Autonomous Navigation, Telemetry, and Control of a Differential-Drive Marine Craft (DDBoat)

Nemanja Krčo, Luigi Agostino Mico, Unai Olalla, Coco Schwarz

BEST Autumn Course – *ROBOTson Crusoe: Let's Explore the Lost City*
ENSTA Bretagne, Brest & Lake Guerlédan, France

Project Repository: `granny-amona`

1 Introduction & Project Objectives

This report details the engineering workflow, software architecture, and field validation carried out by our team during the **BEST Marine Robotics Course** at ENSTA Bretagne and Lake Guerlédan. The overarching goal of the project was to transition from theoretical marine control concepts to a fully autonomous surface vessel capable of navigating to designated GPS waypoints and returning home safely under real-world environmental disturbances.

To achieve reliable autonomy within a compressed timeframe, our team developed the `granny-amona` software suite. Rather than relying solely on minimal single-script loops, we engineered a modular, fault-tolerant ecosystem comprising:

1. **A Bidirectional Telemetry & Motor Server (`server_telemetry.py`)** capable of running both on the physical DDBoat hardware and as a physics-based unicycle simulator (`SimBoat`) for rapid land-based testing.
2. **A Multi-Modal Ground Control Station (`boat-gui/` and `client.py`)** featuring a custom PySide6/PySDL2 graphical interface for the Steam Deck and Linux laptops, complete with live OpenStreetMap tile rendering, real-time sensor sparklines, and gyroscope tilt-steering.
3. **An Autonomous Mission & Race Architecture (`the-race/`)** implementing local tangent-plane coordinate projection, non-blocking UDP supervision, proportional heading/distance control, and multi-layered safety watchdogs.

2 Development Process: Simulation, Teleoperation & Ground Control

2.1 Onboard Telemetry Server & Physics Simulator (`server_telemetry.py`)

We developed `server_telemetry.py` as a drop-in, backward-compatible upgrade to the basic motor server. It supports two interchangeable backend classes sharing a unified interface:

- `RealBoat` **Hardware Backend:** Polls the GPS and 9-DOF IMU in a dedicated background daemon thread while protecting the Arduino serial bus with a mutex lock. It computes real-time roll and pitch angles from the accelerometer:

$$\phi = \text{atan2}(a_y, a_z), \quad \theta_{\text{pitch}} = \text{atan2}\left(-a_x, \sqrt{a_y^2 + a_z^2}\right)$$

and scales the z-axis gyroscope reading (70 mdps/LSB) to stream yaw rate alongside JSON telemetry at 5 Hz.

- `SimBoat` **Unicycle Simulator:** Models the boat on land using discrete Euler integration at 20 Hz ($\Delta t = 0.05$ s). Forward velocity v approaches a target proportional to average thruster command with first-order hydrodynamic inertia, yaw rate is modeled as $\omega = 60.0 \cdot \frac{u_L - u_R}{2 \cdot 255}$ deg/s, and realistic Gaussian noise is injected into simulated GPS coordinates.
- **Protocol & Watchdog Safety:** Communication uses newline-terminated ASCII over TCP (`TCP_NODELAY`) with ping/pong heartbeat timestamps (`P`/`O`). A strict 1.0 **s socket watchdog** automatically zeroes both thrusters if the client disconnects or stalls.



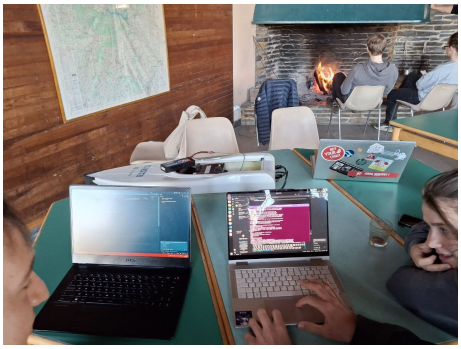


Figure 1: Indoor development and integration of the `granny-amona` software suite on the DDBoat.

2.2 Steam Deck & Laptop Ground Control GUI (`boat-gui/`)

To give our team full situational awareness during field tests, we built a custom dark-themed Ground Control Station using PySide6 and PySDL2 (`boat_gui.py`) as well as a terminal WASD client (`client.py`) and an automated `pytest` verification suite (`tests/test_core.py`):

- **Three Pilot Control Modes (`controls.py`):** Supports *Joystick Mode* (8% deadzone and cubic exponential shaping with ratio-preserving differential thruster mixing), *Tank Mode*, and *Gyro Steering-Wheel Mode*. Gyro mode reads the Steam Deck's 6-axis IMU via SDL2 and fuses accelerometer gravity vectors with gyroscope rates using a **Complementary Filter** ($\alpha = 0.98$):

$$\phi_k = 0.98(\phi_{k-1} + \dot{\phi}_{\text{gyro}}\Delta t) + 0.02\phi_{\text{acc}}, \quad \theta_k = 0.98(\theta_{k-1} + \dot{\theta}_{\text{gyro}}\Delta t) + 0.02\theta_{\text{acc}}$$

  For safety, Gyro mode requires holding a rear **dead-man paddle (`L4/L5/R4/R5`)**; releasing it immediately commands $(0, 0)$.

- **Live Slippy Map & Telemetry Dashboard (`map_widget.py`, `telemetry.py`):** Asynchronously renders OpenStreetMap tiles with local disk caching and an offline grid fallback, while displaying exponentially smoothed speed, GPS Course Over Ground (COG), home distance/bearing, round-trip latency, 60-second sparklines, CSV session logging, and one-click **GPX track export**.

3 Autonomous Navigation & Mission Control (`the-race/`)

3.1 Coordinate Projection, Compass Calibration & Mission State Machine

For the autonomous trials (`the-race/simple_mission.py`), the boat executes a deterministic 10 Hz ($\Delta t = 0.1$ s) loop supervised via non-blocking UDP (`the-race/client.py`):

- **Calibration & Local Tangent Plane:** Raw magnetometer readings are corrected for hard- and soft-iron distortions using our 4-pose calibration parameters (`calib.json`):

$$y = A^{-1}(m_{\text{raw}} - \bar{x}), \quad \psi = -\,\text{atan2}(y_y, y_x)$$

  NMEA GPS coordinates are converted to decimal degrees (`nmea_to_decimal`) with speed filtering disabled and projected into a local meter frame ($x =$ East, $y =$ North).

- **Four-State Autonomous Machine (`STOP`, `MANUAL`, `AUTO`, `BACK_HOME`):** Pressing `R` in the water captures the boat's current GPS fix as `home` $(0, 0)$ and initiates the outbound leg (`AUTO`) toward the target waypoint. Entering within a capture radius of `RADIUS = 4.0 m` automatically transitions the boat to `BACK_HOME` toward $(0, 0)$, stopping the motors upon arrival.
- **Link-Loss & GPS Watchdogs:** Operators can take manual WASD control (`MANUAL`) at any time. If UDP packets from the laptop drop for `LINK_TIMEOUT = 2.0 s` while in `MANUAL`, the boat automatically reverts to its active autonomous leg (`AUTO` or `BACK_HOME`) rather than drifting. If valid GPS fixes are lost for `GPS_TIMEOUT = 10.0 s`, the thrusters shut off.

3.2 Closed-Loop Heading & Distance Control Law

At each 0.1 s step during `AUTO` and `BACK_HOME`, the controller computes the desired bearing angle θ_{goal} and wraps the heading error e_θ into $[-\pi, \pi)$ using `sawtooth`:

$$\theta_{\text{goal}} = \text{atan2}(y_{\text{goal}} - y,\, x_{\text{goal}} - x), \quad e_\theta = \text{sawtooth}(\theta_{\text{goal}} - \theta) = \big((\theta_{\text{goal}} - \theta + \pi) \bmod 2\pi\big) - \pi$$

The squared distance to the target $d^2 = (x_{\text{goal}} - x)^2 + (y_{\text{goal}} - y)^2$ (with a deadband of $d^2 < 0.15\ \text{m}^2 \Rightarrow 0$) modulates a proportional differential control law:

$$u_1 = (\text{speed} + e_\theta) \cdot \min(d^2, 1), \quad u_2 = (\text{speed} - e_\theta) \cdot \min(d^2, 1)$$

The normalized control effort is then scaled by `MAX_PWM` $= 255$ and clamped to $[-255, 255]$.

4 Field Testing & Experimental Results

During our field sessions at Lake Guerlédan, we logged full-rate onboard telemetry to validate sensor calibration, heading estimation, and closed-loop motor response. Table 1 summarizes key events extracted from our recorded run `mission_log_1770842414.csv` (128.67 s duration).

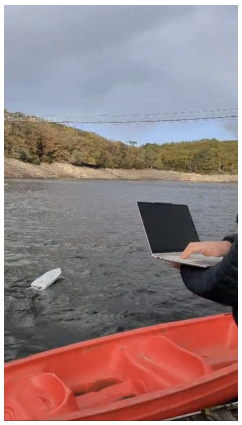


Figure 2: Field testing the autonomous navigation system on the waters of Lake Guerlédan.

Key Technical Takeaways:

- **Sawtooth Wrapping Integrity:** At $t = 45.13$ s and $t = 82.09$ s, the measured heading crossed the $\pm\pi$ rad discontinuity. Thanks to `sawtooth()`, the control signals (u_1, u_2) transitioned smoothly without chattering or windup.
- **Wave-Induced IMU Noise vs. Filtering:** The raw accelerometer a_z channel exhibited high-frequency wave chop oscillations between -12072 and $+7567$ LSB, validating our decision to use complementary filtering in `boat-gui` and exponential speed smoothing.

Table 1: Chronological Summary of Autonomous Field Trial (`mission_log_1770842414.csv`)

Time t (s)	**Local (x, y)**	**Heading θ**	**Inputs (u_1, u_2)**	**Phase & Controller Behavior**
0.01–0.82	$(0.00, 0.00)$	$+1.11 \rightarrow +1.09$	$(4.47, -2.47)$	**Initialization:** Origin established; differential thrust $(u_1 > 0, u_2 < 0)$ initiates right turn toward target bearing.
4.11–4.61	$(-0.63, -5.27)$	$-0.43 \rightarrow -0.28$	$(1.56, +0.44)$	**Heading Convergence:** As heading swings negative toward target, controller reduces differential gap and applies forward thrust.
18.53–18.89	$(-0.63, -5.27)$	$+1.57 \rightarrow +1.47$	$(-2.7, 4.7) \rightarrow (4.7, -2.7)$	**Boundary Reversal:** Crossing the opposite-direction threshold flips sign of e_θ, cleanly reversing thruster polarity.
86.81–87.30	$(-0.63, -5.27)$	$-1.62 \rightarrow -1.58$	$(1.05, 0.95)$	**Steady-State Cruise:** Boat locks onto target bearing $\theta_{\text{goal}} \approx -\pi/2$ rad (due South).
114.6–128.6	$(-0.63, -5.27)$	$-2.46 \rightarrow -2.60$	$(0.00, 2.00)$	**Disturbance Rejection:** External wave/wind yaw perturbations are countered by single-sided starboard thrust.



Figure 3: Photo of the team ”granny-amona” from left to right: Unai, Luigi, Nemanja, Coco.

5 Conclusion

Over the course of one week at ENSTA Bretagne and Lake Guerlédan, our team successfully designed, tested, and deployed a complete autonomous control and teleoperation stack for the DDBoat. By pairing a physics-based simulation mode (`SimBoat`) and a rich Steam Deck telemetry GUI (`boat-gui`) with a resilient UDP autonomous state machine (`simple_mission.py`), we were able to debug coordinate transformations, compass calibration, and control gains safely before committing to open-water runs.