UAV Strategic Deconfliction in Shared Airspace

Reflection and Justification Document

Design Decisions & Architectural Choices

I have chosen the design explained below for its modularity, simplicity in creating & updating the 3D simulation and its resemblance to real world system. Like getting a schedule from outside the program, a stand-alone interface for conflict detection, updating drone position based on its state and using lerp to simulate its steady movement in the simulation. I have not integrated any AI features as using geometric methods ensures reliability and explainable behaviour.

The system is mainly designed in 3 parts,

1. The Simulation (Simulator.py)
2. The Conflict Detection (Conflict_Detector.py)
3. The Control System (main.py)

The Simulator is responsible for generating the 3D graph, drawing features like – drone positions, ground, docks, routes and the simulation time. Mathplotlib handles the 3D graph, plotting the features, and the mouse control features.

The Conflict Detection checks for any conflicts in the given schedule. It works by turning the route of a drone into a set of segments and then checking if any segments conflict with segments of another route.

> **Temporal Check:** Using the start and the end time of all the route, It creates a list of active routes for any point in time and then check if the routes in the same time window are close to each other or not.
>
> **Spatial Check:** The spatial check is performed in 2 stages. First the aabb_overlap() function checks if any of the x, y or z coordinates overlap within the MIN_DISTANCE buffer. If they do not overlay, it means the segments are far enough from each other
>
> If they do overlay each other, then they are checked in the second stage segment_distance(). Here we find the shortest distance between the 2 segments.

This 2-stage checking helps improve the speed for checking conflicts by avoiding the segments which are too far from each other.

The Control System is responsible for getting the schedules, checking for conflicts in the schedule with the conflict detector, initiating the simulator, calculating and updating the new positions of the drone using interpolation time and state tracking.

includes the Visualization and redrawing the updated positions for the drones, the control based of the given schedule. The conflict detection acts as an interface to check if the routes in the schedule are safe and do not conflict.

Testing Strategy

The testing is done for an overall schedule, with 3 test cases, which are included in the schedules folder.

Testing was done for:

- Unit-level validation of distance calculations and AABB overlap logic
- Controlled schedule testing using predefined JSON files with known conflict and non-conflict cases
- Visual verification via the simulator to confirm correct drone movement and timing behaviour

Edge cases considered:

- Routes with overlapping time windows but large spatial separation
- Routes that nearly touch the minimum distance threshold

Scalability Discussion

1. Improving the algorithm for faster and more reliable checks on larger no. of routes. Like using a 3D grid-based system where the cell size is the buffer distance, the cells through which the route passes is marked as active. This helps avoid heavy computation to find distance between the trajectories.
2. AI and pathfinding algorithms can be implemented in the 3D grid system to perform certain tasks or to find a path return to the dock when the battery runs out without disturbing the other drones route.
3. Parallel processing – with large no of active routes checking for multiple routes for conflicts increases efficiency and reduce delay.
4. Live changes to schedules and routes, instead of using a fixed JSON file use live streaming to update schedules and routes while the system is running.