

RCSIM Tier 2 — ESP32-CAM Platform: User Manual

Illustrated user guide: tabs, controls, settings and step-by-step procedures. The screenshots show the English interface.

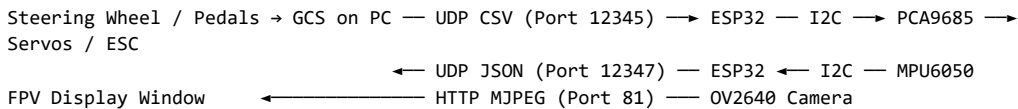
Contents

- 1. Scope and Architecture 3
- 2. Firmware Variants in Repository 3
- 3. Boards and Pinout Limitations 3
 - 3.1. Baseline WROVER Wiring 4
- 4. Configuration and IDE Setup 5
 - 4.1. Wi-Fi and Network Settings 5
 - 4.2. Arduino IDE Environment 5
 - 4.3. Flashing Procedure 6
- 5. Communications Protocol 6
- 6. Initial Bench Setup and GCS Configuration 6
 - 6.1. Control Mapping 7
 - 6.2. Video Streaming 8
- 7. Failsafe Behavior 8
- 8. Advanced Firmware Extensions: V2, V3, and V4 8
 - 8.1. ESP32 V2: Closed-Loop PCA9685 Calibration 8
 - 8.2. ESP32 V3: GPS, Walksnail OSD, and Battery Monitoring 8
 - 8.3. ESP32 V4: CRSF MultiLink Hub 9
- 9. Diagnostics and Recovery 9
- 10. Source Code Verification Trails 9

Edition: 14 October 2026. This manual is grounded in the project codebase and physical hardware photographs. It does not replace physical testing or bench verification of your specific hardware assembly. In preparing this documentation, no live vehicle driving was conducted.

1. Scope and Architecture

Tier 2 encompasses vehicle-side embedded platforms built on the ESP32 microcontroller, including camera-equipped variants. In the baseline implementation, the ESP32 receives steering and throttle commands via Wi-Fi/UDP, drives an external I2C PCA9685 PWM controller, serves an HTTP MJPEG video stream, and transmits IMU telemetry back to GCS.



Extending ESP32 firmware with CRSF capabilities does not convert the platform into Tier 3. Tier 3 specifically covers dedicated RC radio systems (ELRS/mLRS, direct CRSF/MAVLink transceivers, and radio mods). An onboard Raspberry Pi running robotics, SLAM, and autonomy belongs to Tier 4.

2. Firmware Variants in Repository

Base firmware directory: `RCSIM_PC/pc_app/ESP32Arduino/RCSIM_Arduino_ESP32/ESP32 Tier 2/ESP32/`.

- `ESP32/ESP32.ino`: Baseline Wi-Fi/UDP, camera stream, MPU6050 IMU, and PCA9685 control.
- `ESP32V2/ESP32V2autocalibPCA/ESP32V2autocalibPCA.ino`: Adds closed-loop PCA9685 oscillator auto-calibration.
- `ESP32V3+GPS+FPV/ESP32 V3 GPS + FPV`: Adds NMEA GPS, Walksnail digital OSD output, battery monitoring, and multiple IMU drivers.
- `ESP32V4_CRSF_MultiLink/ESP32V4_CRSF_MultiLink.ino`: CRSF routing hub featuring dual-link multiplexing (RF UART + Wi-Fi UDP).
- `ESP32_CRSF_Dongle_Transmitter/ESP32_CRSF_Dongle_Transmitter.ino`: Standalone USB transmitter bridge dongle, not vehicle camera firmware.

These variants are separate programs with distinct pinouts and protocol expectations; they are not a single auto-detecting build. Record your chosen build before wiring. The primary procedure below addresses `ESP32/ESP32.ino`.

3. Boards and Pinout Limitations

The baseline sketch provides pin configurations for `BOARD_AI_THINKER`, `BOARD_WROVER_DEV`, and `LILYGO_TSIMCAM_S3`. In the repository, `BOARD_WROVER_DEV` is enabled by default. Macro names vary across sketch versions; always verify preprocessor definitions.

For `BOARD_WROVER_DEV`, I2C is assigned to **SDA = GPIO13** and **SCL = GPIO14**. The I2C scanner targets the PCA9685 at address **0x40** and the MPU6050 IMU at address **0x68**.

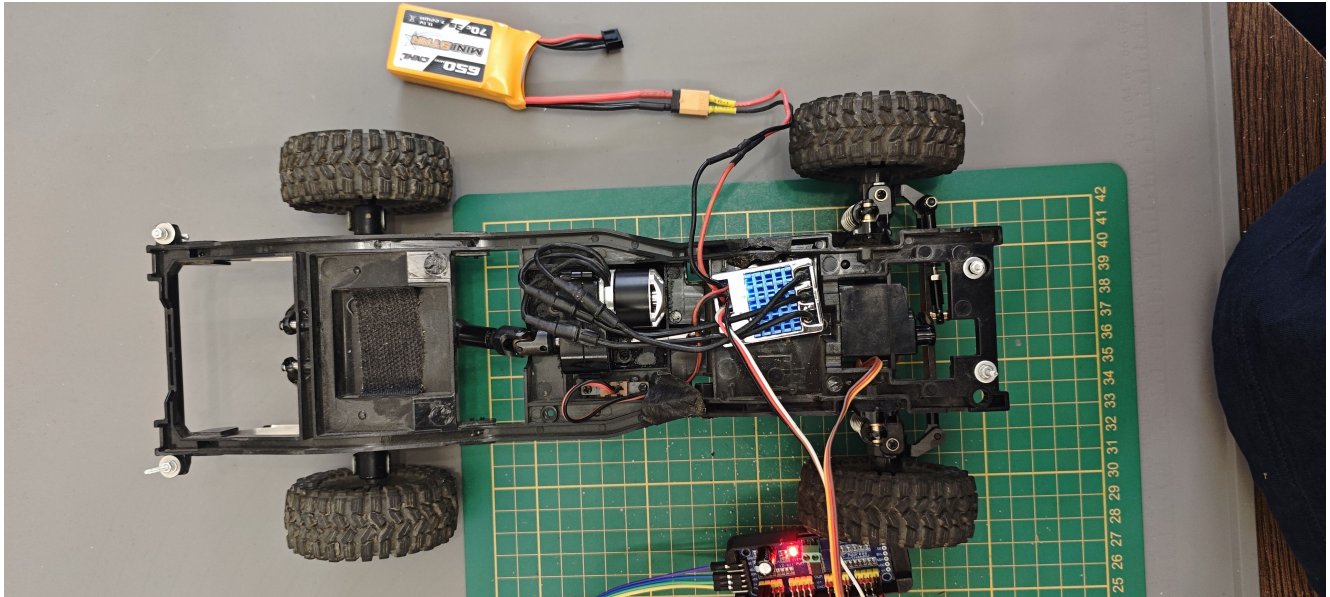
Critical AI-Thinker Pin Conflict: On classic AI-Thinker boards, GPIO21 is assigned to camera data line Y5 and GPIO22 to camera pixel clock PCLK. Attempting to run standard I2C on GPIO21/22 while the camera is active causes a hardware collision that halts camera DMA and corrupts I2C transfers.

3.1. Baseline WROVER Wiring

The following schematic applies to the active BOARD_WROVER_DEV build:

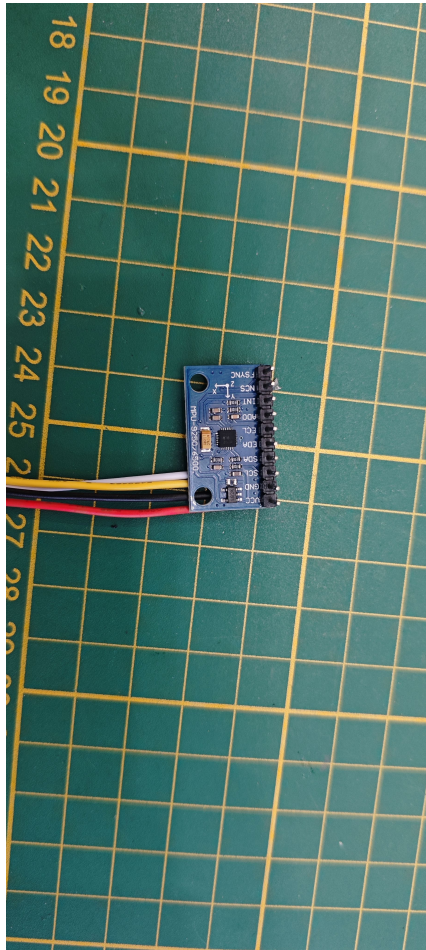
ESP32 GPIO13 (SDA) — SDA PCA9685 — SDA MPU6050
ESP32 GPIO14 (SCL) — SCL PCA9685 — SCL MPU6050
Common Ground — GND (ESP32, PCA9685, MPU6050, ESC signal ground)
PCA9685 CH0 — Steering Servo signal (CH1 in GCS)
PCA9685 CH1 — ESC Throttle signal (CH2 in GCS)
Auxiliary Power — V+ Servo rail per vehicle requirements (BEC / external battery)

PCA9685 channels are zero-indexed: CH0 corresponds to the first integer in the incoming CSV stream. Assign channel functions in GCS to match your physical wiring. Never bridge the ESC battery lead directly to logic pins.



ESP32-CAM module with OV2640 camera prepared for installation

Logic supply (VCC, 3.3V/5V) and servo rail power (V+, 5.0–7.4V) on the PCA9685 must remain electrically isolated. Never power high-current steering servos from the ESP32 3.3V LDO.



ESP32 board and PCA9685 controller mounted on model chassis

Inspect wiring with power disconnected. Camera parallel buses utilize nearly all remaining GPIO pins; pins allocated to the camera cannot be repurposed for auxiliary UART or I2C sensors. Elevate the model wheels for initial bench testing.

4. Configuration and IDE Setup

4.1. Wi-Fi and Network Settings

The baseline sketch defaults to `useStaticIP = true` with example address **192.168.31.111**, gateway **192.168.31.1**, and mask **255.255.255.0**. This must be adapted to your local network. Update `ssid` and `password`, and adjust the static IP or set `useStaticIP = false` to acquire an address via DHCP. Read the assigned IP from the 115200 bps serial console upon boot.

The firmware functions as a network client (station mode). It does not provide an onboard captive portal or web setup page in the baseline build.

4.2. Arduino IDE Environment

1. Install Arduino IDE and the **esp32 by Espressif Systems** package via Boards Manager.
2. Open `ESP32/ESP32.ino`. For V2 or V4, open their respective folders and preserve all header files (`.h`). For V3, ensure the file is placed in a folder matching its sketch name.
3. Install required libraries via Library Manager:
 - **Adafruit PWM Servo Driver Library** (and **Adafruit BusIO**)

- **MPU6050_light**
 - Core libraries (WiFi, WiFiUdp, Wire, esp_camera) are included in the Espressif ESP32 core.
4. Under **Tools** → **Board**, select **ESP32 Wrover Module** (or **AI Thinker ESP32-CAM**). Selecting the board in the IDE does not alter the #define macro inside the sketch; verify both match.
 5. Configure board options: **Partition Scheme: Huge APP (3MB No OTA/1MB SPIFFS)**, **Flash Frequency: 40MHz**, **PSRAM: Enabled** (for WROVER).
 6. Select the serial port, set upload speed to **115200**, and run **Verify**.
 7. If header errors occur, verify library installations. V3 additionally requires **TinyGPSPlus** and the specific IMU library enabled in its macros (e.g. MPU9250_WE, BNO08x, or BMX160).

4.3. Flashing Procedure

1. Connect the ESP32 via its onboard USB or an external 3.3V FTDI / CP2102 adapter (TX → RX, RX → TX, GND → GND).
2. For boards requiring manual boot mode (such as standard AI-Thinker), bridge **GPIO0 to GND**, press **RESET**, and click **Upload** in the IDE. Once flashing completes, remove the GPIO0-GND jumper and reset the board.
3. Open the Serial Monitor at **115200 bps** and reset the board.
4. Verify the startup sequence: Wi-Fi connection, IP address, I2C scanner results (PCA9685 at 0x40, MPU6050 at 0x68), and camera initialization.
5. Keep the module completely still while MPU6050 gyro offsets are being calculated.

5. Communications Protocol

The baseline ESP32UDPStrategy exchanges plain ASCII CSV strings terminated by a newline:

```
1500,1500,1500,1500,1500,1500,1500,1500\n
```

- Control Port: **UDP 12345** on ESP32.
- The parser processes up to 16 channels, clamping inputs to 1000–2000 µs and scaling them to 12-bit PCA9685 register values (205–410 at 50 Hz).
- Telemetry Port: **UDP 12347** on PC.
- IMU telemetry is formatted as JSON:

```
{"imu":{"ax":0.01,"ay":0.02,"az":0.98,"gx":0.1,"gy":0.0,"gz":0.0}}.
```
- The ESP32 sends telemetry every 50 ms to the IP address from which it received the latest control datagram. Telemetry will not transmit until GCS sends its first control packet.

6. Initial Bench Setup and GCS Configuration

1. Select your firmware variant, configure Wi-Fi credentials, flash the board, and open the serial monitor at 115200 bps.
2. Confirm the assigned IP address, I2C bus detection, and camera readiness.
3. Verify video in a web browser at http://ESP32_IP:81/stream. Close the browser tab before connecting GCS.
4. In GCS, open **Configuration** → **Connection Settings**:
 - Set **GCS Mode**: ESP32 (WiFi Lite Mode).

- On the **Tier 2 (WiFi ESP32)** card, select **Tier 2 Classic (Wi-Fi UDP / CSV Telemetry)** and transport **Wi-Fi UDP**.
 - **ESP32 / VPN IP Address:** Enter the board's reported IP.
 - **UDP Control Port:** Enter **12345**.
 - **FPV Camera Port:** Enter **81** (stream URL displays `http://ESP32_IP:81/stream`).
 - In **General** → **Advanced** → **GCS Listen Port**, verify port **12347**.
5. Save the profile, switch to **Cockpit**, select **Manual (Driver)**, and click **Connect**.
 6. Verify telemetry reception and actuate a test servo connected to PCA9685 CH0.
 7. Test failsafe: disconnect GCS or pause control for >500 ms and observe PWM behavior.

6.1. Control Mapping

The screenshot displays the RCSIM Ground Control Station interface. The 'Control Mapping' tab is active, showing a table of actions and their configurations. The right-hand panel provides diagnostics for the connected Moza Racing Ecosystem (SDK), including axis and button status. The bottom of the window features an 'Auto-saving...' indicator and buttons for profile management and diagnostics export.

Action	Assignment	Type	Reverse	Buttons
1 Axis: Channel 1	[Xbox/Microsoft Controller] Axis 0	Bidirectional (e.g. Steering)	☐	Assign
2 Axis: Channel 2 (Brake)	[Xbox/Microsoft Controller] Axis 2	Unidirectional (e.g. Throttle/Brake)	☐	Assign
3 Axis: Channel 2 (Throttle)	[Xbox/Microsoft Controller] Axis 5	Unidirectional (e.g. Throttle/Brake)	☐	Assign
4 Axis: Channel 3	[Xbox/Microsoft Controller] Axis 1	Bidirectional (e.g. Steering)	☐	Assign
5 Button: Channel 4	[Xbox/Microsoft Controller] Axis 0	Bidirectional (e.g. Steering)	☐	Assign
6 Button: Channel 5	[Dev_0300] Button 3	Bidirectional (e.g. Steering)	☐	Assign
7 Button: Channel 6	(none)	Bidirectional (e.g. Steering)	☐	Assign
8 Button: Channel 7 (Position 1 - 1000us)	(none)	Bidirectional (e.g. Steering)	☐	Assign
9 Button: Channel 7 (Position 2 - 1500us)	(none)	Bidirectional (e.g. Steering)	☐	Assign
10 Button: Channel 7 (Position 3 - 2000us)	(none)	Bidirectional (e.g. Steering)	☐	Assign
11 Axis: Channel 8	(none)	Bidirectional (e.g. Steering)	☐	Assign
12 Axis: Channel 9	(none)	Bidirectional (e.g. Steering)	☐	Assign
13 Axis: Channel 10	(none)	Bidirectional (e.g. Steering)	☐	Assign
14 Axis: Channel 11	(none)	Bidirectional (e.g. Steering)	☐	Assign
15 Axis: Channel 12	(none)	Bidirectional (e.g. Steering)	☐	Assign
16 Axis: Channel 13	(none)	Bidirectional (e.g. Steering)	☐	Assign
17 Axis: Channel 14	(none)	Bidirectional (e.g. Steering)	☐	Assign
18 Axis: Channel 15	(none)	Bidirectional (e.g. Steering)	☐	Assign
19 Axis: Channel 16	(none)	Bidirectional (e.g. Steering)	☐	Assign

Save Mapping

Controller Diagnostics

Moza Racing Ecosystem (SDK)

Axes

Axis 0:

Axis 1:

Axis 2:

Axis 3:

Axis 4:

Buttons

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Gears

Gear 0: N

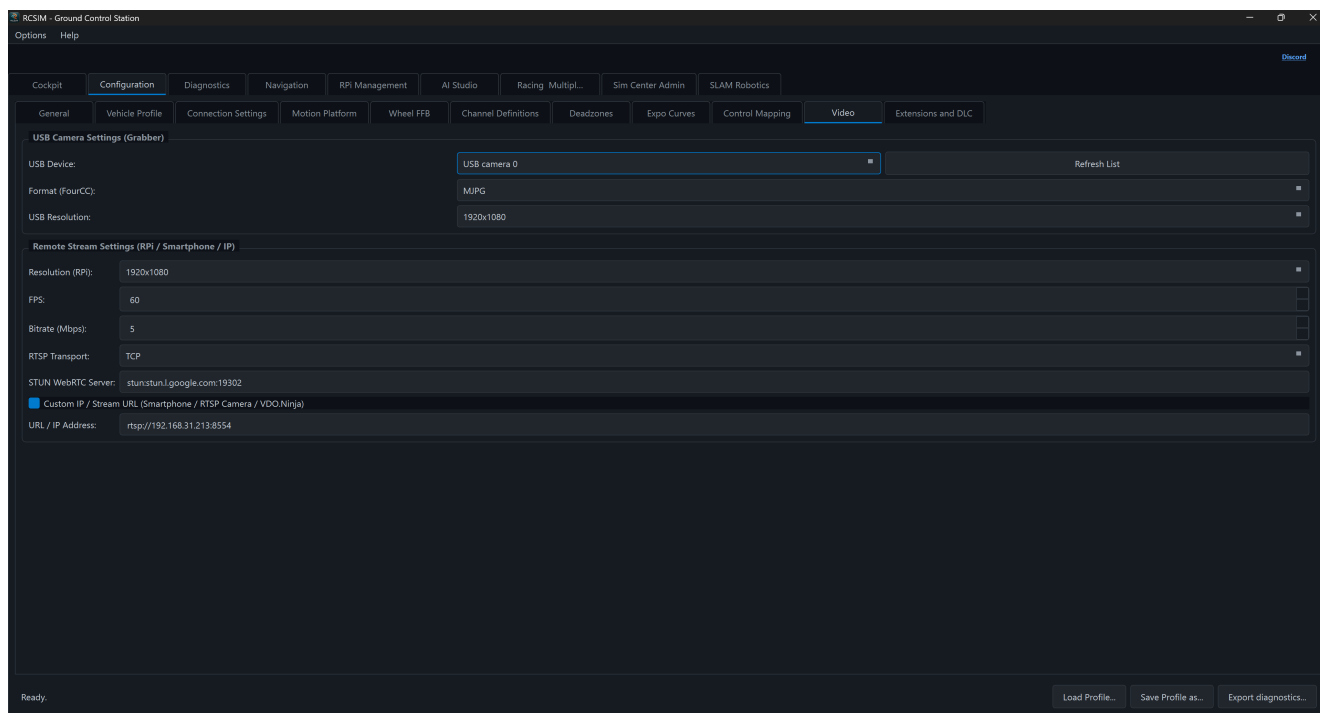
Devices are detected automatically.

Auto-saving... Load Profile... Save Profile as... Export diagnostics...

GCS Control Mapping Example

Under **Channel Definitions**, configure Steering as bidirectional and Throttle/Brake as split unidirectional axes. Under **Control Mapping**, click **Assign**, actuate the physical wheel or pedal, set polarity via Invert, and click **Save Mapping**. Verify neutral zero points and deadzones before powering the ESC.

6.2. Video Streaming



GCS Video Settings Example

Baseline ESP32 firmware serves **HTTP MJPEG**, not RTSP or WebRTC. In Cockpit, configure the stream source to point to `http://ESP32_IP:81/stream`. GCS RTSP and WebRTC fields are ignored by this sketch.

When PSRAM is detected, the sketch initializes the camera at **VGA 640×480**, JPEG quality 12, with dual frame buffers. Without PSRAM, it falls back to **CIF 400×296**, quality 15, and a single buffer.

7. Failsafe Behavior

The baseline firmware implements a **500 ms** watchdog timer. If no valid control packet is received within 500 ms:

- `triggerFailsafe()` executes, forcing all 16 PCA9685 channels to **1500 μ s** neutral PWM.
- This is a software PWM neutralization, not a physical power cut. Ensure your vehicle ESC interprets 1500 μ s as stop/neutral rather than active throttle.
- Resuming valid packets immediately restores active control.

8. Advanced Firmware Extensions: V2, V3, and V4

8.1. ESP32 V2: Closed-Loop PCA9685 Calibration

ESP32V2autocalibPCA.ino connects PCA9685 CH15 to ESP32 GPIO12. Upon boot, it outputs a 1500 μ s pulse on CH15, measures pulse duration with `pulseIn(12, HIGH)`, and iteratively adjusts the PCA9685 internal oscillator frequency until the error is under 3 μ s. Do not connect vehicle servos to CH15. Because GPIO12 is an ESP32 strapping pin, ensure it is not pulled HIGH externally during power-up.

8.2. ESP32 V3: GPS, Walksnail OSD, and Battery Monitoring

ESP32 V3 GPS + FPV adds peripheral telemetry:

- **NMEA GPS:** UART RX on **GPIO32** at **9600 bps**.
- **Walksnail Avatar OSD:** UART TX on **GPIO2**, RX on **GPIO15** at **115200 bps**.
- **Battery ADC:** Analog voltage divider on **GPIO33**. Never feed raw battery voltage (>3.3V) directly to GPIO33.
- Select your IMU via compile-time macros (`USE_MPU6050`, `USE_MPU9250`, `USE_BNO08X`, or `USE_BMX160`).

8.3. ESP32 V4: CRSF MultiLink Hub

`ESP32V4_CRSF_MultiLink.ino` targets the **ESP32-S3** and functions as a dual-link CRSF hub:

- **RF Radio Link:** Hardware UART on RX15 / TX16 at **420,000 bps**.
- **I2C Bus:** Reassigned to **SDA = GPIO8**, **SCL = GPIO9**.
- **Default IMU:** **LSM6DS3** (with MPU6050 compile option).
- **Control Format:** Binary CRSF frames over UDP, rather than ASCII CSV.
- **Watchdog:** 150 ms timeout. CH5 serves as ARM, and CH6 switches between RF, Network, and Auto link arbitration.
- Does not include camera MJPEG streaming.

9. Diagnostics and Recovery

- **No COM Port / Upload Fails:** Check data cable and serial drivers. For AI-Thinker, bridge GPIO0 to GND and cycle reset before upload; remove the bridge to boot.
- **Camera Init Failed (0x105 / 0x20003):** Verify PSRAM settings in IDE, reseal the camera ribbon cable, and check for adequate 5V current (>1.5A peak).
- **I2C Devices Not Detected (0x40 / 0x68):** Verify SDA/SCL pin assignments for your board build. Ensure common ground between ESP32 and PCA9685.
- **GCS Connected but No Servo Motion:** Confirm GCS Mode is ESP32 (WiFi Lite Mode). Verify PCA9685 servo rail power (V+). Check that CH0/CH1 match your assignments.
- **No Video in GCS:** Open `http://ESP32_IP:81/stream` in a browser. Ensure no other client holds the single MJPEG stream lock. Verify port 81 in GCS settings.
- **Latency / Dummy Telemetry:** The baseline GCS ESP32 strategy reports hardcoded ping (5.0 ms) and packet loss (0.0%). Evaluate real link performance using serial console logs.

10. Source Code Verification Trails

- `RCSIM_PC/pc_app/core/comm/esp32_strategy.py:23-30,51-74,83-128` — UDP transport, CSV sending, JSON telemetry decoding.
- `RCSIM_PC/pc_app/core/comm_manager.py:660-668` — ESP32 strategy instantiation.
- `ESP32/ESP32.ino:21-99` — board macros, pin definitions, camera configuration.
- `ESP32/ESP32.ino:116-123,156-162,215-299,301-355` — UDP ports, setup, control loop, failsafe, IMU handling.
- `ESP32V2/ESP32V2autocalibPCA/ESP32V2autocalibPCA.ino:12-14,164-202,296` — PCA9685 CH15 to GPIO12 calibration routine.
- `ESP32V3+GPS+FPV/ESP32 V3 GPS + FPV:18-39` — GPS, OSD, ADC, and multi-IMU definitions.

- ESP32V4_CRSF_MultiLink/ESP32V4_CRSF_MultiLink.ino:31-87,103-122,246-265 — ESP32-S3 pins, dual-link arbitration, CRSF parsing.