MAM15 Robot System — Comprehensive Developer Guide

Executive Summary

The MAM15 system is a **multi-node IoT robot** comprising four independent ESP32-based modules communicating via **LoRa (433 MHz)** and **ESP-NOW**. The architecture separates concerns into distinct functional layers: vision (camera), mobility (motor control), deployment (landing), and user interface (remote control). This document provides complete technical specifications for developers maintaining, extending, or integrating this codebase.

System Overview

Component	Role	MCU	Primary Protocol	Key Function
ESP32-CAM	Vision & Code Relay	ESP32-CAM	WiFi (Web Server) + Serial	Live video stream + 3-char code blink output
MAM15-Landolo	Deployment Unit	ESP32-C3 SuperMini	ESP-NOW Receiver	Servo control + state retention via deep sleep
MAM15-Motorvezerlo	Robot Brain	ESP32 DevKit v1	LoRa RX + ESP-NOW TX	Motor drive control + failsafe + landing relay
MAM15-Tavirianyito	Human Interface	ESP32 DevKit v1	LoRa TX	Button input → LoRa packet generation

Communication Topology

MAM15-Tavirianyito (LoRa TX)
|
v
[433 MHz LoRa]
|
v
MAM15-Motorvezerlo (LoRa RX + ESP-NOW TX)
|
+---> [ESP-NOW] ---> MAM15-Landolo (ESP-NOW RX)
|
+---> Motor PWM (Left/Right forward/reverse)
|
+---> LED Flash (GPIO 22)

---

1. Hardware Architecture

1.1 ESP32-CAM Module

Board: ESP32-CAM (OV2640 sensor)
Primary Function: Live video streaming + 3-character code transmission
Key Pins:

Pin	Function	Notes
GPIO 32	PWDN (Power Down)	Camera power enable (active LOW)
GPIO 13	LED Pin	Status LED (code blink output)
GPIO 12	External Trigger	Trigger input from external device
GPIO 2	External Controller Reset	Reset output to motor controller
GPIO 0–39 (camera pins)	OV2640 bus	Y0–Y9, PCLK, HREF, VSYNC, SDA/SCL

Network Configuration:

- **WiFi Mode:** SoftAP (Access Point)
- **SSID:** "We Are Engineers"
- **Password:** "12341234"
- **IP:** 192.168.4.1 (default AP gateway)

HTTP Ports:

Port	Endpoint	Purpose
80	`/codes`	Web UI + REST API
81	`/stream`	MJPEG video stream
32769	Stream control	Advanced streaming ops
32768	Code control	Code management RPC

Memory:

- **PSRAM:** 4 MB (if detected) – Frame buffer in external PSRAM
- **DRAM:** 320 KB – Fallback if PSRAM unavailable
- **NVS (Preferences):** 4 codes stored persistently

1.2 MAM15-Landolo Unit

Board: ESP32-C3 Super Mini
Primary Function: Servo control + landing sequencing
Key Pins:

Pin	Function	Role
GPIO 2	SERVO1	Landing mechanism servo 1
GPIO 3	SERVO2	Landing mechanism servo 2
GPIO 8	LED Pin	Status indicator (blink on activation)
GPIO 9	Reset Button	Wakeup button (GPIO_INTR_LOW_LEVEL)

Servo Positions (degrees):

- **SERVO_CLOSED_POSITION = 5°** – Locked state
- **SERVO_OPEN_POSITION = 175°** – Deployed state

Power Management:

- **Deep Sleep Mode:** Enabled after landing + LED blink sequence
- **Wakeup Trigger:** Reset button (GPIO 9 low pulse)
- **RTC Storage:** `RTC_DATA_ATTR int bootCount` preserves boot counter across sleep

Boot Sequence:

1. Boot 1: Servos initialize (close), ready for ESP-NOW command
2. Boot N (N>1): Servos close (locked), awaiting reset button press

---

1.3 MAM15-Motorvezerlo

Board: ESP32 DevKit 1
Primary Function: Motor control + communication hub
Key Pins:

Pin	Function	Protocol	Notes
GPIO 18	LoRa SCK	SPI	Clock
GPIO 19	LoRa MISO	SPI	Master In
GPIO 23	LoRa MOSI	SPI	Master Out
GPIO 5	LoRa SS	SPI	Chip Select
GPIO 14	LoRa RESET	Digital	Active LOW reset
GPIO 2	LoRa DIO0	Digital	Packet ready interrupt
GPIO 32	Left Motor Forward	PWM (CH 0)	LEDC channel 0, 50 Hz, 8-bit
GPIO 27	Left Motor Reverse	PWM (CH 1)	LEDC channel 1
GPIO 25	Right Motor Forward	PWM (CH 2)	LEDC channel 2
GPIO 26	Right Motor Reverse	PWM (CH 3)	LEDC channel 3
GPIO 22	LED Flash	Digital	Landing state indicator toggle

Radio Configuration:

- **LoRa Frequency:** 433 MHz (433E6 Hz)
- **Baud Rate (Serial):** 115200 bps

Motor Control:

- **PWM Frequency:** 50 Hz
- **PWM Resolution:** 8-bit (0–255)
- **Speed Levels:**
    - Level 1: 255 (max)
    - Level 2: 120 (medium)
    - Level 3: 40 (slow)

Failsafe Configuration:

- **Timeout:** 300 ms without valid LoRa packet
- **Action:** Motor stop + failsafe flag active

1.4 MAM15-Tavirianyito

Board: ESP32 DevKit 1
Primary Function: Button input → LoRa packet TX
Button Inputs:

Button	Function	Output Bit
Up	Forward	motorCommand bit 0 (left forward)
Down	Backward	motorCommand bit 1 (left backward)
Left	Left Turn	motorCommand bit 2 (right forward)
Right	Right Turn	motorCommand bit 3 (right backward)
Speed	Speed Cycle	speedButtonPressed flag
Landing	Deploy	landingState toggle

LoRa TX Configuration:

- **Frequency:** 433 MHz (same as MAM15-Motorvezerlo RX)
- **Packet Structure:** 6 bytes (see Section 2.2)

---

2. Communication Protocols

2.1 LoRa Packet Specification

Payload Size: 6 bytes
Structure:

Byte	Field	Purpose
0	Robot ID	Device identifier (0x45 = 69 decimal)
1	Motor Command	8-bit bitmask for motor control
2	Speed Button	Boolean flag (0x00 or 0x01)
3	Landing State	Boolean flag (0x00 or 0x01)
4	CRC High	CRC-16 high byte
5	CRC Low	CRC-16 low byte

Motor Command Bitmask (Byte 1):

Bit	Function
0	Left Motor Forward
1	Left Motor Backward
2	Right Motor Forward
3	Right Motor Backward
4–7	Reserved (must be 0)

Example Motor Commands:

Command	Hex	Binary	Motion
Stop	0x00	0000 0000	No motion
Forward	0x05	0000 0101	Both forward (LF + RF)
Backward	0x0A	0000 1010	Both backward (LB + RB)
Left Turn	0x04	0000 0100	Right forward only
Right Turn	0x01	0000 0001	Left forward only

CRC Calculation (CRC-16-CCITT):

- **Polynomial:** 0x1021
- **Initial Value:** 0xFFFF
- **Final XOR:** 0x0000
- **Input Reflection:** True
- **Output Reflection:** True
- **CRC computed over:** Bytes 0–3 (Robot ID through Landing State)

```
CRC16 crcCalculator(0x1021, 0xFFFF, 0x0000, true, true);
crcCalculator.restart();
crcCalculator.add(receivedPacket, 4); // Add first 4 bytes
uint16_t calculatedCRC = crcCalculator.getCRC();
```

2.2 ESP-NOW Protocol (MAM15-Landolo Command)

Direction: MAM15-Motorvezerlo (TX) → MAM15-Landolo (RX)
Payload: 1 byte

Byte	Value	Meaning
0	0x01	Landing Activate (servo open)
0	0x00	Landing Deactivate (servo close)

MAC Address Configuration:

Landoló MAC: 1C:DB:D4:D5:D0:28

Defined in MAM15-Motorvezerlo `settings.h`:

```
#define LANDOLO_MAC_0 0x1C
#define LANDOLO_MAC_1 0xDB
#define LANDOLO_MAC_2 0xD4
#define LANDOLO_MAC_3 0xD5
#define LANDOLO_MAC_4 0xD0
#define LANDOLO_MAC_5 0x28
```

ACK Handling:

- **ACK Code:** 200 (0xC8)
- **Trigger:** Servo successfully opened
- **Effect:** MAM15-Motorvezerlo permanently disables ESP-NOW for this session

---

3. Module Documentation

3.1 ESP32-CAM

File Structure

File	Purpose
`settings.h`	Global config: GPIO pins, WiFi SSID/PSW, server ports, defaults
`camera.h`	OV2640 initialization + camera configuration
`storage.h`	Preferences (NVS) + circular log buffer + code storage
`webserver.h`	HTTP server handlers + MJPEG streaming
`handlers.h`	REST API for codes, control, config
`blink.h`	LED blink task (FreeRTOS) + code transmission
ESP32-CAM_video_stream.ino	Setup + loop (server start)

Key Classes

1. Camera Initialization (`camera.h`)

bool initCamera()

- Configures OV2640 sensor pins and LEDC timing
- Detects PSRAM availability; adjusts frame size accordingly
- Implements retry logic: 20 MHz → 10 MHz → 8 MHz XCLK
- Logs sensor PID, XCLK freq, frame size, JPEG quality
- Returns `false` on all retry failures

2. Code Storage (`storage.h`)

void loadCodes() // Load 4 codes from NVS
void saveCodes() // Persist codes to NVS
void addLog(String msg) // Circular buffer append
String getLogHTML() // Render log as HTML

- **Max Codes:** 4, each 3 characters (e.g., "ABC")
- **Active Code:** Index 0–3 or -1 (inactive)
- **Blink Settings:** BAUD rate (default 300), pause between codes (500 ms)

3. LED Blink Transmission (`blink.h`)

void blinkCode(String code)

- **Protocol:** RS-232-like serial over LED
- **Frame:** START (LOW) + 8 data bits (LSB first) + STOP (HIGH)
- **Bit Delay:** `1000000 / BLINK_BAUD` microseconds
- **Usage:** Relay codes to external LED reader (e.g., ID card scanner)

Web API Reference

1. GET `/codes` – Web UI + code management

GET http://192.168.4.1/codes

Returns: HTML page with 3-panel layout (settings, video stream, log)

2. POST `/codes` – Add or delete code

POST http://192.168.4.1/codes
Content-Type: application/x-www-form-urlencoded

newcode=ABC // Add 3-char code
delete=0 // Delete code at index 0

3. GET `/status` – JSON status

GET http://192.168.4.1/status

Response:
{
"externalTriggerEnabled": false,
"cameraQuality": 30,
"blinkBaud": 300,
"pauseBetweenCodes": 500,
"activeCode": 0,
"shouldBlink": true,
"codes": ["ABC", "DEF", "", ""]
}

4. POST `/control` – Start/stop blink

POST http://192.168.4.1/control
action=start&id=0 // Start blinking code 0
action=stop // Stop blinking
action=reset // Reset motor controller (GPIO 2 pulse)

3.2 MAM15-Landolo

File Structure

File	Purpose
`settings.h`	GPIO pins, servo positions, LED timing, debug flags
servo_control.h	Servo open/close/status methods
led_control.h	LED blink state machine + timer
`communication.h`	ESP-NOW receiver + callback
sleep_manager.h	Deep sleep initialization + wakeup config
`MAM15-Landolo.ino`	Setup + loop + landing activation handler

Key Classes

1. Servo Control (`servo_control.h`)

```
class ServoControl {
void init()
void open() // Write 175° to both servos
void close() // Write 5° to both servos
void setToStartPosition()
bool getIsOpen() const
}
```

- Attaches to GPIO 2 and GPIO 3 via Arduino Servo library
- Two servos controlled in parallel (mirrored positions)

2. LED Control (`led_control.h`)

```
class LedControl {
void init()
void startBlink() // Start 3000 ms blink sequence
void stopBlink()
bool update() // Returns true when sequence complete
bool getIsBlinking() const
void turnOff()
}
```

- **Blink Duration:** 3000 ms
- **On/Off Cycle:** 100 ms on, 100 ms off
- **Total Blinks:** ~15 complete cycles

3. ESP-NOW Communication (`communication.h`)

```
class Communication {
bool init(CommandCallback cb) // Initialize ESP-NOW + register callback
bool sendAck(byte ackCode) // Send 200 (ACK_SERVO_OPENED)
void disconnect() // Deinit ESP-NOW + WiFi off
}
```

- **STA Mode:** WiFi disabled initially (power save)
- **Callback:** `handleCommand(byte cmd)` invoked on packet RX

4. Sleep Manager (`sleep_manager.h`)

```
class SleepManager {
void initWakeupButton() // Configure GPIO 9 with PULLUP
void enterDeepSleep() // esp_deep_sleep_start()
void printBootInfo(int count)
}
```

- **Wakeup:** GPIO 9 falling edge (LOW pulse)
- **RTC Persistence:** `bootCount` survives deep sleep
- **Servos State:** Remain at last commanded position during sleep

State Machine: Landing Sequence

```
[Initial]
|
+-- ESP-NOW packet (CMD = 1) received
|
v
[Activate Landing]
├─ servos.open() → 175°
├─ comm.sendAck(200)
└─ led.startBlink()
|
v
[LED Blink]
├─ 100ms ON / 100ms OFF (×15 cycles = 3000 ms)
└─ led.update() returns true when done
|
v
[Sleep Entry]
├─ comm.disconnect() → WiFi/ESP-NOW off
└─ sleepMgr.enterDeepSleep()
|
v
[Deep Sleep]
├─ Servos remain OPEN (175°)
├─ Await GPIO 9 LOW pulse
└─ Boot counter increments on wakeup
|
v
[Wakeup]
└─ servos.close() → 5° (locked)
```

3.3 MAM15-Motorvezerlo

File Structure

File	Purpose
`settings.h`	Global config: ROBOT_ID, LoRa pins, motor PWM, CRC, debug flags
lora_communication.h	LoRa RX driver + health monitor state machine
packet_handler.h	CRC validation + packet parsing
motor_control.h	Motor PWM + command validation + speed levels
espnow_communication.h	Landing command TX + LED flash + one-shot ACK handler
`failsafe.h`	Timeout watchdog + motor stop trigger
`MAM15-Motorvezerlo.ino`	Composition root + main loop

Key Classes

1. LoRa Communication (`lora_communication.h`)

class LoRaCommunication {
bool init()
bool restart()
void checkHealth()
int parsePacket()
byte read()
void updateReceivedTime()
LoRaState getState() const
bool isHealthy() const
}

enum LoRaState { LORA_OK, LORA_DISCONNECTED, LORA_RECONNECTING }

Health Monitor State Machine:

[LORA_OK]
├─ Every 5000 ms: Check last packet time
└─ If (now - lastRX) > 5000 ms → LORA_RECONNECTING
|
v
[LORA_RECONNECTING]
├─ Attempt restart() up to 3 times
├─ Timeout: 10000 ms
└─ If success → LORA_OK (reset lastReceivedTime)
|
v
[LORA_DISCONNECTED]
└─ Motor stop; wait for next health check

2. Packet Handler (`packet_handler.h`)

```
class PacketHandler {
bool validatePacketSize(int size)
PacketData parsePacket(byte* data)
}

struct PacketData {
byte robotId;
byte motorCommand;
bool speedButtonPressed;
bool landingState;
uint16_t crc;
bool valid;
}
```

- **Robot ID Filter:** Packets with ID ≠ 69 are dropped (logged as warning)
- **CRC Check:** Invalid CRC → `data.valid = false`
- **Validation Order:** Size → CRC → Robot ID

3. Motor Control (`motor_control.h`)

```
class MotorControl {
bool init() // Attach 4 PWM channels
void control(L_fwd, L_bwd, R_fwd, R_bwd)
void stop()
void handleSpeedButton(bool pressed)
void executeCommand(byte motorCmd)
bool validateCommand(byte cmd)
}
```

Command Validation:

- Bit 0 & Bit 1 both set → Conflict (left motor can't go forward + backward)
- Bit 2 & Bit 3 both set → Conflict (right motor can't go forward + backward)
- Valid: 0x00, 0x01, 0x02, 0x04, 0x08, 0x05 (forward), 0x0A (backward), etc.

Speed Cycling:

Button press (falling edge) → Index = (Index + 1) % 3

- Level 0: 255 (max)
- Level 1: 120 (medium)
- Level 2: 40 (slow)

4. ESP-NOW Communication (`espnow_communication.h`)

```
class ESPNowCommunication {
bool init()
void sendLandingCommand(bool state)
void handleLandingState(bool state)
void shutdownPermanently()
bool isPermanentlyDisabled() const
}
```

Landing State Handler (Dual Function):

Input: landingState (from LoRa packet)

1. **ESP-NOW** (if active):
    - Send command (0x00 or 0x01) to MAM15-Landolo
    - Listen for ACK (0xC8 = 200)
    - On ACK received → shutdownPermanently()
2. **LED Flash** (GPIO 22, always):
    - landingState HIGH → digitalWrite(22, HIGH)
    - landingState LOW → digitalWrite(22, LOW)
    - Independent of ESP-NOW status

5. Failsafe (`failsafe.h`)

```
class Failsafe {
void init()
void reset() // Reset timeout timer
bool check() // Returns true if timeout exceeded
bool isActive() const
}
```

Failsafe Logic:

reset() called → lastSafeTime = now()

```
Every loop iteration:
├─ if (now - lastSafeTime) > 300 ms:
│  ├─ failsafeActive = true
│  └─ Motor stop (in main loop)
└─ else: failsafeActive = false
```

Main Control Loop

```
void loop() {
// 1. Health check
lora.checkHealth();
if (lora.getState() != LORA_OK) {
motors.stop();
return;
}

// 2. Try to parse packet
int packetSize = lora.parsePacket();
if (!packetSize) {
if (failsafe.check()) motors.stop();
return;
}

// 3. Update timers
lora.updateReceivedTime();
failsafe.reset();

// 4. Validate size
if (!packetHandler.validatePacketSize(packetSize)) return;
```

```
// 5. Read packet bytes
byte packet[PACKET_SIZE];
for (int i = 0; i < PACKET_SIZE; i++) {
packet[i] = lora.read();
}

// 6. Parse & validate
PacketData data = packetHandler.parsePacket(packet);
if (!data.valid) return;

// 7. Execute
espnow.handleLandingState(data.landingState);
motors.handleSpeedButton(data.speedButtonPressed);
motors.executeCommand(data.motorCommand);
}
```

3.4 MAM15-Tavirianyito

File Structure

File	Purpose
`settings.h`	Button pins, LoRa pins, Robot ID, frequency
button_handler.h / `.cpp`	Button input + debounce logic
`communication.h` / `.cpp`	LoRa packet generation + TX
`MAM15-Tavirianyito.ino`	Setup + loop

Key Classes

1. Button Handler (`button_handler.cpp`)

```
class ButtonHandler {
void init()
void update()
bool isUpPressed() const
bool isDownPressed() const
// ... other button getters
bool isSpeedButtonPressed() const
bool isLandingButtonPressed() const
}
```

- **Debounce:** Typically 20–50 ms per button
- **State:** Tracks current and previous states to detect edges

2. Communication (`communication.cpp`)

```
class Communication {
void init()
void sendCommand(byte motorCmd, bool speedBtn, bool landingBtn)
}
```

Packet Generation:

```
byte packet[6];
packet[0] = ROBOT_ID; // 69
packet[1] = motorCmd; // Bitmask from buttons
packet[2] = speedBtn ? 0x01 : 0x00;
packet[3] = landingBtn ? 0x01 : 0x00;

// CRC of first 4 bytes
CRC16 crc(0x1021, 0xFFFF, 0x0000, true, true);
crc.add(packet, 4);
uint16_t crcVal = crc.getCRC();
packet[4] = (crcVal >> 8) & 0xFF;
packet[5] = crcVal & 0xFF;

LoRa.beginPacket();
LoRa.write(packet, 6);
LoRa.endPacket();
```

Main Loop

```
void loop() {
buttonHandler.update();

byte motorCmd = 0;
if (buttonHandler.isUpPressed()) motorCmd |= 0x05; // LF + RF
if (buttonHandler.isDownPressed()) motorCmd |= 0x0A; // LB + RB
if (buttonHandler.isLeftPressed()) motorCmd |= 0x04; // RF only
if (buttonHandler.isRightPressed()) motorCmd |= 0x01; // LF only

bool speedBtn = buttonHandler.isSpeedButtonPressed();
bool landingBtn = buttonHandler.isLandingButtonPressed();

comm.sendCommand(motorCmd, speedBtn, landingBtn);
delay(50); // Send ~20 times per second
}
```

---

4. Integration & Deployment

4.1 Prerequisites

Software:

- Arduino IDE 2.x or PlatformIO
- ESP32 Boards package (v2.0+)
- Libraries:
  - `esp_now.h` (built-in)
  - `LoRa.h` by Sandeep Mistry
  - `Servo.h` (Arduino standard)
  - `CRC16.h` (ArduinoCRC16)
  - `Preferences.h` (built-in for ESP32)

Hardware:

- 4× ESP32 or Arduino boards (as specified)
- 2× RFM95W LoRa modules (433 MHz)
- 2× Servo motors (SG90 or equivalent)
- 1× OV2640 camera module
- 4× LEDs + resistors (for status indication)
- Power supplies: 5V USB for controllers, appropriate servo power

4.2 Configuration Steps

Step 1: ESP32-CAM Setup

1. **Install board:** Arduino IDE → Board Manager → Search "esp32" → Install "ESP32" by Espressif
2. **Select board:** Tools → Board → ESP32 → "AI Thinker ESP32-CAM"
3. **Edit `settings.h`:**
   const char* AP_SSID = "Your_Network";
   const char* AP_PASSWORD = "Your_Password";
4. **Upload:** Connect USB programmer, set mode to "IO0 GND" for download mode
5. **Verify:** Open serial monitor, see startup messages

Step 2: MAM15-Landolo Setup

1. **Install board:** Boards → ESP32 → ESP32-C3 Dev Module
2. **Update MAC address in MAM15-Motorvezerlo's `settings.h`** (read from MAM15-Landolo via serial on startup)
3. **Upload `MAM15-Landolo.ino`**
4. **Verify:** Serial output shows boot count and servo initialization

Step 3: MAM15-Motorvezerlo Setup

1. **Select board:** ESP32 (same as camera, or your variant)
2. **Install LoRa library:** Library Manager → "LoRa" → Install Sandeep Mistry's version
3. **Verify LoRa pins** in `settings.h` match your wiring
4. **Edit MAC address:** Set `LANDOLO_MAC_*` to MAM15-Landolo's actual MAC
5. **Verify Robot ID:** Ensure `ROBOT_ID` = 69 (must match packets from remote)
6. **Upload `MAM15-Motorvezerlo.ino`**
7. **Verify:** Serial shows LoRa init + health monitoring started

Step 4: MAM15-Tavirianyito Setup

1. **Select board:** ESP32 → ESP32 Dev Module
2. **Verify LoRa pins** in `settings.h`
3. **Edit `ROBOT_ID`:** Must be 69 (same as MAM15-Motorvezerlo receiver)
4. **Upload `MAM15-Tavirianyito.ino`**
5. **Test:** Press buttons; observe LoRa TX activity on serial monitor

4.3 Debugging

Enable all debug output: Edit each `settings.h`:

```
#define DEBUG_ENABLED true
#define DEBUG_LORA true
#define DEBUG_MOTOR true
#define DEBUG_ESPNOW true
#define DEBUG_SERVO true
#define DEBUG_LED true
#define DEBUG_FAILSAFE true
```

Serial Monitoring:

All units at 115200 baud

MAM15-Motorvezerlo startup:
 MOTORVEZÉRLŐ ROBOT INDÍTÁSA
 PWM inicializálás sikeres
 LoRa inicializálva
 ESP-NOW inicializálva
 Motorvezérlő KÉSZEN

MAM15-Tavirianyito button press:
⬅ Gomb: LEFT megnyomva
 Sebesség váltás: 255 → 120

MAM15-Landolo activation:
 PARANCS ÉRKEZETT: 1
 LANDOLÓ AKTIVÁLVA!
 Servók NYITVA (175°)
 LED villogás elindítva

5. Troubleshooting Guide

5.1 LoRa Communication Issues

Symptom: "□ LoRa: Nincs csomag 5 másodperc alatt"

Causes:

1. LoRa module not powered
2. Antenna disconnected or damaged
3. Frequency mismatch (check 433 MHz both sides)
4. SPI pins incorrectly wired

Solutions:

- Verify power supply (3.3V, 500 mA)
- Check SPI connections: SCK, MOSI, MISO, SS, RESET, DIO0
- Test with serial debug: observe "□ LoRa modul újraindítása"
- Use multimeter to test antenna connection

5.2 CRC Errors

Symptom: "□ Hibás CRC - csomag elvetve!"

Causes:

1. Corrupted LoRa RX due to interference
2. MAM15-Tavirianyito CRC calculation bug
3. MAM15-Motorvezerlo CRC library mismatch

Solutions:

- Reduce LoRa spreading factor (lower link budget, less noise resilience)
- Verify CRC library: `CRC16 crc(0x1021, 0xFFFF, 0x0000, true, true)`
- Test with known-good packets (static test data)
- Check MAM15-Tavirianyito CRC generation matches spec

5.3 Motor Control Problems

Symptom: Motors not responding or only one motor works

Causes:

1. PWM pin conflict or not initialized
2. Motor driver H-bridge issue
3. Command validation rejecting packet

Solutions:

- Verify PWM init: "□ PWM inicializálás sikeres" in serial
- Check motor pin voltages with oscilloscope (should see ~100 Hz PWM)
- Validate command byte (check bits 0–3 only)
- Test motors with manual PWM: `ledcWrite(GPIO, 255)` to set full speed

5.4 ESP-NOW Landing Issues

Symptom: MAM15-Landolo servos don't open on landing command

Causes:

1. Incorrect MAM15-Landolo MAC address in MAM15-Motorvezerlo settings
2. MAM15-Landolo not initialized in ESP-NOW RX mode
3. MAM15-Motorvezerlo ESP-NOW init fails

Solutions:

- Capture MAM15-Landolo MAC from serial: `□ Landoló cél MAC: 1C:DB:D4:D5:D0:28`
- Update MAM15-Motorvezerlo `LANDOLO_MAC_*` defines
- Verify MAM15-Landolo receives packets: look for "□ PARANCS ÉRKEZETT: 1"
- Check ESP-NOW debug output for peer add success

5.5 Failsafe Activating Unexpectedly

Symptom: "□ FAILSAFE AKTIVÁLVA - Nincs kommunikáció!" continuously

Causes:

1. LoRa health check failing
2. Packet parsing rejects all packets
3. Radio interference

Solutions:

- Verify LoRa health monitor: "□ LoRa modul sikeresen újracsatlakozott!"
- Check packet size: must be exactly 6 bytes
- Check ROBOT_ID in packet matches settings (69)
- Reduce LoRa TX power or change frequency to avoid interference

6. Code Quality & Maintenance

6.1 Coding Standards

- **Header Guards:** All `.h` files use `#ifndef _H`, `#define _H`, `#endif`
- **Namespaces:** Classes use CamelCase; methods use camelCase; constants use UPPER_CASE
- **Comments:** Hungarian notation emojis for quick visual scanning
- **Debug Macros:** Centralized `DEBUG_*` flags in `settings.h`
- **Circular Dependencies:** Avoided via forward declarations; includes organized top-down

6.2 Testing Checklist

Before deployment:

- [ ] All four nodes boot successfully
- [ ] LoRa health monitor shows "LORA_OK" status
- [ ] Motor commands execute without CRC errors
- [ ] Speed button cycles through 3 levels
- [ ] Landing button activates MAM15-Landolo + servo opens + ACK received
- [ ] Failsafe triggers after 300 ms without packet (motors stop)
- [ ] ESP32-CAM streams video on WiFi
- [ ] LED flash (GPIO 22) toggles with landing state
- [ ] Deep sleep + reset button wake cycle works
- [ ] Telemetry: Serial output shows expected state transitions

6.3 Future Extensions

Possible improvements:

1. **Autonomous Mode:** Replace manual control with waypoint navigation + IMU
2. **Battery Monitoring:** Analog pin for voltage feedback + low-battery failsafe
3. **Logging:** SD card data logging for flight records
4. **OTA Updates:** WiFi-based firmware updates for ESP32-CAM and MAM15-Motorvezerlo
5. **Sensor Fusion:** Add gyro/accelerometer for tilt correction
6. **Redundant Communication:** Secondary LoRa frequency or fallback to long-range RF module

---

7. API Reference

7.1 Motor Control API

motors.init() // → bool: PWM attach success
motors.control(l_fwd, l_back, r_fwd, r_back) // → void: Apply PWM
motors.stop() // → void: All motors OFF
motors.handleSpeedButton(pressed) // → void: Cycle speed levels
motors.executeCommand(motorCmd) // → void: Validate + apply motor command
motors.validateCommand(cmd) // → bool: Check for conflicts

7.2 LoRa API

lora.init() // → bool: Module ready
lora.restart() // → bool: Hard reset + re-init
lora.checkHealth() // → void: Monitor + reconnect if needed
lora.parsePacket() // → int: Bytes available (0 if none)
lora.read() // → byte: Read one byte
lora.updateReceivedTime() // → void: Reset packet timeout
lora.getState() // → LoRaState: Current state enum
lora.isHealthy() // → bool: Module status

7.3 Failsafe API

failsafe.init() // → void: Set baseline time
failsafe.reset() // → void: Clear timeout on valid packet
failsafe.check() // → bool: True if 300 ms timeout exceeded
failsafe.isActive() // → bool: Currently in failsafe mode

7.4 Servo API (MAM15-Landolo)

servos.init() // → void: Attach to GPIO
servos.open() // → void: Write 175° (deployed)
servos.close() // → void: Write 5° (locked)
servos.setToStartPosition() // → void: Set closed on boot
servos.getIsOpen() // → bool: Current position

7.5 ESP-NOW Landing API

espnow.init(callback) // → bool: ESP-NOW ready
espnow.sendLandingCommand(state) // → void: Send 0x00 or 0x01
espnow.handleLandingState(state) // → void: TX command + toggle LED
espnow.shutdownPermanently() // → void: Disable ESP-NOW after ACK
espnow.isPermanentlyDisabled() // → bool: ACK received

---

8. Glossary

Term	Definition
LoRa	Long Range radio modulation; 433 MHz ISM band in EU
ESP-NOW	Espressif's low-power peer-to-peer WiFi protocol
Failsafe	Automatic motor stop if no valid command for >300 ms
Deep Sleep	Minimal power state; wakeup via GPIO interrupt
CRC-16	Cyclic Redundancy Check; error detection over 4-byte payload
PWM	Pulse Width Modulation; 50 Hz frequency for motor speed control
MJPEG	Motion JPEG; streaming video format over HTTP
NVS	Non-Volatile Storage (ESP32 flash partition)
SPI	Serial Peripheral Interface; LoRa module bus
PSRAM	Pseudo Static RAM; external 4 MB on ESP32-CAM

---

9. Version History

Version	Date	Changes
1.0	2025-12-04	Initial release: 4-node system, LoRa + ESP-NOW, motor control + landing

---

Appendix A: Pin Assignment Summary

ESP32-CAM Module

Pin	Function	Usage
GPIO 32	PWDN	Camera power
GPIO 13	LED	Status blink
GPIO 12	External Trigger	External input
GPIO 2	Controller Reset	Reset output

MAM15-Landolo

Pin	Function	Usage
GPIO 2	SERVO1	Servo control
GPIO 3	SERVO2	Servo control
GPIO 8	LED	Status indicator
GPIO 9	Reset Button	Wakeup trigger

MAM15-Motorvezerlo

Pin	Function	Usage
GPIO 18	LoRa SCK	SPI clock
GPIO 19	LoRa MISO	SPI master in
GPIO 23	LoRa MOSI	SPI master out
GPIO 5	LoRa SS	SPI chip select
GPIO 14	LoRa RESET	Module reset
GPIO 2	LoRa DIO0	Packet ready
GPIO 32	Left Motor Forward	PWM channel 0
GPIO 27	Left Motor Reverse	PWM channel 1
GPIO 25	Right Motor Forward	PWM channel 2
GPIO 26	Right Motor Reverse	PWM channel 3
GPIO 22	LED Flash	Landing state

END OF DOCUMENT

© 2025 We Are Engineers Team. Confidential.