6-DOF VLEO Satellite Simulation using Python

1. Summary

This project involves the development of a high-fidelity 6-Degrees-of-Freedom (6-DOF) satellite simulation environment using Python. Unlike simple 3-DOF orbital propagators that only track position (x, y, z), this simulation tracks both **Translational** and **Rotational** dynamics.

The primary objective was to simulate a satellite in a **Very Low Earth Orbit (VLEO)** at 400 km altitude, where atmospheric density is significant. The simulation models realistic disturbance torques (aerodynamic drag interacting with a Center of Pressure offset) and implements an active **Nadir-Pointing Attitude Control System (ACS)** to stabilize the satellite.

2. Core Concepts & Physics Engine

A. Orbital Mechanics (Translation)

The simulation solves Newton's law of universal gravitation to propagate the orbit.

- **Equation:** $F_{\text{grav}} = -\frac{\mu m}{r^3} r$
- **Integrator:** scipy.integrate.solve_ivp is used to numerically solve the differential equations, ensuring high precision over long durations (e.g., 5+ hours).

B. Attitude Dynamics (Rotation)

The satellite's orientation is tracked using **Quaternions** to avoid "Gimbal Lock" (a common issue with Euler angles).

- **Dynamics:** Euler's rotation equations are used: $\tau = I\dot{\omega} + \omega \times (I\omega)$
- **Disturbances:**
  - **Gravity Gradient:** The variation in gravitational pull across the satellite's body.
  - **Aerodynamic Torque:** The "Spice" of this project. Drag force is applied at the Center of Pressure (CoP), which is offset from the Center of Mass (CoM), creating a twisting torque $\left(\tau = r_{offset} \times F_{drag}\right)$

C. Control System

An active **PD (Proportional-Derivative) Controller** is implemented to achieve "Nadir Pointing."

- **Goal:** Keep the satellite's Z-axis sensor locked on the Earth's center while flying over the poles.
- **Mechanism:** The controller calculates the error between the current quaternion and the target quaternion, then applies corrective torque to nullify the error.

3. Code Breakdown

Based on your file structure, the project consists of three distinct implementations:

Code 1: sat_vtk_visualisation.py

- **Description:** This version uses the **Visualization Toolkit (VTK)** for high-end rendering.
- **Key Features:**
    - Loads an external .stl mesh (sat.stl) to visualize the satellite geometry.
    - Renders a 3D Earth with continental outlines and a wireframe grid.
    - **Pros:** Best visual fidelity; professional-grade graphics pipeline.
    - **Cons:** Requires the heavy vtk library dependency; more complex setup.

Code 2: sat_matplotlib_visulaisation.py

- **Description:** A lightweight, pure-Python implementation using matplotlib.
- **Key Features:**
    - Replaces complex 3D meshes with a high-performance marker (Red Dot) to ensure real-time rendering on standard hardware.
    - Focuses on the orbital trajectory (Yellow Trail) and altitude metrics.
    - **Pros:** Runs on any machine with standard Python libraries; zero lag; excellent for quick physics debugging.
    - **Cons:** Less visually detailed than VTK.

Code 3: sat_realistic_orientation_lock.py (The "Spiced" Version)

- **Description:** The advanced simulation that combines complex physics with intuitive visualization.

- **Key Features:**
    - **VLEO Physics:** Enables the "Spice" (Aerodynamic Torque) caused by the thick atmosphere at 400 km.

- **Attitude Arrows:** Renders dynamic vectors (Red/Blue arrows) showing exactly where the satellite is facing versus where it *should* be facing.
- **Earth Rotation:** Simulates the Earth's sidereal rotation, allowing visualization of ground track shift over multiple orbits.
- **Purpose:** To demonstrate the Attitude Control System fighting against atmospheric disturbances in real-time.

4. Conclusion

This project successfully demonstrates the coupling between orbital mechanics and attitude dynamics. It highlights the challenges of VLEO missions—specifically the need for constant active control to counteract aerodynamic instability. The transition from VTK to Matplotlib ensures the simulation is both visually impressive and computationally accessible.