Final Internship Report

Hardware and Software Design, and Functional Pool Testing of a Mini-ROV

Ewan Chapuzet

Master program	Master in Autonomous Mobile Robotics
Institution	ENSTA Campus Brest
Class	2026
Academic supervisor	Lionnel Lapierre
Internship supervisor	Fausto Ferreira
Host laboratory	LABUST, University of Zagreb FER, Croatia

Acknowledgments

I would like to thank the University of Zagreb FER and my internship supervisor, Fausto Ferreira, for the opportunity to carry out this internship within the LABUST laboratory. I also thank Bernard and Marijan for their technical support and availability throughout the project. Finally, I thank Lionnel Lapierre for introducing this opportunity and for his academic guidance during the internship.

Glossary

ROV	Remotely Operated Vehicle.
AUV	Autonomous Underwater Vehicle.
IMU	Inertial Measurement Unit.
ODR	Output Data Rate.
LPF	Low-Pass Filter.
UDP	User Datagram Protocol.
RS-485	Recommended Standard 485.
ROS2	Robot Operating System 2.
BAR30	Blue Robotics BAR30 (MS5837-30BA).
PWM	Pulse Width Modulation.
ESC	Electronic Speed Controller.
BMS	Battery Management System.
NMS	Non-Maximum Suppression.
ONNX	Open Neural Network Exchange.
QoS	Quality of Service.
OTA	Over-The-Air.
DVL	Doppler Velocity Log.
USBL	Ultra-Short Baseline.

Contents

Introduction **1**

Chapter 0: Project Context and Specifications **2**
0.1 Host Laboratory Overview 2
0.2 Requirements and Dimensional Constraints for the Mini-ROV 3

1 Hardware design **4**
1.1 Sourcing, Comparative Analysis, and Component Selection 4
1.2 Mechanical Integration and Space Optimization 6
1.3 Theoretical Performance (Endurance, Depth, and Speed) 7

2 Software Architecture and Embedded System **10**
2.1 Communication Protocols and Network Infrastructure 10
2.2 Sensor Data Acquisition and Control Chain 14
2.3 Real-Time Video Flow Optimization and AI Integration 18
2.4 Safety Algorithms and Degraded Modes (Fail-Safe) 21

3 Iterative Development, Pool Testing, and Outlook **24**
3.1 First Prototype: Assembly and Initial Functional Validation 24
3.2 Second Prototype: Electronics Redesign and System Hardening 28
3.3 Performance Assessment and Recommendations for the Global Project . . 34

Conclusion **36**

A Appendices **37**
A.1 Detailed Component Table and Budget Estimate 37
A.2 Endurance Calculation Assumptions 38
A.3 Additional Mechanical Integration Illustrations 41
A.4 Oscilloscope Measurement at ESP32 Startup 41
A.5 Test Traceability and Analysis Reproducibility 41

Bibliography **43**

Introduction

Marine environment monitoring increasingly relies on robotic platforms able to provide in situ observations at manageable cost. In this context, mini-ROVs are a relevant solution: they allow fast deployment, targeted instrumentation, and fast iteration in the lab. The internship presented in this report follows this approach at LABUST (University of Zagreb), with a clear operational goal: design and validate a compact teleoperated mini-ROV with a clear path toward autonomous functions.

The project is not limited to assembling a prototype. It addresses a full system problem under strong constraints: mass and size compatible with aerial deployment, watertight integration in a reduced volume, reliable communication between surface and robot, stable sensor-actuator chain behavior, and a usable video stream for perception. The internship work therefore combines mechatronic design, distributed software architecture, and experimental validation in a pool.

The development process is iterative. A first version (V1) was used to validate core building blocks and identify dominant limitations in real tests. A second version (V2) then integrated targeted improvements in communication, compactness, software observability, and piloting quality. This V1→V2 cycle is the main thread of the analysis presented here.

The report is organized as follows. Chapter 0 presents the internship context and system requirements. Chapter 1 describes hardware design and mechanical/electrical integration. Chapter 2 details the software architecture, communication protocols, and operational safety mechanisms. Chapter 3 compares these choices with experimental results, contrasts the two prototypes, and proposes an evolution path. The appendices gather the cost data, assumptions, and traceability elements used for validation.

Chapter 0: Project Context and Specifications

0.1 Host Laboratory Overview

The internship took place within the *Faculty of Electrical Engineering and Computing* at the University of Zagreb, known as FER. This faculty is one of Croatia's main academic centers in electronics, control, industrial computing, and robotics. It combines teaching, research, and technology transfer, making it a strong environment for technical projects tightly linked to real-system integration. FER is recognized for sustained scientific activity in these fields, with regular publications in international journals and conferences, and with academic and industrial collaborations that reinforce its influence. In this context, working at FER means operating in an environment where design rigor, experimental validation, and technical documentation are central.

LABUST, the host laboratory, is a FER research unit specialized in underwater systems, marine robotics, and technologies for exploration and intervention in aquatic environments. Its work covers underwater perception, navigation, control, communication, and sensor integration for teleoperated or autonomous robots. The laboratory is part of an active scientific environment, with visible presence in technical literature and sustained publication activity. Focused on applied research, this internship provided a concrete setting where each technical decision required a system-level approach, including power, communication, mechanical integration, pool behavior, and sensor data processing. In this context, the mini-ROV was treated not as an isolated prototype, but as a platform meant to fit into a broader development roadmap aligned with the laboratory's research themes.

0.2 Requirements and Dimensional Constraints for the Mini-ROV

The technical starting point of the project was inspired by an open-source ESP32 mini-submarine platform (*ESP-DIVE on Instructables*) [2]. This reference was useful as a baseline, but the internship objective went beyond replication: the goal was to define a more robust and scalable architecture compatible with the ambitions of the MIMISK project.
The mini-ROV is part of a broader project targeting coordinated deployment of two robots: an aerial drone and an underwater robot. The aerial drone is intended to carry the underwater robot to hazardous or hard-to-access areas where standard marine access is difficult, so that measurements can be performed there. In this context, the internship mission was to design, with support from two team members, an operational ROV including the hardware and software building blocks required for a future transition toward AUV behavior.
The functional requirements selected for this prototype are summarized as follows:

- execute the full onboard processing chain: control, sensor acquisition, video capture, and integration of an AI application for obstacle avoidance;
- ensure reliable communication between the aerial drone and the ROV through a wired link;
- remain compatible with the aerial drone payload capacity (2 kg). Considering cable and fastening system mass, the underwater robot had to stay below 1 kg;
- guarantee system watertightness, front camera integration, and observation capability in aquatic environments;
- provide three-dimensional control (horizontal translation and depth variation) to validate realistic navigation scenarios;
- integrate the minimum sensor set needed for future evolution toward AUV-like behavior.

Beyond performance targets, implementation constraints strongly shaped the design. Budget was not a strict blocker, but a hardware sobriety approach was imposed: avoid over-equipment, prioritize relevant components, and select more expensive options only when technically justified. Finally, the robot geometry had to remain compact and easy to handle to simplify gripping and transport by the aerial drone.
This requirement framework served as the reference for all integration choices presented in the rest of the report, including hardware architecture, communication protocols, and test validation.

Chapter 1

Hardware Design and Mechatronic Integration

Following the preliminary chapter, this part explains how the requirements were turned into a concrete hardware solution. The approach follows three levels: component selection, mechatronic integration, and estimation of expected performance before testing. The final prototype is a 30 cm cylindrical mini-ROV, built around an acrylic tube and ASA-printed parts, with two-thruster horizontal propulsion, front vision, and split control between a Raspberry Pi Zero 2 W and an ESP32-S3.
Depth variation relies on two independently driven 5 mL syringe ballasts. This choice provides controlled buoyancy variation, sufficient for the submersion and resurfacing phases targeted during pool testing.

1.1 Sourcing, Comparative Analysis, and Component Selection

Component selection was not handled as a simple shopping list: it was driven by four constant criteria, namely compactness, mass, reliability, and fast availability in Croatia. In practice, this sourcing constraint pushed purchases toward suppliers with European stock (notably Mouser) to limit integration delays.
For perception, the minimum baseline required for a future transition toward AUV behavior was the estimation of robot orientation, depth, and ballast position. An Adafruit ICM-20948 IMU was initially considered; the version finally deployed and used in testing is the MinIMU-9 v6 (LSM6DSO + LIS3MDL), available in the laboratory. For pressure sensing, packaging constraints led to the standalone MS5837-30BA sensing element with custom watertight integration [4]. The BAR30 campaign described later used the same sensor family, but on a dedicated test firmware and setup rather than on the production electronics configuration.
Onboard vision relies on a Raspberry Pi Camera Module 3 NoIR with wide-angle optics. This choice addresses internal volume constraints while maintaining a useful field of view

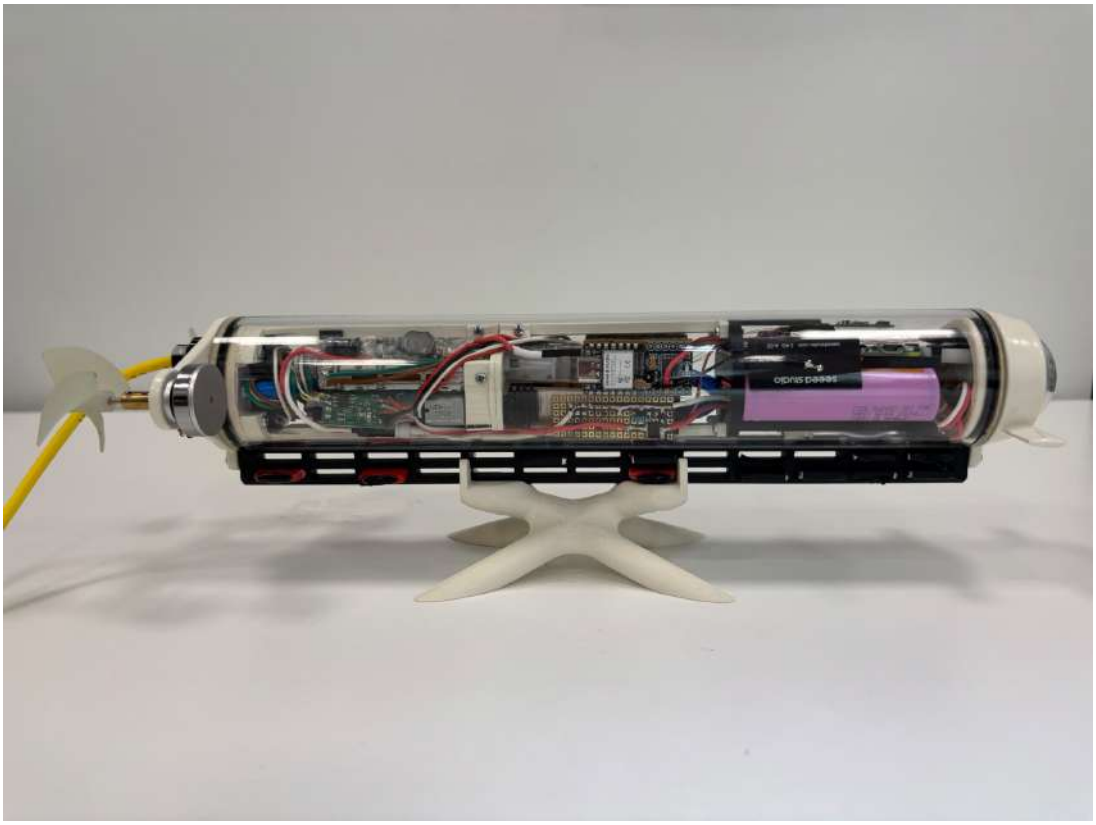

Figure 1.1: Side view of the final prototype, with acrylic tube, internal integration, and external support frame.

in low-light underwater conditions. In the same supervision logic, both syringe ballasts are instrumented with linear potentiometers.

The compute architecture is split across two complementary levels: the Raspberry Pi Zero 2 W handles high-level functions (vision and supervision), while the ESP32-S3 XIAO executes low-level real-time tasks (sensors and actuators). Power is provided by two 18650 Li-ion cells in 2S configuration, protected by a 2S/20A BMS, then distributed through a 5 V/3 A DC-DC converter for sensitive electronics.

For actuation, propulsion is provided by two APISQUEEN UG500 thrusters driven by BlueRobotics Basic ESCs [1]. The two N20 DC motors driving the ballasts are controlled through a DRV8833 H-bridge. Surface-to-robot communication uses RS485 over a 50 m two-wire Blueye tether, via USB-RS485 converters. Finally, a magnetic switch (Hall sensor + MOSFET + transistor) enables power on/off without opening the tube.

Total component cost is about 387 EUR excluding tether, which remains consistent with the objective of a compact and cost-efficient prototype. Full details (unit prices, quantities, and suppliers) are provided in the appendix (Section A.1).

Table 1.1: Main cost items for the final version (excluding tether).

Item	Subtotal (EUR)	Share of total
Propulsion (UG500 + ESC)	110.96	28.7%
Sensors + camera	93.94	24.3%
Control and communication (RPi, ESP32, RS485)	69.18	17.9%
Power chain (cells, BMS, DC-DC)	29.16	7.5%
Structure and mechanics (N20 motors, tube, dome, ASA, syringes, inserts)	84.07	21.7%
Estimated total	**387.31**	**100%**

1.2 Mechanical Integration and Space Optimization

This section presents the mechanical integration logic used to make all subsystems coexist in a tightly constrained volume. The hardware architecture is organized around four blocks: watertight structure, power chain, onboard control, and communication link. Figure 1.2 summarizes this organization and the power flows between blocks.

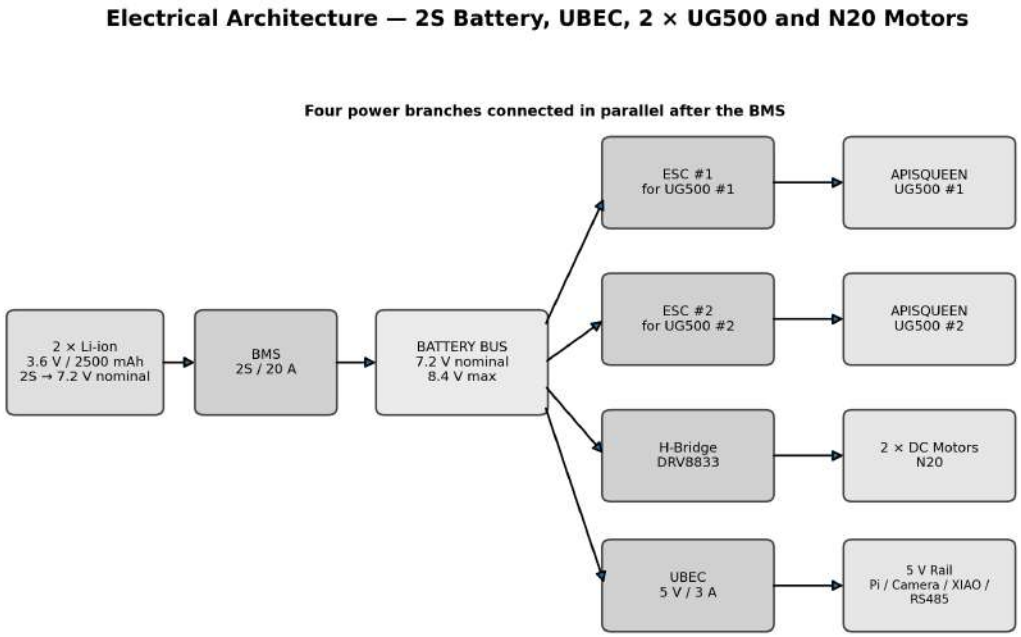

Figure 1.2: Global hardware architecture of the final version: 2S battery pack, power distribution, control, and actuation.

The mechanical structure is based on an acrylic tube closed by a *front cap* with camera dome and an *end cap* gathering cable pass-throughs and ballast hydraulic interfaces. Watertightness is ensured with O-rings. ASA-printed parts were acetone-treated to improve behavior in humid environments.

The ballast system uses two 5 mL syringes, each driven by an N20 motor. Motor-to-syringe transmission uses a non-backdrivable screw-nut mechanism. In practice, this allows

holding intermediate positions without continuous motor current, which reduces power consumption and simplifies the control law.

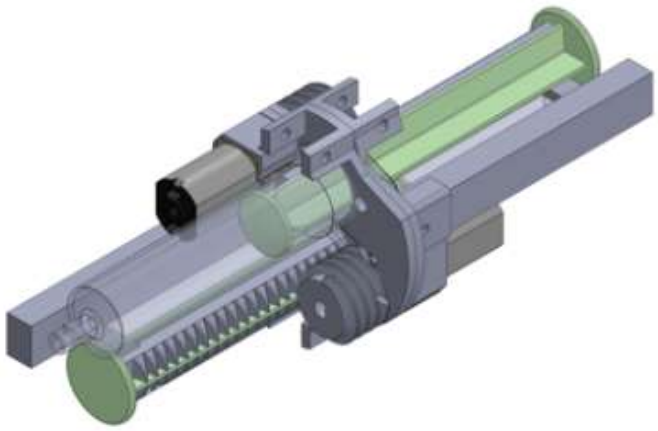

Figure 1.3: CAD model of one syringe ballast module with motorized actuation and mechanical transmission.

Internal integration was the strongest design constraint, with a usable diameter limited to 44 mm. Boards, harnesses, and connectors were therefore organized around the ballast using custom printed supports, to use volume efficiently while keeping maintenance access. Critical subassemblies are interconnected rather than directly soldered, which eases replacement and diagnostics. Additional integration views are provided in the appendix (Section A.3).
The MS5837-30BA pressure sensor was micro-soldered and then encapsulated in sealing resin. The sensing face remains exposed to external water pressure, while communication wires are routed inward to protect the connections. Finally, mass distribution was adjusted to place the center of mass slightly below the geometric center, improving orientation stability. Final measured mass is about 500 g excluding tether.

1.3 Theoretical Performance (Endurance, Depth, and Speed)

The following estimates do not replace Chapter 3 testing; they provide a reference framework. The objective is to define coherent orders of magnitude with the selected architecture, to better interpret campaign results.

1.3.1 Energy Endurance

Energy estimates provide two useful bounds: about 1 h 18 in continuous propulsion regime, and about 4 h 50 for electronics-only load. In real trials, with intermittent propulsion, observed endurance is close to 2 h 30. These values should be read as sizing orders of magnitude, which vary with mission profile. Detailed assumptions are reported in the appendix (Section A.2).

Table 1.2: Summary of endurance values obtained for the final version.

Indicator	**Value**	**Note**
Nominal battery voltage (2S)	7.2 V	2×3.6 V
Full-charge battery voltage	8.4 V	2×4.2 V
Nominal onboard energy	18 Wh	7.2×2.5 Ah
Total reference power	13.79 W	Electronics + propulsion
Theoretical endurance (continuous propulsion)	78.3 min	About 1 h 18
Theoretical endurance (electronics only)	290.9 min	About 4 h 50
Observed endurance in tests	About 2 h 30	Intermittent propulsion

1.3.2 Ballast-Driven Depth Capability

The robot is tuned close to neutral buoyancy in nominal condition. In fresh water ($\rho = 1000\,\mathrm{kg\,m^{-3}}$), with measured mass $m \approx 0.50\,\mathrm{kg}$, displaced volume is approximately:

$$V_0 \approx \frac{m}{\rho} = \frac{0.50}{1000} = 5.0 \times 10^{-4}\,\mathrm{m^3} = 500\,\mathrm{mL}$$

Total ballast capacity is $2 \times 5\,\mathrm{mL}$, i.e., $\Delta V_b = 10\,\mathrm{mL} = 1.0 \times 10^{-5}\,\mathrm{m^3}$. The corresponding water intake adds mass:

$$\Delta m = \rho\,\Delta V_b = 1000 \times 1.0 \times 10^{-5} = 0.01\,\mathrm{kg}$$

Resulting net downward force is:

$$F_z = \Delta m\, g \approx 0.01 \times 9.81 = 0.098\,\mathrm{N}$$

As a first approximation, this negative buoyancy margin is sufficient to initiate and sustain descent. However, ballast capacity alone does not set maximum depth; the practical limit depends on the complete system. The value of 50 m is only the approximate length of the Blueye tether used on this prototype, not a validated operating depth of the robot. The actual depth capability remains subject to structural limits and pressure qualification of the complete robot.

In addition, a separate pressure-chamber test was performed on the acrylic enclosure and indicated resistance to a pressure equivalent to about 60 m of water depth. This result concerns the tested enclosure configuration; it must not be interpreted as a qualification of the fully instrumented robot, its penetrators, or its ballast interfaces.

1.3.3 Theoretical Horizontal Speed

The manufacturer specifies a maximum thrust of 400 g at 24 V. With 2S supply (about 7.4 V), available thrust is therefore much lower. Based on datasheet values (0.7 A at 7 V, 36 mm propeller, 520 KV) and first-order rescaling, static thrust is estimated at about 120 g per thruster, i.e.:

$$F_{\text{prop}} \approx 2 \times 0.120 \times 9.81 = 2.35\,\text{N}$$

Limiting speed is then estimated from thrust/drag equilibrium:

$$F_{\text{prop}} = \frac{1}{2}\rho C_d A v^2$$

In fresh water (matching pool tests, with $\rho = 1000\,\text{kg}\,\text{m}^{-3}$), with tube outer diameter close to 5 cm ($A = \pi(0.025)^2 = 1.96 \times 10^{-3}\,\text{m}^2$) and global coefficient $C_d \approx 1.0$ for a conservative estimate, we obtain:

$$v \approx \sqrt{\frac{2F_{\text{prop}}}{\rho C_d A}} \approx 1.55\,\text{m}\,\text{s}^{-1}$$

This value is an idealized upper bound (without full conversion losses, without dynamic tether penalty, and without control margins). It should therefore be treated as a first-order sizing estimate, not as a measured vehicle speed or a validated operating envelope. The available information is insufficient to establish a defensible experimental speed range without a calibrated velocity measurement and a thruster performance curve at the actual supply voltage.
For reference, propulsion electrical power in 2S regime remains consistent with the previous endurance estimate (two thrusters around 0.7 A each). In fresh water we retain:

- reference propulsion current: $I \approx 2 \times 0.7 = 1.4\,\text{A}$,
- propulsion electrical power: $P_e \approx 7.2 \times 1.4 = 10.08\,\text{W}$,
- frontal area used for calculation: 5 cm tube cross-section.

Finer analyses related to losses, tether dynamics, and control choices are discussed in Chapter 3 to connect these estimates with experimental observations.

Chapter 2

Software Architecture and Embedded System

2.1 Communication Protocols and Network Infrastructure

This part describes the software architecture actually used during testing, based on real implementations on the PC, Raspberry Pi, and ESP32-S3 sides. The design was not driven by a single criterion: it had to support robust control over a long tether, near-real-time video feedback useful for piloting, and safe behavior in case of link failure.
The architecture therefore relies on an intentional transport asymmetry. The long PC-Raspberry link uses RS485 for robustness over long cable runs and in noisy environments. The short Raspberry-ESP link, local to the robot, remains UDP over a dedicated Wi-Fi subnet: this keeps frames lightweight on the microcontroller side while preserving OTA capability for remote ESP firmware upload without wired intervention.

2.1.1 System View and Flow Organization

To remain readable and maintainable, application exchanges are organized into three planes with different timing constraints. The command plane follows PC → Raspberry → ESP and is the most safety-critical one. The telemetry plane follows ESP → Raspberry → PC for supervision. The perception plane carries JPEG video and AI detections from Raspberry to PC, with priority on frame freshness rather than exhaustive frame retention. This organization appears directly in launch scripts. On the PC side, `launch_stack.sh` starts the RS485 mux first, then the video dashboard, then the gamepad sender; in test mode, the ROS2 bridge and rosbag recording are added. On the Raspberry side, the script starts the RS485 mux, then the video sender, then the C++ control application with ncurses supervision. This order is not incidental: it determines local socket availability and therefore the command chain *ready* state.

2.1.2 RS485 PC-Raspberry: Protocol, Arbitration, and Readiness

Both RS485 multiplexers share a deliberately simple binary frame format. Each frame starts with a fixed sequence `"R485"`, followed by a channel identifier and a two-byte little-endian size. Payload is capped at 60000 bytes to avoid processing abnormal packets in case of corruption or software malfunction.

$$\texttt{HEADER} = \texttt{MAGIC(4 bytes)} + \texttt{CHANNEL(1 byte)} + \texttt{SIZE(2 bytes LE)}$$

Channel assignment is fixed in software: one channel for commands, one for video, one for detections, one for status, and one for link control. The latter carries `HELLO`, `HELLO_ACK`, and `CMD_GRANT` messages used to establish the link, validate availability, and arbitrate access to the command path.

Table 2.1: RS485 channels and active transport policy.

Channel	Content	Active policy
1	Operator JSON commands	Schema validation, Raspberry grant-driven sending, latest-only semantics
2	JPEG video	Raspberry-side transmission priority, stale-frame drop
3	AI detections JSON	Output rate capped at 2 Hz toward RS485
4	Robot status JSON	Output rate capped at 2 Hz toward RS485
5	Link control	`HELLO`, `HELLO_ACK`, `CMD_GRANT`, bidirectional supervision

The central choice is grant-based arbitration. Raspberry emits grants at about 24 Hz, and the PC sends only the most recent command at grant time. Command bandwidth is therefore orchestrated, not driven by raw joystick cadence on the PC side. In operation, this avoids command burst accumulation on the uplink and preserves bandwidth for video return.

Timing values are harmonized across modules: heartbeat at 2 Hz, peer timeout at 5 s, an 8 ms window after grant to transmit a new command, and `pc_sender` activity monitoring at 0.7 s. This hierarchy lets the command loop react quickly to anomalies while keeping operator indicators stable enough.

A key field-testing point is the distinction between *rx_alive* and *ready*. *rx_alive* only means frames are still being received from Raspberry; *ready* confirms a truly bidirectional link. This distinction avoids false positives in asymmetric failures (return alive, uplink broken) and prevents displaying "healthy link" when commands no longer reach the robot.

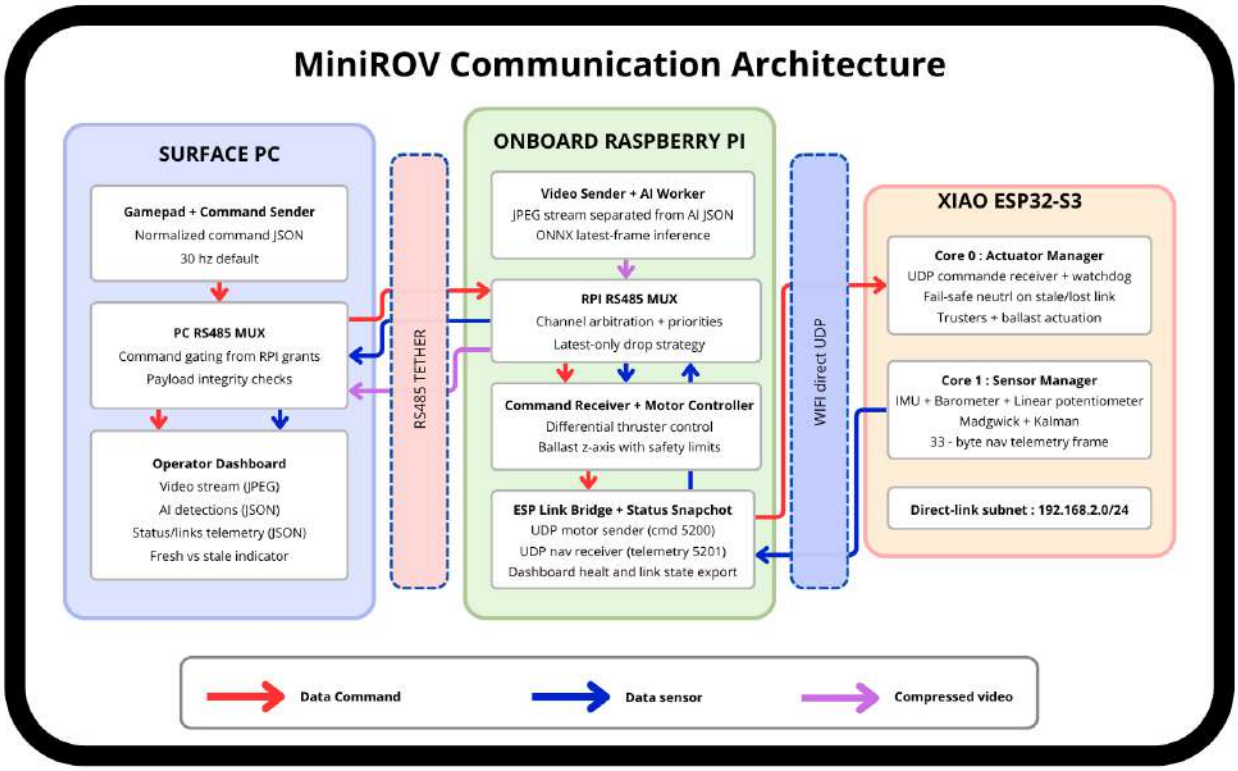

Figure 2.1: Communication software architecture used on the operational branch.

2.1.3 Raspberry-ESP Link: Dedicated Subnet and Fixed Binary Frames

On the short Raspberry-ESP link, the project uses a dedicated static subnet (Raspberry at `192.168.2.150`, ESP at `192.168.2.100`) and two UDP ports: 5200 for actuator commands and 5201 for navigation data. This fixed setup avoids routing ambiguity and greatly simplifies field diagnostics.

On the ESP side, binary format was chosen to keep real-time loops simple and stable. The actuator frame (9 bytes, sync `0xAA`) carries four signed 16-bit commands: left/right thrusters and front/rear ballasts. The navigation frame (33 bytes, sync `0xBB`) gathers quaternion, angular rates, accelerations, magnetic field, depth, and ballast positions.

Both flows target 100 Hz and are measured on Raspberry. The navigation receiver includes a useful operational mechanism: if no frame is received for 2 s while bound to a specific IP, the socket is automatically reopened on `0.0.0.0` to recover from network context changes without manual action. This logic is coupled with command watchdogs: in case of connection loss, commands are neutralized to stop the robot rather than keep an old order active.

2.1.4 Local Inter-Process Communication and Runtime Observability

A significant part of robustness comes from local process organization, not only from inter-board protocols. PC and Raspberry modules mainly exchange through UNIX datagram sockets with explicit paths (incoming/outgoing command, video, detections, status). This decision enables isolated interface testing without adding an external broker.
Raspberry also publishes an RS485 mux JSON state file read by the C++ dashboard. This snapshot exposes key indicators (`serial_connected`, `link_ready`, `command_path_ready`, `pc_sender_active`) that quickly differentiate a serial-port failure, local process issue, or remote-peer failure.
Unified status sent back to PC periodically aggregates operator commands, computed motor outputs, link state, navigation, and AI state. This continuous instrumentation is essential for test reproducibility: it makes it possible to relate command, perception, and robot response afterward, even when some raw logs are incomplete.

2.1.5 Tested Communication Choices and Final Rationale

Choices retained in the operational branch result from progressive filtering between possible solutions and solutions that were truly robust in campaigns. The RS485 path on PC-Raspberry was kept because it proved more stable on long tether during physical validation. Conversely, the I2C option on the Raspberry-ESP segment remained exploratory due to incomplete validation on the final prototype.
RS485 campaign scripts were then used to sweep baud rates, measure losses and RTT, and lock a reproducible configuration. The 3 Mbaud value used in operation corresponds to the chosen compromise between throughput margin, link stability, and reasonable CPU load on Linux hosts.
Usable throughput can be approximated by:

$$B_{\text{useful}} \approx \frac{\text{baud}}{10}\,\eta$$

with η the global transport efficiency (framing overhead and arbitration phases). For 3 Mbaud and $\eta \approx 0.8$, the resulting order of magnitude is about 240 kB/s. This budget is consistent with the retained video configuration (compressed JPEG, bounded size, frame-priority strategy).
Finally, the *latest-only* strategy applied from camera sender to dashboard is as much a piloting choice as a transport choice. In teleoperation, a very recent image, even partially decimated, is more useful than a complete sequence that arrives late. Losing intermediate frames is therefore accepted to keep visual latency bounded.

2.2 Sensor Data Acquisition and Control Chain

2.2.1 Compute Loop Distribution and Real-Time Constraints

The sensor-control chain was built as a distributed system: each computation runs as close as possible to the resource it uses. On Raspberry, the C++ application gathers piloting logic, operator command reception, and supervision status composition. On ESP32, two FreeRTOS tasks run on distinct cores: an actuator task (core 0) and a sensors/filtering/publication task (core 1).
This decoupling is meant to keep operation stable: a transient overload on Linux perception (for example a video processing spike) must not slow motor command application. Conversely, an I2C sensor fault on ESP must not block actuator neutralization. Both loops therefore keep their own cadences and recovery mechanisms.

2.2.2 Operator Input and Command Normalization

On the PC side, `pc_sender.py` does not forward gamepad axes as-is. It first reconstructs each axis profile via `ioctl` (min/max/center/flat zone), then applies normalization, saturation, and deadzone. This step avoids centering bias that varies from one gamepad to another. The JSON command also includes a sequence number and a `sender_ts_ms` timestamp, useful for analyzing losses and timing consistency in campaigns.
On the Raspberry side, `CommandReceiver` adds a second robustness layer: JSON shape validation, value clamp to expected ranges, explicit inversion of some axes (joystick convention to robot convention), and a 250 ms freshness watchdog. Without fresh packets, the command returned to the controller automatically goes back to neutral. This guarantees that a gamepad loss does not leave random persistent values: the robot returns to stop deterministically.

2.2.3 ESP Sensor Acquisition and Fault Recovery

The `SensorManager` module aggregates IMU, barometer, and ballast potentiometers. Initialization is deliberately non-blocking: a missing sensor at startup does not block the whole system. In return, recovery retries are scheduled with long backoff (30 s) to avoid overloading the real-time loop in case of persistent failure.

IMU. The IMU reader explicitly configures registers of all three sub-sensors, then applies physical conversions (rad/s, m/s^2, microtesla). Startup gyroscope calibration is performed on windows consistent with ODR and canceled if platform motion is detected during bias capture. This precaution has direct impact on heading quality, especially in AUTO mode.

Pressure and depth. The production pressure reader uses a non-blocking D1/D2 conversion state machine, with startup surface-pressure calibration. Its production configuration uses the MS5837-30BA at 100 kHz I^2C, OSR2048, and a 100 Hz target rate; depth

is then computed through the hydrostatic relation in fresh water with standard gravity. The dedicated BAR30 characterization campaign used a separate test configuration at 400 kHz I^2C to compare OSR settings. This distinction is important: campaign settings quantify sensor behavior, whereas the values above describe the firmware integrated in the final robot.

Ballast. Both linear ballast sensors are read at 12-bit resolution and normalized. Telemetry also keeps a quantized raw value in the 0..500 range, easier to use for mechanical safety thresholds on the controller side.

2.2.4 Navigation Filtering: From Raw Sensors to Control-Ready State

The onboard navigation filter is designed to produce, at 100 Hz, a compact and stable state directly usable by the Raspberry controller and by analysis tools. The chain is built step by step to stay readable and robust in campaign conditions.
Gyroscope, accelerometer, and magnetometer first pass through lightweight low-pass filters. Gyroscope bias is then adapted slowly, but only in near-static windows (low gyro norm and acceleration close to g). This rule avoids learning false bias while the platform is in dynamic motion.
Attitude is estimated with a Madgwick-type scheme [3]. Magnetometer is not injected blindly: correction is suspended if magnetic norm becomes implausible or exhibits abrupt jump under low dynamics. Progressive re-entry then reactivates fusion when field consistency returns. This mechanism limits yaw jumps caused by local disturbances.
Vertical dynamics is handled with a two-state model,

$$\mathbf{x}_k = \begin{bmatrix} z_k \\ w_k \end{bmatrix},$$

where prediction is fed by gravity-compensated vertical acceleration and correction by filtered barometric measurement. This form remains lightweight for ESP32 and is sufficient for the target pool piloting resolution.
Intermediate iterations tested richer fusion variants, then progressively simplified them to converge toward a more robust campaign configuration. Explored blocks notably included dedicated yaw correction (bounded magnetic innovation with distinct gains in static and dynamic regimes), stronger gyro/magnetometer weighting, and a more explicit vertical coupling with ballast state. These extensions brought occasional gains, but with higher parameter sensitivity and larger performance dispersion across campaigns.
The operational branch therefore retains a deliberately lean chain: lightweight LPFs on sensors, slow gyroscope bias adaptation in calm windows, magnetometer *gating* based on norm plausibility and relative jump, Madgwick for attitude, then a 2-state Kalman filter (depth/vertical speed). For this internship, this compromise provides the best combination of reproducibility, behavior readability, and analysis ease.

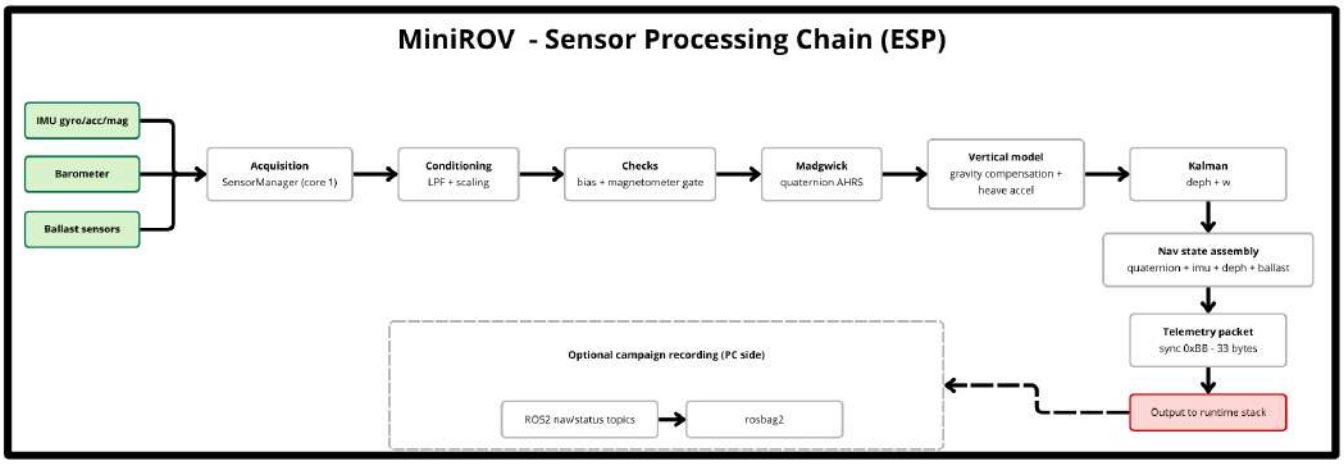

Figure 2.2: Onboard sensing-estimation-telemetry chain.

2.2.5 Raspberry High-Level Control: Mode Logic and Propulsion Mixing

The `MotorController` module structures commands around explicit state logic. Arming gates any non-zero output, MANUAL/AUTO switching is edge-triggered, and AI avoidance is only enabled in AUTO. The code also includes a useful protection during link restart: after fresh command flow returns, the controller requires button release before re-enabling edge detection.

In MANUAL mode, horizontal propulsion follows a differential *surface vessel*-type mixer. Yaw component is modulated with translation level and then bounded, to avoid abrupt propeller reversals in cruise regime. PWM commands then pass through first-order filtering and output deadband, reducing near-zero oscillations.

AUTO mode follows a classical heading-control logic adapted to differential propulsion. The controlled variable is orientation, not raw motor command: the operator adjusts heading reference, then the control layer converts it into left/right action compatible with the platform.

In practice, joystick lateral axis integrates a bounded-rate `target_yaw_deg` reference (90 deg/s), smoothing reference dynamics and limiting behavior discontinuities. This reference is compared to yaw extracted from navigation quaternion; the heading controller (PID structure, mainly used on proportional term here) then produces a rotation demand converted into left/right differential via a virtual wheelbase. Forward speed is held constant to decouple, as much as possible, horizontal trajectory control from attitude control.

AI obstacle avoidance is handled as a bounded perturbation of heading reference. When active, the primary detection applies a limited offset (up to $\pi/3$), then temporally filtered to limit reference oscillations. Stale detections are rejected, and missing valid navigation state cancels AUTO propulsion. This hierarchy preserves coherence between perception, estimation, and action.
Vertical command is then subject to ballast protection: if a potentiometer indicates mechanical end-stop, the command that would push farther is locally neutralized, even if the pilot keeps requesting it.

2.2.6 Raspberry-ESP UDP Chain: Operational Robustness Points

Raspberry UDP sender first applies strong output saturation in $[-255, 255]$, encodes the binary frame, then measures effective cadence. When the process stops, a zero frame is explicitly sent before socket closure to avoid keeping a final non-zero command active on the actuator side.
Raspberry navigation receiver renormalizes received quaternions and reconstructs physical units (rad/s, m/s^2, microtesla, meters). This Linux-side normalization makes visualization and analysis more robust, including for boundary-case packets.

2.2.7 Sensor Validation and Final Configuration Choice

BAR30 campaign results directly oriented production configuration. Final choice does not target absolute minimum noise; it targets best dynamic response compatible with depth-loop stability.

Table 2.2: BAR30 campaign summary used to freeze firmware configuration.

Method	**Measured frequency**	**Mean static noise**	**Dynamic t63**
OSR4096_OPT	50.018 Hz	4.2 mm	63.7 ms
OSR2048_OPT_100HZ	100.023 Hz	6.0 mm	29.5 ms

Moving to 100 Hz on the pressure chain divides dynamic response time by about a factor of 2.2, while increasing static noise from approximately 4.2 to 6.0 mm in the reported campaign. This compromise is considered acceptable for the targeted pool experiments, but it does not establish a general depth-hold accuracy. The tuning is also consistent with the rest of the stack (100 Hz actuator command, 100 Hz navigation telemetry, and ROS2 post-processing). Associated campaign datasets and scripts are listed in the appendix (Section A.5).

2.3 Real-Time Video Flow Optimization and AI Integration

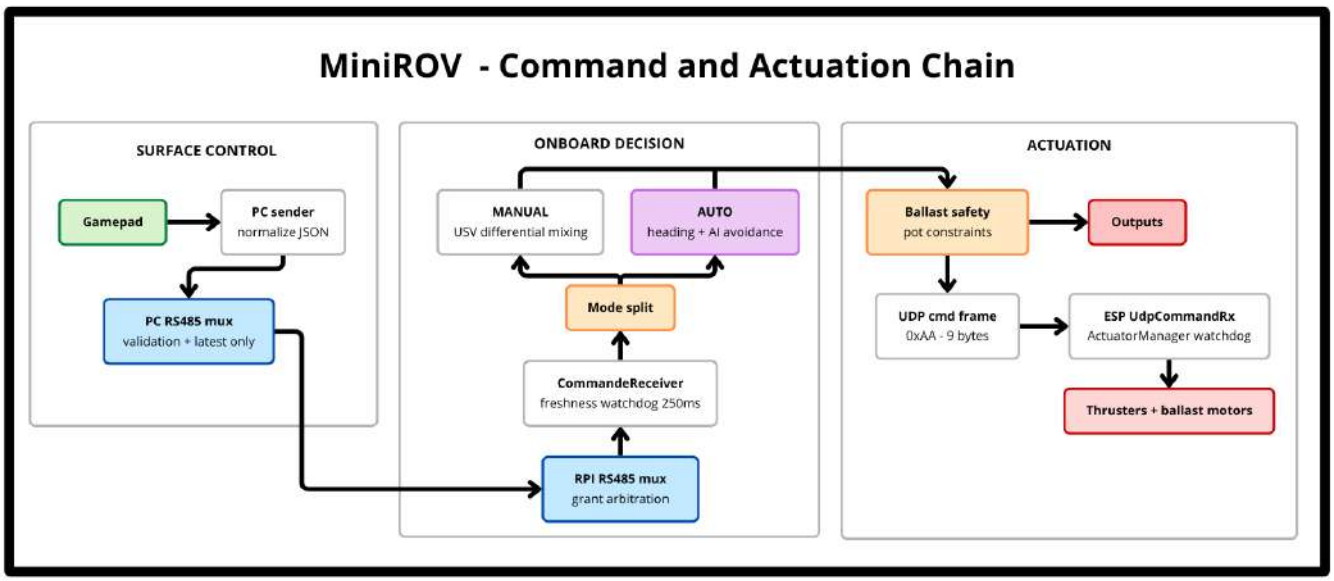

Figure 2.3: Command-to-actuator chain, from gamepad input to thruster and ballast outputs.

This section presents the video/AI chain as a full piloting function. In pool testing, video remains the main perception source for the operator, while AI adds obstacle cues useful for anticipation. The selected sizing therefore targets a clear compromise between real-time readability, transport robustness, and onboard compute load.

2.3.1 Initial Constraint: Keep the View Pilotable on a Shared Link

Video pipeline design started from a simple constraint: return path must remain readable for the pilot even when command, status, and perception share the same RS485 transport. The approach therefore did not maximize image quality, but temporal stability of useful information.

Concretely, this leads to a deliberately compact stream: 320×320 capture, 10 Hz target rate, moderate JPEG quality, and strict payload-size control. This sizing remains consistent with useful serial bus budget and pool operation needs.

2.3.2 Raspberry Camera Pipeline and Tuning Choices

The `video_sender.py` module initializes Picamera2 with minimal buffering (two buffers) and disables internal queueing to avoid accumulation of stale frames. Autofocus is also explicitly managed at startup, with continuous, one-shot auto, or manual modes depending on campaign. The objective is stable visual behavior from one session to another.
The encoding path includes an implementation fallback (OpenCV or `simplejpeg`) to improve deployment reliability on heterogeneous systems. Before mux injection, each frame is checked against RS485 payload limits. Oversized frames are dropped locally and counted. This choice is deliberate: occasional frame loss is preferable to persistent return-path congestion.

2.3.3 *Latest-Frame* Policy and Perceived Latency

The project explicitly applies a *latest-frame* policy at every level. At source, `FrameManager` keeps only the latest available image; the AI worker consumes that image without intermediate queue; Raspberry and PC muxes drain sockets in "most recent frame" mode. The dashboard also displays overwritten-frame counters to make this trade-off visible.
This policy can look counterintuitive if only frame loss is considered. In teleoperation, however, it is more useful: a very recent image, even decimated, is more exploitable than a complete sequence delayed by hundreds of milliseconds. The controlled variable is therefore perceived latency.

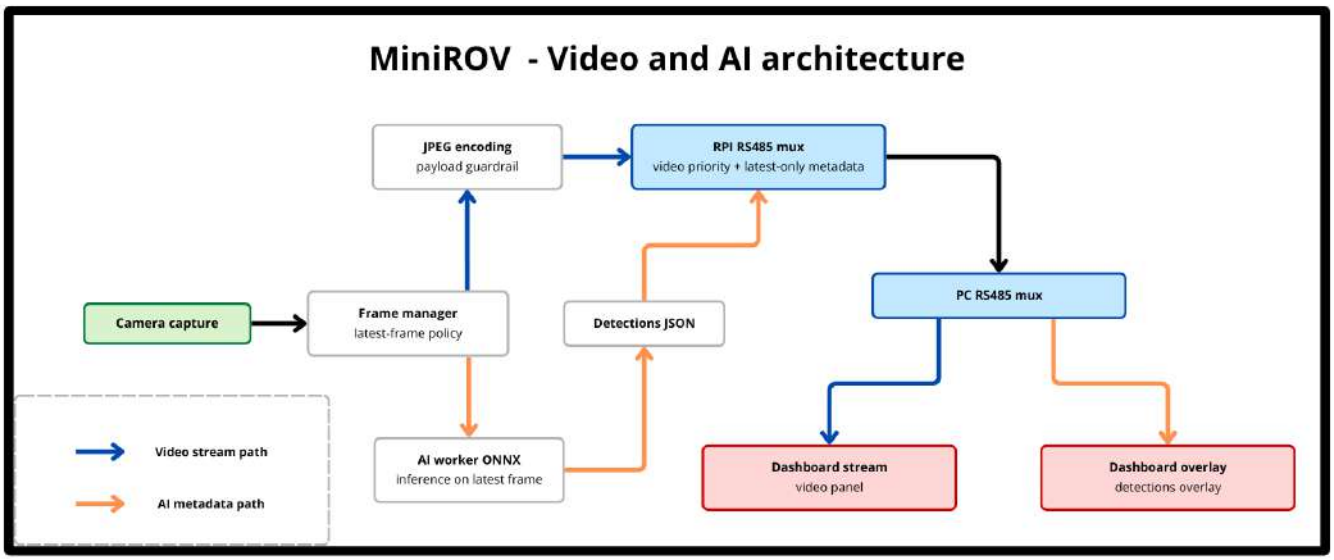

Figure 2.4: Video chain and AI metadata path up to the surface dashboard.

2.3.4 ONNX Inference: Deployment and Parsing Robustness

AI integration is not only a model choice; it includes deployment safeguards. The worker tests multiple model paths, automatically adapts preprocessing to input layout (NCHW/NHWC), and handles different tensor types (float, float16, int8, uint8). If ONNX runtime fails, the video pipeline keeps running, preventing complete supervision loss.
Output parsing is another sensitive point. The code accepts post-NMS and pre-NMS outputs, detects column conventions, applies *objectness* × class score when needed, converts boxes to normalized coordinates, removes invalid boxes, then applies class-wise NMS. This safeguard chain reduces missed detections caused by format differences across model versions.
AI result is then published as JSON on two channels: toward the return transport for operator overlay, and toward a local state file consumed by the Raspberry AUTO controller for avoidance. This duplication is intentional, to decouple visualization needs from local decision needs.

2.3.5 Multi-Flow Arbitration and Load Tracking

Throughput control relies on continuous useful-load tracking:

$$\text{load}_{RS485} = \frac{\dot{V}_{\text{video}}}{B_{\text{useful}}}.$$

Video sender measures effective bitrate and estimates load ratio against configured serial budget. This metric is useful in campaigns because it explains fluidity loss with measurable overload rather than subjective perception.
On Raspberry mux side, arbitration gives strong priority to video, while detections and status are capped at 2 Hz toward RS485. This hierarchy limits visual jitter, since textual metadata is less sensitive than image flow to throughput irregularities.

2.3.6 Surface Dashboard and Freshness Instrumentation

The PC dashboard does not only show image and detection boxes. It keeps freshness indicators for each flow (video, detections, status, command), with temporal smoothing and quantization for stable display. Decode-error, queue-overwrite, and invalid-JSON counters are also visible. This observability is central in testing: it turns subjective impression into objective symptoms.

2.3.7 ROS2 Bridge for Campaigns and Post-Processing

In test mode, `mux_ros_publishers.py` converts mux sockets into ROS2 topics, with a QoS profile adapted to compressed video. This layer publishes both raw JSON and typed messages (depth, quaternion, Euler, potentiometers), which then simplifies analysis scripts

and run-to-run comparisons. Vision topic activation remains optional to limit bag size when only the command-navigation loop is evaluated.

2.4 Safety Algorithms and Degraded Modes (Fail-Safe)

2.4.1 General Approach: Defense in Depth

Software safety is not concentrated in a single module. It is distributed across successive layers, from command generation to actuator application. The principle is simple: at the slightest doubt about command validity or link health, output must return to neutral.
This layered approach provides two operational benefits. First, an isolated failure does not automatically lead to loss of control, because a downstream layer can still block command output. Second, the fallback cause remains traceable, since each layer exposes its own health indicators.

2.4.2 Main Implemented Mechanisms

Beyond watchdogs and local guards, ESP firmware follows a cautious startup policy: safe motor levels before full initialization, Wi-Fi reconnection with bounded delays, and OTA opening only once connection is established. This behavior reduces motor transients at boot and during network recovery.

2.4.3 Analyzed Failure Scenarios

Asymmetric RS485 tether loss. If Raspberry→PC return remains active while PC→Raspberry uplink is broken, heartbeats detect asymmetry. *Ready* state drops on both sides, then command freshness expires at 250 ms, which drives outputs back to neutral. This case is critical in operation because it is hard to diagnose without explicit distinction between *rx_alive* and *ready*.

USB serial adapter loss or renumbering. PC and Raspberry muxes stay in detection loops and fall back to stable `/dev/serial/by-id` paths. The system can therefore recover without full restart when interface comes back, which is valuable in field handling.

ESP-Raspberry Wi-Fi link drop. On ESP side, link loss triggers actuator neutralization and UDP listener stop until healthy state returns. On Raspberry side, command sender and navigation receiver states are published separately; `esp_connected` is true only when both communication directions are operational.

Missing navigation in AUTO mode. The controller forbids autonomous propulsion without valid navigation state. This rule avoids extending a heading command based on potentially invalid orientation estimation.
The concrete mechanisms associated with these scenarios are summarized in the following table.

Table 2.3: Fail-safe mechanisms effectively present in runtime modules.

Module	**Monitored condition**	**Safety action**
CommandReceiver (RPi)	No fresh packet for > 250 ms	Command state reset to zero
MotorController (RPi)	Command link not fresh	Thrusters + ballasts forced to neutral
UdpMotorSender (RPi)	Application stop / socket error	Send one zero frame before close
ActuatorManager (ESP)	Wi-Fi loss or 250 ms command timeout	Actuator neutralization
RPi RS485 mux	Command consumer missing / link not ready	Block command path and force low ready indicator
PC RS485 mux	Invalid payload / link not confirmed	Drop packet, no opportunistic transmission

2.4.4 Role of Operator Interface in Safety

Safety is not only algorithmic; it is also linked to operator interface. Launch scripts enforce startup order (PC then Raspberry), link and frequency indicators remain visible continuously, and the dashboard gathers pilot inputs, motor outputs, and sensor state in one snapshot. This grouping reduces interpretation errors during tests, especially in armed/disarmed and MANUAL/AUTO transitions.

2.4.5 Current Limits and Improvement Path

The current version is robust for internship operation, but it is not yet at industrial hardening level. Raspberry-ESP protocol does not include cryptographic authentication, inter-board timestamps are not absolutely synchronized, and yaw quality remains sensitive to local magnetic disturbances. These limits do not invalidate test outcomes; they define the logical next step: transport hardening, shared timebase, and stronger orientation robustness.

2.4.6 Software Part Summary

The final software architecture can be read as a controlled compromise between three objectives: command reliably, keep useful observation without excessive latency, and return safely without ambiguity. The distributed stack (PC/Raspberry/ESP), explicit flow arbitration, and *latest-only* strategy all follow this common logic.

Methodologically, the most important point is continuous linkage between design, implementation, and behavioral evidence: executed code, runtime indicators, and associated campaigns. This continuity explains observed pool stability and serves as the base for the testing chapter.

Chapter 3

Iterative Development, Pool Testing, and Outlook

Chapter 2 established the software architecture and implementation choices used in operation. This chapter operates at a different level: it documents the experimental approach, compares the two prototypes built during the internship, and connects technical choices to observable test results.
The progression follows an evidence-based logic: V1 as a functional validation baseline, V2 as a robustness iteration, followed by a global assessment and an evolution path.

3.1 First Prototype: Assembly and Initial Functional Validation

3.1.1 Technical Objective and V1 Prototype Configuration

The first prototype was designed as a minimal integration platform to quickly validate core building blocks: hull watertightness, feasibility of two-thruster propulsion, ballast operation, sensor chain, and teleoperation loop from surface station.
The structure used a 50 mm outer-diameter acrylic tube, closed by ASA-printed *front cap* and *back cap*, then acetone-treated to improve surface finish. Horizontal propulsion was provided by two APISQUEEN UG500 thrusters. The robot was organized in three internal zones:

- a rear zone centered on the ballast system,
- a middle zone for battery cells, BMS, and Raspberry Pi,
- a front zone for control electronics (including ESP) and camera.

Final V1 prototype length was 37 cm. This geometry was chosen to keep internal architecture accessible, at the cost of volume and hydrodynamic inertia still needing optimization.

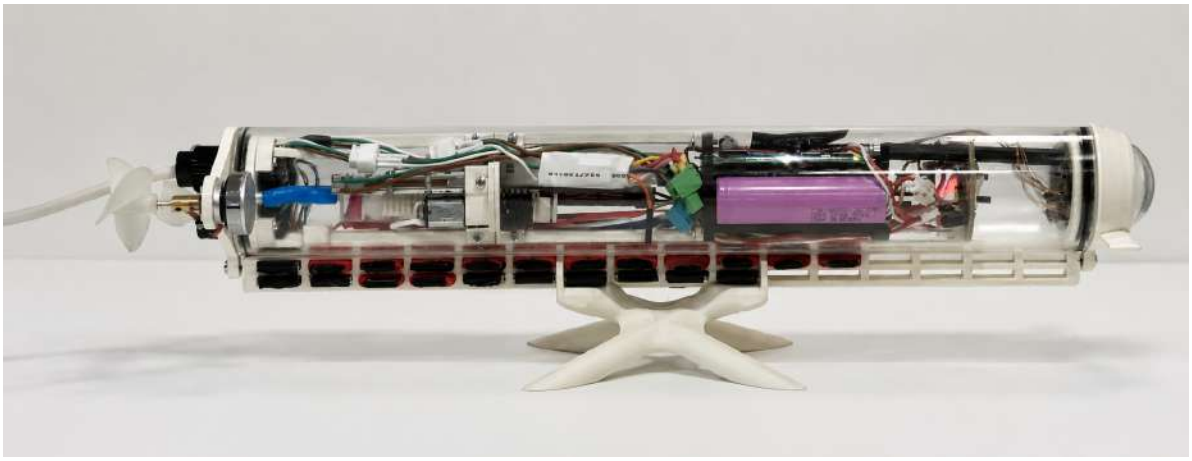

Figure 3.1: V1 prototype: first complete integration of the mechanical and electronic chain.

3.1.2 Communication Architecture and V1 Link Limitations

On V1, the surface-Raspberry link used an approximately 8 m Cat6 cable connected through an RJ45→USB 2.0 converter on the Raspberry side. ESP-Raspberry link used UDP on the local onboard network, in a deliberately basic version without advanced hardening layer.
This configuration met its rapid-start objective, but also fixed prototype limits: tether stiffness, parasitic drag, and behavior variability caused by mechanical cable-robot interaction during maneuvers.

3.1.3 V1 Validation Campaign

Pressure watertightness validation. Before final integration, a watertightness test was performed on the assembled hull (tube + caps + pass-throughs), then confirmed in immersion. The result was positive at this stage: no structural leakage was observed during V1 test sequences.

Qualitative pool testing. Pool tests were carried out in a functional-validation logic rather than strict metrology. Observations converge on the following points:

- the robot moves correctly in horizontal motion,
- control remains usable but still approximate,
- ballast operates and enables submersion/resurfacing transitions,
- perceived response time is consistent with teleoperation objective,
- communication with PC base station is operational.

The main dynamic limiting factor observed is the Cat6 tether: low flexibility and unfavorable buoyancy introduce significant parasitic force, which perturbs trajectory and degrades fine piloting quality.

First vision and AI tests. A first camera chain was deployed on V1 to validate acquisition and video return. Associated AI tests mainly secured integration (formats, cadence, visualization) rather than demonstrating robust use in dynamic mission conditions. This point is important for correct result interpretation: AI level becomes pre-operational on the next generation (V2), but still requires consolidation before mission-grade use.

3.1.4 Barometer Characterization: From Sensor Validation to Firmware Choice

In addition to pool tests, a dedicated BAR30 characterization campaign was conducted to characterize depth-reading behavior (accuracy, noise, effective frequency, dynamics). This campaign was instrumented outside the piloting loop to isolate the sensing chain. The need came from a datasheet remark: increasing frequency could reduce precision. Measurements therefore aimed to quantify this trade-off explicitly for the robot use case.

Figure 3.2: BAR30 measurement setup used for static and dynamic characterization campaign.

Results show a clear noise/reactivity trade-off depending on OSR. The 50 Hz mode (OSR4096) reduces static noise ($\sigma \approx 4.2$ mm), while the 100 Hz mode (OSR2048) divides dynamic response time t_{63} by about 2.2 (29.5 ms versus 63.7 ms), with noise still compatible with vertical control objective. The increase in noise at 100 Hz is measurable, but remains compatible with the resolution targeted for the present pool experiments. It is therefore an engineering compromise rather than evidence that the 100 Hz mode is intrinsically superior in every application. This result, together with the faster dynamic response, guided the production configuration later used on V2. Scripts and datasets associated with this campaign are referenced in the appendix (Section A.5).

3.1.5 V1 Summary

V1 fulfilled its role as an integration baseline: mechanical feasibility validated, sensor/actuator chain operational, first effective teleoperation loop established. However, limitations in fine control, trajectory quality, and surface-link robustness motivated a targeted redesign leading to V2 prototype.

3.2 Second Prototype: Electronics Redesign and System Hardening

3.2.1 V1 → V2 Evolution Logic

The second prototype keeps the functional architecture validated on V1 (dual thruster, ballast, Raspberry/ESP pair), but introduces targeted changes to address root causes observed in campaign conditions: tether constraints, communication robustness, internal compactness, and software-chain stability.

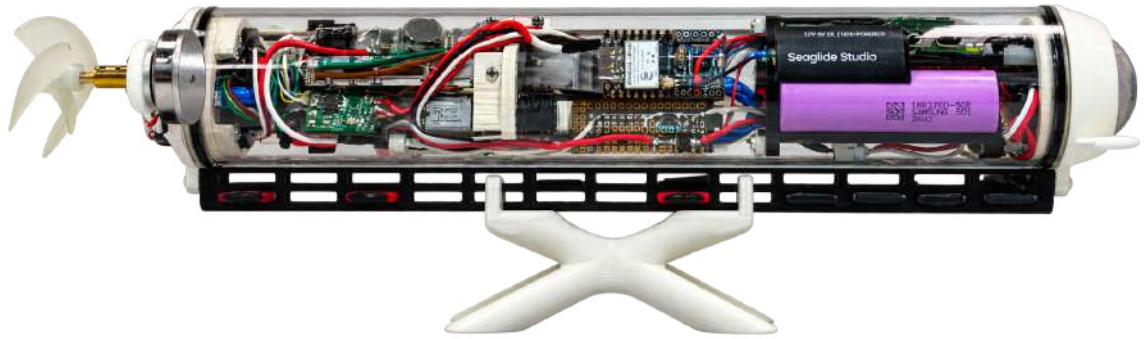

Figure 3.3: V2 prototype: more compact integration and reorganized electronics.

3.2.2 Hardware Evolution and Internal Integration

V1 → V2 transition was conducted through short iterations, each targeting one root cause observed in pool tests (internal volume, tether behavior, control robustness). V2 internal reorganization notably moved electronics around the ballast zone, reducing total length by about 7 cm (from 37 cm to 30 cm). This volume reduction decreases required trim mass to reach neutral buoyancy and lowers overall hydrodynamic footprint.
Sensor integration was also hardened: IMU properly fixed above Raspberry Pi, pressure sensor integration stabilized, and harness routing cleaned to limit mechanical and electromagnetic disturbances.

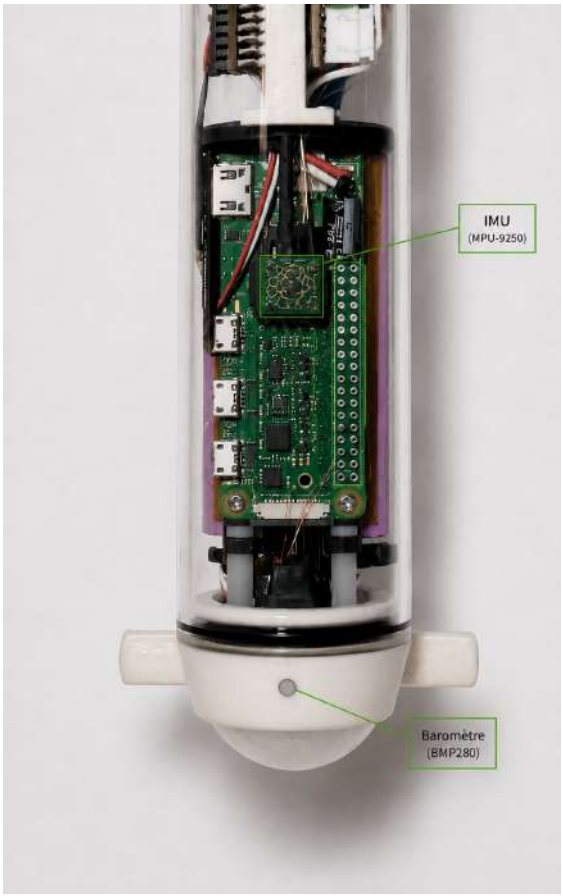

Figure 3.4: V2 integration detail: IMU and pressure sensor positioning in the front zone.

In the propulsion chain, ESCs were replaced between iterations to address limits observed in early tests (control stability and operational behavior). V2 retains a more robust and more reproducible solution for prolonged campaigns.

3.2.3 Surface-Link Change and Software Impact

The most significant V2 evolution is the move from Cat6/USB to RS485 over two-wire Blueye tether, consistent with existing aerial robot infrastructure in the project. This change is not limited to the physical layer: it requires explicit software architecture to handle constrained serial transport that is not natively multi-flow.
In practice, this introduced frame multiplexing, explicit bidirectional control (readiness, heartbeat, grants), command-freshness watchdogs, and strict degraded modes. Implementation details are given in Chapter 2; from the testing chapter viewpoint, the main effect is a clear improvement in operational stability and link-incident traceability.

3.2.4 Software Evolution Oriented to Operation

Without repeating Chapter 2 details, the essential difference between V1 and V2 is the transition from a validation stack to an operational stack. Gamepad input normalization

was reinforced, a full operator dashboard was added on PC side, and robot state reporting was structured for live diagnostics.
In parallel, ROS2 integration and rosbag recording enabled quantitative test exploitation. AI-enriched video reached a pre-operational level: object detection works, but the full function is not yet validated as robust autonomous behavior in dynamic mission conditions and still requires improvement. Finally, updates to sensor and control chains (MANUAL/AUTO modes, obstacle-assist activation) improved global consistency between perception, decision, and action.

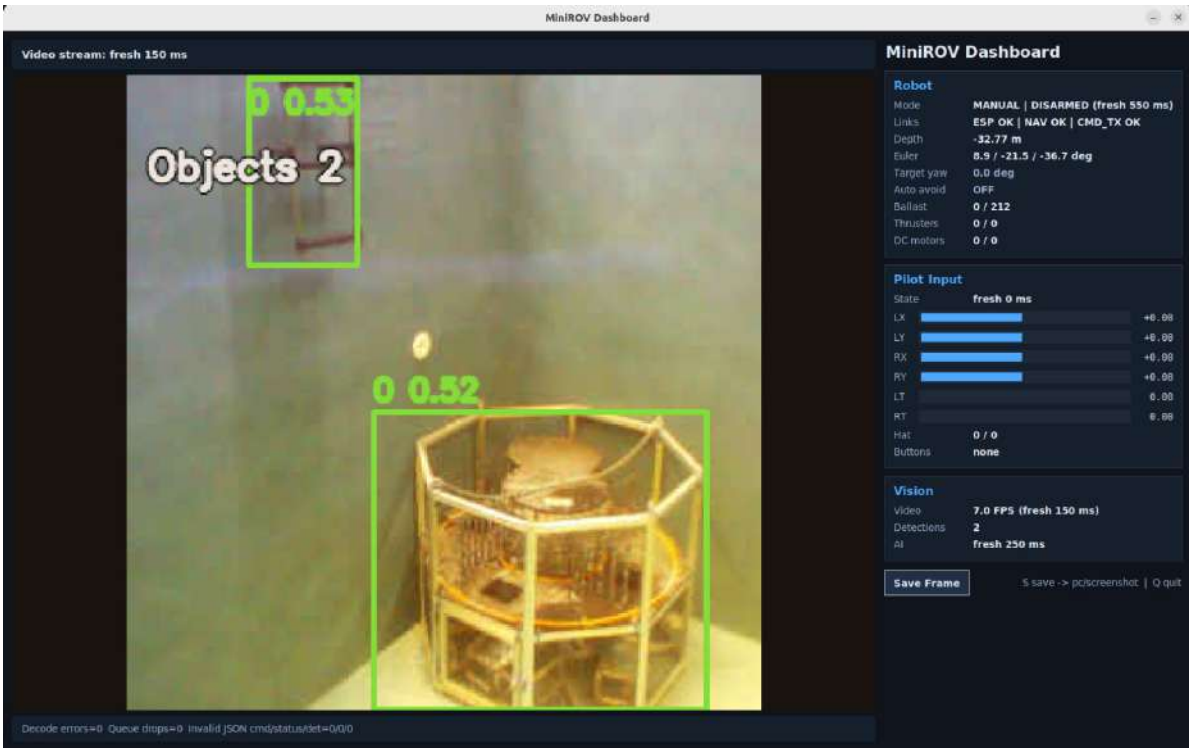

Figure 3.5: V2 dashboard in operation: video stream, AI boxes, robot status, and synchronized pilot inputs.

3.2.5 V2 Pool Tests: Qualitative Observations

V2 pool tests confirm clear progress relative to V1. Piloting is cleaner and more responsive, trajectory is less disturbed by the tether, and the video chain remains stable during motion. AI detection showed positive signs, with objects effectively detected in the operator stream. The main limit at this stage is schedule-related, not conceptual: obstacle avoidance was not pushed to full closed-loop validation in pool conditions within internship duration. AI tests were concise: they show a good start (objects detected), but errors remain and the function is not yet mission-operational.

3.2.6 Quantitative Results from Rosbags

The major V2 contribution for analysis is availability of exploitable rosbags, enabling transition from mainly qualitative validation to quantitative performance reading. Three

analyses were conducted. Associated data paths and scripts are centralized in the appendix (Section A.5).

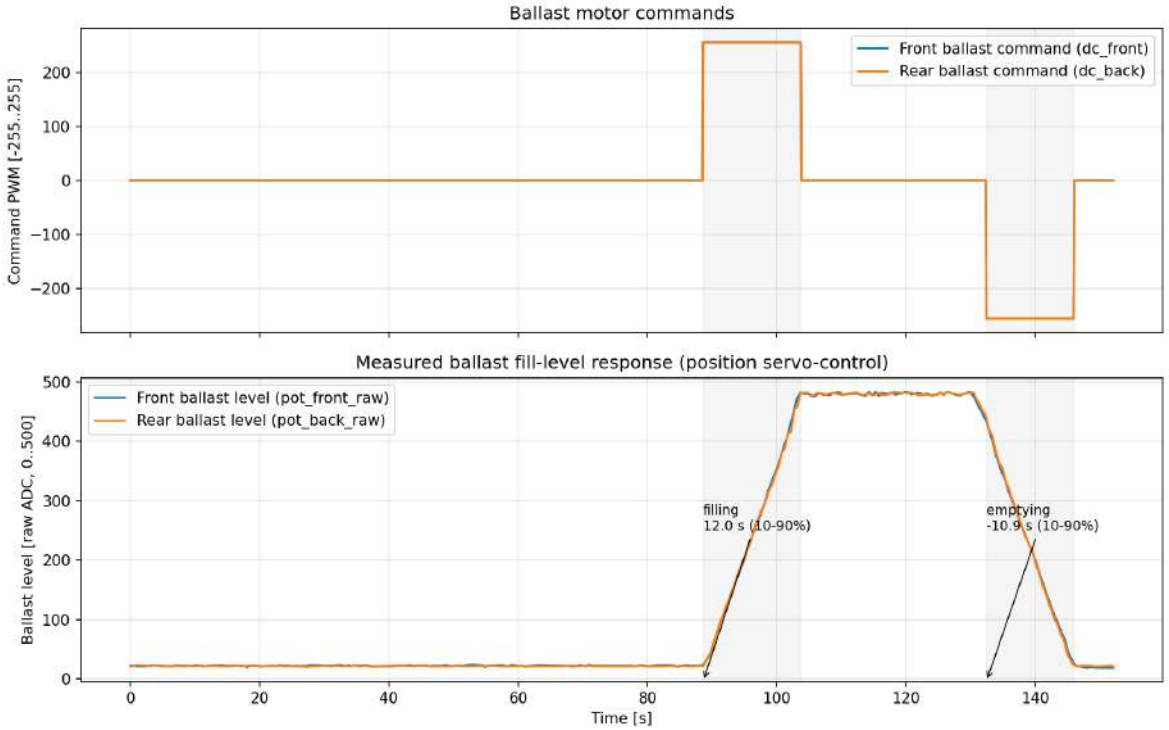

Figure 3.6: Ballast response to saturated PWM steps: command (top) and potentiometer feedback (bottom).

(1) Ballast actuator response. Figure 3.6 compares PWM commands (dc_front, dc_back) and potentiometer positions (pot_front_raw, pot_back_raw). Two full-scale steps (± 255) were applied, with hold until steady state. Both channels follow requested direction without notable divergence, confirming electromechanical symmetry of the subsystem. Transition from near-empty (~ 20) to saturation (~ 485) is observed with 10–90% response time of 12.0 s in fill and 10.9 s in drain. Absence of oscillation and overshoot at end-stops indicates well-damped behavior, close to first-order dynamics, compatible with future closed-loop vertical control.

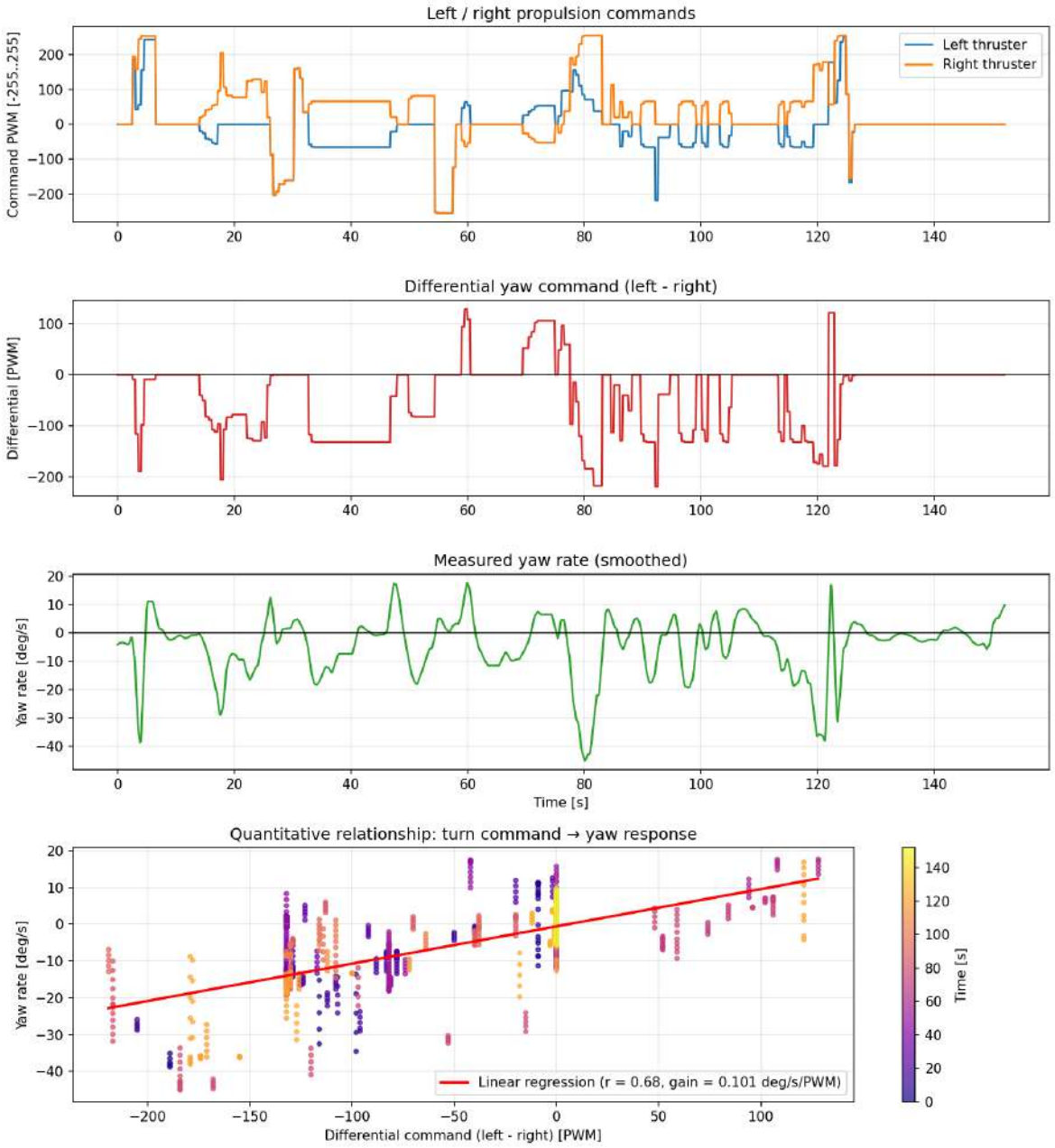

Figure 3.7: Relation between differential command ($T_L - T_R$) and yaw rate: time series and linear regression.

(2) Yaw response to differential command. Figure 3.7 isolates the steering chain from differential thruster command to measured yaw rate. Linear regression over the full run gives Pearson correlation $r = 0.68$ and positive gain of 0.101 deg/s per differential PWM unit. Sign is correct (no inversion), and trend remains monotonic. Dispersion level around regression line is consistent with real vehicle dynamics (inertia, damping, coupling with translation, and mission disturbances), rather than with command defect.

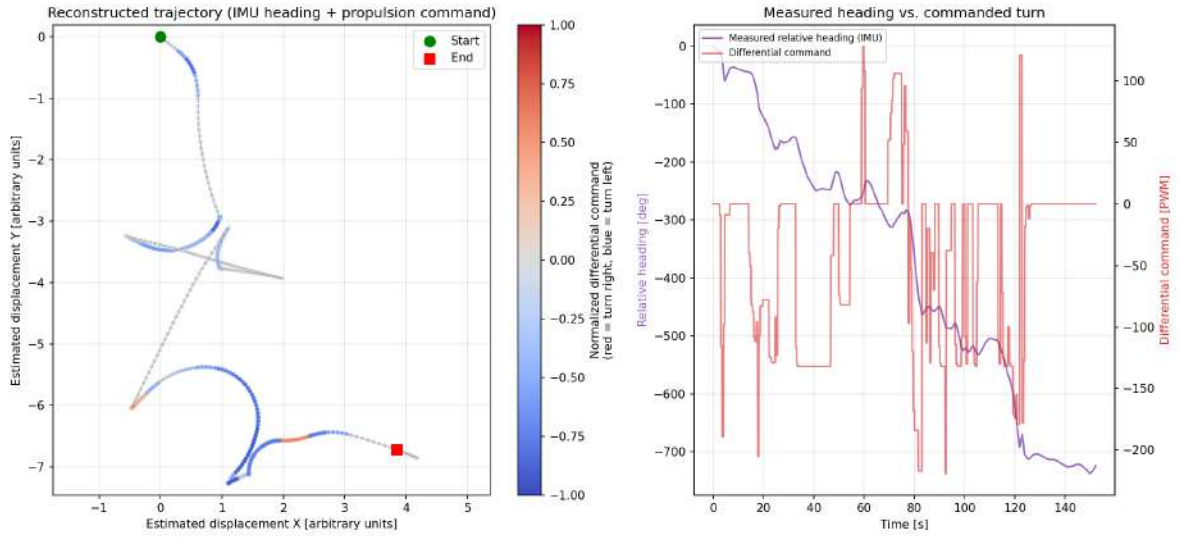

Figure 3.8: Trajectory reconstructed from IMU heading and propulsion commands (relative units).

(3) Reconstructed trajectory and kinematic consistency. Without absolute position sensor (no DVL/USBL), a qualitative 2D reconstruction was performed by *dead-reckoning*, combining unwrapped IMU heading and normalized surge command using a unicycle model:

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta.$$

Axes remain in arbitrary units (no metric speed calibration for thrusters), but structural consistency is clear: quasi-rectilinear segments for low differential command, and continuous-curvature loops during sustained differential command phases. The relative heading vs differential-command plot confirms expected causality between piloting action and robot response. This temporal + geometric convergence is strong evidence of consistency of the chain *mixer* → *thrusters* → *hull* → *IMU*.

3.2.7 Synthetic V1/V2 Comparison

The following comparison summarizes the most significant differences between the two iterations, limited to points with direct impact on test quality and operational robustness.

Table 3.1: Comparison of the two prototype iterations on key testing criteria.

Criterion	**Prototype V1**	**Prototype V2**
Surface link	Cat6 + RJ45→USB 2.0, 8 m	RS485 on two-wire Blueye tether
Dynamic behavior in pool	Functional but strongly influenced by cable	Cleaner piloting, tether influence substantially reduced
Internal integration	Architecture validated, volume still large (37 cm)	Compact reconfiguration (30 cm), better mass/volume control
Operator chain	Basic functional feedback	Full dashboard + diagnostics + rosbag recording
Vision / AI	Initial feasibility validation	Pre-operational live integration, object detection effective but errors remain
Evidence level	Mostly qualitative	Qualitative + quantitative (rosbag, dynamic responses)

Two key lessons emerge from this table. First, V2 gains are not local: they simultaneously involve mechanics (compactness), link behavior (tether robustness), and software operation (observability and post-processable data). Second, transition to quantitative evidence (rosbags and dynamic analyses) changes prototype nature: the goal is no longer only to "make it work", but to characterize system behavior with objective indicators.
This transition motivates the next section, focused on global assessment and evolution recommendations.

3.3 Performance Assessment and Recommendations for the Global Project

3.3.1 What Is Demonstrated at the End of the Internship

At the end of both iterations, the project has a concrete operational baseline: compact dual-thruster platform, active ballast, stable sensor chain, prototype-level hardened surface link, and software environment enabling both live operation and offline analysis. The V1→V2 transition is not cosmetic: it reflects measurable system robustness gains useful for the next MIMISK phases.

3.3.2 Current Identified Limits

Three dominant limits remain in current configuration:

- ESP↔Raspberry Wi-Fi link, kept to ease OTA, remains sensitive to ESP antenna radio quality,
- RS485 surface link meets inter-robot compatibility needs, but requires complex software management for heterogeneous bidirectional flows,

- IMU calibration, although functional, still needs deeper work to improve orientation robustness in long campaigns.

3.3.3 Priority Evolutions at Constant Architecture

(1) ESP↔Raspberry: Wi-Fi for maintenance, I2C for data. A relevant path is to keep Wi-Fi for remote flashing and maintenance, while moving sensor/command traffic to a local wired link (I2C). Initial integration already exists; completing it would decouple real-time robustness from radio quality.

(2) Surface↔robot: toward two-wire Ethernet transport. RS485 was a pragmatic choice to converge quickly with the aerial robot baseline. For next steps, migration toward two-wire Ethernet transport (Blue Robotics Fathom-X / LX200V20 board track) has strong impact potential: software simplification, true network full-duplex, direct SSH access, and much larger throughput margin than multiplexed serial link.

(3) IMU calibration and orientation qualification. Current calibration is sufficient for internship tests, but deeper work is needed for long missions: richer acquisition protocol, inter-session repeatability, and systematic validation of yaw drift under different magnetic contexts.

3.3.4 Evolutions Considered for a Next Generation

If requirements allow a modest size increase, several system gains become accessible: slightly larger tube, global-shutter camera adapted to marine vision, more capable Raspberry platform for heavier ROS2 and AI workloads, dedicated onboard Ethernet interface, and increased motor count to improve current handling and stabilization during video inspection.
This trajectory remains compatible with the aerial-drone transport objective if conducted with strict discipline on mass, size, and volume distribution. It is therefore a plausible and technically coherent continuation of achievements obtained during this internship.

Conclusion

This internship aimed to turn a mini-ROV concept into a technically consistent platform that can be tested in real conditions and reused for future project work. This objective was achieved. The development process covered the full system chain: watertight mechanical integration in a constrained volume, compact electrical architecture, distributed PC/Raspberry/ESP software stack, and progressive pool validation.

The main contribution of the internship is the iterative V1→V2 structure. The first prototype played its role as a functional validation baseline (propulsion, ballast, sensors, communication). The second prototype turned this baseline into a more robust and better instrumented platform, with improved piloting stability, more compact integration, and stronger analysis capability supported by rosbag recordings.

In terms of results, testing shows end-to-end consistency of the command-response chain: repeatable ballast dynamics, yaw response with correct sign and expected trend, and kinematic reconstruction consistent with applied commands. In parallel, the barometer campaign made it possible to fix an acquisition/filtering trade-off suitable for vertical navigation by explicitly linking firmware configuration to measured behavior.

The identified limits are clearly defined. The ESP↔Raspberry Wi-Fi link remains sensitive to radio quality; the surface RS485 link is effective in operation but introduces non-negligible software complexity; inertial calibration is functional but still needs improvement for long and heterogeneous campaigns. These limits do not invalidate the achieved results, but they define the technical priorities for next steps.

The evolution path is therefore clear: (i) separate maintenance and data flows between Wi-Fi and a local wired bus onboard, (ii) converge toward a two-wire Ethernet surface link to simplify multi-flow transport and increase bandwidth margin, (iii) reinforce orientation/sensor qualification and stabilization in disturbed environments, and (iv) increase compute/perception capacity on a slightly larger platform generation if mission requirements allow it.

In summary, the internship delivers a concrete, reproducible, and usable result: an operational mini-ROV whose architecture, observed performance, and current limits are documented with a level of evidence suitable for continued research-driven development.

Appendix A

Appendices

A.1 Detailed Component Table and Budget Estimate

The following table summarizes the budget estimate for the final prototype version. Fasteners already available in the laboratory are not included in the cost estimate. The tether is not included in the total.

		Supplier	Price per piece	Pieces need	Total
Motors	APISQUEEN thruster ug500	https://ww	20,54 €	2	41,08 €
	N20 Gear Motor	https://ww	3,00 €	2	6,00 €
Controler motors	Bluerobotic Basic esc	https://blu	34,94 €	2	69,88 €
	Dual H Bridge DC Controller	s_tl	1,50 €	1	1,50 €
Powering	18650 Li-Ion Cell	https://www.r	10,00 €	2	20,00 €
	BMS 2S 20A	Aliex	0,60 €	1	0,60 €
	DC DC converter 5v 3A	https://hr	8,56 €	1	8,56 €
Sensors	Pololu MinIMU-9 v6 (LSM6DSO and LIS3MDL Carrier)	https://hr.r	37,40 €	1	37,40 €
	Barometer (MS583730BA01-50)	https://hr.r	10,02 €	1	10,02 €
	linear potentiometer	https://hr	1,74 €	2	3,48 €
	rpi camera module 3 noiR wide	https://hr	43,04 €	1	43,04 €
Microcontroleur	ESP32S3 seed studio xiao	https://hr	7,30 €	1	7,30 €
	Raspberry pi zero 2 w	https://hr	12,90 €	1	12,90 €
Communication	USB to RS485 Adapter PCB	https://ftdi	23,74 €	2	47,48 €
Materials	Acrylic Dome	Aliex	2,97 €	1	2,97 €
	PVC Tube iner diameter 44mm thickness 3.6mm 35mm		20,00 €	1	20,00 €
	ASA material 3D print	https://azuref	22,90 €	1	22,90 €
	Threaded insert M2 4mm	https://hr.m	0,28 €	15	4,20 €
	M2 Screws	TBD			- €
	M1.6 Screws	TBD			- €
	syring 5ml	https://www	14,00 €	2	28,00 €
				total	387,31 €

Figure A.1: Capture of the source table used to track components, quantities, and costs.

Table A.1: Detailed component and estimated cost breakdown of the mini-ROV (excluding tether).

Item	Qty	Unit (EUR)	Subtotal (EUR)
APISQUEEN thruster UG500	2	20.54	41.08
N20 DC motor	2	3.00	6.00
BlueRobotics Basic ESC	2	34.94	69.88
Dual H-Bridge DC controller	1	1.50	1.50
Li-ion 18650 cell	2	10.00	20.00
BMS 2S 20A	1	0.60	0.60
DC-DC converter 5V 3A	1	8.56	8.56
Pololu MinIMU-9 v6	1	37.40	37.40
MS5837-30BA barometer	1	10.02	10.02
Linear potentiometer	2	1.74	3.48
RPi Camera Module 3 NoIR Wide	1	43.04	43.04
ESP32-S3 Seeed Studio XIAO	1	7.30	7.30
Raspberry Pi Zero 2 W	1	12.90	12.90
USB-RS485 converter	2	23.74	47.48
Acrylic dome	1	2.97	2.97
PVC tube (44 mm inner diameter, 3.63 mm wall thickness)	1	20.00	20.00
ASA material (3D printing)	1	22.90	22.90
M2 threaded inserts (4 mm)	15	0.28	4.20
5 ml syringe	2	14.00	28.00
Estimated total			**387.31**

A.2 Endurance Calculation Assumptions

Endurance calculations are provided as sizing estimates. They help compare operating scenarios but remain dependent on real command profiles during testing.

Robot Power Consumption and Autonomy — 2S Battery / UG500 / UBEC

Parameter	Value	Unit	Calculation / Source	Status	Comment
Battery nominal voltage	7.2	V	2 × 3.6 V	Defined	2S Li-ion
Battery full voltage	8.4	V	2 × 4.2 V	Defined	Fully charged
Battery capacity	2.5	Ah	2 × 2500 mAh	Defined	Nominal capacity
Battery energy	18	Wh	7.2 V × 2.5 Ah	Calculated	Nominal stored energy
5 V electronics load	3.29	W	Previous component-level estimate	Estimated	Pi / camera / XIAO / RS485 etc.
Battery power for electronics	3,713155555555555	W	3.29 / 0.90 + 7.2 × 0.008	Calculated	Includes UBEC efficiency and quiescent current
UG500 #1 reference power	5.04	W	7.2 V × 0.7 A	Reference	0.7 A is published at 7 V; 7.2 V is an extrapolation
UG500 #2 reference power	5,04	W	7.2 V × 0.7 A	Reference	Same assumption
Total propulsion reference power	10.08	W	2 × UG500	Calculated	ESC losses excluded
TOTAL ROBOT REFERENCE POWER	13,793155555555556	W	3.71 + 10.08	Calculated	N20 + H-bridge excluded
THEORETICAL AUTONOMY	**78,29970420111755**	**min**	**18.0 Wh / 13.79 W × 60**	**CALCULATED**	**≈ 1H18 ≈ 78.3 min**
Autonomy — electronics only	290.857731070309	min	18.0 / 3.71 × 60	Calculated	**≈ 4H50 with no propulsion**

AUTONOMY CALCULATION: Battery energy = 7.2 × 2.5 = 18.0 Wh. Electronics power from battery = 3.71 W. Two UG500 reference power = 10.08 W. Total = 13.79 W. Autonomy = 18.0 / 13.79 = 1H18 = 78.3 min.

Figure A.2: Capture of the endurance spreadsheet (consumption assumptions and results).

Table A.2: Summary of energy assumptions and associated endurance values.

Parameter	**Value**	**Comment**
Nominal battery voltage	7.2 V	2S pack (2 × 3.6 V)
Full-charge voltage	8.4 V	2 × 4.2 V
Battery capacity	2.5 Ah	2 × 2500 mAh
Nominal energy	18.0 Wh	7.2 × 2.5
Estimated electronics load	3.29 W	Pi, camera, XIAO, RS485
Battery power for electronics	3.71 W	Converter efficiency included
UG500 reference power (per unit)	5.04 W	Assumption at 7.2 V
Reference propulsion power (2 thrusters)	10.08 W	ESC losses not explicitly counted
Total reference power	13.79 W	3.71 + 10.08
Theoretical endurance (reference propulsion)	78.3 min	About 1 h 18
Theoretical endurance (electronics only)	290.9 min	About 4 h 50
Observed operational endurance	About 2 h 30	Intermittent propulsion

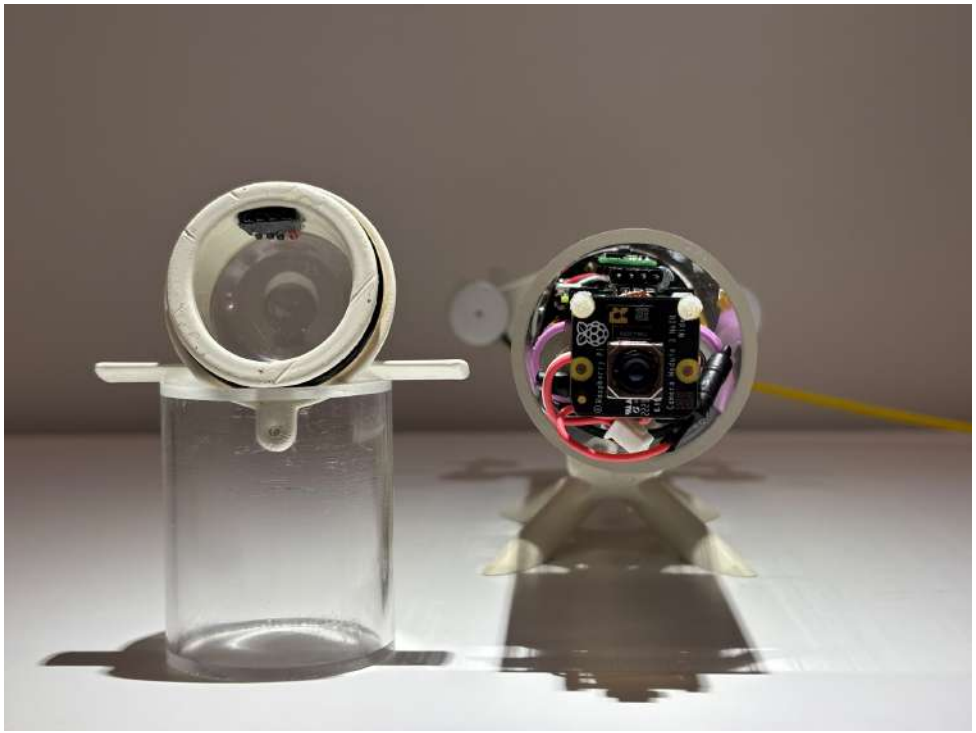

Figure A.3: Front cap detail: dome, camera, and mechanical interfaces.

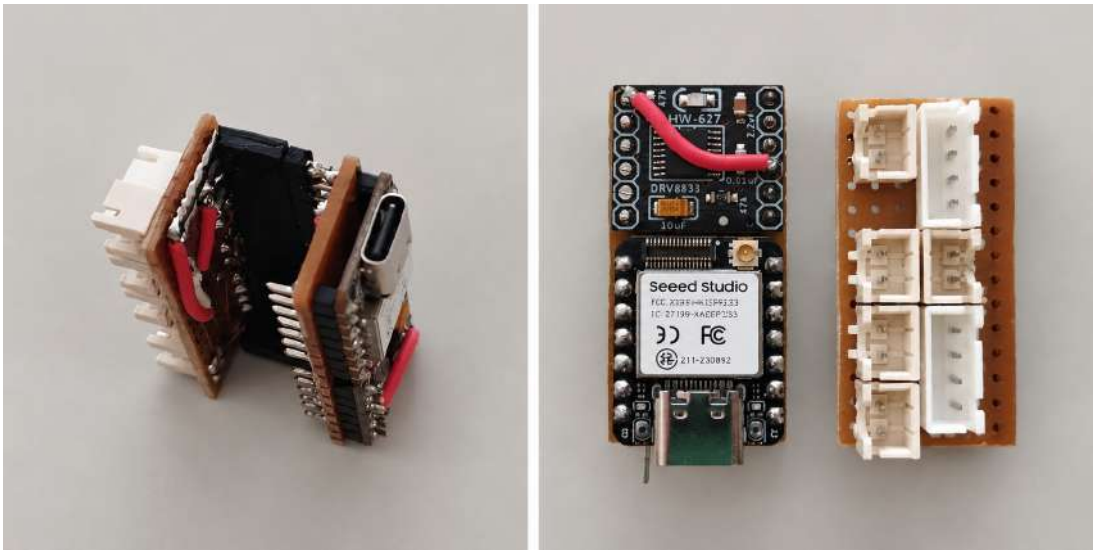

Figure A.4: Compact control protoboard (ESP32-S3 XIAO and motor interface).

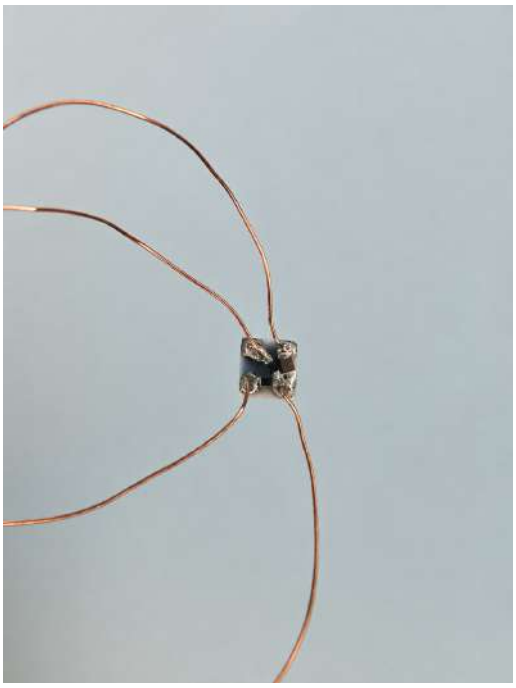

Figure A.5: MS5837-30BA pressure sensor micro-soldering before encapsulation.

A.3 Additional Mechanical Integration Illustrations

A.4 Oscilloscope Measurement at ESP32 Startup

Figure A.6: Oscilloscope capture showing a transient/noise event during control-chain initialization.

This measurement was used during the diagnostic phase between prototypes. Full interpretation and impact on the V1/V2 transition are discussed in Chapter 3 to avoid redundancy in the hardware chapter.

A.5 Test Traceability and Analysis Reproducibility

Results presented in Chapter 3 rely on artifacts stored in the repository to enable verification and replay of processing steps. Main elements are:

- campaign reports and documentation: `documentation_resultats/`,
- rosbag test datasets: `pc/rosbag/`,
- rosbag post-processing script (IMU, motors, depth plots, video export): `pc/ros2/postprocess_rosba`
- command-response analysis script (ballast, yaw, trajectory): `pc/ros2/analyze_command_response.p`
- pressure/depth characterization campaign: `test_campaigns/bar30_frequency_accuracy/`.

This organization guarantees that quantitative report figures (Chapters 2 and 3) are not isolated outputs, but traceable products of a documented test pipeline. It facilitates cross-campaign comparison and replay of analyses during future hardware and software iterations.

Bibliography

[1] APISQUEEN. *UG500 underwater thruster: product specifications*. 2024. URL: https://www.apisqueen.com/products/underwater-thruster-ug500 (visited on 08/25/2026).

[2] ESP-DIVE. *Build an ESP32 RC submarine with FPV camera*. 2024. URL: https://www.instructables.com/ESP-DIVE-Build-an-ESP32-RC-Submarine-With-FPV-Came/ (visited on 08/25/2026).

[3] Sebastian O. H. Madgwick. *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. Tech. rep. University of Bristol, 2010. URL: https://x-io.co.uk/downloads/madgwick_internal_report.pdf (visited on 08/25/2026).

[4] TE Connectivity. *MS5837-30BA Pressure Sensor: Datasheet*. TE Connectivity. 2015. URL: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Datasheet%2FMS5837-30BA%2FEnglish%2FMS5837-30BA.pdf (visited on 08/25/2026).