FlytBase UAV Strategic Deconfliction System

Code Documentation

---

1. Introduction

This document describes the internal design, structure, and working of the **UAV Strategic Deconfliction System** developed for the FlytBase Robotics Assignment 2025. The system acts as a **pre-flight authority** to verify whether a drone's planned mission is safe to execute in shared airspace by checking for spatial and temporal conflicts with other drones.

The implementation is written in **Python** and follows a modular, readable, and scalable architecture aligned with real-world **UTM (Unmanned Traffic Management)** principles.

---

2. System Objectives

- Validate waypoint-based UAV missions before takeoff
- Detect conflicts in **space and time**
- Provide clear and explainable conflict reports
- Visualize drone trajectories and conflict zones
- Allow extension to real-world hardware and large-scale deployments

---

3. Project Structure

```
src/
 └── deconfliction.py


docs/
  ├── FlytBase_Code_Documentation.pdf
 └── Reflection_and_Justification.pdf
```

The deconfliction.py file contains the complete strategic deconfliction logic.

---

4. Core Modules and Functions

4.1 Data Models

Waypoint

Represents a single point in a drone's mission trajectory.

```
Waypoint = {
   "x": float,
   "y": float,
   "t": float
}
```

- x, y → spatial coordinates
- t → timestamp in seconds

(Altitude z can be added for 3D / 4D extension.)

Mission

Defines a drone's complete flight plan.

```
Mission = {
   "drone_id": str,
   "waypoints": List[Waypoint],
   "t_start": float,
   "t_end": float
}
```

4.2 Configuration Parameters

```
SAFETY_BUFFER = 2.0      # meters
TIME_STEP = 0.5          # seconds
TIME_THRESHOLD = 0.25    # seconds
```

- **Safety Buffer**: Minimum allowed separation distance between drones
- **Time Step**: Resolution for trajectory interpolation
- **Time Threshold**: Acceptable time overlap window

These parameters are configurable to match regulatory or operational requirements.

---

5. Trajectory Interpolation

Purpose

Drones move continuously between waypoints. Conflict detection must therefore consider **intermediate positions**, not just waypoint coordinates.

Implementation

$p(t) = p1 + (p2 - p1) \times ((t - t1) / (t2 - t1))$

The generate_trajectory() function samples intermediate positions using linear interpolation.

Benefits

- Accurate conflict detection
- Deterministic and computationally efficient
- Easy extension to 3D

---

6. Conflict Detection Logic

6.1 Spatial Conflict Check

The spatial distance between two drones is calculated using Euclidean distance:

$\mathrm{sqrt}((x2 - x1)^2 + (y2 - y1)^2)$

A spatial conflict exists if:

Distance ≤ SAFETY_BUFFER

6.2 Temporal Conflict Check

Temporal overlap is verified using a time threshold:

abs(t_primary - t_other) ≤ TIME_THRESHOLD

This avoids false positives when drones pass the same location at different times.

6.3 Combined Conflict Condition

A conflict is reported **only when both conditions are satisfied**:

Spatial Overlap AND Temporal Overlap

---

7. Conflict Explanation Engine

When a conflict is detected, the system records:

- Conflicting drone ID
- Time of conflict
- Location of conflict
- Minimum separation distance

Example Output


```
{
  "status": "CONFLICT",
  "with_drone": "Drone_A",
  "time": 12.5,
  "location": [10.2, 9.8],
  "distance": 1.3
}
```


This makes the decision **transparent and auditable**.

---

8. Query Interface

The primary interface used by mission planners:

```
def check_mission_clearance(primary_mission, simulated_flights):
```

```
return status, conflict_report
```

Return Values

- CLEAR → Mission approved
- CONFLICT → Mission rejected with details

---

9. Visualization Module

The visualization component uses **Matplotlib** to display:

- Primary drone trajectory
- Simulated drone trajectories
- Conflict locations highlighted in red

Purpose

- Visual validation of logic
- Clear demonstration during evaluation
- Supports both conflict and conflict-free scenarios

---

10. Testing Strategy

Test Scenarios

- Conflict-free mission
- Spatial overlap without time overlap
- Exact safety buffer threshold
- Multiple simultaneous conflicts

Robustness Considerations

- Handles overlapping time windows
- Avoids numerical instability
- Deterministic outputs

---

11. Scalability Considerations

To support **tens of thousands of drones**, the following enhancements are recommended:

- Distributed microservices architecture
- Spatial indexing (KD-tree / R-tree)
- Parallel conflict detection
- Streaming telemetry ingestion (Kafka / MQTT)
- Fault-tolerant state management

---

12. Hardware Integration Mapping (Conceptual)

Real World Component	Code Representation
GPS Coordinates	(x, y, z, t)
Mission Plan	Waypoints
Live Telemetry	Trajectory updates
LiDAR / Sensors	Tactical override layer

This mapping ensures smooth transition from simulation to real deployment.

---

13. AI-Assisted Development

AI tools were used to:

- Accelerate initial development
- Validate geometric and temporal logic
- Improve documentation clarity

All AI outputs were critically reviewed and manually tested.

---

14. Conclusion

This codebase demonstrates a **clean, scalable, and explainable** strategic deconfliction system suitable for real-world UAV operations. Its modular design allows seamless integration with cooperative GPS tracking and onboard sensor-based tactical safety layers.

End of Code Documentation