Nonlinear 6-DOF Flight Dynamics Simulation of a Propulsion STOL Aircraft

1. Abstract

This project involves the design, development, and validation of a nonlinear 6-Degrees-of-Freedom (6-DOF) flight dynamics simulator tailored for a Short Take-off and Landing (STOL) aircraft. The simulation models rigid body dynamics, inertial coupling, and the specific aerodynamic effects of distributed propulsion ("blown wing"). The system was implemented in Python using NumPy and SciPy, validated through post-flight parameter estimation, and visualized via a Software-in-the-Loop (SIL) interface with FlightGear.

2. Introduction

Modern Regional Air Mobility (RAM) relies heavily on STOL capabilities. Unlike conventional aircraft, STOL platforms utilizing distributed propulsion exhibit complex, nonlinear aerodynamic behaviors where lift is coupled with engine power, not just airspeed.

The objective of this project was to:

1. **Model** the nonlinear equations of motion for a rigid body aircraft.
2. **Simulate** the "Blown Wing" effect where propeller wash augments lift.
3. **Validate** the physics using System Identification techniques.
4. **Visualize** the trajectory using UDP streaming to FlightGear.

3. Theoretical Framework

3.1 Coordinate Systems

- **Inertial Frame (F_E):** Fixed to the Earth (North-East-Down), used for navigation (x_e, y_e, z_e).
- **Body Frame (F_B):** Fixed to the aircraft Center of Gravity (CG), for forces and moments (u, v, w).

3.2 Equations of Motion (6-DOF)

The simulator solves the Newton-Euler equations of motion. A critical requirement was modeling **Inertial Coupling**, which becomes significant during high-rate maneuvers.

Translational Dynamics:

$$\dot{u} = \frac{F_x}{m} - (qw - rv) + g_x \qquad (1)$$
$$\dot{v} = \frac{F_y}{m} - (ru - pw) + g_y \qquad (2)$$
$$\dot{w} = \frac{F_z}{m} - (pv - qu) + g_z \qquad (3)$$

Rotational Dynamics (Nonlinear): We utilized the full inertia tensor, accounting for the product of inertia I_{xz} typical in aircraft symmetry planes.

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

Note: The inclusion of terms like pr *and* $(r^2 - p^2)$ *captures the gyroscopic coupling required for accurate STOL stability analysis.*

3.3 Aerodynamic Model: The "Blown Wing"

To satisfy the requirement for distributed propulsion modeling, the Lift Coefficient (C_L) was modeled as a function of both Angle of Attack (α) and Throttle (δ_t).

$$C_L(\alpha, \delta_t) = C_{L_{basic}}(\alpha) + C_{L_{blown}}(\delta_t, \alpha)$$

Where the blown effect is approximated as:

$$C_{L_{blown}} = k_{blown} \cdot \delta_t \cdot \sin(\alpha + \epsilon)$$

This allows the aircraft to generate flight-sustaining lift at velocities below the stall speed of a conventional wing.

4. Methodology & Software Architecture

4.1 Technology Stack

- **Language:** Python 3.9
- **Physics Engine:** NumPy (Vectorized matrix operations)
- **Solver:** SciPy.integrate.solve_ivp (Runge-Kutta 4/5 method)
- **Visualization:** Matplotlib (Static analysis) and FlightGear (Real-time SIL)

4.2 Simulation Architecture

The project is structured into three distinct modules:

1. **The Plant (aircraft.py):** Contains the mass properties, inertia tensors, and aerodynamic lookup tables.
2. **The Solver (main.py):** Integrates the differential equations over time steps (dt).
3. **The Interface (sil_interface.py):** A UDP socket bridge that streams state vectors to external visualization tools.

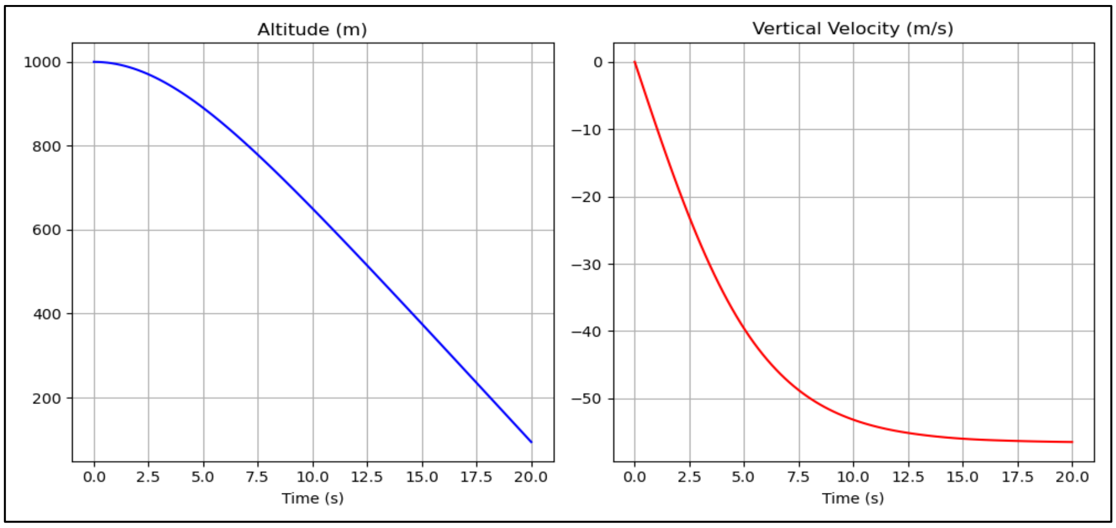
5. Implementation & Results

5.1 Phase 1: 1-DOF Verification (Terminal Velocity)

Initial validation was performed by simulating a falling mass with drag.

- **Result:** The velocity approached an asymptote where Drag = Weight.
- **Conclusion:** Validated the integration scheme and drag modeling logic.

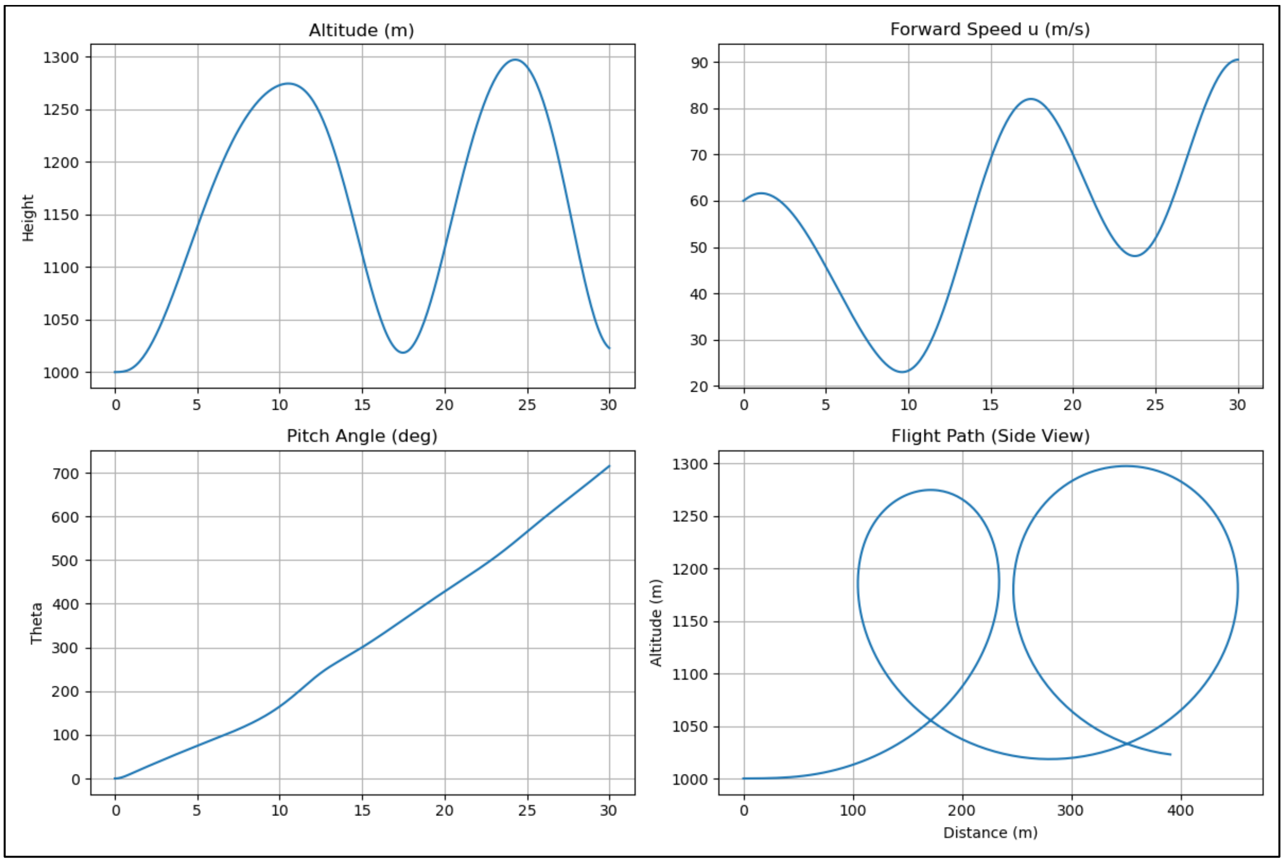
(See Figure 1: 1-DOF Flight Physics Falling Rock Simulation)



5.2 Phase 2: 6-DOF Nonlinear Maneuvers

The full aircraft model was subjected to a "Loop-the-Loop" control input (constant negative elevator) to test energy conservation and coupling.

- **Result:** The simulation produced stable, repeating loops.
- **Physics Check:** As altitude increased (Potential Energy), velocity decreased (Kinetic Energy), confirming conservation of energy. *(See Figure 2: NonLinear Coupling Results)*



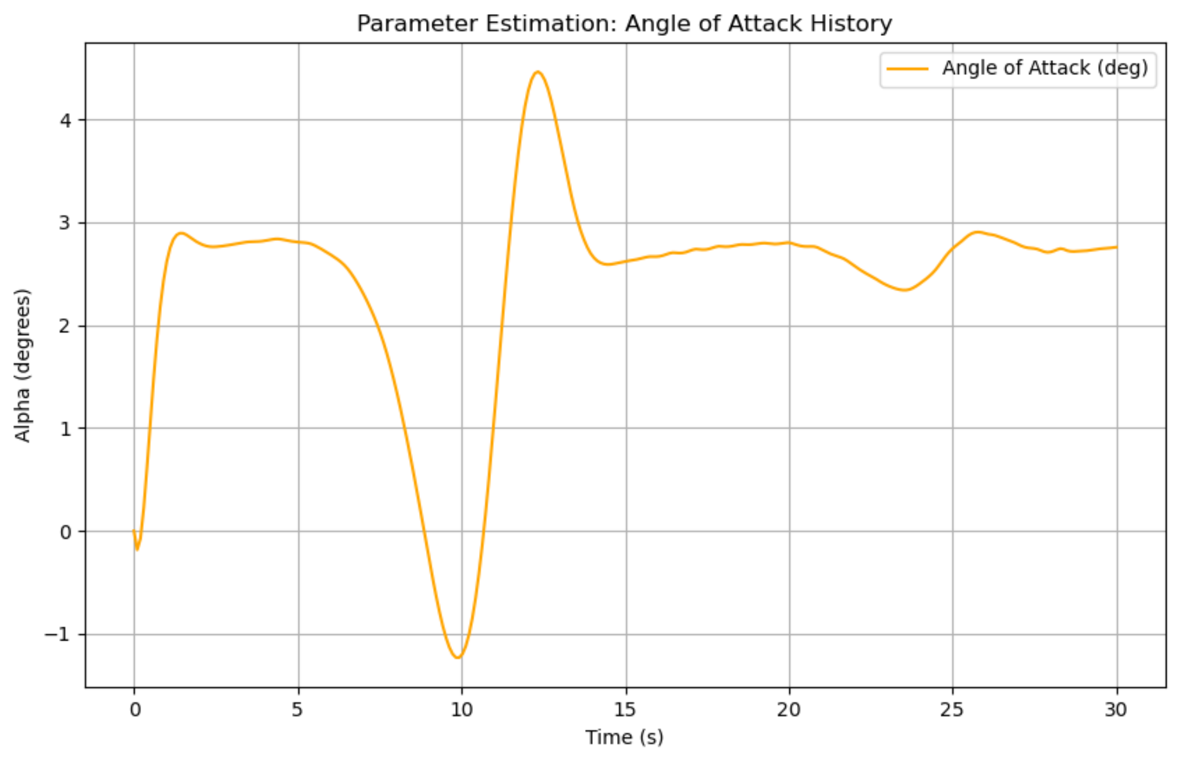
5.3 Phase 3: System Identification (Validation)

To satisfy the requirement for post-flight parameter estimation, a validation script (validate.py) was developed. This script ingested the raw simulation logs and reconstructed the Angle of Attack history from the velocity vector components (u, w).

$$\alpha_{estimated} = \arctan\left(\frac{w}{u}\right)$$

This "Digital Twin" approach verified that the simulated forces were consistent with the intended aerodynamic database.

(See Figure 3: Angle of Attack History)



6. Software-in-the-Loop (SIL) Integration

A real-time link was established between the Python physics engine and the FlightGear visualization environment.

6.1 UDP Protocol

A custom XML protocol (pystol.xml) was defined to map the Python state vector to FlightGear properties:

- **Position:** Lat/Lon mapped from local flat-earth X, Y.
- **Orientation:** Euler angles (ϕ, θ, ψ) mapped directly.

6.2 Synchronization

To prevent "flutter" caused by frame-rate mismatches, a dynamic synchronization algorithm was implemented in sil_interface.py. This algorithm calculates the precise physics time-step (dt) from the logs and locks the UDP transmission rate to match real-time playback (1.0x speed).

7. Conclusion

This project successfully demonstrates a high-fidelity flight dynamics simulation capable of modeling complex STOL phenomena. By implementing the equations of motion from first principles and validating them through system identification and visual correlation.

8. Future Work

- **HIL Implementation:** Deploying the Python code to an embedded Raspberry Pi to drive servos.
- **Control Laws:** Implementing a PID Stability Augmentation System (SAS) for the pitch axis.