Chris Tian & Timothy Hutapea
EE250
8 December 2025

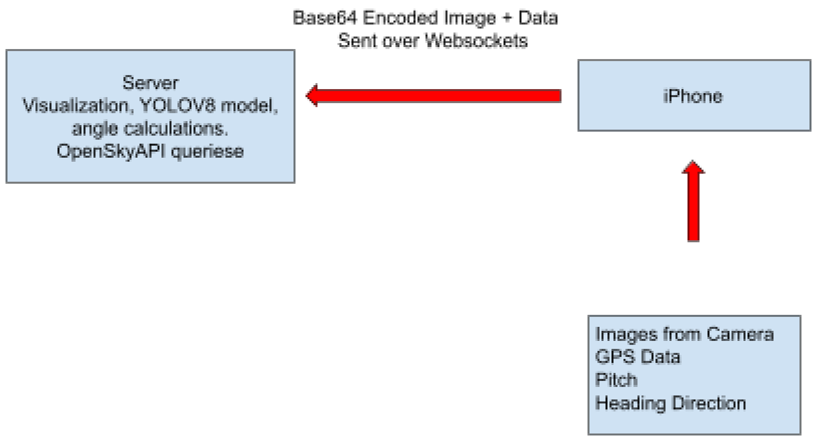
EE-250 Final Project: Plane Detector

1. Introduction

The objective of the Plane Detector is to create an end-to-end IoT system capable of identifying aircraft using visual data and location correlation. Our project uses an iPhone camera to scan the sky. If a plane is visually detected, then the system identifies the aircraft by its callsign by comparing the iPhone's orientation and GPS location with live air traffic data. The final output is visualized on a web-based map that plots the plane relative to the user.

2. System Architecture & Block Diagram

Our project consists of two physical nodes communicating over a local network.



3. Technical Implementation

Hardware

We used a Mac Laptop as the processing node and an iPhone as the sensing node. Our approach minimized costs by using already-owned hardware. The nodes communicate via WebSockets, which is natively supported by Expo and allows for low-latency, data streaming compared to HTTP requests.

Data Collection

We used expo-camera for video capture, expo-location for GPS coordinates, and expo-sensors DeviceMotion to track the phone's pitch and heading.

Serialization: Camera frames are converted into Base64 strings and packaged into a JSON string with the sensor telemetry.

Signal Processing & Machine Learning

Processing occurs in real-time on the server:

- Computer Vision: Incoming Base64 images are decoded and passed to a custom YOLOv8 model. We trained this model ourselves using a dataset of publicly available aircraft images to specifically detect planes in the video feed.
    - https://universe.roboflow.com/chris-tfdku/flying_object_dataset-eajp7
- Field of View Filtering: Once a plane is detected by the model, the system performs a field of view calculation. Using the phone's GPS and pitch angle, we calculate a vector to determine the visible sky segment.
- Data Correlation: The system queries the OpenSky API for live air traffic. Then, it calculates the distance and angle between the phone and planes within 10 miles to find the best match between the API data and the direction the phone is facing.

Visualization
The data is visualized on a map using Folium. The map centers on the camera's location and plots markers for detected planes, displaying their callsigns.

4. X-Factor
We found datasets of airplane images and trained a YOLOv8 model for aircraft detection. We also wrote a script to correlate 3D orientation with API data to identify the specific plane and flight info.

5. Reflection & Limitations
The biggest limitation is that the system only works with daylight. The computer vision model relies on the camera quality and cannot detect planes at night. Additionally, there is inherent latency in the video stream due to Base64 encoding/decoding overhead.

The biggest challenge was connecting the iPhone and the Python backend. We had to learn Expo and React Native from scratch to access the hardware sensors. In hindsight, using a dedicated camera module would have simplified the stack.

Video Demo:
https://youtu.be/alE3j5fuAaA