Reflection Document - Drone Conflict Detection & Visualization

1. Project Overview

This project was developed as part of the FlytBase internship assignment to create a strategic deconfliction system. The system's objective is to serve as a final authority for verifying a drone's mission for safety in shared airspace. The solution is implemented as a flybase assignment.

4D simulation, which checks for conflicts in 3D spatial coordinates (x, y, z) over time. This comprehensive analysis ensures that the primary drone's planned mission is safe from potential collisions with other simulated flights.

2. Design Choices

- **Modular Architecture**: The codebase is logically organized into dedicated modules, which was a key design decision to ensure the solution is scalable and maintainable.

  io.py handles data ingestion, trajectory.py manages path generation, detector.py performs

  the core conflict checks, and visualizer.py is responsible for all animation and plotting.

- **Data Format**: JSON was chosen for mission data due to its simplicity, human readability, and ease of parsing in Python.
- **Trajectory Modeling**: For simplicity and clear visualization, a constant velocity model was assumed between waypoints. This allowed for straightforward linear interpolation to create a continuous trajectory for all drones.
- **4D Visualization**: Matplotlib was selected for visualization due to its powerful 3D plotting capabilities, which were essential for demonstrating the 4D (3D space + time) nature of the project and the spatiotemporal evolution of conflicts.

3. Testability and Quality Assurance

My testing strategy involved creating distinct scenarios to validate the system's core functionality. I designed input JSON files to represent both conflict-free missions and missions with multiple intersections at the same spatial and temporal points. This allowed me to verify that the detector.py module correctly identifies and logs conflicts. The codebase also includes proactive error handling for potential failure modes and edge cases, such as an invalid JSON structure or a mission with fewer than two waypoints.

4. Scalability Discussion

To handle real-world data from tens of thousands of commercial drones, the current architecture would require several enhancements. The current brute-force conflict check has a complexity of O(n^2), which is not scalable. Architectural changes would include:

- **Spatial Indexing**: Implementing a spatial indexing data structure, such as a **KD-Tree** or **Quadtree**, would allow for more efficient proximity queries, speeding up conflict detection.

- **Distributed Computing**: A distributed computing framework (e.g., Dask, Apache Spark) would be necessary to distribute the trajectory generation and conflict analysis workloads across multiple machines.
- **Real-Time Data Ingestion**: The system would need a real-time data ingestion pipeline to process live drone telemetry and flight plan updates, rather than relying on static JSON files.

5. Tools and Resources Used

- **Programming Language**: Python 3.x
- **Libraries**: NumPy (for efficient numerical calculations), Matplotlib (for 4D visualization)
- **Development Environment**: VS Code
- **AI Assistance**: AI tools were used to aid the development process. Specifically, an AI assistant was used for code suggestions and, most importantly, for debugging the complex animation logic in

visualizer.py. The AI's contributions were critically evaluated and refined to ensure they met the project's specific requirements.

6. Lessons Learned

This project was a significant learning experience. I gained a deeper understanding of spatio-temporal data processing and the challenges of modeling and visualizing dynamic systems. A key lesson was in structuring a complete software project with clear, modular components and appreciating the complexity of creating a stable 4D animation in Matplotlib. The iterative debugging process for the visualization component was particularly insightful, highlighting the importance of robust code even in a demonstration context.

7. Future Improvements

- **Conflict Resolution**: Instead of just detecting conflicts, the system could suggest alternative flight paths to avoid collisions.
- **Web Dashboard**: A web-based interface could be built to allow users to upload missions and view real-time conflict simulations.
- **Advanced Algorithms**: Implement more complex and efficient algorithms for pathfinding and conflict detection to improve performance and accuracy.