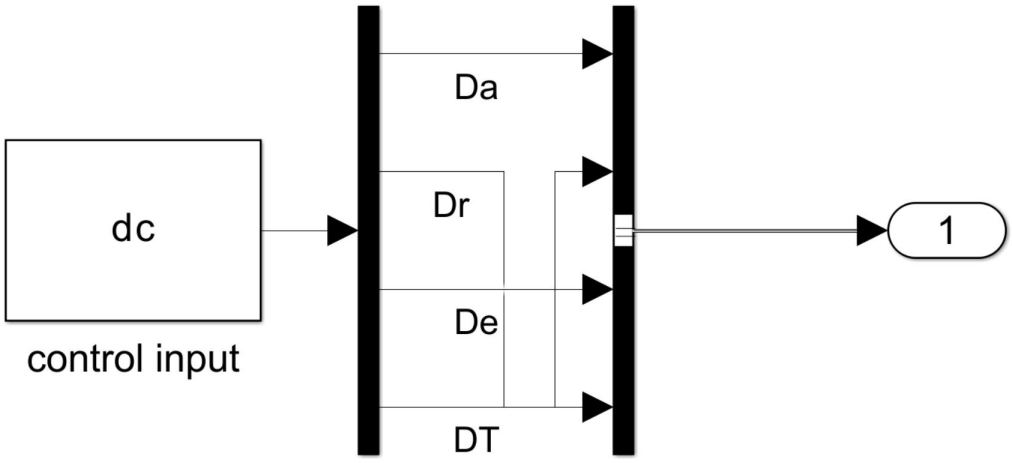


Figure 216: Control inputs sub-block

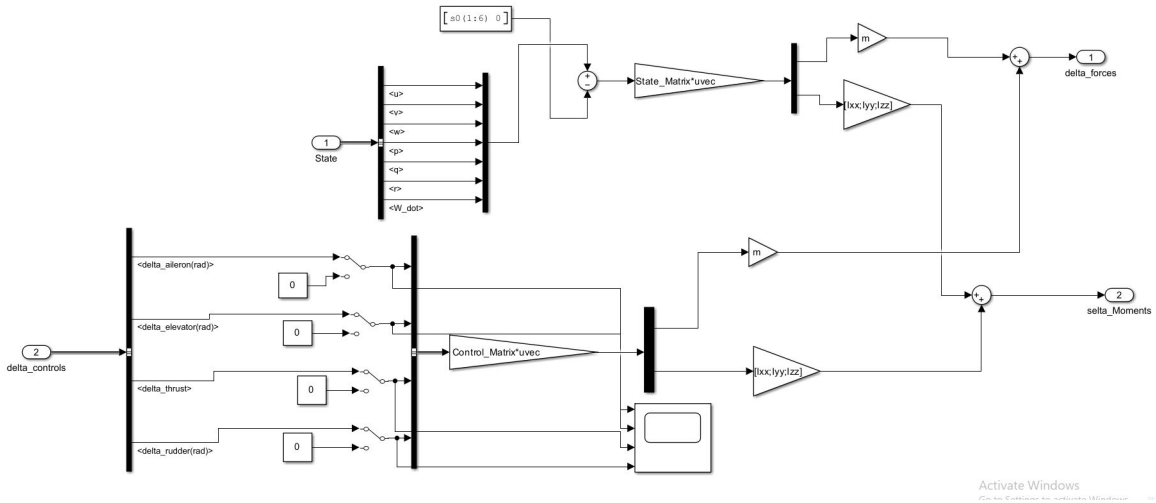


Figure 217: Aircraft Model sub-block

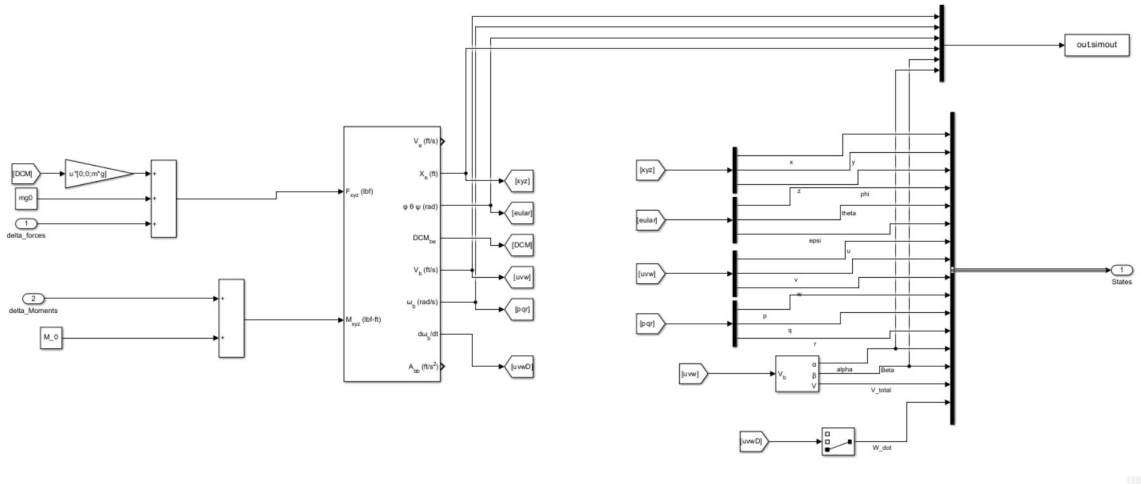


Figure 218: Rigid Body Dynamics Model sub-block

9.2.3 Autopilot Pitch and altitude control

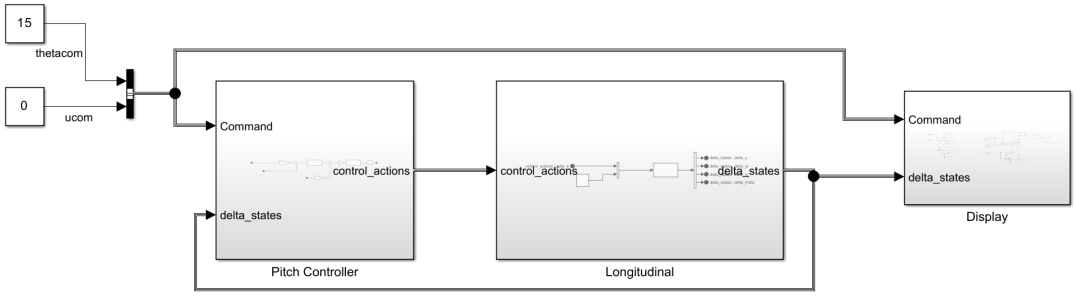


Figure 219: Autopilot pitch control loop

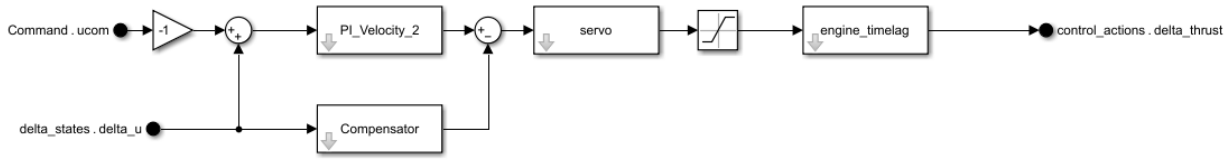


Figure 220: Velocity Controller

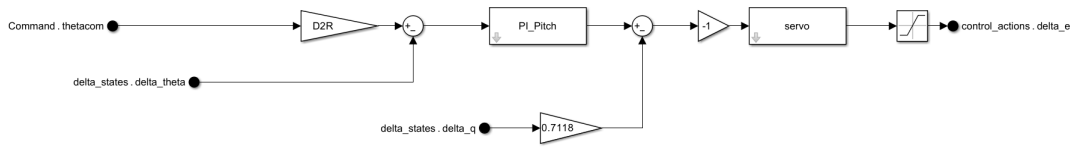


Figure 221: Pitch controller

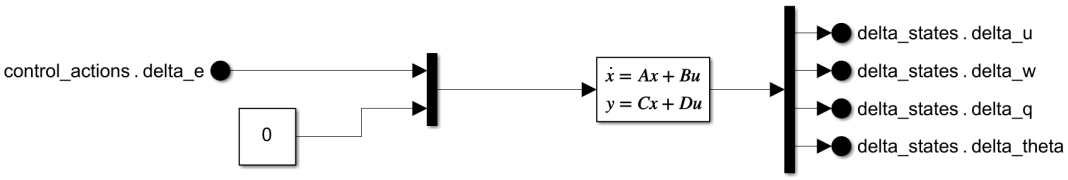


Figure 222: Longitudinal state space

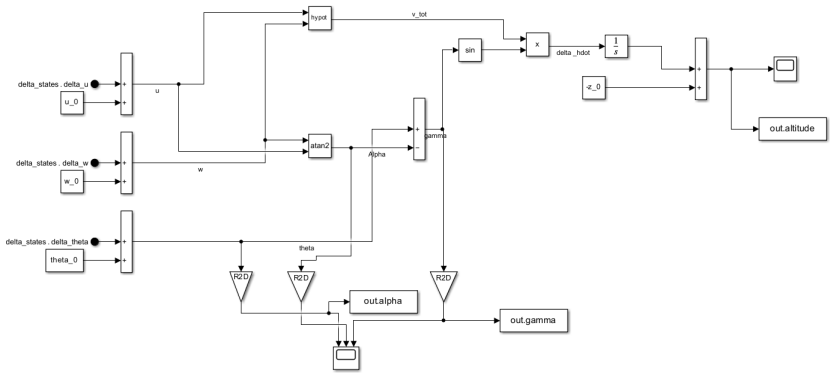


Figure 223: Altitude calculations and display

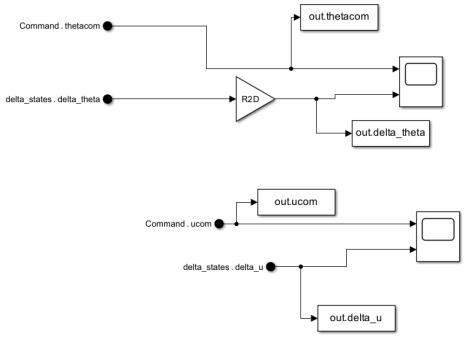


Figure 224: Theta θ and velocity (u) displays

9.2.4 Simulation Model for Linear and Non-Linear

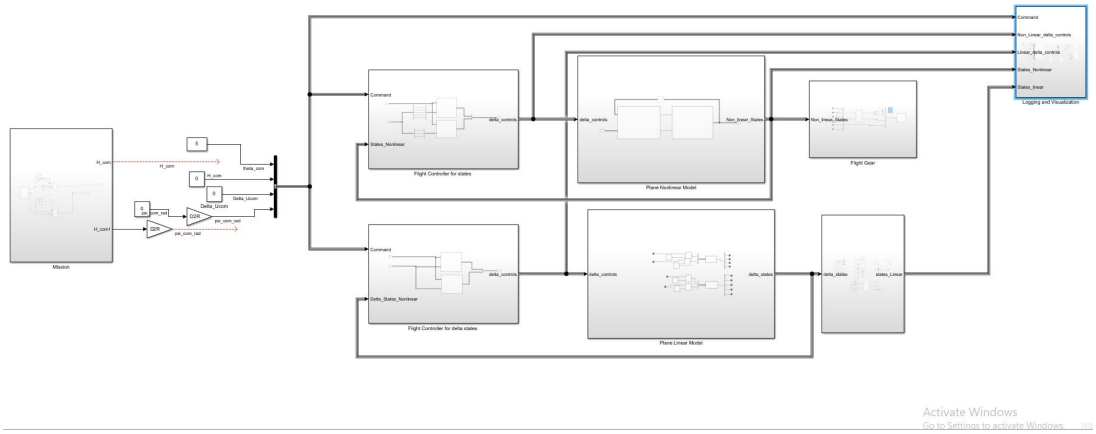


Figure 225: The complete model

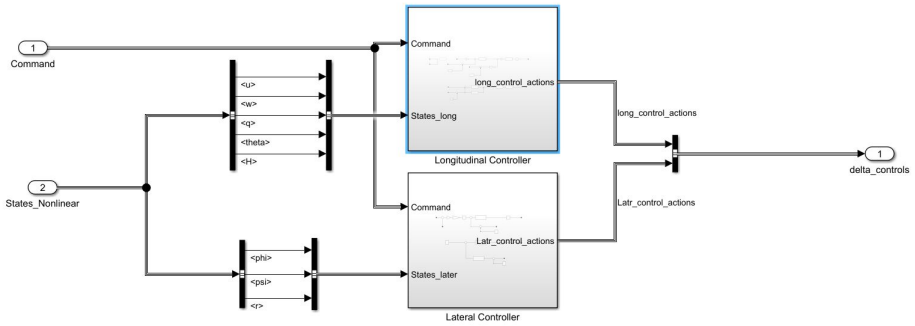


Figure 226: Flight Controller for states

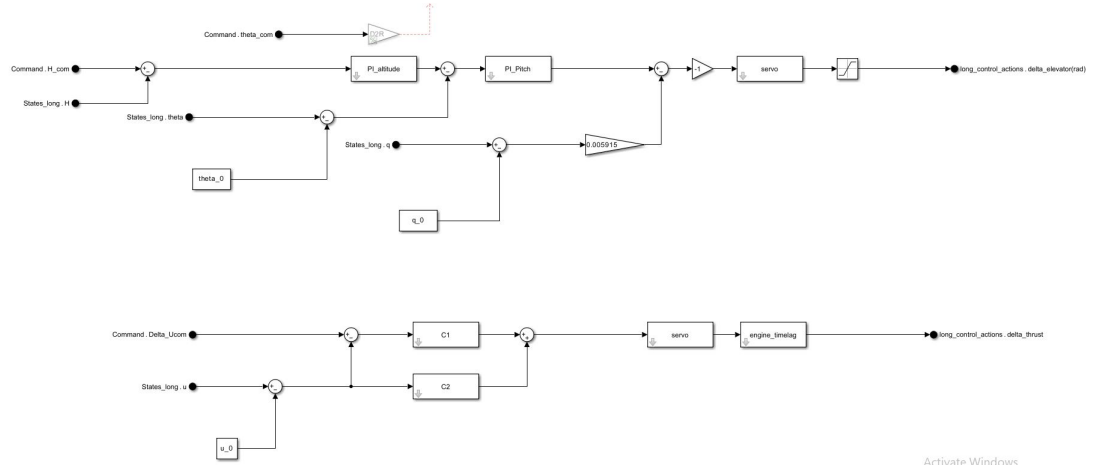


Figure 227: Longitudinal Controller for states

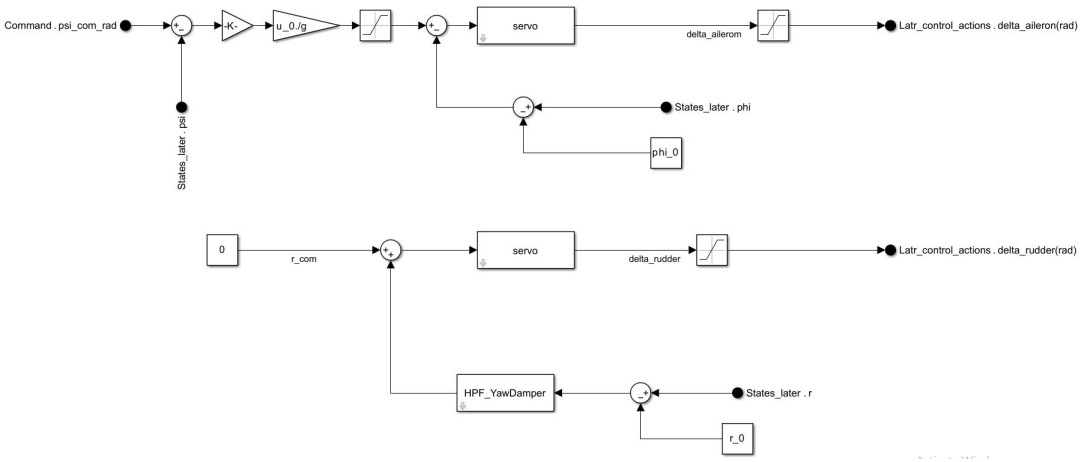


Figure 228: Lateral Controller for states

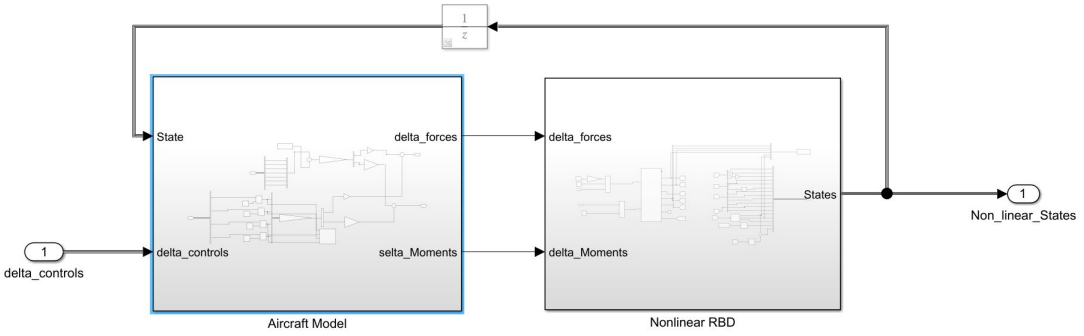


Figure 229: Plane Non-Linear Model

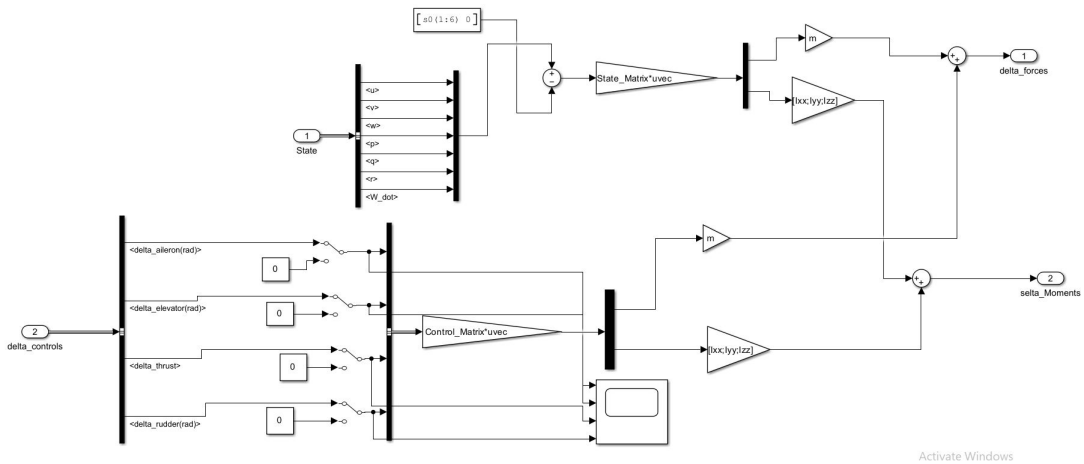


Figure 230: Aircraft Model

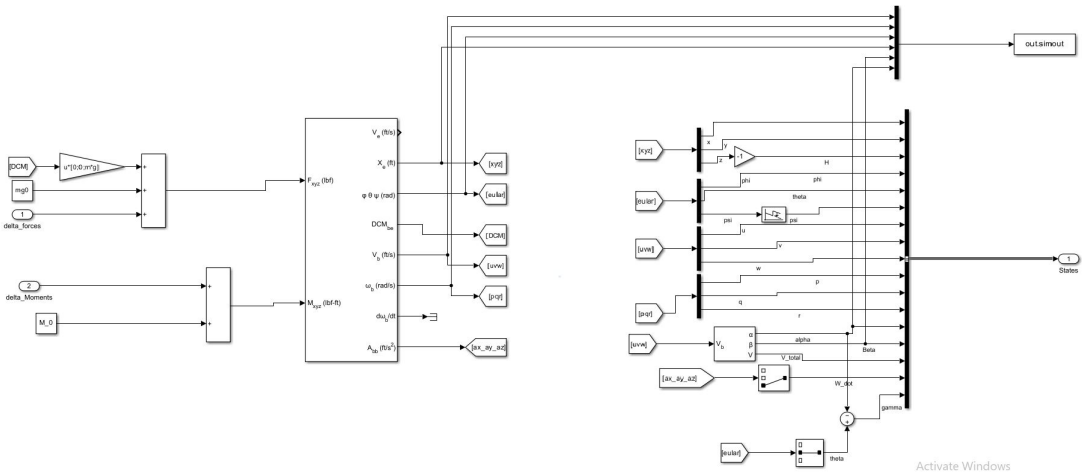


Figure 231: Non-Linear Rigid Body Dynamics

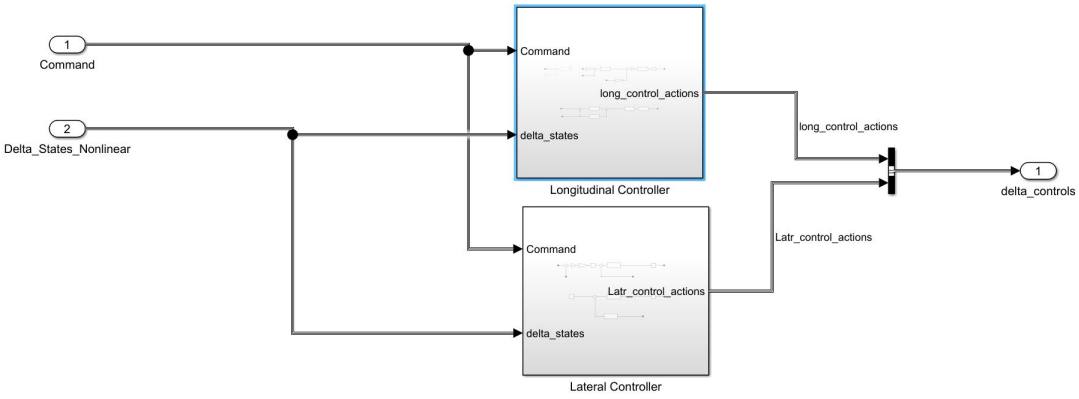


Figure 232: Flight Controller for delta states

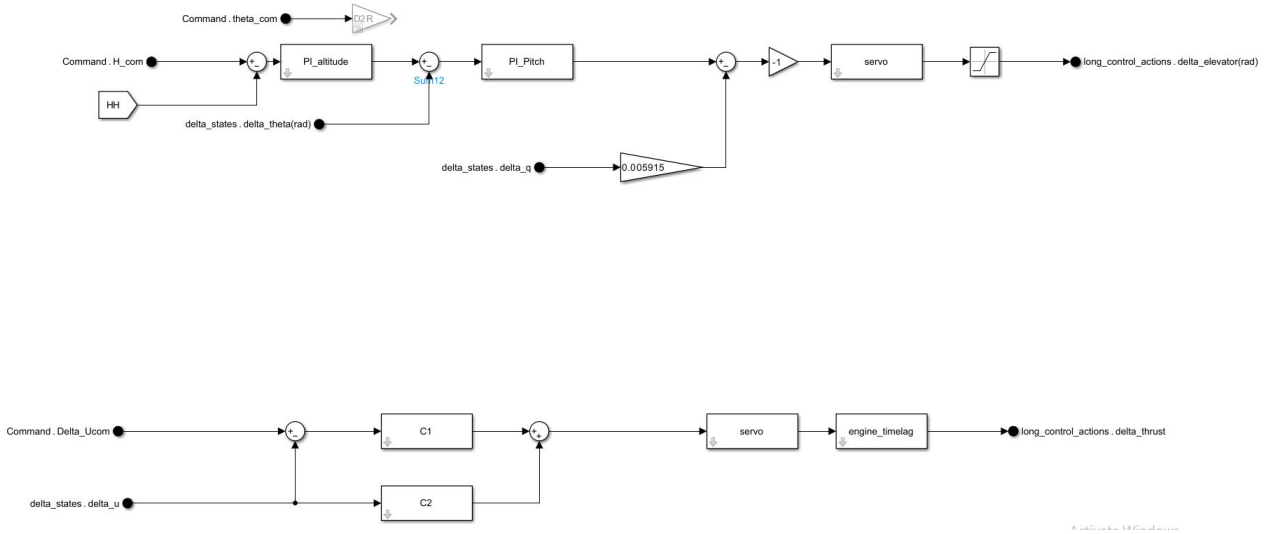


Figure 233: Longitudinal Controller for delta states

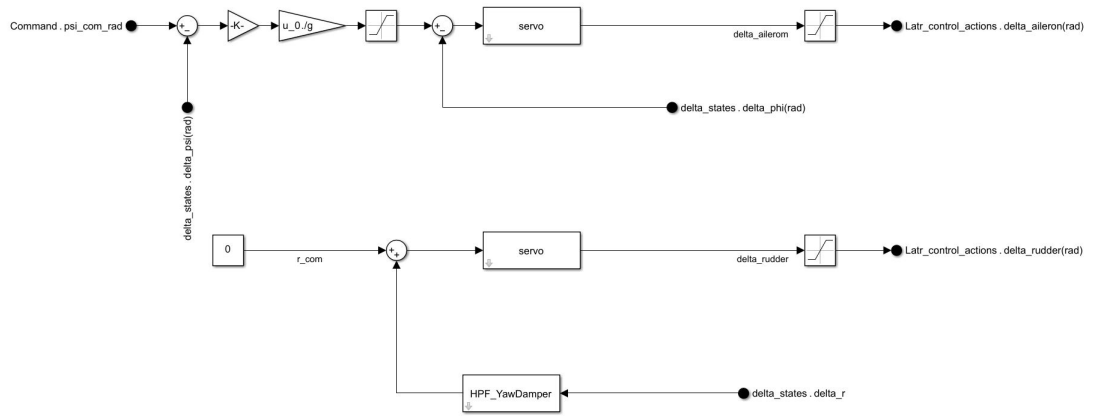


Figure 234: Lateral Controller for delta states

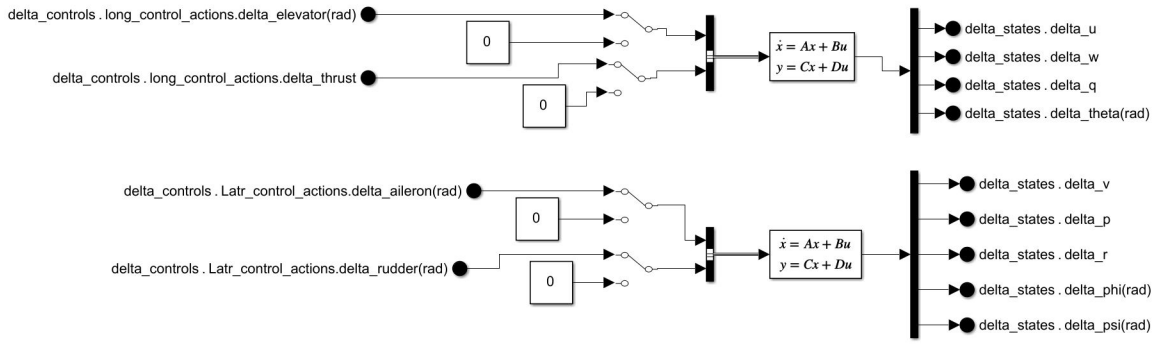


Figure 235: Airplane Linear Model

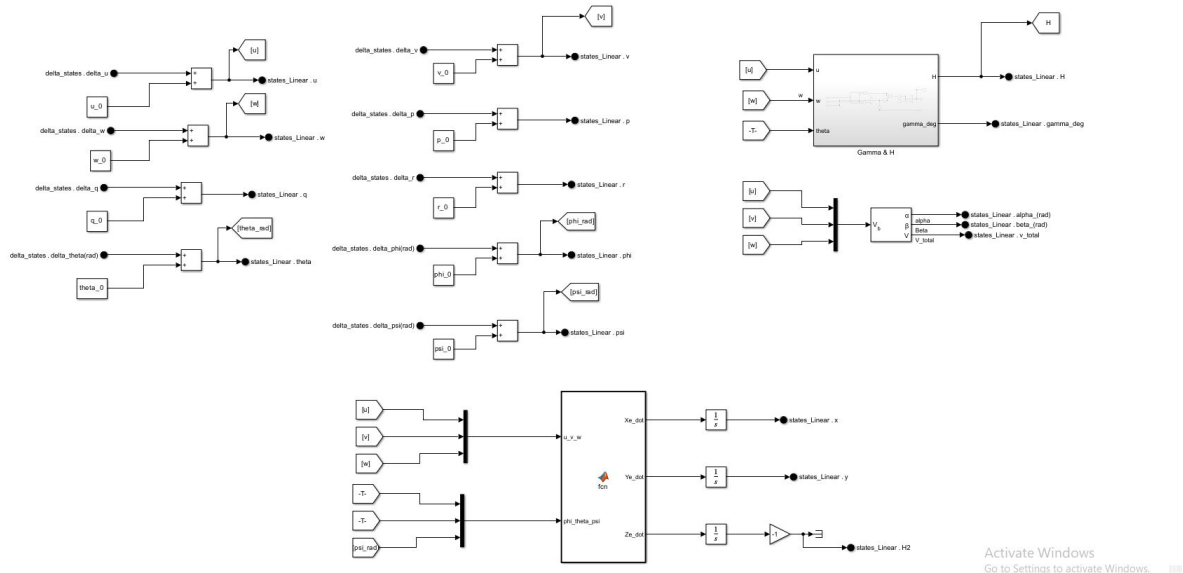


Figure 236: Delta States to States Calculations

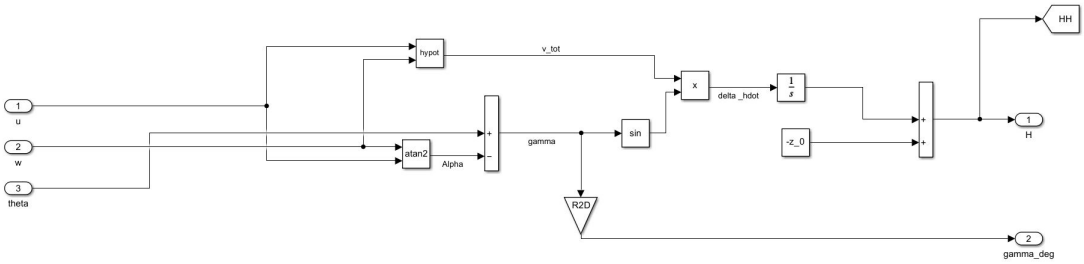


Figure 237: Gamma and H Calculations

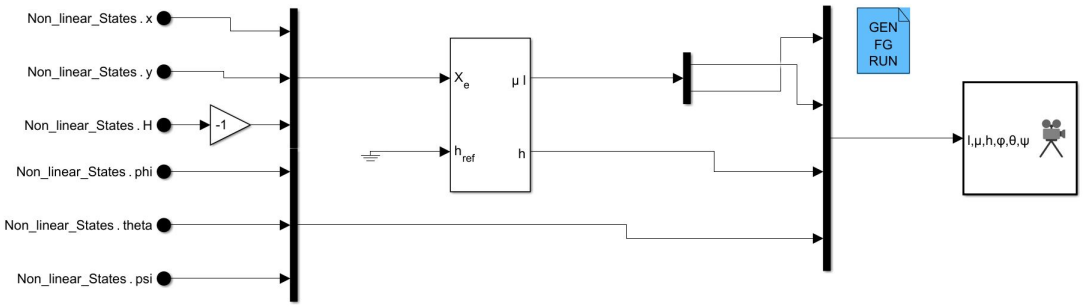


Figure 238: Send Data to Flight Gear

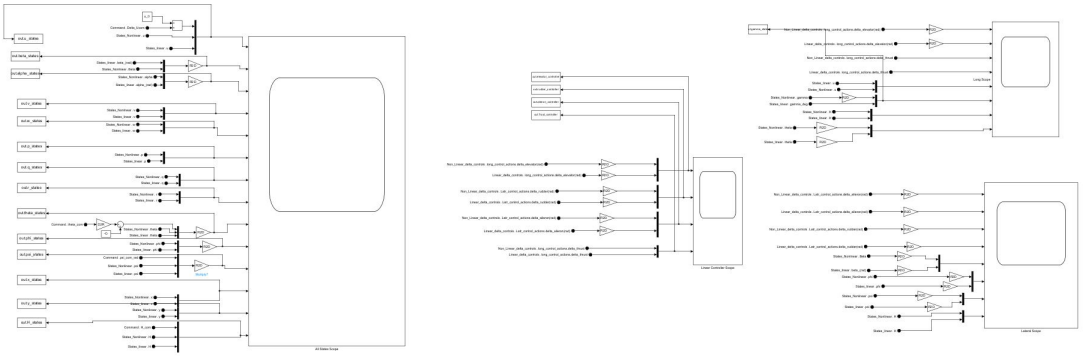


Figure 239: Logging and Visualization

10 Bonus

10.1 Task2 Bonus

Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is a widely used metric for measuring the differences between values predicted by a model or an estimator and the actual observed values. It quantifies the

magnitude of error in a way that emphasizes larger discrepancies. The RMSE is particularly useful when comparing the accuracy of different models or when assessing the quality of a model's predictions.
The formula for calculating RMSE is given by:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}$$

where:

- x_i represents the predicted values,
- y_i represents the observed values,
- n is the number of data points.

The RMSE value is always non-negative, and a lower RMSE indicates a better fit between the predicted and observed values. RMSE is sensitive to outliers, as larger errors have a disproportionately high influence on the overall metric due to the squaring of the differences.

Normalized Root Mean Squared Error (NRMSE)

Normalized Root Mean Squared Error (NRMSE) extends the RMSE by normalizing the error, allowing for a comparison across different datasets or models with varying scales. Normalization provides a relative measure of error, making it easier to interpret the results in context.
The NRMSE is calculated as follows:

$$\text{NRMSE} = \frac{\text{RMSE}}{\max(y) - \min(y)}$$

In this formula:

- $\max(y)$ and $\min(y)$ represent the maximum and minimum observed values, respectively.

By normalizing the RMSE, the NRMSE provides a percentage of the total range of the observed data. This metric is especially useful when evaluating models across different datasets, as it accounts for variations in the magnitude of the data. A lower NRMSE indicates a more accurate model, while values closer to 1 suggest a poor fit.

10.2 Task4 Bouns

10.2.1 Model Linearizer for Longitudinal Mode

This section presents the state-space and transfer function representations of the longitudinal mode using the Model Linearizer.

State-Space Representation

The state-space model for the longitudinal mode is structured as follows:

State Variables: x_1: u -x_2: w - x_3: q - x_4: θ

Input Variables: u_1: δ_e u_2: δ_t

Output Variables: y_1: q - y_2: θ - y_3: u - y_4: w

The complete state-space model for the longitudinal mode is illustrated in Figure 240.

Linearization Result:

A =

	x1	x2	x3	x4
x1	0	0	-27.15	-32.13
x2	0	0	501.3	-1.74
x3	0	0	0	0
x4	0	0	1	0

B =

	u1	u2
x1	1.18	5.05e-05
x2	-21.8	-2.2e-06
x3	-1.4	3.02e-07
x4	0	0

C =

	x1	x2	x3	x4
y1	0	0	1	0
y2	0	0	0	1
y3	1	0	0	0
y4	0	1	0	0

D =

	u1	u2
y1	0	0
y2	0	0
y3	0	0
y4	0	0

Figure 240: State-space representation of the longitudinal mode using the Model Linearizer

Transfer Function Representation The transfer function of the longitudinal mode, derived from the state-space model, is shown in Figure 241. This representation provides an alternative view of the system dynamics in the frequency domain.

Linearization Result:

From input "u1" to output...

$$y1: \frac{-1.4}{s}$$

$$y2: \frac{-1.4}{s^2}$$

$$y3: \frac{1.18\ s^2 + 38.01\ s + 44.98}{s^3}$$

$$y4: \frac{-21.8\ s^2 - 701.8\ s + 2.436}{s^3}$$

From input "u2" to output...

$$y1: \frac{3.02e\text{-}07}{s}$$

$$y2: \frac{3.02e\text{-}07}{s^2}$$

$$y3: \frac{5.05e\text{-}05\ s^2 - 8.199e\text{-}06\ s - 9.702e\text{-}06}{s^3}$$

$$y4: \frac{-2.2e\text{-}06\ s^2 + 0.0001514\ s - 5.255e\text{-}07}{s^3}$$

Figure 241: Transfer function of the longitudinal mode using the Model Linearizer

10.2.2 Model Linearizer for Lateral Mode

This section presents the state-space and transfer function representations of the lateral mode using the Model Linearizer.

State-Space Representation

The state-space model for the lateral mode is structured as follows:

State Variables: x_1: V - x_2: p - x_3: r - x_4: ϕ - x_5: ψ

Input Variables:u_1: δ_a - u_2: δ_r

Output Variables: y_1: ψ - y_2: p- y_3: ϕ - y_4: r - y_5: V

The complete state-space model for the lateral mode is illustrated in Figure 242.

Linearization Result:

A =

	x1	x2	x3	x4	x5
x1	0	27.15	-501.3	32.13	0
x2	0	0	0	0	0
x3	0	0	0	0	0
x4	0	1	0.05416	0	0
x5	0	0	1.001	0	0

B =

	u1	u2
x1	0	11.35
x2	0.229	0.254
x3	0.0285	-0.614
x4	0	0
x5	0	0

C =

	x1	x2	x3	x4	x5
y1	0	0	0	0	1
y2	0	1	0	0	0
y3	0	0	0	1	0
y4	0	0	1	0	0
y5	1	0	0	0	0

D =

	u1	u2
y1	0	0
y2	0	0
y3	0	0
y4	0	0
y5	0	0

Figure 242: State-space representation of the lateral mode using the Model Linearizer

Transfer Function Representation

The transfer function of the lateral mode, derived from the state-space model, is shown in Figure 243. This representation provides an alternative view of the system dynamics in the frequency domain.

Linearization Result:

From input "u1" to output...

$$y1: \frac{0.02854}{s^2}$$

$$y2: \frac{0.229}{s}$$

$$y3: \frac{0.2305\ s}{s^3}$$

$$y4: \frac{0.0285}{s}$$

$$y5: \frac{-8.069\ s^2 + 7.407\ s}{s^4}$$

From input "u2" to output...

$$y1: \frac{-0.6149}{s^2}$$

$$y2: \frac{0.254}{s}$$

$$y3: \frac{0.2207\ s}{s^3}$$

$$y4: \frac{-0.614}{s}$$

$$y5: \frac{11.35\ s^3 + 314.7\ s^2 + 7.092\ s + 1.331e\text{-}14}{s^4}$$

Figure 243: Transfer function of the lateral mode using the Model Linearizer

10.2.3 Comparisons with Manual Linearization

In this section we comparison between the Model Linearizer and manual linearization approaches for both the state-space model and transfer function of the longitudinal and lateral modes. The manual linearization process, detailed in Section 6.

State-Space Model:
In general, the state-space models for both the longitudinal and lateral modes are similar between the Model Linearizer and manual linearization. However, some variations are observed, particularly in the values of the state variables p, q, and r. Notably, the Model Linearizer approximates very small values (e.g., 0.001) as zero, which introduces slight differences in the state variables. These approximations can affect the accuracy of the model, especially in cases where small values are significant.

Transfer Function:
For the transfer functions, both methods yield mostly similar results. However, some differences arise due to the approximations in Simulink, particularly with the values of p, q, and r. The Model Linearizer's tendency to approximate small numbers to zero introduces discrepancies in certain transfer function coefficients, which may impact specific transfer function outputs. Despite these minor differences, the overall behavior of the system remains consistent across both approaches.