

Resource-Constrained Edge Computing Architecture for Multi-Sensor Autonomous Marine Monitoring

Advaith Rajagopalan

BASIS Independent Silicon Valley

San Jose, California, USA

rajagopalan.advaith@gmail.com

Abstract—Long-term autonomous monitoring of marine debris mandates the use of simultaneous high-bandwidth sensing, neural network-based perception, autonomous navigation, and communication. This paper describes the embedded resource architecture of a low-cost catamaran autonomous surface vehicle (ASV) featuring a Raspberry Pi 5 processor, three RGB cameras, an MLX90640 thermal sensor, a pair of Blue Robotics T200 thrusters, and a metadata-based telemetry channel. Specifically, we analyze on-Pi inference at 150 ms per 640×640 frame (6.7 FPS), with 340% multi-core CPU utilization, resident memory usage of 1.3 GB, and sustained temperature of 70 °C; develop a power budget, which identifies the T200 thruster pair as a major energy sink (>90% during transit) and Pi inference as a major compute sink; and design a metadata-only telemetry protocol such that 300–600 byte JSON detection packets suffice for <6 KB/s at 10 Hz. We show that mission endurance is limited to 45–90 minutes during station-keeping investigation, 25–40 minutes during exploration, and 8–16 minutes during transit; and that compute and energy bottlenecks lie in different subsystems and are mission-phase dependent, allowing mission-aware allocation and scheduling of resources.

Index Terms—autonomous surface vehicle, edge AI, embedded systems, Internet of Things, marine monitoring, power budget, Raspberry Pi, resource-constrained computing.

I. INTRODUCTION

Continuous monitoring of floating marine debris requires mobile autonomous vehicles capable of sensing, real-time perception, navigation, and wireless communication for extended time periods [1], [2]. Embedding these functions in a single low-cost platform with sufficient performance relies on careful management of compute, power, and communication resources.

This paper addresses the resource architecture of a catamaran ASV running an embedded marine debris detector on a Raspberry Pi class platform [3], which incorporates two Pi HQ RGB cameras, a Pi AI Camera, MLX90640 thermal array, a pair of Blue Robotics T200 brushless thrusters [4], and on-Pi inference using the YOLOv8n algorithm [5]; the system’s perception accuracy and failures have been described elsewhere [6]. Hull design, mission staging, and control algorithm will be described in companion work; our focus here will be on the system resource architecture: namely, how the Raspberry Pi 5 handles compute, power, and communication.

Our task entails balancing three mutually constraining factors. First, **compute**: CPU-only Pi inference sets a stringent upper limit on perceptual throughput orders of magnitude lower

than GPU implementations [7], [8]. Second, **power**: a single lithium polymer battery must provide enough capacity for driving thrusters consuming over 700 W at peak power [4], in addition to sensing and inference at two orders of magnitude lower consumption [9], [10]. Third, **communication**: marine operations feature radio channels with insufficient bandwidth for transmitting live video [11], [12].

Two key technical contributions are made. First, we report on embedded inference, describe power consumption, and devise a metadata-based communication protocol. Second, we identify an essential yet counterintuitive architectural property: the compute bottleneck (on-Pi inference) and the energy bottleneck (brushless thrust) exist in separate subsystems and are constrained by separate mission phases. This separation implies inference-on-demand scheduling and metadata-only telemetry rather than over-provisioning.

The rest of the paper is organized as follows. In Section II, we describe the system resource architecture. Embedded inference is discussed in Section III. Power consumption is analyzed in Section IV. Communication is covered in Section V. Sections VI and VII discuss trade-offs and conclusions.

II. SYSTEM RESOURCE ARCHITECTURE

The architecture consists of six hierarchical layers connected by data, control, and power flows (Fig. 1). The sensing layer utilizes two Pi HQ RGB cameras via a CSI multi-camera adapter, a Pi AI Camera, an MLX90640 thermal array on a pan-tilt mount, and a GPS receiver and IMU module for position and heading; sensor output streams reach the Raspberry Pi (Pi) using CSI, I²C, and serial buses. The Raspberry Pi 5 compute engine [3] handles all onboard software: YOLOv8n object detection [5], thermal processing, sensor fusion, navigation decision logic, and the mission state machine. Offloading PWM signals for the actuators from the Pi 5 compute hub to the PCA9685 frees up Pi’s CPU to perform high-throughput inference and decisions without generating PWM in real-time. The actuation layer contains two T200 brushless thrusters equipped with their electronic speed controllers (ESCs) and pan-tilt servos controlling the thermal camera head. The power layer delivers a 4S LiPo battery via an XT60 connector splitter to two unregulated propulsion-bus wires and to a regulated 5 V electronics rail via a Hobbywing UBEC. The communication layer sends out confirmed detections in JSON format.

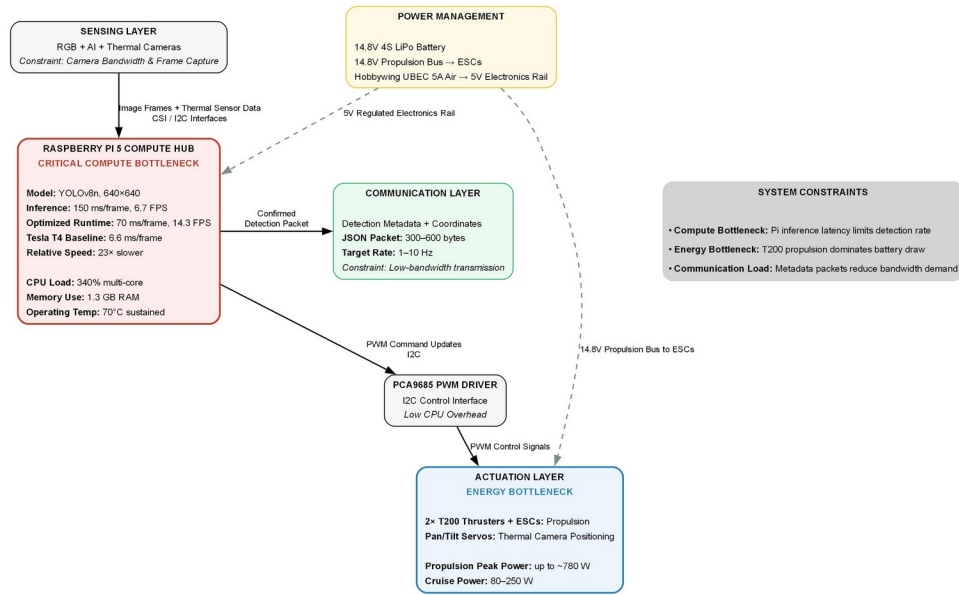


Fig. 1. System resource architecture. Six subsystem layers (sensing, Pi 5 compute hub, PCA9685 PWM driver, actuation, power management, communication) connected by data, control, and power flows. The Raspberry Pi 5 is the compute bottleneck (YOLOv8n inference latency limits detection throughput); the T200 thruster pair is the energy bottleneck (>90% of mission energy during transit). Solid arrows: data and control. Dashed arrows: power.

From the above decomposition, three important architectural attributes stand out. First, the high-current propulsion path and low-current electronics path remain separated and regulated downstream of the battery, reducing the coupling of thruster-induced voltage transients into the computation and sensing rails. Second, offloading PWM from the Pi 5 to the PCA9685 allows the former to concentrate on inference and control tasks. Third, and perhaps most importantly, the system features a compute bottleneck in the Raspberry Pi 5 and an energy bottleneck in the thruster pair, decoupled spatially (they belong to different subsystems) and temporally (at different stages of the mission).

Specifically, the compute bottleneck is imposed by YOLOv8n inference latency and caps the detection throughput at 6.7 FPS in the current design configuration (see Section III). The energy bottleneck is determined by the power consumed by T200 thruster pair, comprising >90% of the power budget during transit and 70–80% during exploration (see Section IV). As a result, the compute bottleneck is most heavily loaded during the station-keeping investigation stage when thrusters operate at a reduced throttle; energy bottleneck is most critical during transit when inference is idle or duty cycled. As these bottlenecks rarely coincide, both constraints can be addressed through careful phasing of the mission.

III. EMBEDDED INFERENCE CHARACTERIZATION

YOLOv8n inference is the major computation task on Raspberry Pi 5. Table I presents per-frame computation costs in various configurations on the training platform GPU and the Raspberry Pi 5 CPU. The measurement for Tesla T4 is from per-frame YOLOv8n inference benchmark in our companion perception study [6].

TABLE I
YOLOv8n Inference Computation Costs Across Platforms

Platform / Configuration	Inference (ms/frame)	FPS	Notes
Tesla T4 GPU	6.6	151.5	Training reference [6]
Pi 5, CPU-only (deployed)	150	6.7	Measured
Pi 5, optimized	70	14.3	NCNN/quant. projection
Pi 5, unoptimized	250	4.0	Throttling projection

Measurements were taken on a passively cooled Raspberry Pi 5 using the deployed YOLOv8n PyTorch runtime at 640×640 input after warmup, with inference time measured over sustained repeated frames. The platform produced 6.7 FPS with 150 ms per-frame inference time, 340% multi-core CPU utilization (3.4 out of 4 Cortex-A76 cores [3]), 1.3 GB of memory use, and steady-state CPU temperature of 70 °C. The configuration is ~23× slower than the Tesla T4 reference. There are two key takeaways here.

First, continuous multi-camera scanning is impractical. Given three RGB cameras and one Raspberry Pi inferring on them at 6.7 FPS sequentially, the inference on full frames of all cameras cannot be completed within available compute resources at decent frame rates. As a consequence, the architecture uses staged perception: the pan-tilt thermal sensor conducts an omnidirectional sweep (computationally cheap) and provides thermal bearing candidates for YOLOv8n confirmation. This is expected to reduce inference duty cycle by an order of magnitude or more, fitting inference workload into compute capacity.

Second, optimization opportunities are available with some price tag. Optimization paths such as NCNN export and INT8 quantization may provide additional headroom, but these

remain projections for this platform. The 70 ms / 14.3 FPS inference cost in Table I reflects this projected headroom. Conversely, thermal throttling triggered upon reaching the BCM2712 threshold may bring the number of FPS down to 4 or less; 250 ms / 4.0 FPS is the corresponding pessimistic estimate. These two points mark the boundaries of achievable range in performance around the 150 ms point.

340% sustained CPU utilization means that the inference workload fully occupies more than three Raspberry Pi 5 cores [3]. Hence, there is little room left on the host CPU for navigation, logging, telemetry. To address that, the architecture uses PCA9685 to offload PWM signal generation (negligible host costs), performs thermal and IMU sampling at low frequencies using I²C link, and conducts telemetry encoding only upon successful detection confirmation. The 1.3 GB resident memory requirement is consistent with 4 GB / 8 GB RAM configurations of Raspberry Pi 5.

IV. POWER BUDGET AND MISSION ENDURANCE

The boat employs a 4S LiPo battery rated 14.8 V, 6000 mAh, and 88.8 Wh. Safe discharge leaves roughly 20% as a reserve to ensure cell longevity, so useful onboard energy is treated as approximately 70–80 Wh after reserve and electronics-regulator losses. We adopt 75 Wh as the planning baseline.

Subsystem power consumptions are summarized in Table II and are obtained from the manufacturer datasheets [3], [4], standard component rating catalogs for peripherals, and platform-level estimates. The T200 thruster pair is the largest subsystem in terms of power, ranging 80–250 W during cruising and ~780 W when using full throttle [4]. All other subsystems aggregate their consumption from 6 to 18 W when powered.

TABLE II
Subsystem Power Consumption

Subsystem	Idle (W)	Active (W)
Pi 5 + SSD/M.2 HAT	4–6	8–12
2× T200, cruise	—	80–250
2× T200, full throttle	—	~780
2× Pi HQ + AI cameras	1.3–2.5	2.5–5.0
MLX90640 thermal	0.05–0.15	0.1–0.2
Pan-tilt servos + PCA9685	<0.2	1–7
Adapter/extension boards	0.5–1	1–2
UBEC/regulator overhead	—	~10%

The mission-phase power and corresponding endurance based on 75 Wh are aggregated in Table III. Investigation involves mostly motionless station-keeping and inference operations consuming 50–100 W and giving endurance of 45–90 minutes. Exploration requires low-speed propulsion and detection operations taking 120–175 W and providing endurance of 25–40 minutes. Transit in cruise mode is the most energy-intensive phase consuming 275–535 W, with endurance of 8–16 minutes.

TABLE III
Mission-Phase Power Consumption and Endurance from 75 Wh Budget

Mission Phase	Power (W)	Endurance (min)
Investigation (slow / station-keep)	50–100	45–90
Exploration (low cruise + sweep)	120–175	25–40
Transit (medium / high speed)	275–535	8–16

Propulsion appears to be the bottleneck. During transit, thrusters consume more than 90% of all system energy, during exploration 70–80%, and both modes require no more than 10% of their energy for inference and detection. Any optimizations of the hardware such as NPU offload, lowering clock rates, and AI cameras with on-sensor pre-processing will have little impact on the energy budget because the major part of the energy consumption belongs to propulsion. To extend the mission endurance, the propulsion subsystem should be modified using thrust schedule optimization, hull hydrodynamics design, and installing solar PV modules [9], [10].

In this system, computing resources and energy are completely decoupled. Inference is performed mostly during investigation, while propulsion is maximal during transit. Thus, there is almost never a case where these two become bottlenecks simultaneously, and the mission scheduler need not take any decisions to balance computation and energy consumption directly. However, despite solar PV modules being installed in the same area with sensors and cameras, solar augmentation remains a propulsion-side intervention [9], [10].

V. COMMUNICATION ARCHITECTURE

Telemetry via radio, RF modem, or cellular connection over an offshore range runs over channels of bandwidth far below that needed for video streaming, and the connection is intermittent at best [11], [12]. The communication protocol design focuses on metadata-only transfer: the video feed is recorded to an onboard SSD, and metadata about any detection and the current state of the system are sent to the shore via wireless channel.

The packet structure and contents are defined in Table IV, which includes fields like a timestamp, robot identifier, GPS coordinates, heading direction, detected class, detection confidence, bounding box information, the camera where the object was found, optional thermal context, current health status (battery voltage, CPU temperature), and current mission phase.

TABLE IV
Detection Packet Fields

Field	Purpose
timestamp	Detection time
robot_id	Originating ASV identifier
lat, lon, heading_deg	GPS position and camera bearing
detection.class	Object type (bottle, debris, foam)
detection.confidence	Model confidence score
detection.bbox	Bounding box in image
detection.camera_id	Source camera (1–3)
thermal.max_temp_C, mean_temp_C	Thermal context (optional)

Field	Purpose
system.battery_V, cpu_temp_C	Health monitoring
mission_phase	Current FSM state

Target latencies and throughput for this type of wireless channel are shown in Table V, with separate requirements defined for critical motor command loop and action-relevant detection packets. Detection packets are typically generated at one-per-second frequency. At this rate, the throughput amounts to less than 1 KB/s, and at the maximum of ten-per-second rate it amounts to 3–6 KB/s. Both are compatible with low-rate marine telemetry links, including LoRa, RF modem, and basic cellular links, pending link testing [11], [12].

TABLE V
Latency Targets and Link Throughput

Channel	Target
Internal motor control	<50 ms (on-board)
Detection metadata uplink	<500 ms
Mission logging	1–5 s
Detection bandwidth, 1 Hz	<1 KB/s
Detection bandwidth, 10 Hz	3–6 KB/s

Three principles define this architecture. First, the bandwidth scales with the content and not raw video footage: streaming 640×640 RGB video with 6 FPS would require bandwidth of 7 MB/s, whereas transmitting the same detection events as metadata would use only 5 KB/s, a reduction of 3 orders of magnitude. Second, storage is much cheaper compared to wireless channel, and therefore onboard NVMe provides hundreds of minutes of video storage for subsequent processing, while wireless channel transmits only actionable information. Third, graceful degradation of the system performance is provided by splitting telemetry tasks from autonomous actions in cases of connection outages.

VI. DISCUSSION

As mentioned before, compute, energy, and bandwidth requirements have distinct dependencies on each other. Projected inference speed improvements from optimizations like NCNN + INT8 inference provide additional compute resources but do not change energy consumption, since inference is already insignificant in comparison to propulsion energy budget. Reduced propulsion energy increases endurance but reduces coverage range and slows transit. Telemetry throughput reduction has minor effects on bandwidth since it is already unused most of the time and could potentially reduce detection responsiveness.

The design decisions follow from these observations. The staged perception allows managing the compute bottleneck by limiting the inference duty cycle. Inference on demand is used in order to avoid prolonged CPU usage and related high idle power consumption. Metadata-only telemetry takes care of bandwidth requirements and separates link reliability from the autonomy task. Such a multi-constraint pattern is characteristic of agricultural, environmental, and surveillance robotics on Pi-class platforms [13], [14].

Validated and unvalidated assumptions. Inference latency of 150 ms, frames per second of 6.7, CPU utilization of 340%, 1.3 GB resident memory, and 70 °C CPU temperatures are the experimental measurements on the Pi 5 platform running YOLOv8n in real conditions. The Tesla T4 latency of 6.6 ms is taken from the evaluation report of perception system presented in [6]. The projected Pi 5 optimizations with respect to NCNN and INT8 remain unvalidated on this platform. Power consumptions of various subsystems are provided by their datasheets [3], [4] and standard component ratings. Mission-phase total power consumptions are calculated from this information. Total energy budget and endurance of mission phases are projections, and actual mission-level power consumption needs further validation as well as radio-link characterization.

VII. CONCLUSION

This paper describes the architecture of the embedded computing system of a low-cost autonomous surface vehicle designed for monitoring marine debris using the simultaneous consideration of computing, energy, and bandwidth constraints. Real performance figures for the platform’s inference are provided. The overall energy budget defines propulsion as the largest consumption subsystem and sets endurance limits for different mission phases. Finally, the designed metadata-only communication protocol is within the sub-6 KB/s bandwidth typical of offshore radio telemetry. The key architectural feature of the design lies in the fact that computing and energy bottlenecks are in different subsystems and active during different mission phases. Therefore, it is possible to satisfy both of the limitations via phase-aware scheduling without over-provisioning. Open-water mission-level validation, including measured power consumption under load and characterized radio performance, is the next stage of this work.

References

- [1] J. R. Jambeck et al., “Plastic waste inputs from land into the ocean,” *Science*, vol. 347, no. 6223, pp. 768–771, 2015, doi: 10.1126/science.1260352.
- [2] L. Lebreton et al., “Evidence that the Great Pacific Garbage Patch is rapidly accumulating plastic,” *Scientific Reports*, vol. 8, Art. no. 4666, 2018, doi: 10.1038/s41598-018-22939-w.
- [3] Raspberry Pi Ltd., “Raspberry Pi 5 Product Brief,” Apr. 2026. [Online]. Available: <https://pip.raspberrypi.com/documents/RP-008348-DS-raspberry-pi-5-product-brief.pdf>
- [4] Blue Robotics, “T200 Thruster,” product datasheet. [Online]. Available: <https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/>
- [5] G. Jocher, A. Chaurasia, and J. Qiu, “Ultralytics YOLOv8,” GitHub repository, 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [6] A. Rajagopalan, “Deployment-Based Hard-Negative Stress Test of Marine Debris Detection Using Water-Dependent Negative Examples for Resource-Constrained ASVs,” accepted, *Proc. 2026 IEEE Canadian Atlantic Ocean Symposium (CAOS)*, Halifax, NS, Canada, 2026.
- [7] D. K. Alqahtani, M. A. Cheema, M. A. Rodriguez, and A. N. Toosi, “A Comprehensive Evaluation of Deep Learning Object Detection Models on Heterogeneous Edge Devices,” *arXiv:2409.16808*, 2026.
- [8] M. T. Okano, W. A. C. Lopes, S. M. Ruggero, O. Vendrametto, and J. C. L. Fernandes, “Edge AI for Industrial Visual Inspection: YOLOv8-Based Visual Conformity Detection Using Raspberry Pi,” *Algorithms*, vol. 18, no. 8, Art. no. 510, 2025, doi: 10.3390/a18080510.

- [9] J. F. P. Fernandes et al., "A smart energy management system for surface unmanned vehicles for border surveillance missions," *Scientific Reports*, vol. 15, Art. no. 23684, 2025, doi: 10.1038/s41598-025-08579-x.
- [10] F. Sotelo-Torres, L. V. Alvarez, and R. C. Roberts, "An Unmanned Surface Vehicle (USV): Development of an Autonomous Boat with a Sensor Integration System for Bathymetric Surveys," *Sensors*, vol. 23, no. 9, Art. no. 4420, 2023, doi: 10.3390/s23094420.
- [11] L. Parri, S. Parrino, G. Peruzzi, and A. Pozzebon, "Low Power Wide Area Networks (LPWAN) at Sea: Performance Analysis of Offshore Data Transmission by Means of LoRaWAN Connectivity for Marine Monitoring Applications," *Sensors*, vol. 19, no. 14, Art. no. 3239, 2019, doi: 10.3390/s19143239.
- [12] S. Pensieri et al., "Evaluating LoRaWAN Connectivity in a Marine Scenario," *Journal of Marine Science and Engineering*, vol. 9, no. 11, Art. no. 1218, 2021, doi: 10.3390/jmse9111218.
- [13] N. Tahir and R. Parasuraman, "Edge Computing and Its Application in Robotics: A Survey," *Journal of Sensor and Actuator Networks*, vol. 14, no. 4, Art. no. 65, 2025, doi: 10.3390/jsan14040065.
- [14] A. S. Seisa, S. G. Satpute, and G. Nikolakopoulos, "A Kubernetes-Based Edge Architecture for Controlling the Trajectory of a Resource-Constrained Aerial Robot by Enabling Model Predictive Control," *arXiv:2301.13624*, 2023.