Reflection & Justification Document

## 1. Design Decisions and Architectural Choices

The system is designed using Python for rapid prototyping, numerical computation, and visualization. The architecture separates core responsibilities into three main modules: data structures (Waypoint and Mission classes), conflict detection (check_conflict), and visualization (animate_missions). This modularity ensures scalability, testability, and maintainability.

The decision to use `datetime` objects for temporal tracking allows real-world timestamps and seamless integration with external flight systems. The use of 3D coordinates (x, y, z) is essential for real-world UAV operations, as airspace is inherently three-dimensional.

## 2. Spatial and Temporal Checks

Temporal conflict detection is based on overlapping time intervals between mission segments. For any two segments from different drones, we compute the intersection of their time intervals. If there is no overlap, the segments are ignored.

Spatial conflict detection involves computing the minimum distance between two drones during the time overlap. Velocity vectors are calculated for each segment, and relative position and velocity are used to compute the time `t*` at which the drones are closest during the interval. We check whether the Euclidean distance between the drones at `t*` is below the buffer threshold (default: 15 meters). If so, a conflict is recorded.

## 3. AI Integration

No AI techniques are integrated into the current prototype. However, AI could be applied in future developments, particularly for predictive conflict resolution, path optimization, and learning-based traffic forecasting. For instance, reinforcement learning could dynamically reroute drones based on historical traffic patterns.

4. Testing Strategy and Edge Case Considerations

Unit tests were performed using multiple synthetic scenarios. The system was validated against:

- Direct conflicts (intersecting trajectories at the same time).
- Near-miss cases (trajectories close in space but separated in time).
- Altitude-separated paths.
- Parallel trajectories with same speed.
- Zero-duration segments (to ensure no divide-by-zero errors).

Edge cases like overlapping waypoints, near-zero velocity, and vertical-only movement were explicitly tested.

5. Scalability Considerations

To scale the system for real-world use (e.g., tens of thousands of drones), the following enhancements are needed:

- **Spatial Partitioning**: Use spatial indexing structures like Quadtrees, K-D Trees, or R-trees to restrict pairwise conflict checks to nearby drones.
- **Temporal Batching**: Divide the airspace timeline into epochs and restrict conflict detection to relevant time windows.
- **Parallelization**: Utilize multithreading or distributed systems to parallelize conflict checks across

GPU or cloud nodes.

- **Database Integration**: Use real-time databases (e.g., InfluxDB, TimescaleDB) to ingest live telemetry data and query historical movements.

- **Efficient Algorithms**: Adopt optimized collision-checking libraries or physics engines for large-scale simulations.

## Conclusion

This system serves as a functional and extensible prototype for 3D drone conflict detection. With appropriate enhancements and integrations, it can serve as a core component in UTM (Unmanned Traffic Management) systems for autonomous air traffic control.