CS408 - 2025 Spring Term Project (Phase-1)

Name, Surname & ID of group members: Mehmet Altunören (22539), Orhun Burak Kıyanç (22538), Deniz Kaan Yılmaz (32117)

System Architecture and Connections

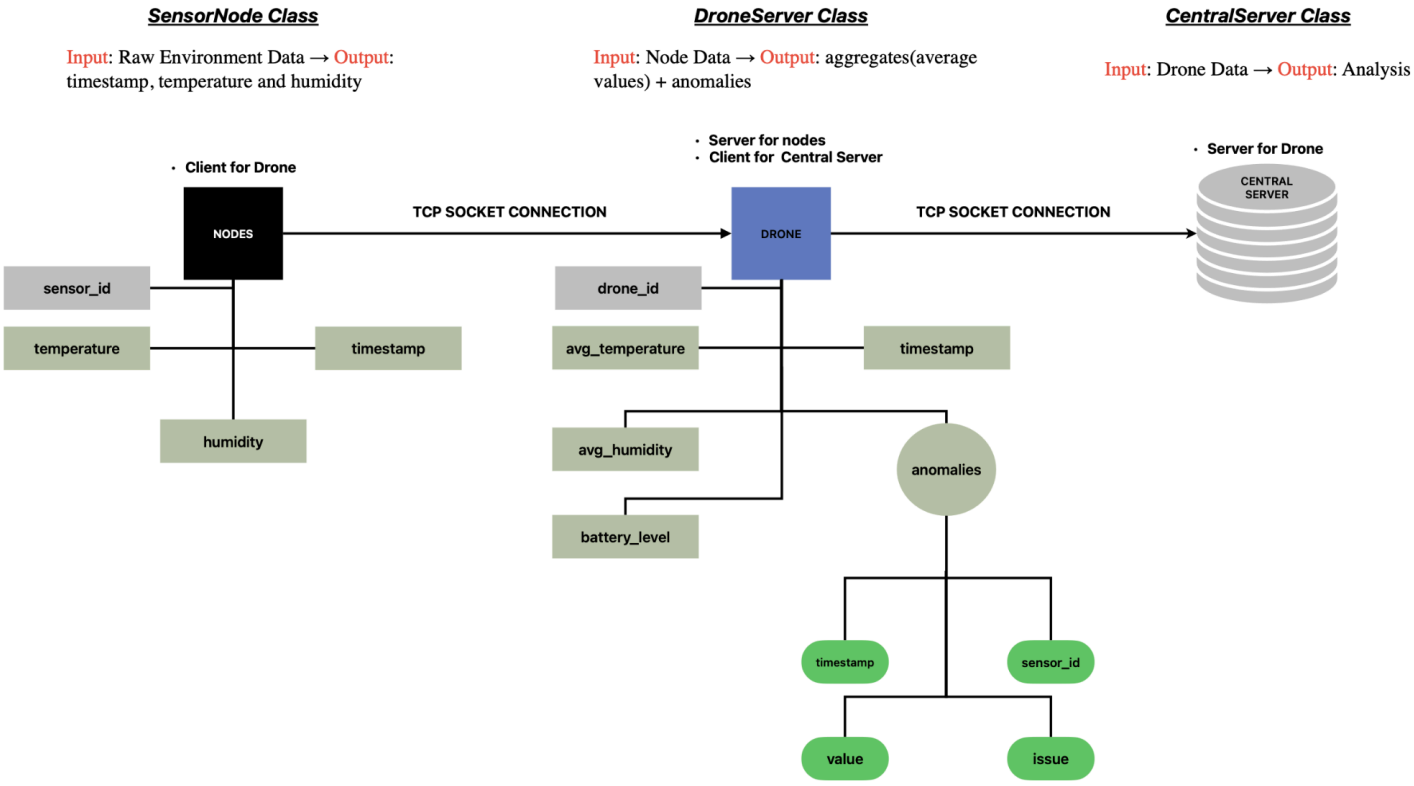
This project simulates and environmental monitoring system composed of:

- **Sensor Nodes** that simulate environmental data (e.g., temperature, humidity)
    - Connects to the drone via TCP and sends data at regular intervals.
    - Does not contain GUI; works via the command line.
- A **Drone** acting as a mobile edge computing unit that collects and processes sensor data.
    - Receives data from sensor nodes (as TCP Server).
    - Processes data: calculates average, check for anomalies.
    - Simulates battery level. If low, go to “returning to base” state.
    - Transmits processed data to the central server (as TCP client).
    - Includes GUI: data stream, anomaly list, battery status and log panel.
- A **Central Server** that receives and visualizes processed data from the drone.
    - Receives summary data from the drone.
    - Presents data to the user as a table/graph/log with GUI.

All communication will occur over **TCP** sockets. Data will be transmitted in **JSON** format for structure and flexibility.

- Multiple Sensor Nodes (TCP clients) send data to the Drone.
    - Manage connections concurrently via threads.
- The Drone (TCP server for sensors, TCP client to Central Server) processes data.
- Central Server (TCP server) receives and visualizes processed data.

The Diagram of the System



JSON Data Formats (Sensor to Drone, Drone to Server)

1- Sensor to Drone

- sensor_id: Unique sensor ID (string)
- timestamp: Timestamp in ISO 8601 format
- temperature: Temperature value (float)
- humidity: Humidity value (float)

```
{
 "sensor_id": "sensor_01",
 "timestamp": "2025-04-09T14:25:00Z",
 "temperature": 24.7,
 "humidity": 48.2
}
```

2- Drone to Server

- drone_id: Unique drone ID (string)
- avg_temperature: Average temperature value (float). Averages of last N data.
- avg_humidity: Average humidity value (float). Averages of last N data.

- anomalies: List of anomalies (sensor_id, issue, value, time)


```
{
 "drone_id": "drone_alpha",
 "timestamp": "2025-04-09T14:25:10Z",
 "average_temperature": 25.1,
 "average_humidity": 50.3,
 "anomalies": [
  {
   "sensor_id": "sensor_02",
   "issue": "temperature_too_high",
   "value": 102.3,
   "timestamp": "2025-04-09T14:24:50Z"
  }
 ],
 "battery_level": 79
}
```


Connection Parameters & Communication

Communication is managed via TCP with configurable IPs and ports. Example parameters:

- Component: Sensor Node
  - Connection Route: Sensor → Drone
  - Role: TCP Client
  - IP: Drone's IP
  - Port: 3400
- Component: Drone
  - Connection Route: Sensor → Drone → Server
  - Role: TCP Client & Server
  - IP: 127.0.0.1 (or customize IP)
  - Port_1: 3400 for Sensor Nodes
  - Port_2: 3500 for Central Server
- Component: Central Server
  - Connection Route: Drone → Server
  - Role: TCP Client & Server
  - IP: 127.0.0.1
  - Port: 3500

Modules & Classes

- **Sensor Nodes**
  - **Class:** SensorNode
  - **Task:**Receive environmental data and send it to the drone over TCP.
  - **Inputs:** Drone IP, port, data sending interval.
  - **Outputs:** JSON data packets.
  - **Sub-Functions:**
    - connect_to_drone()
    - collect_environmental_data()
    - send_data()
    - handle_reconnection()

- **Drone**
  - **Class:** DroneServer
  - **Task:** Receives data from sensors, processes it, performs battery check, sends it to server.
  - **Inputs:** Sensor data(JSON), battery threshold, server IP/Port
  - **Outputs:** Processed summary data (anomalies, averages)
  - **Sub-Modules and Sub-Functions:**
    - **EdgeProcessor**
      - update_readings()
      - compute_averages()
      - detect_anomalies()
    - **BatteryManager**
      - Return-to-Base Behavior: In the event that the battery level drops below the configured threshold, the BatteryManager module triggers a “Returning to base” state. In accordance with Option A, the Drone will close all active connections with sensor nodes and stop receiving data. These disconnections are logged, and sensor nodes are expected to attempt reconnection periodically. Data transmission and processing will resume once the battery level is restored.
      - consume()
      - check_status()

        - **DroneClient**
            - send_to_server()
        - **DroneGUI**
            - update_table()
            - highlight_anomalies()
            - display_battery()
            - log_panel()
- **Central Server**
    - **Class:** CentralServer
    - **Task:** Receives data from the drone and displays it on the GUI.
    - **Inputs:** TCP data from the drone.
    - **Outputs:** Visualized table/graph, log
    - **Sub-Functions:**
        - **listen_for_data()**
        - **parse_data()**
        - **log_event**
        - **update_GUI()**

Potential Issues

- **Concurrency & Thread Management (on DroneServer)**
    - **Problem**: The DroneServer must handle data from multiple SensorNode connections simultaneously.
    - **Impact**: Poor concurrency handling could result in race conditions, deadlocks, or data loss.
- **Queue Management & Data Overflow**
    - **Problem**: If sensor data is sent at a high frequency, the Drone may not process data in time.
    - **Impact**: Memory consumption could rise, leading to potential buffer overflows or system crashes.
- **JSON Parsing Errors**
    - **Problem**: Malformed JSON packets can cause parsing failures.
    - **Impact**: Dropped packets or system crashes.
- **Latency in Visualization (Central Server)**
    - **Problem**: Frequent or large data transmissions can affect GUI responsiveness.
    - **Impact**: Laggy or frozen GUI, resulting in poor user experience.