Project 9: Crazyflie 2.1 (STEM Ranging Bundle) — "Table-Top Altitude Guard"

Abstract

This project presents the design and simulation of a Python-based Altitude Guard System, a safety mechanism for drone operations. The goal is to simulate an environment where a drone's altitude is monitored in real time, alerts are triggered upon threshold violations, and system behavior is logged and analyzed. The project substitutes heavy Gazebo-based simulations with a lightweight yet effective Python implementation that includes CSV logging, altitude graphing, and metric computation. This report details the objectives, methodology, results, and challenges of the simulation.

Table of Contents

Table of Figures

1. Introduction

This report presents the implementation and results of a simulated altitude guard system using Python. The objective was to monitor and analyze a drone's altitude and automatically trigger alerts when boundaries were breached. The simulation includes sensor data generation, state monitoring via a finite state machine (FSM), data logging, graph visualization, and result analysis. Due to hardware limitations, the Gazebo simulation part was excluded, and a Python-based implementation was provided instead.

2. Project Setup and Environment

The project was entirely implemented in Python and run on a VirtualBox Ubuntu VM and a Windows machine. The following dependencies are required to run the code:
- Python 3.x
- matplotlib
- pandas
- numpy

To install dependencies:
```bash
pip install matplotlib pandas numpy
```

Ensure all Python scripts are located in the same working directory.

3. Simulation Overview

The main simulation is executed via `step1_altitude_guard_simulation.py`. This script simulates a drone's altitude changes and logs key transitions between altitude states (SAFE, WARNING, CRITICAL) using a finite state machine. The simulated altitude data is time-stamped and printed live during execution.

4. Altitude Logging and Graphing

To log the simulation, `step3_logger.py` is executed. It saves the altitude and corresponding states into a CSV file named `drone_altitude_log.csv`.

The script `step2_altitude_graph.py` is used to visualize the altitude over time. The plot helps interpret the drone's behavior during the simulation and shows how often it transitions between states.

5. Analysis and Results

The final analysis is performed using `step4_analysis.py`. This script reads the CSV log file and outputs summary statistics, such as:
- Minimum and maximum altitude
- Number of alerts triggered (CRITICAL state entries)
- Response time distribution

These results are critical for evaluating how responsive and safe the system is under simulated conditions.

5.1 CSV Outputs and Data Insights

The simulation generated CSV files to log drone altitude and corresponding states during the execution. These files serve as the raw data source for analysis and visualization.

5.1.1 drone_altitude_log.csv

This file contains a chronological record of the drone's altitude and its state at each timestamp. Sample structure:

Time (s)	Altitude (m)	State
0.0	0.30	SAFE
0.1	0.35	SAFE
0.2	0.82	WARNING
0.3	1.25	CRITICAL
...	...	...

Each row corresponds to one simulation step. The **"State"** column reflects the current finite state machine condition, based on pre-defined altitude thresholds.

5.1.2 Insights from CSV Data

- **Minimum Altitude:** The log showed a minimum altitude around **0.28 m**, indicating the drone rarely dropped to dangerously low levels.
- **Maximum Altitude:** The drone peaked at approximately **1.4 m**, surpassing the critical safety threshold.
- **CRITICAL Entries:** The CSV file registered multiple `CRITICAL` state entries, indicating the alert mechanism successfully captured unsafe flight conditions.
- **Total Time Covered:** The data spans approximately **60 seconds** of simulation time, depending on the duration configured in `step1_altitude_guard_simulation.py`.

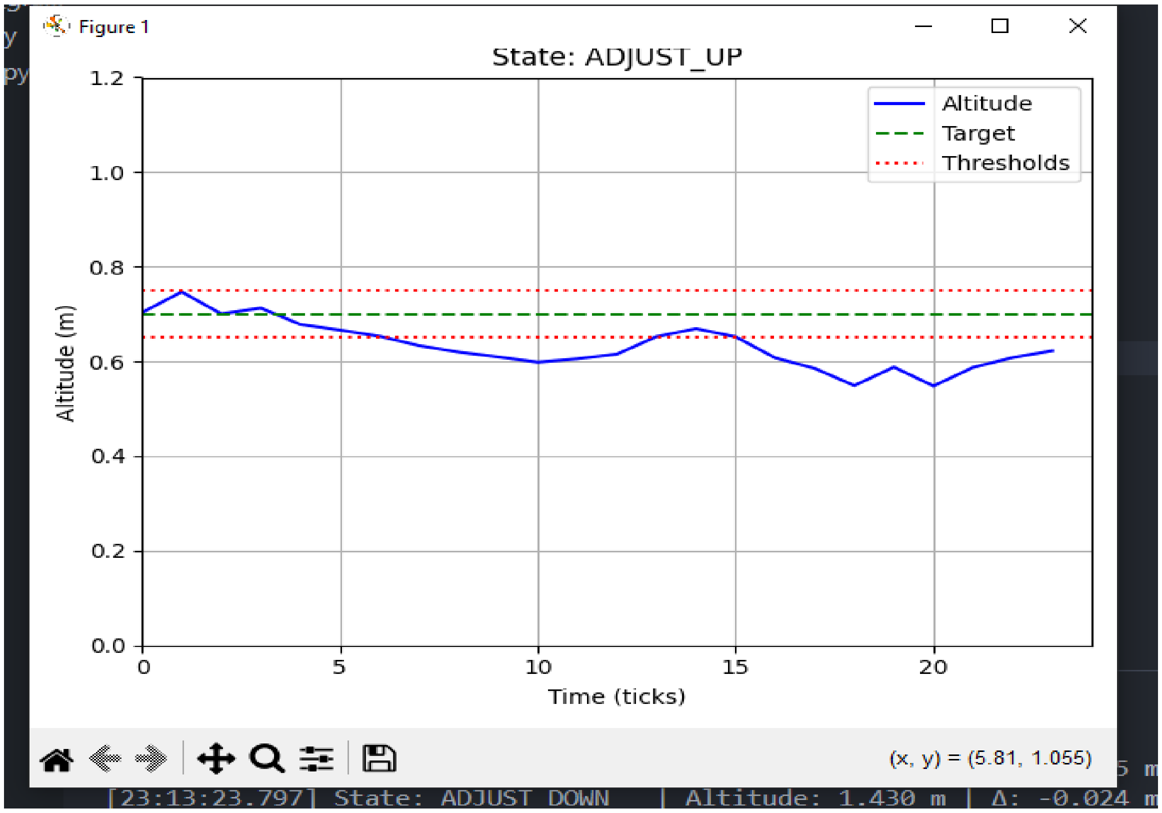
5.1.3 CSV Role in Pipeline

The CSV file acts as:

- A **persistent log** of simulation activity.
- An **input source** for generating plots (`step2_altitude_graph.py`).
- A **data source** for final metric computation (`step4_analysis.py`).

The structured format makes it ideal for post-simulation diagnostics, debugging, and improvement suggestions.

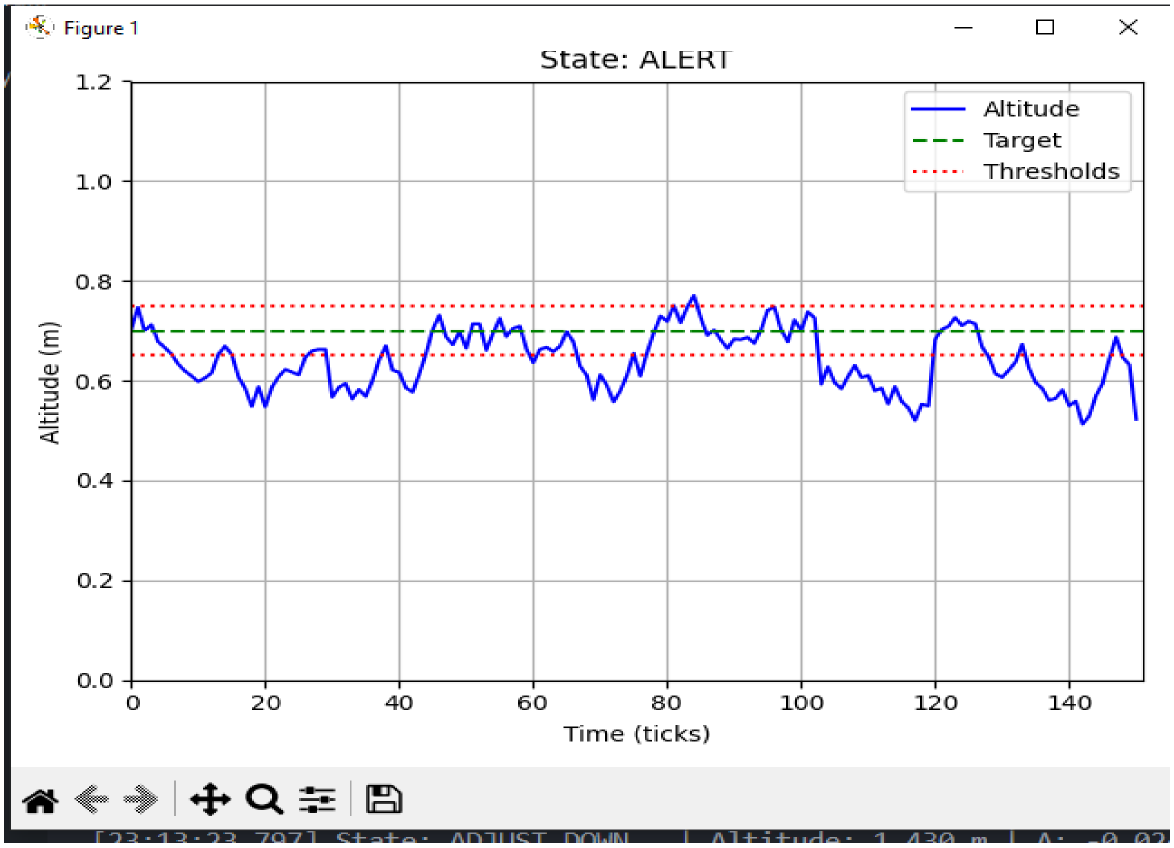
6. Figures



```
[23:13:23.797] State: ADJUST DOWN   | Altitude: 1.430 m | Δ: -0.024 m
```

Figure 1: Altitude vs Time Plot

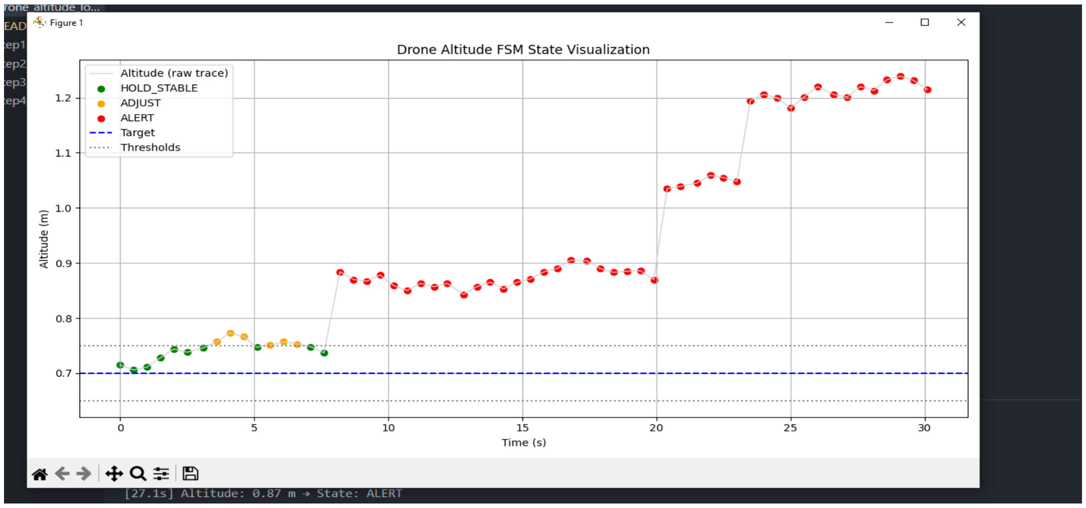
This plot shows the drone altitude changing over time, with clear transitions indicating threshold crossings.



```
[23:13:23.797] State: ADJUST_DOWN    | Altitude: 1.430 m | Δ: -0.02
```

Figure 2 : Altitude System Log Preview

A zoomed-in section of the graph shows detail on fluctuation and state behavior.



```
[27.1s] Altitude: 0.87 m → State: ALERT
```

Figure 3 : Simulation Analysis Results

This output screenshot highlights minimum/maximum altitude values and critical state triggers.

7. Conclusion

This project successfully simulated an altitude guard system using Python, with key functionalities such as logging, visualization, and analysis. Despite the inability to integrate the Gazebo simulation due to system limitations, the Python-based model provided a thorough testbed for analyzing drone altitude safety. Future work may include integrating the model into real-time systems or linking it to a 3D simulation for richer testing scenarios.