Ground Control Station (GCS)

1. 🛠️ Core Components Overview

Your GCS consists of three main modules:

1. **Terminal Window (Command & Control):**
    - Displays telemetry logs, system messages, and command outputs.
    - Allows you to send simple commands (e.g., "arm", "takeoff", "land").
2. **Camera Feed Window (Video Streaming):**
    - Uses your **custom camera controller script** on Raspberry Pi.
    - Streams video to any **open localhost port**.
    - GCS Electron app receives and displays the live feed.
3. **Map Window (Navigation & Mission Planning):**
    - Uses **Leaflet.js** or another open-source mapping framework.
    - Displays UAV location, waypoints, and flight path.
    - Allows mission planning (future feature: click-to-set waypoint).
4. **Backend Connectivity:**
    - **Python backend**: Handles communication with hardware (sensors, telemetry, camera controller).
    - **JavaScript (Electron frontend)**: GUI to display all windows and user interaction.
    - **Data Bridge**: WebSockets or ZeroMQ between Python ↔ JS.

---

2. ⚙️ Data Flow & Protocols

Video Streaming

- **Protocol**: MJPEG / RTSP / WebRTC (choose based on latency requirement).
- **Algorithm (Video)**:
    1. Camera Controller captures frames on Pi.
    2. Encodes frames → pushes to localhost port (e.g., `http://localhost:8080/video`).
    3. GCS requests stream → renders in Electron video window.

Telemetry (Control & Feedback)

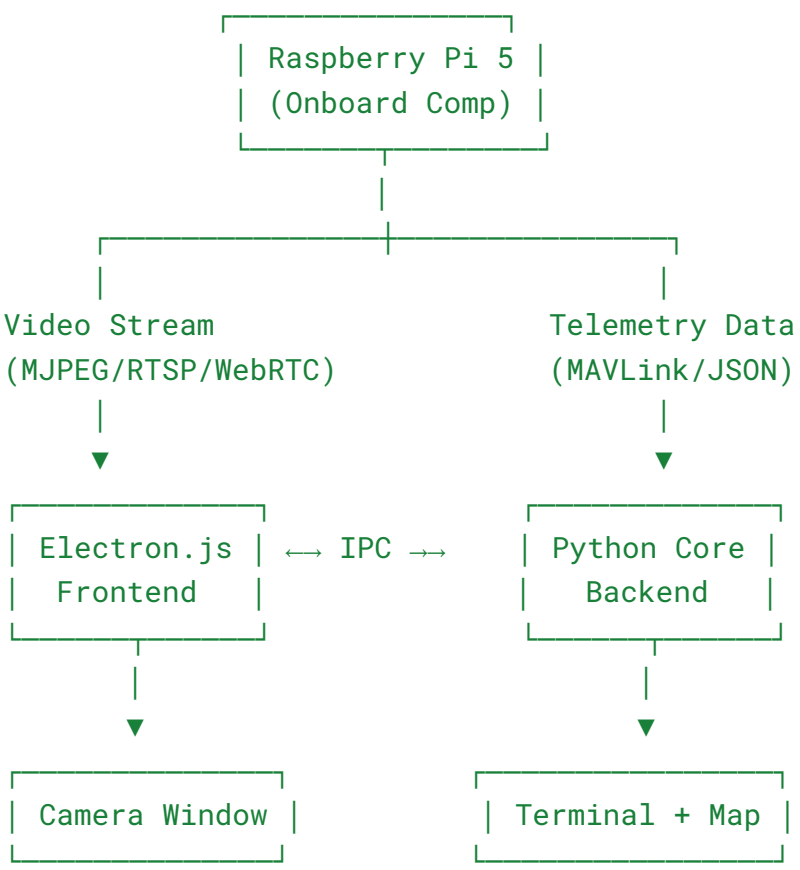
- **Protocol**: MAVLink (standard for drones), or custom JSON via UDP/TCP/WebSocket.
- **Algorithm (Telemetry)**:
    1. Raspberry Pi gathers data (GPS, IMU, battery).
    2. Sends telemetry packets → backend server.
    3. Backend parses → forwards to frontend terminal/map.
    4. Frontend displays data in terminal + plots UAV position on map.

Command Uplink (GCS → UAV)

- **Protocol**: UDP or WebSocket for low latency.
- **Algorithm (Commands)**:
    1. User inputs command in terminal (e.g., `/arm`).
    2. Frontend → backend via IPC/WebSocket.
    3. Backend → UAV over UDP/MAVLink.
    4. UAV executes → sends status back (e.g., "ARMED").

---

3. 🔄 High-Level System Flow



---

4. 📝 Step-by-Step Quickstart Guide (Layman Terms)

Step 1: Set Up Raspberry Pi 5

- Install **Python 3.11+**, `opencv-python`, `flask` (for camera streaming).

Run your **camera controller script** to stream video:

```
python3 camera_stream.py --port 8080
```

-
- Pi now streams video at `http://<pi-ip>:8080/video`.

Step 2: Backend (Python)

- Write a **telemetry server** (UDP or WebSocket).

Example:

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("0.0.0.0", 14550))  # MAVLink default port
while True:
    data, addr = sock.recvfrom(1024)
    print("Telemetry:", data)
```

-

Step 3: Electron Frontend

- Create **3 windows** in Electron:
    - `TerminalWindow` → connects to backend → prints logs.
    - `CameraWindow` → loads `http://<pi-ip>:8080/video`.
    - `MapWindow` → loads Leaflet.js map + overlays UAV position.

Sample Electron window creation:

```
const { BrowserWindow } = require("electron");

function createCameraWindow() {
  let win = new BrowserWindow({ width: 800, height: 600 });
  win.loadURL("http://localhost:8080/video");
}
```

-

Step 4: Data Bridge

- Use **WebSockets** (`socket.io`) for frontend ↔ backend.
- Commands flow:
    - User types `/takeoff`.
    - Electron sends command → Python backend.
    - Python backend → Pi → UAV.

Step 5: Map Integration

Use **Leaflet.js**:

```
var map = L.map('map').setView([28.6139, 77.2090], 13);
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png').addTo(map);
L.marker([28.6139, 77.2090]).addTo(map).bindPopup('UAV Location');
```

- Update marker position on telemetry data.

---

5. 📑 Algorithms Summary

Video Stream Algorithm

`Capture frame → Encode → Send to port → Electron fetch → Display in camera window`

Telemetry Algorithm

`UAV sensors → Pi → Python server (UDP/WebSocket) → Electron frontend → Terminal/Map`

Command Uplink Algorithm

`Electron command input → Python backend → Pi → UAV`

---

6. 📡 Protocols Used

- **MJPEG/RTSP/WebRTC** → Video stream.
- **UDP + MAVLink / JSON** → Telemetry.
- **WebSocket (socket.io)** → Backend ↔ Frontend communication.
- **Electron IPC** → Internal window messaging.

---

7. ✅ Final Notes

- Start small: **just camera + telemetry logs** first.
- Then add **maps & waypoints**.
- Finally, add **mission planner & autonomous control**.