Design and Implementation of OREO: An Autonomous Four-Wheel-Drive Rover for Waste Sorting and Navigation.

Steve Muya
Team lead, hardware

Ibrahim Gichemba
Chassis design and prototyping

Jimmy Kimeu
Software

Andrew Muchangi
Software, computer vision

Anilla Wambui
Software, documentation

Robotics Dojo 2026 Competition, Jomo Kenyatta University of Agriculture and Technology (JKUAT), Kenya

***Abstract*—This paper presents OREO, a fully 3D-printed, four-wheel-drive autonomous rover built by Team Asterix (Team 3) for the Robotics Dojo 2026 waste-management challenge. The robot reads a waste image on a display screen, loads the matching material, crosses a dynamic obstacle zone through a gate or over a ramp, and offloads into the correct colour-coded section without human intervention. Computation is divided between a Raspberry Pi 5, which runs ROS 2 Jazzy, slam_toolbox, AMCL, Nav2 and a py_trees mission tree, and an ESP32-S3, which handles motor control, encoder and IMU processing and safety stops. A YOLO26n detector compiled for a Hailo-8L accelerator recognises plastic, metal and organic waste in about 23 ms per frame. We describe the mechanical, electrical, software and vision subsystems, report a float-model precision of 0.68 and recall of 0.72 on the competition test split, and document the engineering problems we met, including a Raspberry Pi 5 that would not boot from its buck converter until a 1000 µF output capacitor was added.**

***Index Terms*—autonomous mobile robot, ROS 2, SLAM, Nav2, behavior tree, YOLO, Hailo-8L, 3D printing**

I. INTRODUCTION

The 2026 Robotics Dojo challenge is waste management, and it asks a great deal of a small robot. Robotics Dojo is the project-based training programme at JKUAT under the AFRICA-ai-JAPAN Project, supported by JICA, and its challenge grows every year: line following and mazes in 2022, ROBOKEN standards in 2023, ROS and SLAM in 2024, and autonomous operation with vision and rough terrain in 2025.

This year the robot must drive to a display screen, decide whether the waste shown is plastic, metal or organic, and collect the matching load at its loading station. It then crosses a zone containing a dynamic obstacle and chooses between a short route through a gate and a longer route over a ramp. Finally it offloads into the correctly coloured section and stops in the finish zone. Six skills underpin this: object detection, material loading, terrain navigation, dynamic obstacle avoidance, path planning and offloading.

We are Team Asterix (Team 3), five team members with one robot, which we call OREO. The contributions of this paper are: (1) a compact, fully 3D-printed four-wheel-drive platform with a servo-driven offloader; (2) a power system sized from measured loads around a 4S lithium-ion pack; (3) a ROS 2 software stack combining SLAM mapping, AMCL localisation, Nav2 and a behaviour-tree mission planner; and (4) an on-board YOLO26n waste detector running on a Hailo-8L accelerator, which leaves the Raspberry Pi CPU free for navigation.

II. SYSTEM OVERVIEW AND DESIGN STRATEGY

We designed backwards from the scoring: whatever the judges score, the robot has to do reliably. That produced a short list of requirements, each of which became a subsystem (Table I).

TABLE I
MISSION REQUIREMENTS AND DESIGN RESPONSES

Requirement	Design response
Waste detection	YOLO26n on Hailo-8L, majority vote over recent frames
Loading	Nav2 dock goal, then a short drive with the ultrasonic stop disabled
Terrain	Four driven spike-tread wheels; motors sized for a 7° incline
Dynamic obstacles	LiDAR costmaps in Nav2 and a 15 cm ultrasonic stop
Route choice	Gate check with a timed wait in the behaviour tree; ramp otherwise
Offloading	Camera colour check, 180° turn, servo-opened hopper

Two decisions shaped everything else. First, the computation is split in two (Fig. 1): the Raspberry Pi 5 does the thinking (SLAM, navigation, vision and mission logic) and an ESP32-S3 does the time-critical work (motors, encoders, IMU and the distance-based emergency stop). Second, we rehearsed in simulation before trusting the hardware, using Gazebo [1], [12], so most software faults were found in simulation rather than on the arena floor.

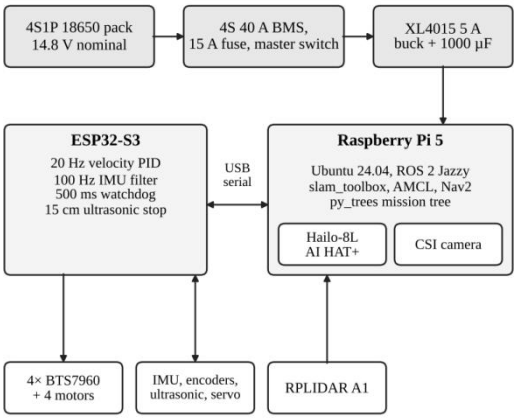


Fig. 1. System architecture of OREO. Power flows from the battery through the BMS to the motor drivers and, through the buck converter, to the Raspberry Pi 5. The ESP32-S3 and the Pi exchange commands and feedback over a single USB serial link.

III. MECHANICAL DESIGN

A. Chassis and drivetrain

The robot is one parametric assembly modelled in Autodesk Inventor. Every part is positioned with constraints, so a change in one place moves everything that depends on it. The body is a vented box of 240 × 190 × 81 mm with a removable cover for wiring access (Fig. 2 and Fig. 3). Slots and holes in the top plate provide cable pass-throughs and mounting points. Four motor bracket holders carry the drive motors, each fitted with an 85 mm spike-tread tyre. The finished robot weighs about 3 kg and occupies roughly 0.26 × 0.30 m on the floor.

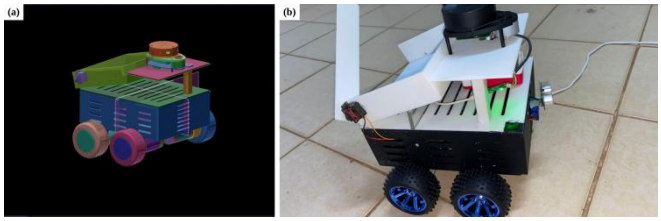


Fig. 2. OREO: (a) CAD render of the chassis and (b) the built robot with the hopper open.

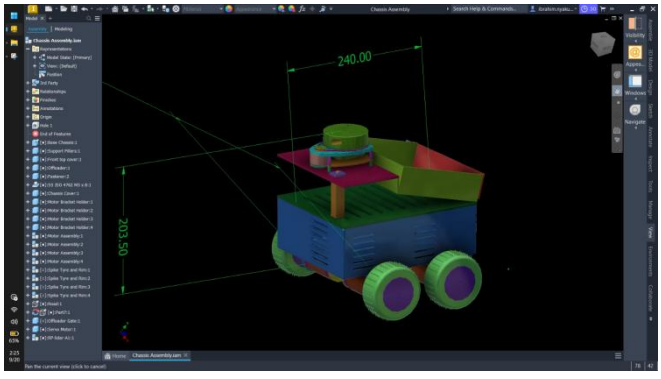


Fig. 3. Inventor assembly with the main chassis dimensions annotated.

We printed the entire robot. A design change goes from CAD to a new part in hours, vent slots and motor brackets print in one piece, and a broken part is reprinted from the same file. Infill can be tuned to keep the weight down, and a single fastener size (M3 × 8 socket-head screws) keeps assembly and spares simple.

B. LiDAR mount and a late design change

An RPLIDAR A1 sits on a raised plate so that its 360° view clears the body. In our first version the plate was held by a single support pillar. Late in the build we added a second, identical pillar on the opposite side of the same base, so the plate is now supported from both sides (Fig. 4a). The aim was a stiffer mount and a steadier, level view for the LiDAR, the sensor on which all of our navigation depends.

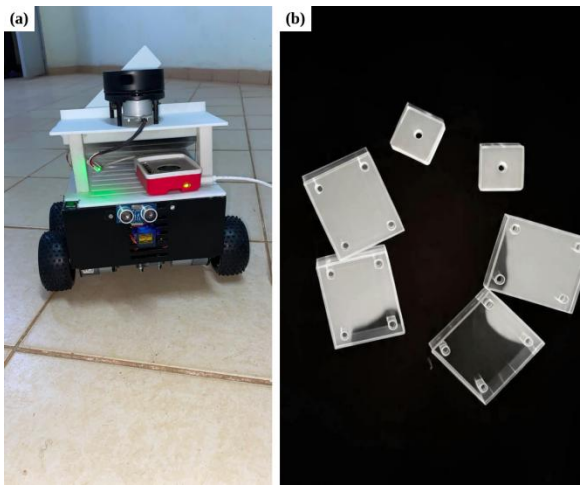


Fig. 4. (a) The built robot with the second support pillar on the opposite side of the LiDAR base. (b) Printed chassis plates and parts before assembly.

C. Offloader

The offloader is a tray with a hinged gate swung open by a single MG90S servo. The robot turns 180° at the offload section so that the hopper opens into it. The servo moves from 162° to 90°, holds for 2 s and returns.

IV. ELECTRICAL SYSTEM AND FIRMWARE

A. Power distribution

The power system was sized from the loads. The robot runs from four 18650 lithium-ion cells in series (4S1P, 14.8 V nominal, 16.8 V fully charged) behind a 4S 40 A balancing BMS. A 15 A blade fuse and a master toggle switch follow the BMS, and every subsystem shares a single ground busbar. Power wiring is 18 AWG and signal wiring is 24 AWG. Table II lists the measured and estimated loads.

TABLE II
CURRENT BUDGET ON THE 4S BUS

Subsystem	Nominal (A)	Peak (A)
Rear motors (2, encoded)	1.0	4.6
Front motors (2, open-loop)	1.0	4.6
Raspberry Pi 5, LiDAR, camera	1.2	2.0
Servos (hopper, ultrasonic pan)	0.3	0.9
ESP32-S3, IMU, ultrasonic	0.1	0.15
Total	**3.6**	**12.25**

At 3.6 A the 2500 mAh pack runs for about 42 minutes, against a course measured in minutes. The 10C cells can deliver 25 A, so even the 12.25 A worst-case stall stays under half the pack limit. The plastic 18650 spring holders heat and melt at these currents, so we bypassed the negative springs with soldered 18 AWG wire.

The Raspberry Pi 5 needs 5 V at up to 5 A and is fed from an XL4015 5 A buck converter with a heatsink, branched off the main switch ahead of the motor drivers so that motor noise stays away from the computer. The ESP32-S3 is powered by, and talks to, the Pi over one USB cable (see Section VIII for the start-up problem this converter caused). Each wheel has its own BTS7960 43 A driver; we rejected L298N drivers because their 2 A per-channel limit could not survive a combined 9.2 A stall. Because 16.8 V exceeds the 12 V motor rating, the firmware caps PWM at 180 of 255, limiting the motors to 16.8 × 180 / 255 = 11.86 V.

B. Motor sizing

The motors were checked against a deliberately pessimistic case: an 8 kg robot (more than twice the real mass) on a 7° ramp with a rolling-resistance coefficient of 0.05. The torque required per motor is

$$\tau_m = m\, g\, (\sin\theta + C_{rr} \cos\theta)\, r / N \qquad (1)$$

where m is the mass, g gravitational acceleration, θ the incline, C_{rr} the rolling-resistance coefficient, r = 42.5 mm the wheel radius and N = 4 the number of driven wheels. Equation (1) gives 0.143 N·m per motor. A 2.5× allowance for skid-steer turning on sand raises this to 0.358 N·m (3.65 kgf·cm), well inside the 10 kgf·cm rating of the 12 V JGB37-520 gearmotors. The speed requirement is modest: 33.25 m of arena path in 180 s needs only 0.185 m/s.

C. Low-level firmware

The ESP32-S3 runs C++ firmware on the Arduino framework and handles everything that cannot wait for the Pi. The rear wheel encoders trigger interrupt handlers kept in internal RAM, so no ticks are lost. A velocity PID loop runs at 20 Hz, matching the controller period of the Pi's ros2_control configuration. The IMU is sampled at about 100 Hz; roll and pitch come from a complementary filter [15] and yaw from gyro integration.

The drivetrain is deliberately uneven: encoded 110 RPM motors (JGB37-520) on the rear axle and plain 200 RPM motors (37GB-3530) on the front. Early on, shared PWM channels let the faster front wheels drag the rear ones. Giving every wheel its own driver fixed this, and the open-loop front motors are scaled by 0.55 (110/200) so that all four wheels turn at the same surface speed.

The firmware also owns safety. If the Pi is silent for 500 ms, the motors stop; a stall or runaway lasting half a second cuts the drive; and an ultrasonic sensor stops the robot at 15 cm inside the firmware, within one control loop, because a round trip through the Pi is too slow to avoid a collision. The mission tree disables this auto-stop only while docking at the loading station. The ESP32's own heading hold is off by default because it fought the small steering corrections Nav2 was already making.

V. SOFTWARE, MAPPING AND NAVIGATION

A. Software stack

The software runs on the Pi 5 under Ubuntu 24.04 and ROS 2 Jazzy [1] with CycloneDDS as middleware. A ros2_control hardware plugin talks to the ESP32 over serial, so Nav2 [2] sees OREO as an ordinary differential-drive base.

B. Mapping

We map the field with slam_toolbox [3] in one slow drive of about three minutes, steering by keyboard at roughly 0.2 m/s and finishing near the start point. A timed mapping run saves the map automatically shortly before the deadline. In simulation (Fig. 5) the saved map matched the true walls to within a mean error of 3 cm. We then clean the map by erasing the gate once, so that the costmaps learn its real state live, and place every checkpoint at least 0.5 m from the walls, because the robot only spins and backs up when squeezed closer than that.

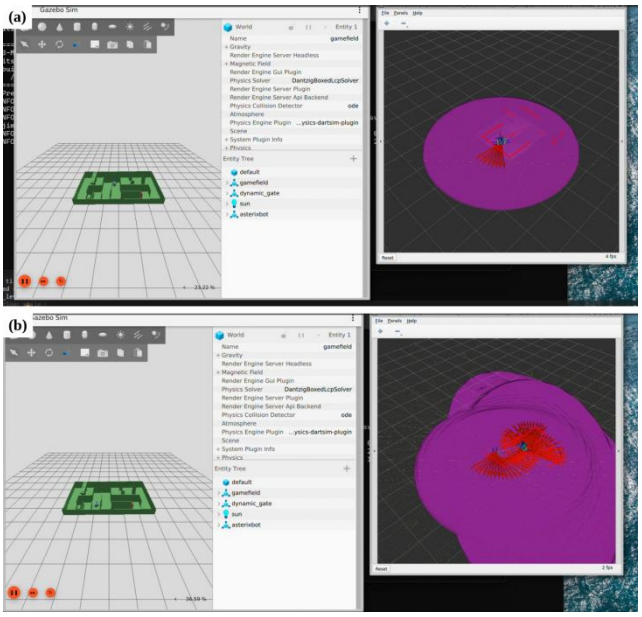


Fig. 5. Mapping in simulation: Gazebo (left) and the RViz map and scan (right), (a) early in the run and (b) later, as the arena walls fill in.

C. Localisation and navigation

On a saved map the robot localises with AMCL [4], [14] and navigates with Nav2: the NavFn planner, a Regulated Pure Pursuit controller with a rotation shim [5], and a collision monitor. Before each run a scan-matching script finds the starting pose on the map, and a mean scan-to-wall distance below 3 cm confirms a good fit. Three fixes made the largest difference. First, the LiDAR sees the robot's own chassis about 0.2 m behind it and slam_toolbox has no minimum-range setting, so we mask those beams in a filter before SLAM; otherwise the robot maps itself as a wall. Second, the IMU reads about 4° of pitch on flat ground, which made our tilt filter discard every scan beyond 2.8 m; we measure the resting offset (200 samples) and subtract it, and real 7° ramps

still trigger the filter as they should. Third, rays with no return now clear old costmap marks, so a person who has walked away stops blocking the path.

D. Mission behaviour tree

Mission logic is a py_trees behaviour tree [6] that follows the course in order: classify the waste, go to the loading zone, cross the obstacle zone, take the gate or the ramp, offload, finish and stop. At the loading zone the robot docks 0.25 m in, waits for the load and backs out. In the obstacle zone, if the ultrasonic reads below 15 cm, the tree pauses for 2 s and retries up to five times. The gate check is the only branch: if the gate opens within the wait time the robot takes the short route, and otherwise it backs out and follows the ramp waypoints.

At the offload section the camera first checks the colour of the section (blue for plastic, green for organic, red for metal). The robot then turns 180°, opens the hopper, waits 2 s and closes it. The run ends with a controlled stop in the finish zone that holds zero velocity until odometry has been still for a full second, so the stop is demonstrated and not merely commanded. Each stage can be run alone through a *stages* parameter, which let us test detection and offloading in isolation long before the whole course was ready.

VI. COMPUTER VISION

We built two waste models. The reasons for abandoning the first are the most useful vision lesson of the project.

A. Model 1: a transfer-learning classifier

The first model is a MobileNetV2 [7] classifier trained by transfer learning on the public Custom Waste Classification Dataset [13] (plastic, metal, organic), exported to TensorFlow Lite (2.5 MB) and run on the Pi's CPU at 224 × 224 input. It gave a working answer quickly, but it has a flaw that cannot be trained away: a classifier must always choose one of its three classes. Pointed at an empty wall, it still answers plastic, metal or organic. A 0.60 confidence threshold and a five-frame vote reduced these false answers without removing them, and the model also competed with SLAM and Nav2 for the CPU. It is now kept as a fallback.

B. Model 2: a Dojo YOLO26n detector

We then adopted the Robotics Dojo's pretrained YOLO26n [8], [9] waste detector, trained on the competition's own rdj2026 dataset [10] with classes metal, organic and plastic. A detector can report that nothing is present and returns a box and a score for every object. Its end-to-end one-to-one head produces final detections without non-maximum suppression, which keeps post-processing cheap and deterministic on the Pi. The model takes 640 × 640 letterboxed RGB input.

C. Compilation for the Hailo-8L

The AI HAT+ runs only Hailo Executable Format files, so the model was exported to ONNX (opset 21, end-to-end head kept) and compiled on an x86-64 laptop with Hailo Dataflow Compiler 3.34.0 and Model Zoo v2.20.0 [11]. We calibrated on 256 competition images, because 512 images needed about 9 GB and crashed our 12 GB virtual machine, and compiled on the CPU at a reduced optimisation level without AdaRound. The depthwise layers, head convolutions and all six outputs stay at 16-bit precision (a16_w16), since detection heads lose most of their accuracy at 8 bits. A custom decoder converts the six raw output tensors (box and class maps at strides 8, 16 and 32) into boxes. On 40 test images it reproduced the floating-point ONNX output to within 0.00003 px, so any later accuracy loss can only come from quantisation.

D. On-robot integration

The camera frame goes to the Hailo chip, and the most common class over the last five frames becomes the published waste type. The mission tree accepts a label after three matching readings and can fall back to a default type after a timeout, so the run never freezes at the screen. If the AI HAT is not detected, the same node falls back to the TFLite classifier. A separate HSV colour check on the same camera reads the offload-section colour and the gate state without involving the neural network.

VII. EXPERIMENTAL RESULTS

A. Waste detection

TABLE III
PERCEPTION PERFORMANCE

Measurement	Result
Precision / recall (float model, rdj2026 test split: 153 images, 269 objects)	0.68 / 0.72
Recall per class	metal 0.96, plastic 0.80, organic 0.51
Decoder vs. float ONNX (40 images)	≤ 0.00003 px
Inference on Hailo-8L, device open	≈ 23 ms / frame
Throughput on Hailo-8L	43 fps
Camera to boxes, live on the Pi	≈ 38 ms / frame
Time to a confirmed label	≈ 1.5 s

Metal is recognised almost perfectly, while organic is the weak class because organic items vary most in shape and colour. Speed is no longer a constraint: the chip needs 23 ms per frame, so our 0.5 s publishing interval, not the model, sets how quickly a label is confirmed. On the robot, metal and plastic labels stayed steady for 10 to 20 consecutive readings while items were swapped.

B. Fallback classifier

Fig. 6 and Table IV show the fallback classifier on a laptop webcam. The bottle was labelled plastic with near-certain confidence. The frying pan and the pink comb were labelled correctly most of the time, but confidence was lower and several readings fell below the 0.60 threshold and were reported as uncertain, which is the behaviour we rely on to avoid acting on a doubtful label.



Fig. 6. Live webcam tests of the Model 1 classifier: (a) a plastic bottle, (b) a metal frying pan and (c) a pink plastic comb.

TABLE IV
LIVE WEBCAM TESTS OF MODEL 1

Item	Class	Confidence	Uncertain
Water bottle	plastic	99–100 %	0 of 9
Frying pan	metal	60–95 %	1 of 11
Pink comb	plastic	64–89 %	2 of 9

Readings are those visible in the terminal captures of each test.

C. Mission performance

The behaviour tree runs all mission checkpoints in sequence: waste detection, loading, obstacle zone, gate or ramp decision, and offloading. Detection and offloading together ran start to stop in 57 s on a test map on the robot. Table V gives the simulated route times for the three gate cases. Our fake-robot test bench runs five scenarios (metal with an obstacle, plastic with a closed gate, organic with an open gate, a wrong bin colour that is warned about, and a wrong bin colour that is refused), and each ends as designed.

TABLE V
MISSION TIMING

Scenario	Time (s)
Open gate (simulation)	17.5
Closed gate, ramp route (simulation)	69
Gate closes mid-route (simulation)	92
Detection and offloading, test map (robot)	57

VIII. CHALLENGES AND LESSONS LEARNED

A. The Raspberry Pi 5 that would not boot

Our most stubborn problem looked like a wiring fault and turned out to be a power-quality one. We fed the Pi 5 from the XL4015 buck converter, which steps the battery voltage down to 5 V. Power clearly reached the Pi, but not enough of it: the board never booted, which made the fault hard to diagnose because the supply appeared to be working.

Our reading is that the converter could hold 5 V at rest but sagged when the Pi 5 demanded its start-up surge. The Pi 5 is specified for 5 V at 5 A, which is the converter's rated limit, so there was almost no margin to absorb a transient. The remedy was inexpensive: a 1000 µF capacitor across the converter's output pins gives the Pi a small local energy reserve during start-up. With the capacitor fitted, the Pi 5 powered up and booted. The lesson is to measure a supply while it is working, not only while it is idle, and to treat a computer's start-up current as a requirement separate from its running current.

B. Other problems

- *The robot mapped itself.* The LiDAR saw the chassis behind it, so SLAM drew the robot's outline as walls. Masking those beams before SLAM fixed it.
- *A phantom ramp.* A 4° resting IMU pitch made the tilt filter cut every scan; subtracting a measured offset fixed it, and the offset must be re-measured if the IMU is remounted.
- *Wheels fighting each other.* Mixed 110 and 200 RPM motors on shared PWM channels dragged each other; four drivers and a 0.55 scale solved it.
- *A stale costmap mark* beside the classify point stopped the final turn and ended one run; letting empty rays clear old marks removed the problem.
- *Two software stacks, one robot.* A launch over SSH skipped our shell settings and fell back to a different DDS, so Nav2 rejected goals. One start script now exports the right settings, kills leftover nodes and refuses to start on a nearly full disk.
- *Compiling without a GPU.* Cutting the calibration set to 256 images and lowering the optimisation level let the Hailo compile fit a laptop.
- *A loose ribbon.* On one boot the AI HAT was not detected until its PCIe ribbon cable was reseated.

IX. CONCLUSION

This paper described OREO, a fully 3D-printed four-wheel-drive rover that performs the Robotics Dojo 2026 waste-sorting mission autonomously, from classifying waste on a display to offloading it into the matching bin. Dividing the work between a Raspberry Pi 5 for SLAM, Nav2 navigation and vision and an ESP32-S3 for closed-loop motor control and safety kept time-critical control independent of heavier computation. A YOLO26n detector compiled for the Hailo-8L runs at about 23 ms per frame, with a float-model precision of 0.68 and recall of 0.72, while a MobileNetV2 classifier provides a CPU fallback. Hardware iteration resolved two significant problems: a 1000 µF output capacitor gave the Pi 5 the start-up reserve it needed to boot, and a second support pillar stiffened the LiDAR mounting base. Together, the mechanical, electrical, software and vision subsystems form an integrated platform that meets the mission requirements, and the design decisions and failures recorded here may help future Dojo teams build faster.

ACKNOWLEDGMENT

The authors thank the Robotics Dojo committee and trainers at JKUAT for the tutorials, the pretrained rdj2026 waste detector and the guidance that carried the team from a bare chassis to a working robot. We are grateful to JICA and JKUAT for hosting the internship and supporting the parts, and to earlier Dojo teams whose written papers showed what a good build looks like.

REFERENCES

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Sci. Robot.*, vol. 7, no. 66, Art. no. eabm6074, May 2022.

[2] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The Marathon 2: A navigation system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2020, pp. 2718–2725.

[3] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *J. Open Source Softw.*, vol. 6, no. 61, p. 2783, 2021.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.

[5] S. Macenski, S. Singh, F. Martín, and J. Ginés, "Regulated pure pursuit for robot path tracking," *Auton. Robots*, vol. 47, pp. 685–694, 2023.

[6] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton, FL, USA: CRC Press, 2018.

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 4510–4520.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 779–788.

[9] Ultralytics, "Ultralytics YOLO documentation." [Online]. Available: https://docs.ultralytics.com

[10] Robotics Dojo, "rdj2026 dataset, version 2," Roboflow Universe. [Online]. Available: https://universe.roboflow.com/nikita-kamau/rdj2026

[11] Hailo Technologies, "Hailo Dataflow Compiler 3.34.0 and Hailo Model Zoo v2.20.0 documentation." [Online]. Available: https://hailo.ai

[12] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2004, pp. 2149–2154.

[13] wasifmahmood01, "Custom Waste Classification Dataset," Kaggle. [Online]. Available: https://www.kaggle.com

[14] D. Fox, "Adapting the sample size in particle filters through KLD-sampling," *Int. J. Robot. Res.*, vol. 22, no. 12, pp. 985–1003, 2003.

[15] R. Mahony, T. Hamel, and J.-M. Pflimlin, "Nonlinear complementary filters on the special orthogonal group," *IEEE Trans. Autom. Control*, vol. 53, no. 5, pp. 1203–1218, Jun. 2008.