CS 408 PROJECT PHASE 3

Ertem Ata Kavaz – 32790

Ege Çetinkaya - 32661

Karya Üstünçelik - 30903

System Overview

The system consists of the following components:

Sensor Nodes: Lightweight, headless scripts that simulate temperature and humidity sensors. They periodically send data to the drone via TCP. If a connection fails, they retry until successful.

Drone Gateway: This is the core processing unit. It receives data from sensors, keeps a rolling window of the last 10 readings per sensor, calculates averages, detects anomalies, and forwards results to the central server. If the drone’s simulated battery drops below a threshold (15%), it stops forwarding data and queues it instead. Once charged back to 100%, it resumes normal operation.

Central Server: A GUI application that receives aggregated data from the drone. It displays live readings, logs events, and provides visual graphs for each sensor node.

All communication is based on TCP, and JSON is used as the format for data exchange. This makes the system portable and easy to run on any machine with Python.

Summary of Functionalities

Real-time TCP data flow between components.

Drone-side edge processing: 10-reading averages and anomaly detection.

Battery simulation: drains 1% per second, pauses at 15%, resumes at 100%.

Queuing mechanism: summarized data is buffered during charging and sent later.

GUI features: sensor table, battery status, real-time log panel (drone); live graphs and logs (central server).

Headless sensors with retry-on-failure logic.

Explanation of Core Functions

Sensor Node:

python3 sensor.py --drone\_ip --drone\_port --interval --sensor\_id test --count

generate\_sensor\_data(sensor\_id)

Input: sensor\_id (string)

Output: A dictionary with simulated temperature, humidity, and timestamp

Purpose: Randomly generates realistic environmental readings for the sensor.

sensor\_loop(sensor\_id, drone\_ip, drone\_port, interval)

Input: Sensor ID, drone’s IP and port, interval in seconds

Output: Sends data to the drone over TCP periodically

Purpose: Maintains a persistent connection with the drone, sends a new reading every few seconds, and retries on failure.

Drone Gateway

python3 drone.py --serverip --serverport --listenip --listenport

Stats class

* Manages a rolling window of 10 values per sensor.
* Automatically calculates averages and counts anomalies.
* Fields: avg\_t, avg\_h, anom, last.

make\_sender()

* Input: Server IP, port, log queue, charging event, buffer list
* Output: A callable function that tries to send JSON to the central server
* Purpose: Sends summaries when battery is healthy, queues when battery is low or connection fails.

flush\_queue()

Input: Server IP, port, queue buffer, log queue

Output: Sends queued messages

Purpose: Called once battery is fully recharged to flush stored summaries to the central server.

listener()

Input: Listen IP, port, sensors dictionary, log queue, sender, listen/stop events

Output: Accepts incoming sensor connections and processes data

Purpose: TCP server logic for handling multiple sensor clients concurrently.

battery()

Input: GUI, log queue, listener event, charging event, queue buffer, server IP/port

Output: Modifies battery percentage every second

Purpose: Simulates battery behavior. At 15%, disables listener and activates queuing. At 100%, re-enables listener and flushes the queue.

GUI class (drone)

Purpose: Displays sensor readings, battery status, and log messages in real-time.

Automatically updates every second using Tkinter's after method.

Central Server

python3 central.py --ip --port

logRecord()

Input: A string message

Output: Appends timestamped message to shared logs list

Purpose: Used by all server-side processes to track events.

processDrone()

Input: A socket and address

Output: Parses JSON lines from the drone

Purpose: Reads incoming messages, stores them in the shared state, and triggers logs.

csGUI()

Purpose: Builds the GUI for the central server using Tkinter.

Shows a scrollable event log, a live data table, and graph panels per sensor.

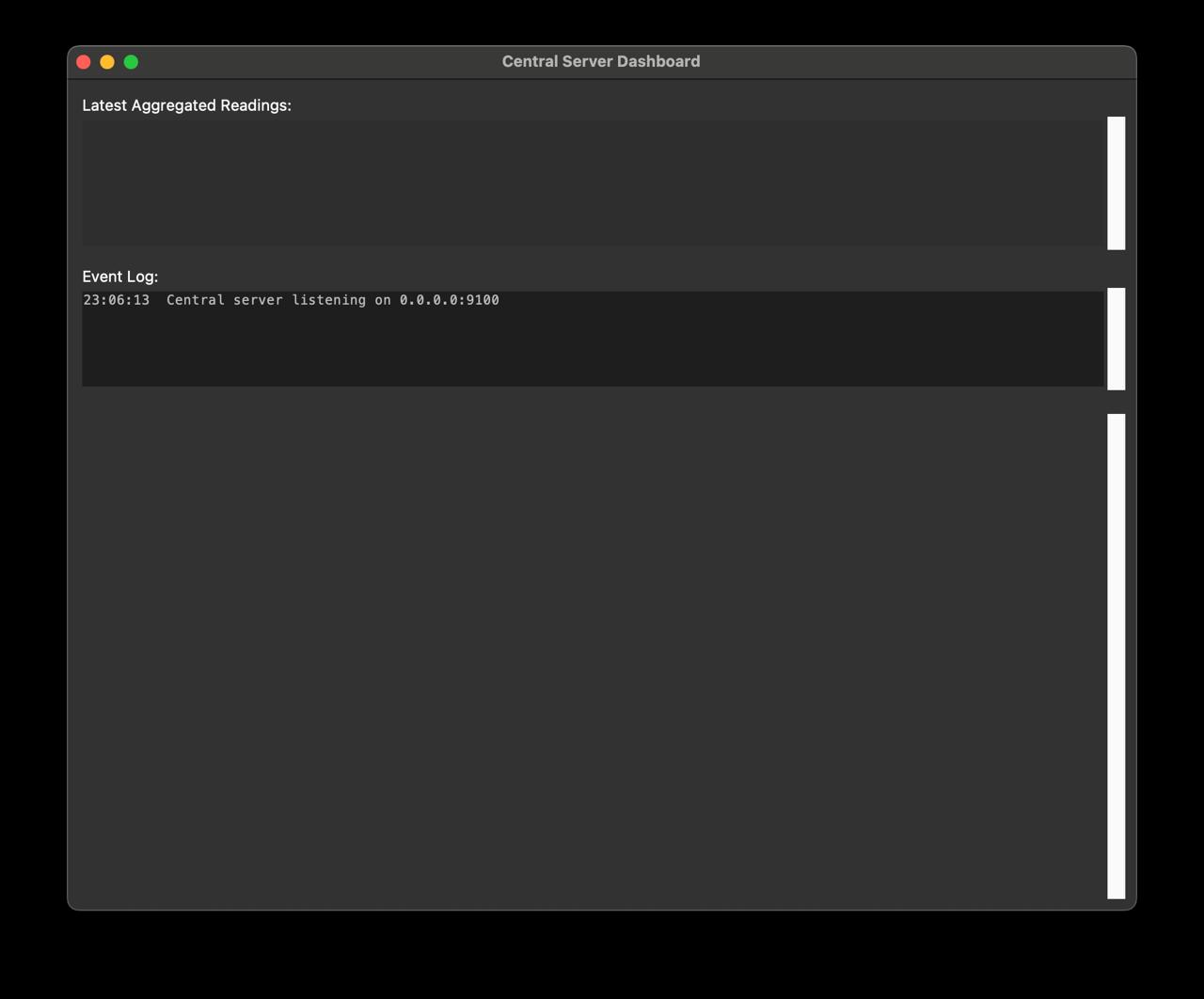
initTCPServer()

Input: IP and port to bind

Output: Opens socket server for drone

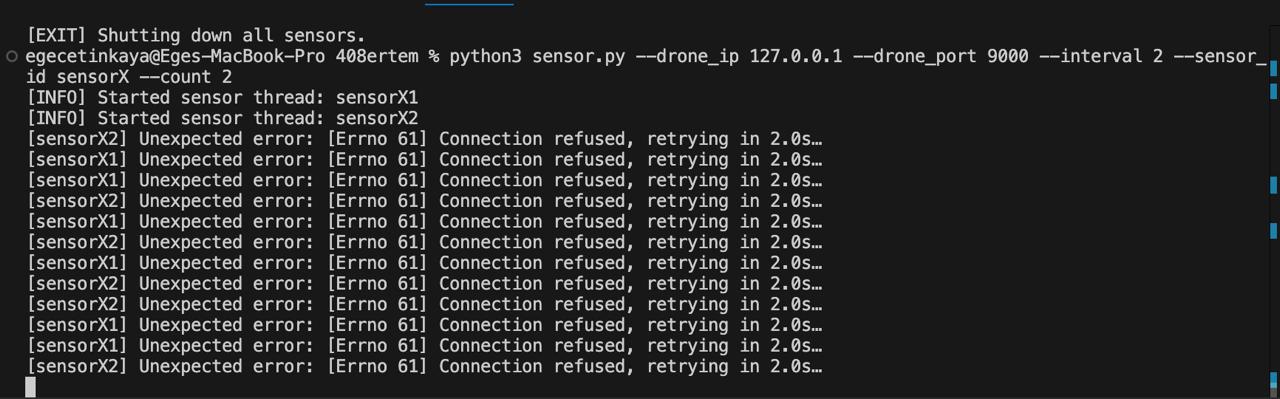
Purpose: Accepts TCP connections and starts a thread for each.

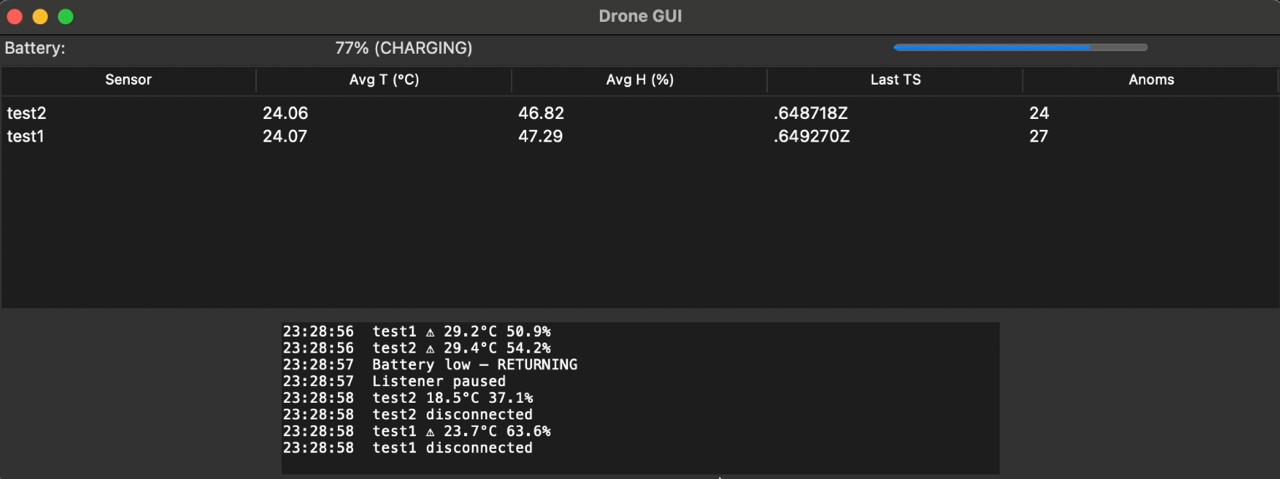
Grapgical User Interface:

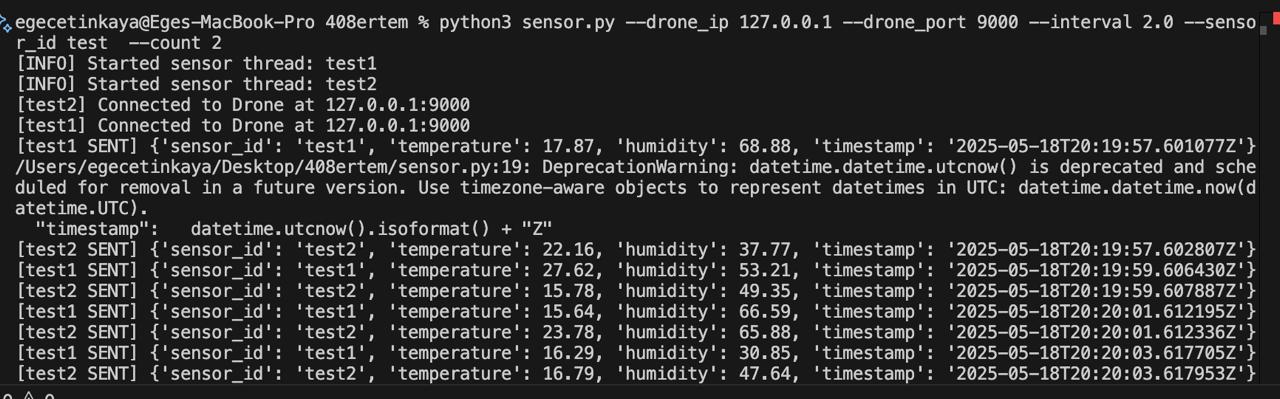
This is the initial Central Server Dahboard:

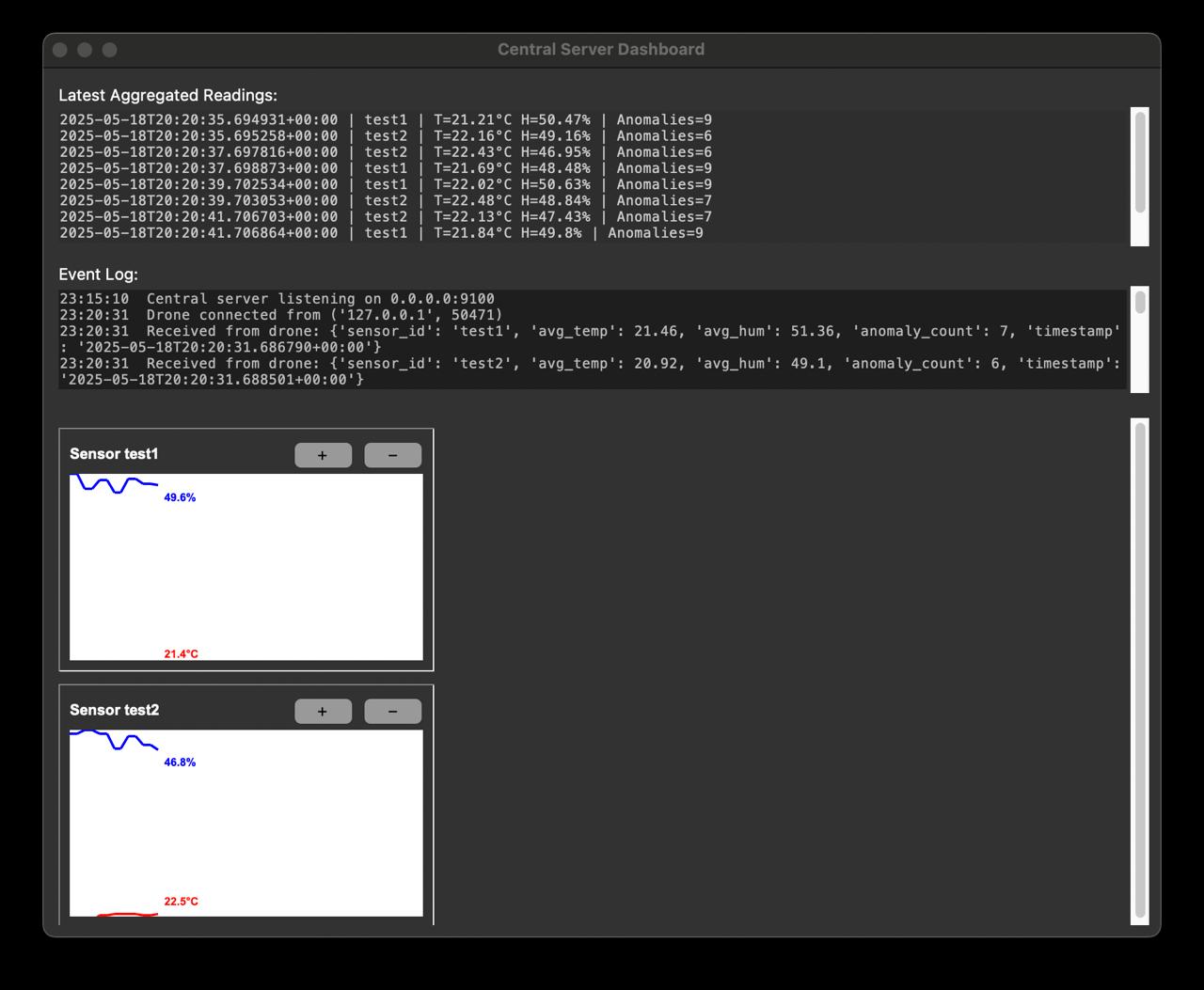
This is the initial Drone Graphical User Interface:

Sensors when there is no drone connected:

The drone GUI while the drone is charging:



The Sensor’s initial connection with the drone: 

The Central Server Dashboard after connection:

Final Notes

The entire system works in a modular and event-driven structure. It has been tested with multiple sensor clients, real-time anomalies, and battery simulations. Data is never lost—even if the drone is charging, the queue ensures forwarding once battery is restored. GUIs are designed for ease of grading and understanding.

This project helped reinforce socket programming, GUI design with Tkinter, real-time data processing, and multithreading in Python.