UAV Strategic Deconfliction in Shared Airspace

1. Introduction

This document details the design, implementation, testing, and potential future enhancements for the UAV Strategic Deconfliction System developed as part of the FlytBase Robotics Assignment 2025.

2. Design Decisions and Architectural Choices

The primary objective of this system is to act as a final authority for verifying the safety of a primary drone's planned waypoint mission against the flight paths of multiple simulated other drones in shared airspace. The system must check for conflicts in both space and time.
The core architecture revolves around a central deconfliction service that receives the primary drone's mission details and the flight schedules of simulated drones. The key design decision was to process all drone trajectories offline before the primary mission begins, allowing for a comprehensive check against all known potential conflicts within the mission's time window. This offline check provides a "clear" or "conflict detected" status before takeoff.
The system is structured modularly, and key modules include:

- Data Ingestion/Preparation: Handling the input of primary drone waypoints, time window, and secondary drone waypoint/time step data. Functions like **generate_primary_trajectory** and **generate_secondary_trajectories** are responsible for converting waypoint lists and time data into standardized time-sequenced trajectories.
- Trajectory Representation: Representing trajectories as a list of time-sequenced (x, y, z, t) tuples. Linear interpolation is used to generate points between specified waypoints/time steps for smoother simulation and more granular conflict checking.
- Conflict Detection Logic: Implementing the core spatial and temporal checks. This is handled by functions like **check_intersection**, which identifies periods of temporal overlap between trajectories, and **determine_conflict**, which applies the spatial buffer distance check within those temporal overlaps. The **get_position_at_time** function is a critical helper for querying drone positions at specific time points along their trajectories.
- Conflict Reporting: Generating detailed explanations of detected conflicts, including location, time, and involved secondary drones.

- Visualization & Simulation: Providing a visual representation of the primary mission, simulated drone trajectories, and highlighting conflict locations and times. The **UAVDeconflictionVisualizer** class encapsulates the logic for setting up and running a 3D animation using matplotlib. This includes displaying drone positions, trails, timestamps, and conflict markers.
- Reporting/Saving Results: Functionality to save conflict details to files, such as text or CSV formats.

This modular architecture facilitates testing, maintenance, and potential future expansion. The use of 3D coordinates and time (4D space-time) was incorporated for the extra credit component.

3. Implementation of Spatial and Temporal Checks

- Temporal Check: The primary temporal check is performed by the **check_intersection** function. It determines the overlapping time window between the primary drone's trajectory time range and each secondary drone's trajectory time range. If there is no temporal overlap (start_time >= end_time), no conflict is possible between those two drones. If an overlap exists, the function generates a series of time samples within this overlapping window.

- Spatial Check: The spatial check is performed within the temporal overlap found by **check_intersection**. The **determine_conflict** function takes the potential intersection points (time samples and corresponding drone positions) and iterates through them. For each time point, it calculates the Euclidean distance between the primary drone's position and the secondary drone's position using the **calculate_distance** function. A conflict is detected if this calculated distance is less than a defined safety buffer distance. The buffer distance used in the example code and test suite is 2.0 units.

The conflict detection is thus a two-step process: first, identify if drones are operating in the same time window, and then check if they come too close spatially during that overlap. Detected conflicts are recorded with their specific time, location, distance, and the involved secondary drone's ID.

4. Testing Strategy and Edge Cases

The approach taken involves developing an automated test suite to cover various scenarios. This suite defines multiple test cases, each with a primary mission, secondary flight schedules, and an expected outcome.

The automated test suite iterates through these predefined cases, executes the deconfliction logic for each, and then compares the detected result against the expected result for that case. The results (PASS/FAIL/ERROR) and conflict details are logged to timestamped files, typically in CSV format, for review.

- Test cases include:
  The code includes error handling for conditions like insufficient waypoints or mismatched waypoint/timestamp lists. The trajectory generation and position retrieval functions also handle cases where a requested time is outside a drone's active time window.
  - Clear Mission: No expected spatial or temporal overlaps resulting in conflicts.
  - Direct Collision: Drones intended to occupy the exact same space at the exact same time (these should be flagged).
  - Buffer Zone Boundary Cases: Drones passing close to the buffer distance threshold, both just inside (should conflict) and just outside (should not conflict).
  - Temporal Separation: Drones occupying the same spatial region but at significantly different times, with no overlap in their active mission time windows.
  - Vertical Conflicts: Conflicts occurring primarily due to altitude (Z-axis) proximity.
  - Parallel Paths: Drones flying parallel to each other at various separation distances.
  - Late/Early Starting/Ending Drones: Drones whose mission time windows only partially overlap or are outside the primary drone's window.
  - Zero-Length Segments/Stationary Drones: Waypoints that are identical, representing a drone hovering at a specific location.
  - Complex 3D Maneuvers: Trajectories involving changes in all three dimensions simultaneously.
  - High-Density Traffic: Cases with multiple secondary drones potentially interacting with the primary drone.

5. Use of AI and Resourcefulness

Aligned with the assignment's expectation to utilize AI-assisted tools to expedite development within the timeframe, various AI platforms were strategically integrated into the project workflow. While the core logic algorithm was designed manually, AI tools facilitated rapid implementation, testing, and documentation.

The process involved several key applications of AI:

- Boilerplate Code Generation: “Gemini” was first used to formulate a detailed prompt based on the manual core logic. This prompt was then provided to “Claude AI” to generate the initial boilerplate code structure for the system, significantly accelerating the setup phase.
- Debugging and Refinement: As development progressed, DeepSeek was employed to assist in identifying and resolving bugs and glitches encountered in the codebase, contributing to the system's robustness and error handling.
- Test Case Design: DeepSeek was also leveraged to generate a variety of test cases, helping to ensure comprehensive coverage of different conflict scenarios and supporting the Test Case Design requirement.
- Documentation Assistance: Finally, NotebookLM was utilized to aid in structuring and drafting project documentation, including sections of this reflection and justification document.

Crucially, all AI contributions underwent critical evaluation and refinement. Code snippets and suggestions from AI were rigorously reviewed against the established design principles and assignment requirements. Debugging solutions were validated through testing before integration, and AI-generated test cases were examined to ensure their relevance and coverage. This iterative process of leveraging AI for speed and scale while maintaining manual oversight and validation ensured the quality and correctness of the final system, demonstrating resourcefulness in utilizing modern development aids.

6. Scalability Discussion

Scaling this system to handle real-world data from tens of thousands of commercial drones would require significant architectural changes and enhancements. The current implementation, which appears to load all trajectories into memory and perform pairwise checks in a single process, would quickly become computationally prohibitive and exhaust memory resources with such a large number of drones and potentially more granular trajectories.

Key requirements for scaling would include

- Distributed Computing: Distributing the workload across multiple servers or nodes. Conflict checks between different pairs of drones can often be performed in parallel. This could involve frameworks like Apache Spark or Dask.
- Spatial Indexing: Implementing spatial data structures (e.g., k-d trees, R-trees, quadtrees/octrees) to quickly query for drones whose trajectories are geographically close to the primary drone's path within relevant time windows. This would drastically reduce the number of pairs that need to be checked.

- Temporal Indexing: Combining spatial indexing with temporal data structures or using spatio-temporal indexing techniques to efficiently query for drones that are both spatially and temporally near the primary drone.
- Real-time Data Ingestion Pipelines: Developing robust pipelines to ingest flight plan data from tens of thousands of drones continuously or near real-time. This would require systems capable of handling high data throughput and ensuring data freshness.
- Fault Tolerance and High Availability: The system must be resilient to failures. This would involve redundant servers, data replication, and mechanisms for handling node failures without interrupting the service.
- Scalability of Conflict Resolution Algorithms: While the current system only detects conflicts, a real-world system might also need to resolve them (e.g., suggesting alternative routes). These resolution algorithms would also need to be scalable to handle complex scenarios involving many drones simultaneously.
- Database/Data Storage: Migrating from in-memory storage and file-based data to a scalable database solution capable of storing and querying large volumes of spatio-temporal trajectory data efficiently.

In summary, scaling would transform the system from a single-process simulation into a distributed, highly available service leveraging advanced indexing techniques, real-time data processing, and potentially more sophisticated, scalable conflict resolution algorithms.