Technical Design Paper: Eigen, an Autonomous Waste Management Robot for Robotics Dojo 2026

Victoria Rotich, Rimax Mudhune, Shalom Kiptanui, and Collins Chemweno (Team Eigen)

***Abstract*—This paper documents the design of Eigen, a four-wheel autonomous robot that maps an arena, classifies a waste type, loads the matching item, chooses between a ramped and a gated route, and offloads into a colour-coded bin. A Raspberry Pi 5 runs the Robot Operating System 2 (ROS 2) with Simultaneous Localization and Mapping (SLAM) and Navigation 2 (Nav2), while an STM32 microcontroller handles real-time motor control. Ackermann steering, a gravity-fed rear gate driven by one servo, and Zenoh middleware complete the design. Development followed a simulation-first approach and favoured proven software packages. SLAM, Nav2 navigation, and waste detection with a You Only Look Once (YOLO) model on the AI HAT+ have been demonstrated, and the paper reports the failures, mitigations, and lessons recorded during development.**

I. Introduction

Robotics Dojo is a project-based robotics training program hosted at the Jomo Kenyatta University of Agriculture and Technology (JKUAT), in which student teams build robots for an annually changing challenge that mirrors real industrial problems. The 2026 challenge addresses urban waste management. A robot must complete a 3 min mapping phase, during which manual control is permitted, followed by a 3 min fully autonomous phase [1].

In the autonomous phase, the robot navigates to a display screen, classifies the waste type shown (plastic, metal, or organic), and drives to the loading station for that type. It then crosses ramps and mixed terrain or takes a gated flat route, avoids dynamic obstacles that represent workers, and deposits the load in the correct colour-coded offloading zone. The robot must fit within a 30 cm cube, use a self-contained supply of at most 24 V, and log timestamps for key events [1].

Team Eigen designed a robot named Eigen for this challenge. Section II explains the design strategy and its trade-offs. Section III describes the vehicle design, including the mechanical, electrical, software, and vision subsystems, and the lessons learned. Section IV reports the testing performed, the results to date, and the reliability analysis. Section VII describes how the system communicates its state to users.

II. Design Strategy

A. Challenge Requirements and System Elements

Table I links each task to the capability it requires and to the subsystem that provides it. The software stack uses the Robot Operating System 2 (ROS 2), with Simultaneous Localization and Mapping (SLAM) Toolbox for mapping [6] and Navigation 2 (Nav2) for planning and control [2]. Development uses laptops running the Windows Subsystem for Linux (WSL), the drivetrain uses geared direct current (DC) motors, and a 3S lithium polymer (LiPo) battery supplies power. Sensing uses a Light Detection and Ranging (LiDAR) sensor and a camera, and the classifier runs on a Raspberry Pi Artificial Intelligence Hardware Attached on Top (AI HAT) accelerator. The mapping shows that navigation underpins four of the seven tasks, which determined the order of development.

Table I

Scored tasks and supporting subsystems

Task	Required capability	Subsystem
Mapping	Build a usable map manually	RPLIDAR, SLAM Toolbox
Waste detection	Classify plastic, metal, organic	Pi Camera, AI HAT
Terrain	Climb ramps, cross gravel and sand	Drivetrain, wheels
Obstacles	Detect and avoid moving obstacles	Nav2 costmaps
Loading	Reach the correct station, secure load	Nav2, rear bin
Route decision	Wait for the gate or divert	Behaviour tree
Depositing	Identify bin colour, offload	Camera, gate servo

The robot therefore consists of eight elements: the frame, the drivetrain and steering, the offloading mechanism, the power system, the control electronics, the sensors, the mapping and navigation software, and the waste detection system.

B. Reliability versus Complexity

Eigen uses established packages instead of custom algorithms: SLAM Toolbox for mapping [6], Nav2 for planning and control [2], ros2_control for the motor interface [7], and the robot_localization extended Kalman filter for odometry fusion [8]. A dedicated STM32 microcontroller handles timing-critical loops, which keeps the motor control independent of the load on the Linux computer. The offloading mechanism uses one servo and a hinge, and gravity performs the unloading, which removes linkages that could jam.

Work was ordered by risk. Reliable mapping and navigation came first because four tasks depend on them. The offloading gate followed, and detection and decision logic were integrated last. Reports from previous Robotics Dojo teams showed that odometry from wheel encoders alone drifts on loose ground and that coiled motor cables corrupt encoder signals [12]. Eigen fuses an inertial measurement unit (IMU) with the encoders and keeps encoder wiring short and separate from motor power leads.

C. *Capability versus Robustness*

The rulebook prohibits external communication during the autonomous phase [1]. Vision processing on a remote computer is therefore unsuitable. Eigen runs the classifier on the Raspberry Pi AI HAT+, which contains a Hailo neural processing unit [11] and offloads inference from the central processing unit (CPU) that runs Nav2. This adds capability without a network dependency.

Ackermann steering was selected for the front axle, accepting a larger turning radius than differential drive in exchange for stable motion on ramps and loose surfaces, and for slower wheel wear due to lower friction. Simulation before field trials allowed navigation parameters to be tuned without risk to the hardware. Table II summarises the main capability decisions and their robustness costs.

Table II

CAPABILITY VERSUS ROBUSTNESS DECISIONS

Decision	Capability gained	Robustness cost and response
Onboard AI HAT	Classification without a network	Added heat and load; monitor temperature
Ackermann steering	Stable motion on ramps	Larger turning radius; set radius from 0.35 m to 0.50 m
Zenoh middleware	Development from WSL	Version mismatch risk; start-up checklist
STM32 and Pi split	Real-time motor control	Extra link; command time-out stops motors (see Appendix, safe stop)
Single-servo gate	Simple, light offloading	No positive latch in transit (servo holding torque keeps the gate shut)
Gazebo digital twin	Tuning before hardware existed	Simulation gap; repeat key tests on the robot

III. VEHICLE DESIGN

A. *Design Process*

The team followed six stages: Computer-aided design (CAD) and a Unified Robot Description Format (URDF) model, procurement against a bill of materials (BOM), fabrication and assembly, subsystem testing, and integration with Nav2. The URDF model of a four-wheel Ackermann rover with the LiDAR and bin was created first and used in Gazebo while the hardware was being manufactured.

Fig. 1 shows CAD screenshots of Eigen made in Autodesk Inventor, and Fig. 2 shows the assembled robot.

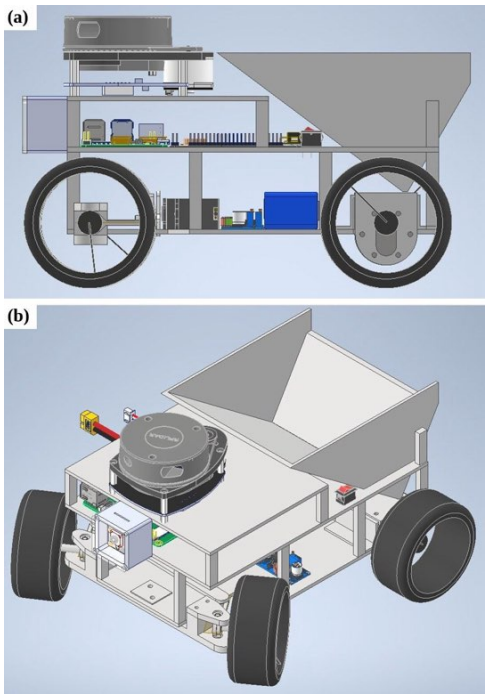


Fig. 1 *CAD of Eigen in Autodesk Inventor. (a) Side view showing the three decks, the rear bin, and the wheel mounts. (b) Isometric view showing the RPLIDAR, the front steering assembly, and the rear bin.*

Fig. 2 shows the assembled robot, as built from the CAD model.

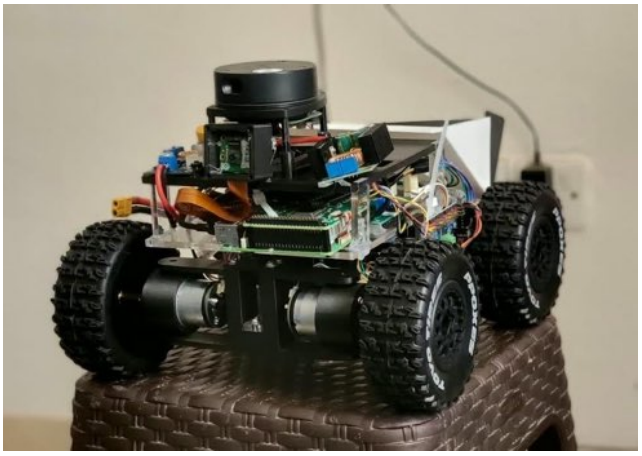


Fig. 2 *Assembled Eigen robot. The raised RPLIDAR gives an unobstructed 360° view, and the laser-cut acrylic base carries the electronics below the sensor deck.*

Table III lists the key specifications, and Fig. 3 shows the URDF model that served as the digital twin for simulation.

Table III

KEY SPECIFICATIONS OF EIGEN

Property	Value
Dimensions	30 cm x 28 cm x 21.5 cm,
Compute	Raspberry Pi 5 with AI HAT+
Low-level control	STM32F401 Black Pill
Perception	RPLIDAR, Pi Camera V2, BNO085 IMU, wheel encoders
Drive	Four geared DC motors, 112 mm wheels

Steering	Ackermann, one MG996R servo
Offloading	Hinged rear gate, one MG996R servo
Power	3S LiPo, 11.1 V, 2.2 Ah
Software	ROS 2, Zenoh, SLAM Toolbox, Nav2

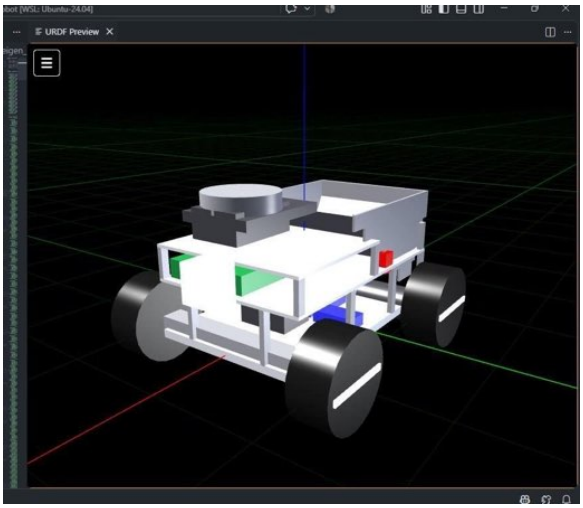


Fig. 3 *URDF model of the four-wheel Ackermann rover with the LiDAR and rear bin. The model lets navigation be tuned in Gazebo before the physical robot is complete.*

B. Mechanical Design

1) Frame

The base is a 5 mm laser-cut acrylic plate, chosen because it is inexpensive, quick to fabricate, electrically insulating, and rigid enough to avoid vibration that would degrade sensor data. Three-dimensionally printed polylactic acid (PLA) parts form the motor mounts, sensor mounts, and couplings. Printed parts allow quick revision, while the flat acrylic gives precise hole patterns. The sensor deck raises the RPLIDAR above the electronics, and the overall dimensions are 30 cm x 28 cm x 21.5 cm including the LiDAR, which satisfies the 30 cm limit [1].

Fig. 1 shows the layered layout. The heavy parts are placed on the lowest deck to keep the centre of mass low, the Raspberry Pi and control electronics sit on the middle deck, and the RPLIDAR sits on the top deck above the electronics. The rear bin is open at the top and ends in the hinged gate. Fig. 4 shows the three-dimensionally printed parts and the axle assembly, in which two geared motors are held by printed mounts and connected to the wheels through couplings.

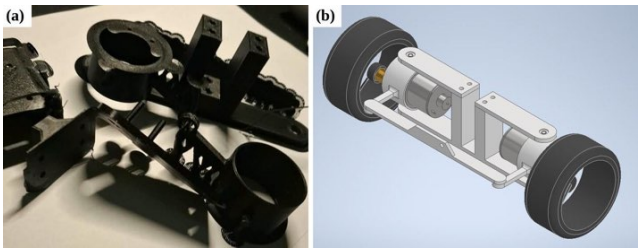


Fig. 4 *(a) 3D-printed PLA parts: motor mounts, couplings, and brackets. (b) CAD of the axle assembly with two motors, printed mounts, and linkage.*

2) Drivetrain and Wheels

Four geared DC motors with quadrature encoders drive 112 mm rubber wheels with a knobbed tread for grip on loose surfaces. The maximum linear speed follows from the wheel radius and motor speed:

$$v = 2\pi r N / 60 \qquad (1)$$

where r is the wheel radius in metres and N is the wheel speed in revolutions per minute. For Eigen with a 0.056 m radius and 111 revolutions per minute (RPM), the maximum velocity is 0.65 m/s. The ramps in the arena require torque rather than speed, so the Nav2 speed limit is set well below this value at 0.5 m/s.

3) Steering

A single MG996R servo, with a printed linkage, steers the front wheels in an Ackermann arrangement. The minimum turning radius is limited by the wheelbase and the maximum steering angle:

$$R_{\min} = L / \tan \delta_{\max} \qquad (2)$$

where L is the wheelbase and $\delta_{\max}$ is the maximum steering angle of the inner wheel [3]. With L = 0.20 m and $\delta_{\max}$ = 30°, the minimum radius is 0.35 m. Nav2 must be configured with a minimum turning radius no smaller than this value, which explains why the default configuration, written for differential drive, produced a radius that was too large during early tests.

4) Offloading Mechanism

Waste is placed on the robot from outside at the loading station, so the robot needs no manipulator. The rear bin ends in a hinged gate connected to the second MG996R servo. At the offloading zone, the servo rotates the gate downward, the gate becomes a ramp, and the waste slides out under gravity. Fig. 5 shows the CAD model of the bin, gate, and servo horn. The servo must hold the gate against the load, and the holding torque is estimated from

$$\tau = (m\, g\, l / 2) \cos \theta \qquad (3)$$

where m is the combined gate and load mass, g is gravitational acceleration, l is the gate length, and θ is the gate angle from horizontal.

The MG996R provides about 11 kg·cm at 6 V, so the design keeps the required torque below half of value .

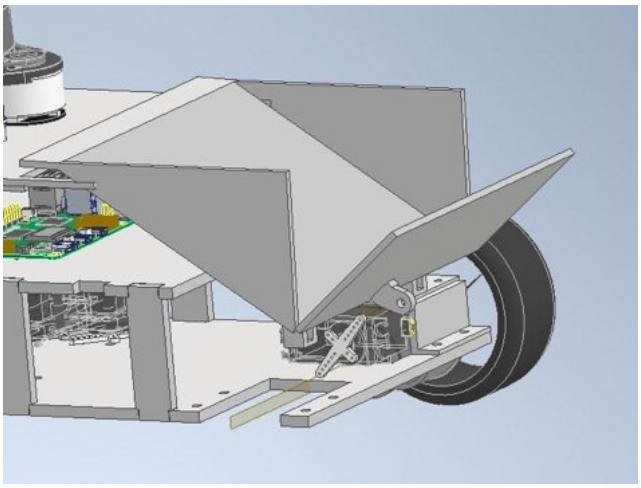


Fig. 5 *CAD view of the rear bin with the hinged offloading gate and servo. A single actuator and gravity replace a lifting mechanism, which reduces the number of failure points.*

C. Electrical and Power System

Fig. 6 shows the power architecture. A 3S LiPo battery feeds an XT60 connector, a fuse, and a double-pole master switch. The 11.1 V rail supplies the XY-160D dual-channel motor driver and two buck converters. An XL4015 converter supplies the 5 V logic rail for the Raspberry Pi 5, the AI HAT, the RPLIDAR, the Pi Camera, the STM32, and the IMU, and a second converter supplies the servo rail, which feeds the PCA9685 servo driver and the two servos. The battery energy is

$$E = V\,Q \tag{4}$$

where V is the nominal voltage and Q is the capacity. For 11.1 V and 2.2 Ah, E = 24.4 Wh per pack.

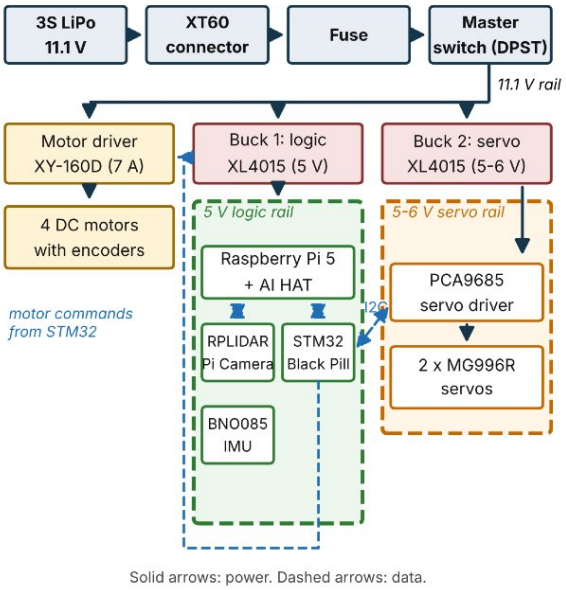


Fig. 6 *Power architecture of Eigen. Solid arrows show power and dashed arrows show data. DPST: double-pole single-throw; I2C: Inter-Integrated Circuit bus.*

The 5 V rail load was estimated from typical manufacturer values (Table IV) using

$$P = I\,V \tag{5}$$

where P is power in watts and I is current in amperes.

Table IV

ESTIMATED 5 V RAIL LOAD

Load	Power (W)	Current (A)
Raspberry Pi 5 under load	10.0	2.0
AI HAT+ accelerator	3.0	0.6
RPLIDAR scanning	2.5	0.5
Pi Camera and logic boards	2.0	0.4
MG996R (2)	13.3	2.66
Total (excluding servos)	**17.5**	**3.5**
Total (including servos)	**30.8**	6.16

The MG996R servos draw a stall current of 2.66 A each at 5 V, so the 5 V rail would be overloaded if the servos were supplied from it. This necessitated the need for an extra buck converter which powered the servos.

The circuit was first built on a breadboard and then transferred to a perforated board with Japan Solderless Terminal (JST) connectors and screw terminals (Fig. 7). The new board was designed to provide more efficient component placement and better use of space than the previous larger perfboard. Fig. 8 shows the wiring on the electronics deck. Connectors allow any module to be replaced without desoldering, which simplifies replacement and repair.

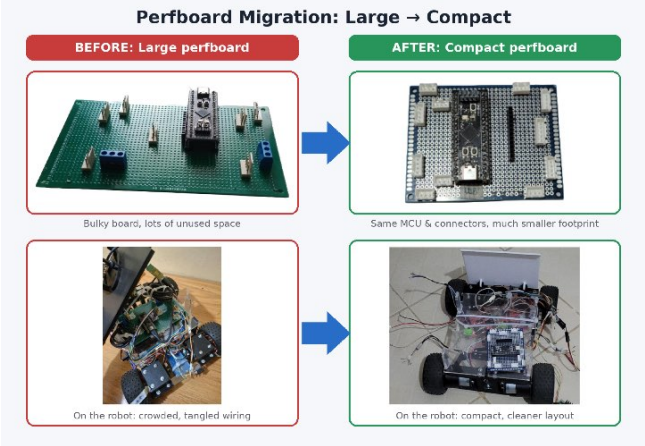


Fig. 7. *Electronics deck after the transfer from the breadboard, showing the compact perforated board installed on the robot.*

D. Control and Sensing

The STM32F401 Black Pill executes proportional-integral-derivative (PID) speed control for each wheel from encoder feedback and generates the pulse-width modulation (PWM) signals for the motor driver. It also commands the servos through a PCA9685 16-channel PWM driver. The Raspberry Pi 5 sends velocity commands and receives wheel state over a serial data link. A BNO085 nine-axis IMU provides orientation with onboard sensor fusion. The RPLIDAR provides a 360° scan for mapping and obstacle detection, and the Pi Camera V2 supplies images for classification.

E. Software Architecture

The software runs on ROS 2 with Zenoh as the middleware instead of the default Fast Data Distribution Service (Fast DDS) [10]. Zenoh allows the full stack to run from Windows through Windows Subsystem for Linux 2 (WSL2), so all team members can visualise the robot in the ROS Visualization tool (RViz) and debug it from their laptops. The team observed a positive effect on communication after the switch. Fig. 8 shows the data flow, and the next subsection describes the Zenoh connection between the robot and the laptops. The ros2_control framework [7] hosts a drive controller and the IMU broadcaster. The extended Kalman filter in robot_localization fuses wheel odometry with the IMU and publishes the filtered odometry used by SLAM Toolbox and Nav2. The twist_mux node arbitrates velocity commands between Nav2, the joystick, and a tracking input, which allows an operator to take control during the mapping phase only.

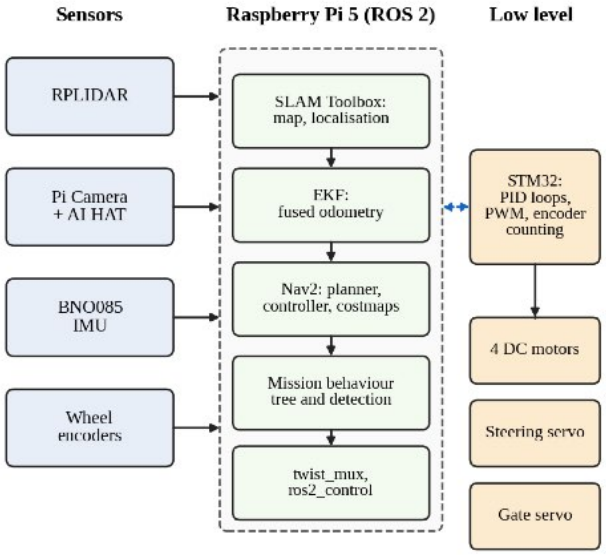


Fig. 8 *Software architecture. All planning and perception run on the Raspberry Pi 5, and only motor control and servo outputs run on the STM32, so a navigation fault cannot stop the PID loops.*

F. Zenoh Communication Architecture

Eigen runs a Zenoh router on each side of the link. When the launch script is started on the Raspberry Pi, it automatically sets up a Zenoh router on the Pi. When command_station.py is started on a laptop, it likewise sets up a router on the laptop. The two routers communicate with each other, so the topics published on the Pi are visible on the laptop and the topics published on the laptop are visible on the Pi (Fig. 9).

This lets every team member view the live map, costmaps, camera image, and topic list in RViz and send navigation goals, while planning and perception keep running on the Pi. Teleoperation is the exception: it runs on the Pi itself, for the reason given below. Both routers must run compatible Zenoh versions, which is why the start-up checklist in Table II is used.

Clock synchronisation. Each ROS 2 message carries a timestamp from the clock of the machine that sent it. The Pi and the WSL2 instance on the laptop do not share a clock, and their timestamps were found to be far apart. The problem appeared during teleoperation from the laptop: the robot moved at first and then stopped. The drive controller ignores velocity commands that are older than its command time-out (cmd_vel_timeout), and commands stamped by the laptop clock appeared stale to the Pi. Increasing cmd_vel_timeout stopped the robot from halting, but it affected navigation, because teleoperation commands were still accepted for the longer period. The team therefore runs teleoperation on the Pi itself, where all nodes share one clock. Synchronising both machines to a common time source, for example with a network time client such as chrony, removed the cause .

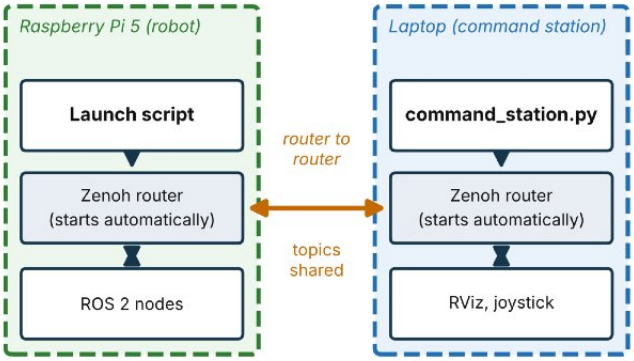


Fig. 9 *Zenoh architecture. A router starts automatically on the Pi with the launch script and on the laptop with command_station.py, and the two routers exchange topics.*

G. Mapping, Localization, and Navigation

During the 3 min mapping phase, an operator drives the robot while SLAM Toolbox builds an occupancy grid from the scans and filtered odometry [6]. A loop through the arena is driven so that loop closure can correct accumulated drift. The map is saved, together with the pose graph, and loaded in localization mode for the autonomous phase. Nav2 then plans a global path and follows it with a local controller over inflated costmaps, with recovery behaviours for blocked paths [2].

Nav2 executes navigation through behaviour trees, which describe each decision as an Extensible Markup Language (XML) structure. The route decision of the competition fits this model: at the junction, the robot checks the Path 2 gate, waits for the standard waiting time announced by the organizers [1], and otherwise diverts to Path 1. Nav2 already provides a tree that pauses near a blocked goal and waits for the obstacle to clear, which serves as the template [9].

The global planner is the Smac Hybrid-A* planner with the Reeds-Shepp motion model. It plans paths that respect the Ackermann turning limit and allow reversing, and it produced the best plans for Ackermann kinematics among

the planners tried [4]. Four local controllers were then compared (Table V). The Dynamic Window Approach controller (DWB), the default in ROS 2 Humble, was found unreliable by earlier Robotics Dojo teams and was written for differential drive only [5]. Model Predictive Path Integral (MPPI), the default in Jazzy, was judged much better by those teams and is described as state of the art in the Nav2 documentation [4], but it demands more computation and struggled around bends. Adding intermediate goals reduced its tendency to hug walls. Regulated Pure Pursuit (RPP) follows the planned path almost exactly, which was also observed in Gazebo, supports the Ackermann motion model, and needs little computation. However, the Jazzy implementation cannot set acceleration limits directly, and its switch between forward and reverse motion is abrupt enough to shift parts inside the chassis. The Nav2 velocity smoother removed the speed problem but delayed motion commands until a collision occurred. Vector Pursuit (VP) responded better than MPPI with little tuning and supports acceleration limits natively, but it left the planned path and often entered the inscribed obstacle zone, and it also reversed abruptly. MPPI was selected for the eigen robot.

Table V

LOCAL CONTROLLER COMPARISON

Controller	Strengths	Problems found
DWB	Default in Humble	Flaky; differential drive only
MPPI	Strong plans; default in Jazzy	High compute; weak on bends; wall hugging
RPP	Tracks path closely; Ackermann; low compute	No acceleration limits; abrupt reversing; smoother adds lag
VP	Responsive; acceleration limits	Leaves path; abrupt reversing

H. Drive Modes: Ackermann and Differential

Eigen supports two drive modes. In Ackermann mode, the front servo steers the front wheels, and the Ackermann steering controller from ros2_controllers [7] converts the velocity commands from Nav2 into a steering angle and wheel speeds, with the minimum turning radius set from (2). In differential-drive mode, the robot turns through the speed difference between the left and right wheels, as assumed by the default Nav2 configuration.

The mode is chosen with an argument when the launch script is started on the Raspberry Pi, so the team can pivot between Ackermann and differential drive at launch. Ackermann mode was selected for the competition because of its stable motion on ramps and loose surfaces, and differential-drive mode remains available when a smaller turning radius matters more (Section II-C).

I. Waste Detection and Mission Logic

A You Only Look Once (YOLO) object detector runs on the Hailo accelerator of the AI HAT+ [11] and returns a bounding box, a class identifier, and a confidence score for each object in the camera image. Class 0 corresponds to metal, class 1 to organic waste, and class 2 to plastic. The robot is designed to accept a result only after consecutive agreeing frames, which reduces the effect of single misclassifications. Bin colour identification uses hue-saturation-value (HSV) thresholding in OpenCV and is planned for integration after the classifier. Fig. 10 shows the full mission logic, including the gate decision.

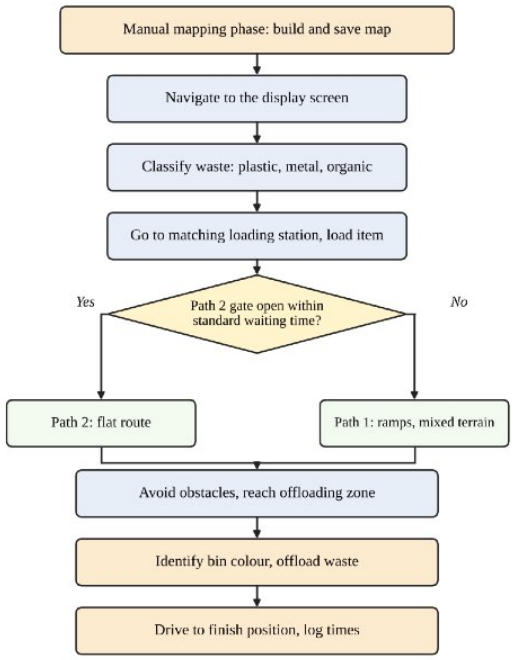


Fig. 10 *Autonomous mission logic of Eigen, including the gate decision.*

A mission node logs a timestamp for each required event: start, detection, loading, terrain, obstacle response, gate decision, gate waiting, depositing, finish, and total time [1]. The node is also designed to disable joystick input at the start of the autonomous phase, so that no manual command reaches the drive controller.

J. Lessons Learned

- **Power first.** The 5 V converter was sized for typical loads, and the margin proved too small. The power budget should be measured before the integration of new loads.
- **Check supplies early.** The supplied camera was faulty, the acrylic ran out, and the filament did not match the printer. Each caused delay, and a spare critical sensor and an early materials check would have avoided them.
- **Bolts instead of glue.** The steering linkage detached at one end and was glued back. Earlier teams also advise bolted joints because they can be adjusted and removed [13].
- **Simulate first.** The Gazebo model allowed Nav2 and SLAM to be tuned before the robot was complete.
- **Match software versions.** Mixed ROS 2 distributions or middleware versions on the laptop and the Pi can prevent discovery, so a start-up checklist is used.

- **Check the clocks and the command source.** The clocks of the Pi and the WSL2 laptop were far apart, so teleoperation from the laptop stopped after a short movement. Raising cmd_vel_timeout hid the symptom but let teleoperation commands affect navigation. Running teleoperation on the Pi removed the problem, and the clocks should be compared at every start-up.
- **Configure for the real kinematics.** Navigation defaults assume differential drive, and the first indoor trials produced a turning radius that was too large for the Ackermann chassis. Limits such as the minimum turning radius should be set from the vehicle geometry before tuning begins.
- **Use removable connectors.** Moving from the breadboard to a more compact perforated board with JST connectors and screw terminals improved component placement and freed usable space compared with the previous larger board. A failed module can now be replaced in minutes instead of being desoldered.
- **Repeat key tests on the robot.** Gazebo allowed early tuning, but a simulation gap remains, so the planner, the controller, and the map quality must be confirmed on the physical platform.

IV. Experimental Results

A. Testing Method

Testing proceeded from unit tests of individual parts, to integration tests of the combined platform, to simulation, and finally to field trials; Fig. 11 shows some of the initial test images. The motors, encoders, and steering servo were exercised on the bench. The software was tested in Gazebo with a model of the gamefield and then on the robot in a house corridor and rooms. Quantitative acceptance tests for navigation success rate, offloading success rate, and classification accuracy were scheduled before the competition.

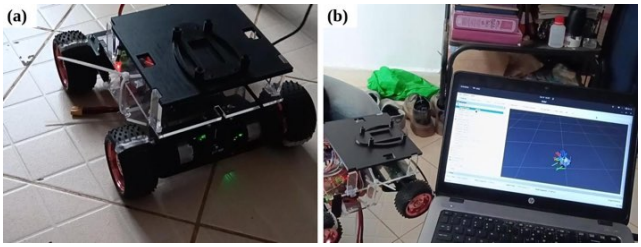


Fig. 11 *Initial tests of Eigen. (a) Early test of the assembled chassis on the floor. (b) IMU orientation shown live in RViz on a laptop, which confirmed the sensor and the Zenoh link before navigation work began.*

B. Results

Table VI

Test results

Test	Outcome
Wheel PID	Robot drives in a straight line error in cm over 2 m
Ackermann steering	Turns left and right.
IMU integration	Orientation visualised live in RViz
Gazebo SLAM	Occupancy map of the gamefield produced (Fig. 12)
Nav2 on robot	Navigation completed with a fixed steering angle (Fig. 13)
Camera	Live compressed image stream available to ROS 2
Zenoh middleware	Full topic list visible from a laptop through WSL
Waste detection	YOLO model on the AI HAT+ detected organic and plastic items with confidence 0.57–0.78; one false positive (Fig. 14)
Offloading gate	Modelled and fabricated; mounting and offloading trials

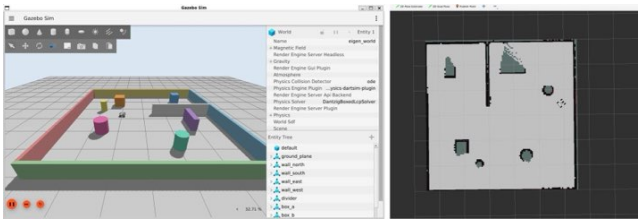


Fig. 12 *(a) Gazebo gamefield with walls, divider, boxes, and cylinders. (b) Occupancy map built by SLAM Toolbox. The map shows that scan matching resolves the walls and the obstacles, which is the prerequisite for the navigation tasks.*

The simulated map in Fig. 13 contains the outer walls, the divider, and the obstacles at their correct positions, which supports the use of the digital twin for tuning. It also shows the Nav2 costmap on the physical robot, with inflated obstacle regions around the walls and furniture, together with the live camera view. Navigation was successful.

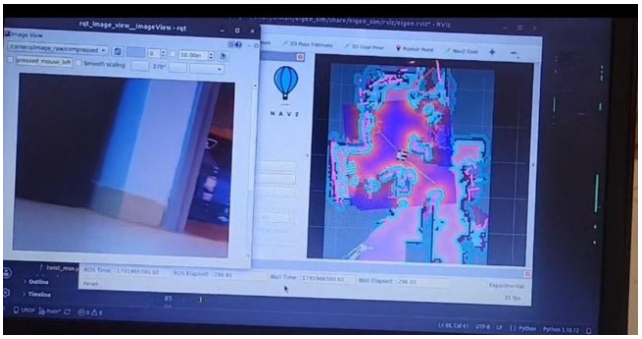


Fig. 13 *RViz view during the indoor trial: the Nav2 costmap with inflated obstacles (right) and the live camera feed (left). The figure confirms that mapping, localization, and perception run together on the robot.*

An informal test used real objects and images shown on a phone screen, which resembles the display screen of the competition. Fig. 14 shows the results. Seven images produced ten detections. Nine were correct, with confidence between 0.57 and 0.78 (mean 0.68), and one was a false positive on a finger (class 1, confidence 0.70). Table VII summarises the test. Metal waste (class 0) was also teste .

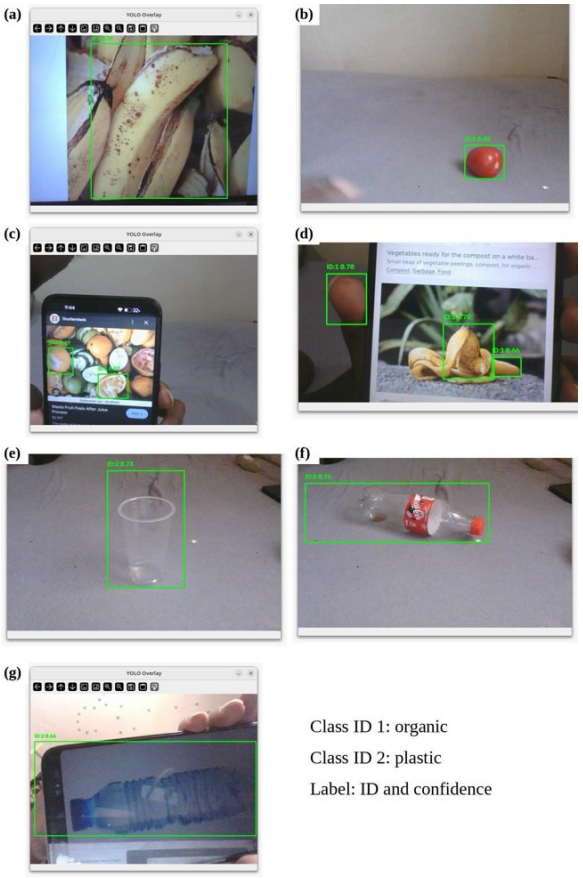


Fig. 14 *Detection tests with the YOLO model on the AI HAT+. (a) Banana peel, 0.78. (b) Tomato, 0.60. (c) Orange peels on a phone, 0.69 and 0.57. (d) Banana peel on a phone, 0.70 and 0.66, with a false positive on a finger, 0.70. (e) Plastic cup, 0.73. (f) Plastic bottle, 0.76. (g) Bottle on a phone, 0.66. Real objects and screen images were both detected.*

Table VII

INFORMAL DETECTION TEST

Class	Detections	Confidence
Organic (1)	6 correct	0.57–0.78
Plastic (2)	3 correct	0.66–0.76
False positive	1 (finger)	0.70
Metal (0)	3 correct	0.67- 0.75

The false positive has a higher confidence than the average correct detection, so a confidence threshold alone cannot remove it. Eigen therefore restricts the search to the region of the display screen and requires several consecutive agreeing frames before it accepts a class. A larger test set with a count per class and a confusion matrix is planned before the competition.

C. Map of the Gamefield and Its Improvement

Fig. 12 shows the Gazebo gamefield and the map built from it, which served as the reference while the robot was being completed. Fig. 15 shows the map of the gazebo gamefield of the competition that the robot has been producing.

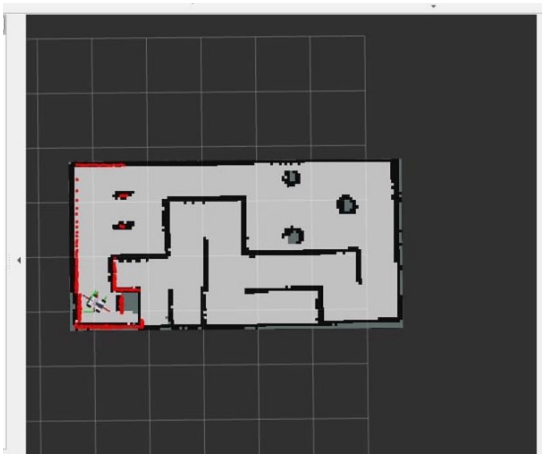


Fig. 15 *Occupancy map of the gazebo gamefield built by SLAM Toolbox on the robot. Walls, inner dividers, and obstacles are mapped, which shows that the map is good enough for navigation.*

The map of the physical gamefield was improved in stages (Fig. 16). In the first map, stray scan rays extend far beyond the arena and the walls are doubled and uneven. In later maps the outline closes, the inner walls become straight, and the obstacles appear as separate marks. The main cause of distortion identified in Table VIII is pose drift from wheel slip on loose ground. The measures against it are the fusion of the IMU with the wheel encoders in the extended Kalman filter, scan matching in SLAM Toolbox, and driving a loop through the arena so that loop closure can correct accumulated drift.

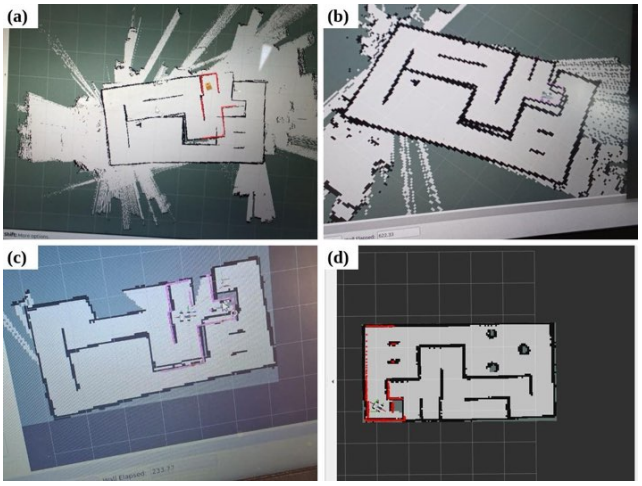


Fig. 16 *Evolution of the gamefield map from (a) the first map to (d) the latest. Stray scan rays outside the arena disappear and the walls become straight and closed, which shows the effect of the measures in the text.*

D. Reliability and Robustness Analysis

The team analysed the failure modes observed during development and those expected in the competition. Table VIII lists the most important ones, and Table VI summarises the results obtained so far. The thermal limit of the 5 V converter is the clearest case: the converter became hot, and the RPLIDAR then failed to launch until the converter had cooled. Power cycling after a few minutes restored normal operation, which confirms a temperature-related current

limit. The converter output leads were also thin, which added resistance and heat. The corrective actions are XT60 connectors, thicker leads, airflow, and a reduced load on this rail where a second buck converter was added. A second case is the clock offset between the Pi and the WSL2 laptop, which stopped teleoperation from the laptop (Section III-F).

Table VIII

Failure mode analysis

Failure mode	Effect	Mitigation
5 V converter overheats	LiDAR fails to start	XT60 connectors, thicker leads, airflow, and a second converter, which fully mitigated the overheating
Faulty Camera	Camera was nonoperational	Spare camera was used, checked-up at start.
Steering linkage detaches	Loss of steering	Replace glue with bolts
Wheel slip on loose ground	Pose drift	IMU fusion, scan matching
Clock offset between the Pi and WSL2	Teleoperation stops after a short movement	Run teleoperation on the Pi; compare clocks at start-up
Joystick active in autonomy	Rule violation	Disable joystick input at start

Table IX lists the acceptance tests planned before the competition.

Table IX

Planned acceptance tests

Test	Metric	Target
Mapping	Field area covered	95 %
Goal navigation	Success over 10 runs	90 %
Waste classification	Accuracy per class	90 %
Gate decision	Correct decisions	10 of 10
Offloading	Successful offloads	9 of 10
Full autonomous run	Time to finish	Under 180 s

Each test in Table IX will be run on the physical robot in the gamefield and judged against its target, and the mission-node timestamps will supply the times.

- **Mapping.** The operator drives the arena once within the 3 min mapping phase. The saved occupancy grid is compared with the known gamefield layout, and coverage is the share of the field area that is correctly mapped. The target is 95 %.
- **Goal navigation.** From the saved map, the robot is sent to a goal in localization mode. A run succeeds if it reaches the goal without contact and without operator input. The target is 9 successes in 10 runs, with the time of each run recorded.
- **Waste classification.** Organic, plastic, and metal samples are shown on the display screen under different lighting, and a result counts only after the consecutive-frame check. Accuracy is the share of correct results per class, with a confusion matrix, and the target is 90 % for each class.
- **Gate decision.** The junction is tested with the Path 2 gate closed and open. The decision is correct when the robot waits for the standard waiting time and then uses Path 2 if it opens, and diverts to Path 1 otherwise. The target is 10 correct decisions in 10.
- **Offloading.** With the bin loaded, the robot offloads at the colour-coded zone, and a trial succeeds when the load leaves the bin without jamming. The target is 9 successful offloads in 10.
- **Full autonomous run.** The whole autonomous phase is run from start to finish, and the total time comes from the mission-node log. The target is under 180 s, which is the length of the autonomous phase.

Remaining validation covers ramp climbing, gravel and sand traversal, moving-obstacle response, the gate decision, offloading repeatability, and a full timed run. Each test will record success rate and time, and the results will update this paper.

V. Acknowledgements

Team Eigen thanks the Robotics Dojo organisers and mentors for the training sessions, the Raspberry Pi 5 and sensor kit, and access to laser cutting and 3D printing.

VI. References

[1] Robotics Dojo, “Robotics Dojo 2026: Arena rules and regulations,” ver. 0.9, Jomo Kenyatta Univ. Agric. Technol., Juja, Kenya, 2026.

[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] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

[4] S. Macenski, T. Moore, D. V. Lu, A. Merzlyakov, and M. Ferguson, “From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2,” *Robot. Auton. Syst.*, vol. 168, Art. no. 104493, 2023.

[5] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

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

[7] ros2_control Development Team, “ros2_control documentation.” [Online]. Available: https://control.ros.org

[8] T. Moore and D. Stouch, “A generalized extended Kalman filter implementation for the Robot Operating System,” in *Proc. 13th Int. Conf. Intell. Auton. Syst. (IAS-13)*, Padua, Italy, 2014.

[9] Open Navigation LLC, “Nav2 documentation: Behavior trees.” [Online]. Available: https://docs.nav2.org

[10] Open Robotics, “rmw_zenoh: ROS 2 middleware based on Zenoh.” [Online]. Available: https://github.com/ros2/rmw_zenoh

[11] Raspberry Pi Ltd., “Raspberry Pi AI HAT+ documentation.” [Online]. Available: https://raspberrypi.com/documentation/accessories/ai-hat-plus.html

VII. Appendix—Situational Awareness

Users trust an autonomous robot when they can see what it is doing and why. Eigen supports this in the following ways.

- **Autonomy statement.** Operators are told that the robot drives itself after the mapping phase, and that the only human actions are loading the waste and pressing the stop control.
- **Visible state.** The live map, planned path, costmaps, camera image, and the classified waste type are displayed in RViz, so an observer can confirm that the robot understands its surroundings and its task.
- **Event log.** The timestamps of detection, loading, gate decision, offloading, and finish are recorded, so each run can be audited and compared.
- **Roles.** The operator loads the correct item at the loading station and keeps clear of the path. The robot handles navigation, route choice, and offloading.
- **Limits.** The robot has been validated on a limited set of waste items and conditions. Unusual lighting, items outside the three classes, and unexpected obstacles may cause errors, and a safe stop is preferred over a guess.
- **Safe stop.** A master switch and a fuse remove power immediately, and the drive controller stops when commands cease.
- **Honest framing.** The robot is a machine that follows its design. It does not reason beyond its programmed behaviours and has no intent.