UAV Deconfliction System

Design & Architecture

I have used Python as the programming language due to its ease of integration with graphical GUI libraries. The code follows a modular architecture, where different functionalities are encapsulated into separate modules, all of which are eventually invoked in the main.py file.
The key modules include:

- **Temporal Check**: Handles time-based conflict detection.
- **Spatial Check**: Ensures spatial consistency and collision avoidance.
- **Data Loading**: Manages dataset handling and preprocessing.
- **Dataset Generation**: Creates test datasets dynamically for robust evaluation.
- **Visualization**: Plots drone trajectories in shared airspace.

For visualization, I have used **Matplotlib** and **Plotly**, enabling interactive and static visual representations of drone movements. To simplify logic and minimize dataset complexity, drone paths are assumed to follow **linear interpolation**.

Project Structure

The project directory consists of the main source code along with two key folders:

- test/ – Contains datasets and scripts to generate random test cases for robust evaluation.
- resources/ – Stores videos, GIFs, images, and HTML files necessary for visualization.

A detailed directory structure is provided in the **README.md** file for better understanding.

Execution & Output

Running main.py generates a detailed report highlighting:

1. Conflict detection results.
2. Locations and timestamps of potential collisions.
3. The primary drone's collision details, including proximity and impact assessment.

This modular approach ensures maintainability, flexibility, and scalability for future enhancements.

Spatial & Temporal Checks

Spatial Conflict Check (spatial_check.py) :- The is_spatial_conflict function evaluates whether two positions violate a predefined safety buffer.

- Input:
  - pos1: Position of the first drone (tuple with at least 3 coordinates).
  - pos2: Position of the second drone.
  - buffer: Minimum safe distance between the two drones.
- Logic: Uses the Euclidean distance formula to calculate the separation between drones. If the distance is less than the safety buffer, a conflict exists.

- Output: Returns True if a conflict is detected, otherwise False.

Temporal Conflict Check (temporal_check.py) :- The *is_temporal_conflict* function determines whether two time windows overlap. This is crucial for detecting potential conflicts in shared airspace.

- Input: Two time windows, each represented as a tuple of start and end times: (t_start, t_end).
- Logic: If one window starts after the other ends, they do not overlap. Otherwise, a conflict exists.
- Output: Returns True if a conflict is detected, otherwise False.

Future AI Integration

AI can enhance the simulation with **predictive conflict detection, autonomous collision avoidance, real-time decision-making, continuous improvement,** and **multi-sensor fusion**, ensuring safer and smarter airspace management.

Testing & Edge Cases

Standard Tests :- Validated core functionality under normal conditions:

- Single Drone: Smooth trajectory generation & visualization.
- Multiple Drones (No Conflicts): Correct path rendering.
- Basic Conflict Detection: Accurate identification.

Edge Cases :- Ensured robustness against extreme scenarios:

- High-Density Traffic: Optimized interpolation (300 → 150 points) & spatial partitioning.
- Hovering Drones: Refined conflict logic to include stationary drones.

Scalability

The current testing approach works well for small-scale simulations but may face challenges as drone count increases. To enhance scalability, optimizations like parallel processing, spatial indexing, and efficient data handling will be essential for handling large-scale airspace scenarios.

- Modular Tests: Scalable unit tests for trajectory (position_at()) & conflict detection.
- Edge Case Coverage: Robust handling of zero-speed drones & boundary conditions.
- Visual Validation: Effective manual conflict checks for ≤10 drones.
- Data Handling: Used NumPy and Pandas for efficient and scalable data handling in the simulation.