Design Spec

Jeffrey Li

April 2025

The *Drone Flight Path Simulator (DFPS)* combines several coordinated software components to provide mission planners with an interactive 3D environment for designing, visualizing, and evaluating SUAS flight paths. This document explains how each major part of the system functions and how the components interact to support stealthy, effective mission planning.

1 Mission Control Backend

The *Mission Control Backend* coordinates high–level application behavior. It is implemented as a Python/Django service that exposes web endpoints for the user interface, handles incoming configuration data, and routes simulation requests to the underlying engines.

When a user modifies a flight path or mission parameter, the Mission Control Backend validates the input, retrieves relevant environmental and sensor data, invokes the simulation engine, and returns computed results such as detection likelihood, battery consumption, and mission success metrics.

2 Simulation Engine

The *Simulation Engine* is responsible for modeling the drone's movement along a candidate flight path and computing mission outcomes. It samples the path over time, applies weather and terrain data, and evaluates audio and visual detectability using configured sensor and threat models.

The engine provides a statistical estimate of mission success, pointwise detection risk along the path, and aggregate measures such as overall stealth score. Results are returned to the Mission Control Backend for use by the visualization layer.

3 Terrain and Environmental Model

The *Terrain and Environmental Model* provides the context in which the simulation runs. It combines base map data with optional elevation models and environmental parameters such as time of day, visibility, and weather conditions.

When the user selects or configures a mission area, this component loads the relevant map tiles and associated metadata. The Simulation Engine queries this model to determine how terrain and environmental factors influence line of sight, sound propagation, and sensor effectiveness at each point along the flight path.

4 Detection and Sensor Model

The *Detection and Sensor Model* encapsulates the rules and parameters that govern how detectable the drone is and how well its onboard sensors can perform mission tasks. It includes audio and visual signature models, sensor ranges, and configurable threat characteristics.

For each sample of the flight path, this model provides estimates of probability of detection and effective sensing performance. These values are used by the Simulation Engine to compute the color–coded risk overlays that are displayed in the user interface.

5 Mission Planning Interface

The *Mission Planning Interface* is the primary user–facing page. It is implemented as a web application using React with Redux for state management. GeoDjango and the Mapbox Maps API are used to display a 3D map of the mission area and the drone's path.

On this page, users can draw or edit flight paths, inspect altitude and distance information, and view real–time updates to detection likelihood and mission success metrics as parameters change. The interface also displays key mission elements such as home base, target area, and geofenced boundaries.

6 Flight Path Management

The *Flight Path Management* component handles the creation, storage, and retrieval of flight plans. Users can start from a sample path, draw a new path directly on the map, or upload a predefined route.

Each flight plan stores geometry, waypoints, associated mission parameters, and metadata such as creation time and author. The Mission Control Backend uses this component to persist plans to the database and to load them when a user wants to review or modify previous missions.

7 User Accounts and Security

The *User Accounts and Security* subsystem manages authentication, authorization, and access control for DFPS. It supports profile–based access using credentials stored in the database and may incorporate two–factor authentication for additional security.

Only authenticated users can create, modify, or delete flight plans. Security constraints such as geofencing, forbidden flight zones, and altitude limits are enforced both at the backend and in the user interface. Alert mechanisms notify users when a path violates operational or safety constraints.

8 Data Persistence and Logging

The *Data Persistence and Logging* component defines how DFPS stores mission–related data. It uses a relational database to maintain user accounts, saved flight paths, configuration presets, and plugin metadata.

In addition, flight data logging records relevant metrics such as path geometry, timestamps, speed, altitude, and detection scores. These logs can be used for post–mission analysis, training, and future improvements to the simulation models.

9 Extensibility and Plug–In Architecture

The *Extensibility and Plug–In Architecture* allows DFPS to evolve as mission requirements change. Detection models, environmental factors, or additional mission parameters can be introduced as plug–ins without altering the core system.

Each plug–in exposes a well–defined interface so that the Simulation Engine can incorporate its outputs into mission evaluations. This design supports future integration of real–time weather feeds, new sensor types, or specialized threat models while keeping the main codebase stable and maintainable.