UAV Strategic Deconfliction System: FlytBase Assignment 2025

Hey there, I'm Devashish, and if you're reading this, you're diving into a project I poured my heart into over two intense, non-stop days. I mean, coffee-fueled nights, debugging at 3 AM, and that rush when the visualization finally clicked and all things fit together perfectly, yes, that's me. The goal was simple : make drone mission deconfliction **clear and effortless**. No confusing outputs, no wasted time, just a tool that instantly tells you if your mission is safe, and *why*.

I didn’t do this alone. I used AI (yep, ChatGPT) as my co-pilot. It was like having a second brain on demand, helping me code and test ideas faster, and debug tricky bits even when my own brain wanted to crash. I focused on the core logic, the design, and the creative side, while AI handled the grunt work. That’s why I could pull together a fully working system with CLI, GUI, and probabilistic engines, all in 2 days.

I aimed to make this intuitive, so anyone can pick it up quickly without needing to be a drone expert. Load your data, run the checks, and see the results clearly, whether through a quick command line output or the full app visualization. Yeah, with the tight timeline, there are probably a few spots that could be smoother, but I focused on the essentials and delivered something solid that works.

What It Does

This is a **strategic deconfliction system**. You give it a leader mission (your UAV’s waypoints + time window) and a set of other drone trajectories. It checks if the mission is safe by looking at:

- **Space** → Are two drones dangerously close?
- **Time** → Do they hit the same area at the same time?

And instead of just yelling *“collision”*, it explains exactly when, where, and with which drone the conflict happens, plus a severity score, which according to me would be how the real drone systems would work.

I implemented two main approaches for detection. The linear one uses straightforward path checks with cubic splines for smooth trajectories. The Gaussian one adds probability clouds, especially useful if your data includes velocities, to account for uncertainties like where a drone might actually be. This helps avoid potential collisions more reliably.

The app lets you visualize everything in 3D with time as the fourth dimension, sliding through to see how things evolve using a slider which is easy to use and much easier to visualize and understand. My whole plan was to build something that helps you visualize this problem and solve it for you, and in the process also let you be a part of it. It solves the problem and keeps the transparency of how I did it, that's how autonomous systems should work.

Key Features I Built In

The heart of this project lives in two classes:

1. **DronePath: The Linear Engine**

This is the **deterministic baseline**.

- It takes waypoints (x, y, z, t) and fits **cubic splines** through them.
- The spline makes the path smooth and realistic (no jagged teleporting).
- At any given time t, I can query the drone's position from the spline.
- Collision check = compare leader's spline with every other drone's spline. If the **Euclidean distance** between them drops below a threshold, it's flagged.
- Works well when you just want **yes/no** answers fast.

Why it matters: this is the simplest and fastest way to check conflicts. It's exact, clean, and reliable for quick baseline checks.

2. GaussianPoints : The Probabilistic Engine

This is where I leveled it up. Instead of treating drone paths as "perfect lines," I modeled them as **probability clouds**.

- Each waypoint has position (and optionally velocity). Velocity is key, because it helps estimate how uncertainty grows over time.
- Around each spline path, I wrap a **Gaussian distribution** (mean = spline position, covariance = uncertainty in x/y/z).
  Conflict check = compute the **Mahalanobis distance** between two trajectories.
    - If it falls inside the confidence region (say 95%), that means there's a high probability of collision.
    - This is more realistic than just Euclidean distance because it accounts for noise, drift, and velocity variation.

Why I chose this:
I'd been learning about **probability densities and Kalman filters**, and it clicked, drones don't fly perfectly, so neither should my math. Gaussian mode makes the system closer to how real airspace safety works and not just *"will they collide"*, but *"what's the chance they might collide?".*

Why Having Both is Right

- **Linear mode** is great for fast checks, debugging, or cases where you trust the data.
- **Gaussian mode** is right for **real-world conditions**: wind, GPS drift, velocity uncertainty.
- Together, they give me flexibility where I have deterministic when I need it and probabilistic when accuracy matters.

My Assumptions and Why They Make Sense

To make this work effectively, I assumed drones need at least position and velocity data for accurate flight modeling and without velocity, paths can be unpredictable. That's why the Gaussian mode uses probabilities to estimate possible positions, reducing collision risks. If velocity isn't there, it defaults sensibly.

Another thing, datasets might have missing values at certain times. My code fills those gaps with mean values from nearby points, keeping trajectories continuous without breaking the system.

Finally, drones don't fly in straight lines between points; there are always velocity adjustments. So, I used cubic splines to create realistic, curved paths that hit every waypoint precisely.

These choices keep things practical and robust, based on making the system reliable even with imperfect data.

Getting It Set Up

1. Clone the repository from my github

```
git clone https://github.com/DevashishHarsh/Drone-Deconflictor
cd Drone-Deconflictor
```

2. Install the required files (create a virtual environment if required)

```
pip install numpy scipy matplotlib pyqt5 pyqtgraph
```

How It Works

1. Load JSON datasets (leader + drones).
2. Build smooth trajectories (splines).
3. Compare leader's path with drones:
    - Linear → Euclidean distance threshold.
    - Gaussian → Probabilistic check using confidence levels.
4. Report conflicts (time, place, severity).
5. Visualize it all in 3D+time with the GUI.

It starts with loading the primary mission – waypoints with x, y, z, and the time window. Then, it pulls in simulated drone schedules from JSON files or hardcoded data. Trajectories are built using cubic splines for interpolation, giving smooth x(t), y(t), z(t) functions.

For checks: sample the paths, calculate minimum distances. In linear mode, it's basic euclidean distances against a threshold. Gaussian uses covariance for uncertainty, with a distance metric that factors in probabilities like checking if overlaps exceed a confidence level.

The app renders this in 3D, with a slider for time scrubbing. Conflicts show up highlighted, and you get details in a table. Math-wise, it's optimized for closest approaches, but kept efficient for quick runs.

This setup draws from how real systems handle uncertainty, but I tailored it to be lightweight and run on any machine. *(Just make sure to handle gaussian points correctly)*

Thinking About Scale

For the real-world with thousands of drones, this prototype handles small sets well, but scaling means architectural shifts. Think distributed processing across machines, real-time data streams, and smarter indexing for fast queries. Add fault tolerance with backups and efficient algorithms to avoid slowdowns. In a month, I could expand this into a full deployment-ready system.

Future Work: Sky's the Limit

- Real-time API (xAI-inspired).
- Wind/terrain integration.
- Multi-primary support.

Wrapping Up: My Thoughts on This

Building this in two days was intense, but I'm thrilled with the outcome. A working app, solid CLIs, and tools that cover the assignment perfectly. I used some AI assists for initial setups, but the real work was mine, refining until it felt right. There might be minor issues from the rush, like edge cases I didn't catch, but the foundation is strong. If this impresses, let's chat about building bigger things. Thanks for taking a look, I hope you enjoy using it as much as I did creating it.

Devashish