Malkiabots Technical Paper

Robotics Dojo Competition 2026

Mercy Murimi, Joy Oluga, Lyn Kamau, Jordana Owilla, Melisa Kinda

ABSTRACT

This paper presents an autonomous waste management robot developed for the 2026 Robotics Dojo Competition, whose theme is waste management and off-loading. The robot is built to identify individual pieces of waste, classify each as organic, plastic, or metal, and deliver it to the off-loading zone for its type. Its structure, electronics, and software are designed to work together as one system, built to perform dependably within the constraints of the predetermined game field. The software runs on ROS2 Jazzy, which lets the perception, classification, planning, and control nodes exchange data as independent, interchangeable modules. An RPLiDAR scanner feeds SLAM, which builds a map of the field and tracks the robot's position within it. Nav2 then plans and carries out obstacle-free routes between the waste and the off-loading zones, so the robot can travel entirely on its own. Development favored fast iteration and catching problems early. Before any hardware was committed, the robot was modeled and simulated in Gazebo, with RViz used to display sensor data and the robot's state. These simulations tested its motion, its navigation strategy, and the full detect, classify, collect, and off-load sequence, helping the team find and fix design flaws early and move toward a working build sooner.

KEYWORDS

ROS 2 Jazzy, SLAM, Nav2, behaviour trees, waste classification

I. INTRODUCTION

Waste management is shifting from manual handling toward automated, data-driven operations. Robots now sort material at recycling facilities and move it between processing stages. This reduces worker exposure to hazardous waste, keeps sorting consistent over long shifts, and produces data on what is being discarded [1], [2]. For the sector, this means higher recovery rates, lower handling costs, and operations that can scale with growing urban waste volumes. These gains depend on one capability: the robot must find an item and decide what it is. Detection and classification is therefore the central challenge of this work. Waste varies in shape, colour, and texture, is often partly hidden or soiled, and is seen under uneven lighting. A single misclassified item contaminates the stream it is placed in.

This year's Robotics Dojo Competition takes waste management and off-loading as its theme. The robot must be fully autonomous and fit strictly within a $300 \times 300 \times 300$ mm envelope. It must cross different types of terrain while carrying a piece of waste, classify that waste as organic, plastic, or metal, and off-load it at the zone designated for its type. The waste items vary in shape, fit within about 8 cm, and weigh up to 150 g.

Each component has a defined role in meeting these requirements. A Raspberry Pi 5 [3] is the main computer and runs the task logic, mapping, and navigation. A Raspberry Pi Camera Module v2 [4] supplies the images, which the Pi streams over Wi-Fi to a laptop that runs the classification model. An RPLiDAR [5], [6] scans the field for mapping, localisation, and obstacle avoidance. An MPU6050 IMU [7] reports orientation. Four JGB37-520 geared DC motors with encoders and a 65 mm wheel kit [8] give the traction to cross uneven ground under load, and their encoders provide wheel odometry. Two L298 dual H-bridge modules [9] drive the motors. An Arduino Nano [10] handles real-time motor, encoder, and servo control. Four 18650 Li-ion cells [11] power the drive motors, and a 27 W power bank powers the Raspberry Pi 5 and the devices on its USB ports. A servo [12] opens the off-loading gate.

ROS 2 Jazzy [13] connects these components as independent nodes. A behaviour-tree model [14] runs detection and classification. It sequences the steps of search, detect, classify, select the zone, transport, and off-load. It also provides fallbacks, such as re-scanning when classification confidence is low. Two interfaces drive the robot. Teleoperation publishes velocity commands from a keyboard or joystick and is used for commissioning and manual control. For autonomous navigation, SLAM [15], [16] builds a map from the RPLiDAR scans and estimates the robot's pose in it. Nav2 [17]–[19] then plans a global route across a costmap and follows it with a local controller that avoids obstacles. Its own behaviour tree runs recovery actions when the robot is blocked. Both interfaces send the same velocity commands to the Arduino Nano, so control can pass between them without changes to the lower layers. The robot is first simulated in Gazebo and visualised in RViz on a model of the game field, before any hardware is committed.

For off-loading we chose a 3D-printed ramp inclined at 43.5° and closed by a 38 g detachable gate. At the zone, the servo lifts the gate and gravity slides the waste off. This needs one actuator and little power, and it fits within the size limit.

This work makes the following contributions:

(1) A compact autonomous waste-handling robot that fits within a 300 mm cube and is designed to travel over varied terrain under load.

(2) A behaviour-tree-based detection and classification pipeline, running on a laptop CPU fed by a compressed camera stream from the robot, that sorts waste into organic, plastic, and metal categories.

(3) A ROS 2 Jazzy navigation setup that combines teleoperation with SLAM and Nav2 for autonomous travel to the correct off-loading zone.

(4) A ramp-and-gate off-loading mechanism whose geom-

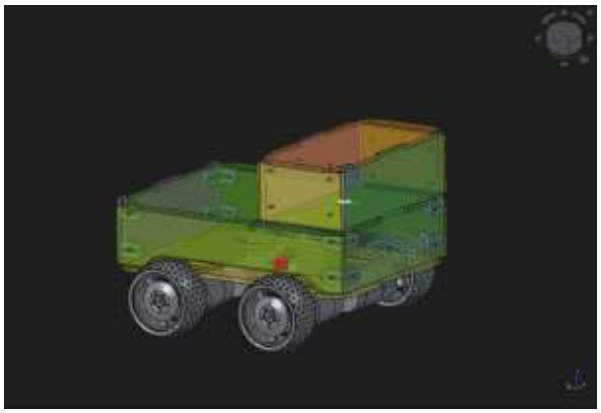


FIGURE 1. CAD VIEW OF THE ROBOT CHASSIS AND CARGO BED.

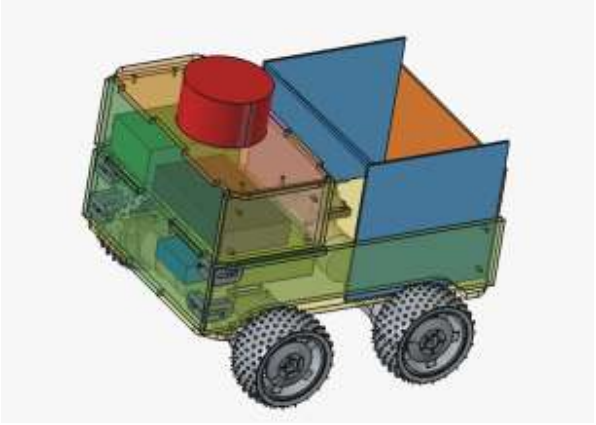


FIGURE 2. CAD MODEL SHOWING THE SENSOR, ELECTRONICS, BATTERY, RAMP, AND GATE.

etry and actuator are justified by calculation.

II. RELATED WORK

Robots are already used in the waste industry. Commercial sorting systems from companies such as ZenRobotics and AMP Robotics pair cameras and machine learning with robotic pickers that remove items from a moving conveyor [1], [2]. These systems are large, fixed in place, and tied to facility infrastructure. Research on waste classification has produced public image datasets such as TrashNet and TACO, and classifiers trained on them [20], [21]. Most of this work classifies images and says little about how a small robot applies a classifier while moving through a field.

On the navigation side, ROS underpins research and education platforms such as the TurtleBot series [22], [23]. Nav2 provides a modular stack for planning, control, and recovery behaviours [18], [24]. Behaviour trees, borrowed from game development, are increasingly used to structure robot tasks. They are easy to read and extend, and they make recovery from a failed step explicit [14]. Competition robots such as those in RoboCup and Eurobot show how perception, planning, and actuation can be combined under tight size and time limits [25], [26].

Two gaps remain. Sorting robots are stationary and expensive, and most mobile waste platforms are larger vehicles built to collect rather than sort. Few designs combine detection, classification, travel over uneven terrain, and sorted off-loading in a small, low-cost, fully autonomous robot. Our design addresses this gap with a four-wheel-drive base, LiDAR-based navigation, behaviour-tree task control, and a ramp-and-gate mechanism, all within the 300 mm limit of the competition.

III. DESIGN STRATEGY

The design followed from the competition rules. The robot must fit in a 300 mm cube, cross varied terrain, classify waste, and deliver it to the correct zone. This called for a compact layout, a low centre of mass, and few moving parts. Figure 2 shows the resulting layout.

The robot is built on a 200 $\times$ 270 mm base plate in three levels. The base holds the four drive motors and the Li-ion pack. A middle level above the front half holds the Raspberry Pi 5 and the power bank. An upper deck at 130 mm carries the RPLiDAR, which sits at the highest point (176 mm above the base plate) so its scan is not blocked by the body. The rear half of the robot is an open bay that holds the ramp. The structure is made of laser-cut panels with interlocking tabs and slots, which allows quick assembly and easy part replacement. With its wheels, the robot measures about 254 $\times$ 271 $\times$ 228 mm, inside the 300 mm cube with margins of 46, 29, and 72 mm.

Locomotion uses four JGB37-520 motors [8], one per wheel, with large treaded 65 mm wheels. Section IV-A shows that the motors provide about 4.5 times the force needed to climb a 15° slope, and that tyre grip, not motor torque, is the limiting factor. The battery sits low in the base to keep the centre of mass low.

The off-loading ramp is a 3D-printed plate inclined at 43.5° that runs from the upper deck level down to the base plate across the open rear bay. A 38 g detachable gate closes its lower end (Figure 4). The steep angle makes the waste slide off for any friction coefficient below $\tan 43.5^\circ = 0.95$. The gate is light enough that a servo delivering only 0.29 kg·cm can lift it (Section IV-A).

The robot senses its surroundings in layers. The RPLiDAR provides the scans for SLAM and mapping [15], [16]. Wheel encoders give odometry, and the IMU gives orientation. Together they keep the pose estimate reliable when wheels slip on rough ground. The Pi Camera captures the waste, and a laptop that receives the camera stream over Wi-Fi classifies it.

Computation is split into two tiers. The Raspberry Pi 5 runs the behaviour tree, SLAM, and Nav2, and streams the camera feed to a laptop that runs the classifier. The Arduino Nano handles real-time actuation of the motors and servo and reads the encoders. The two are linked by a USB serial connection. Power is also split. The Li-ion pack feeds the drive motors through two L298 modules, and a 27 W power bank feeds the Raspberry Pi 5 and its USB devices. No voltage converter is needed (Section IV-B).

Before any hardware was built, the robot was described as a URDF [27], [28] and tested in Gazebo on a model of the game field [29]–[31], with RViz used to inspect sensor data

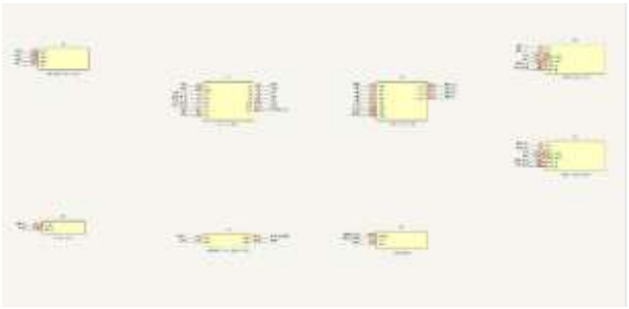


FIGURE 3. WIRING SCHEMATIC OF THE ROBOT.

and the robot's state [32]. This allowed the navigation setup and the detect–classify–transport–off-load sequence to be checked early.

Figure 3 shows the wiring schematic and Table 1 lists every connection in the robot, grouped by component. Driver 1 runs the front pair of motors and Driver 2 runs the rear pair. Both drivers take the same control lines from the Nano, so the robot is driven as a left side and a right side: D5 sets the power of the left motors and D6 the power of the right motors, while D9 and D10 set the left direction and A0 and A1 the right direction. All grounds are joined to one common reference.

Table 1. Wiring connections of the robot

Component	Pin / wire	Connects to
Arduino Nano		
Nano	D0 (RX)	USB serial, not used for anything else
Nano	D1 (TX)	USB serial, not used for anything else
Nano	D2	Front-left encoder, channel A
Nano	D3	Front-right encoder, channel A
Nano	D4	Front-left encoder, channel B
Nano	D5 (PWM)	ENA of Driver 1 and ENA of Driver 2
Nano	D6 (PWM)	ENB of Driver 1 and ENB of Driver 2
Nano	D7	Front-right encoder, channel B
Nano	D8	Rear-left encoder, channel A
Nano	D9	IN1 of Driver 1 and IN1 of Driver 2
Nano	D10	IN2 of Driver 1 and IN2 of Driver 2
Nano	D11	Gate servo signal
Nano	D12	Rear-right encoder, channel A
Nano	D13	Not connected
Nano	A0	IN3 of Driver 1 and IN3 of Driver 2
Nano	A1	IN4 of Driver 1 and IN4 of Driver 2
Nano	A2	Rear-left encoder, channel B
Nano	A3	Rear-right encoder, channel B
Nano	A4	IMU SDA
Nano	A5	IMU SCL
Nano	A6, A7	Not connected
Nano	5V	Driver 2 logic +5V, encoder VCC (all four), IMU VCC, servo power
Nano	GND	Common ground
Nano	VIN	Not connected
Nano	USB	Raspberry Pi USB port
Driver 1 (front): L298N		
Driver 1	ENA	Nano D5
Driver 1	IN1	Nano D9
Driver 1	IN2	Nano D10
Driver 1	IN3	Nano A0
Driver 1	IN4	Nano A1
Driver 1	ENB	Nano D6
Driver 1	+12V (VS)	Battery plus
Driver 1	GND	Common ground
Driver 1	OUT1, OUT2	Front-left motor
Driver 1	OUT3, OUT4	Front-right motor
Driver 2 (rear): 7 A dual H-bridge		
Driver 2	ENA	Nano D5
Driver 2	IN1	Nano D9
Driver 2	IN2	Nano D10
Driver 2	IN3	Nano A0
Driver 2	IN4	Nano A1
Driver 2	ENB	Nano D6
Driver 2	Logic +5V	Nano 5V
Driver 2	Logic GND	Nano GND
Driver 2	9-24V	Battery plus
Driver 2	PGND	Battery minus
Driver 2	OUT1, OUT2	Rear-left motor
Driver 2	OUT3, OUT4	Rear-right motor
Encoders		
Front-left encoder	A / B	Nano D2 / D4
Front-left encoder	VCC / GND	Nano 5V / Nano GND
Front-right encoder	A / B	Nano D3 / D7
Front-right encoder	VCC / GND	Nano 5V / Nano GND
Rear-left encoder	A / B	Nano D8 / A2
Rear-left encoder	VCC / GND	Nano 5V / Nano GND
Rear-right encoder	A / B	Nano D12 / A3
Rear-right encoder	VCC / GND	Nano 5V / Nano GND
IMU (MPU6050 on GY-521)		
IMU	VCC	Nano 5V
IMU	GND	Nano GND
IMU	SDA	Nano A4
IMU	SCL	Nano A5
IMU	XDA, XCL, AD0, INT	Not connected
Gate servo		
Servo	Signal	Nano D11
Servo	Power (red)	5V
Servo	Ground (brown / black)	Common ground
Battery and power		
Battery	Plus	Driver 1 +12V, Driver 2 9-24V
Battery	Minus	Driver 1 GND, Driver 2 PGND, common ground
Raspberry Pi 5		
Raspberry Pi 5	USB port 1	Arduino Nano (USB)
Raspberry Pi 5	USB port 2	Lidar (USB)
Raspberry Pi 5	USB port 3	Camera (USB)
Raspberry Pi 5	Power input (USB-C)	Power bank
Common ground (all joined)		
Common ground	Joined points	Battery minus, Nano GND, Driver 1 GND, Driver 2 PGND and logic GND, all four encoder GND, IMU GND, servo ground

IV. VEHICLE DESIGN

A. Mechanical Design

Structure. The values in this section are read from the CAD model. The base plate is 200 × 270 × 5 mm. A middle plate sits at 65 mm above the front half, and an upper deck at 130 mm carries the RPLiDAR. The side panels are 5 mm thick. The wheelbase is 130 mm and the track is 217 mm between wheel centres. The axles sit 19.8 mm below the base plate, so with 65 mm wheels the base plate is 52 mm above the ground. The robot measures about 254 × 271 × 228 mm including the wheels and the RPLiDAR.

Locomotion. Each wheel is driven by a JGB37-520 12 V 110 RPM motor with encoder and a 65 mm wheel [8]. Pixel Electric lists 10 kg·cm of torque, 6–110 RPM, 0.45–1 A, and 7–15 W per motor [8]. Other suppliers list the same motor with a rated torque of 3.9 kg·cm, a stall torque of

12 kg·cm, and a stall current of 2.4 A. All calculations below use the lower rated value of 3.9 kg·cm. The wheel radius is $r = 0.0325$ m.

Top speed at the wheel ($n = 110$ RPM):

$$v = \pi dn/60 = \pi(0.065)(110)/60 = 0.37 \text{ m/s} \quad (1)$$

Tractive force per wheel at rated torque ($\tau = 3.9$ kg·cm $= 0.382$ N·m). Four wheels give $4 \times 11.8 = 47.1$ N:

$$\begin{aligned} F_w &= \tau / r \\ &= 0.382/0.0325 = 11.8 \text{ N} \end{aligned} \quad (2)$$

Required force on a slope. The CAD-based mass estimate is 2.82 kg for the robot and 2.97 kg with a 150 g payload (plate masses from CAD volumes at 1.18 g/cm^3 for acrylic, 3D-printed parts at the gate's density, supplier masses for the motors, and assumed masses for the electronics). We use a design mass $m = 3.0$ kg. The slope is $\theta = 15^\circ$ and the rolling-resistance coefficient is $C_{rr} = 0.10$ for rough ground:

$$\begin{aligned} F_{\text{req}} &= mg(\sin\theta + C_{rr}\cos\theta) \\ &= 3.0(9.81)(0.259 + 0.097) = 10.5 \text{ N} \end{aligned} \quad (3)$$

This is 2.6 N per wheel, or about 0.87 kg·cm per motor. The force margin is $47.1/10.5 = 4.5$. The grip limit, with a friction coefficient $\mu = 0.6$ between tyre and ground:

$$F_{\max} = \mu mg\cos\theta = 0.6(29.4)(0.966) = 17.1 \text{ N} \quad (4)$$

Since $10.5 \text{ N} < 17.1 \text{ N} < 47.1$ N, tyre grip is the limit, not motor torque. This is why the design uses large treaded wheels. Skid steering needs extra force to turn on high-friction ground, and the force margin leaves room for this.

Stability. The tip-over angle is $\theta_{\text{tip}} = \arctan(b/h)$, where b is the horizontal distance from the centre of mass to the pivot axle and h is the centre-of-mass height above the ground. The CAD-based estimate puts the centre of mass 88 mm above the ground and 29 mm behind the front axle, because the electronics and battery sit in the front half. When descending nose-first, the robot tips about the front axle at $\theta_{\text{tip}} = \arctan(29/88) = 18^\circ$ (22° with the payload on the ramp). This is only slightly above the 15° design slope, so steep descents should be avoided. When climbing, the robot tips about the rear axle at 49°, and sideways at 51°.

$$\theta_{\text{tip}} = \arctan(b/h) \quad (5)$$

Off-loading ramp. The ramp is a 3D-printed 2 mm plate with side supports and a separate detachable gate (Figure 4). From the CAD model, the ramp is 165 mm wide and its plate is 174 mm long along the slope. It drops 120 mm over a 126 mm run, from the upper deck level (130 mm) to the base plate. The slope measured from the plate's face is 43.55°, which we round to $\alpha = 43.5^\circ$. The gate measures $165 \times 100 \times 2$ mm (33 cm^3) and weighs 38 g. The waste slides when $\tan\alpha > \mu_s$, where μ_s is the friction coefficient between waste and ramp:

$$\tan 43.5^\circ = 0.95 > \mu_s \quad (6)$$

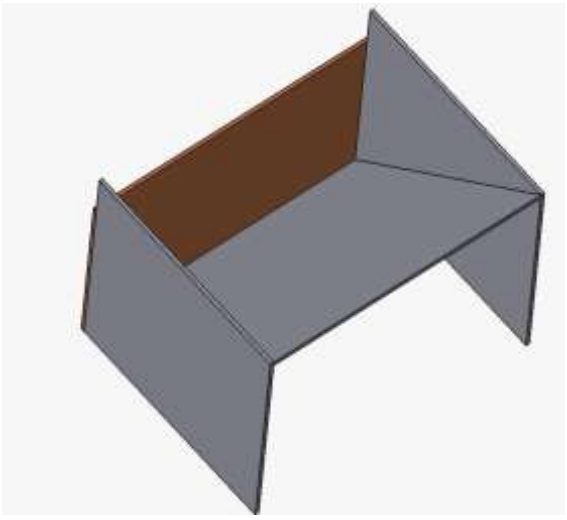


FIGURE 4. RAMP DESIGN WITH DETACHABLE GATE, INCLINED AT 43.5°.

For a typical μ_s of up to 0.5, the minimum angle is $\arctan(0.5) = 26.6^\circ$. The 43.5° ramp has a margin of $0.95/0.5 = 1.9$. At $\mu_s = 0.5$ the acceleration down the ramp is $a = g(\sin\alpha - \mu_s\cos\alpha) = 9.81(0.689 - 0.363) = 3.2$ m/s^2. An item therefore clears the 80 mm gate opening in $t = \sqrt{2 \times 0.08/3.2} \approx 0.22$ s.

The servo lifts the gate against its own weight and the friction from the waste pressing on it. In the worst case, with no friction on the ramp, the waste presses on the gate with $m_w g \sin\alpha = 0.15(9.81)(0.689) = 1.01$ N. The friction coefficient at the gate is assumed to be $\mu_g = 0.3$:

$$\begin{aligned} F_{\text{lift}} &= m_g g + \mu_g m_w g \sin\alpha \\ &= 0.038(9.81) + 0.3(1.01) = 0.68 \text{ N} \end{aligned} \quad (7)$$

The largest item is about 80 mm, so the gate must lift at least 85 mm to let it pass. With a crank-and-link mechanism turning through 180°, the lift is $2r_c$, so the crank radius is $r_c = 42.5$ mm:

$$\begin{aligned} \tau_{\text{servo}} &= F_{\text{lift}} \cdot r_c \\ &= 0.68 \times 0.0425 = 0.029 \text{ N·m} = 0.29 \text{ kg·cm} \end{aligned} \quad (8)$$

With a safety factor of 3, the servo needs at least 0.88 kg·cm. The chosen servo [12] provides **[x] kg·cm at [y] V**, a margin of **[x / 0.29]**. Figure 5 shows the free-body diagrams for the robot on a slope and for the waste and gate on the ramp.

Lesson learned: Our first design attached the ramp to slope from the layer that holds the RPLiDAR to the gate at the rear of the robot. In that model the LiDAR layer sits at 130 mm and the rear support, where the gate would go, tops out at 72 mm, 165 mm further back. The slope would therefore have been $\arctan(58/165) = 19^\circ$, or about 21° if measured from the top surface of the deck ($\tan\alpha = 0.35$–0.38). By Equation 6, waste would slide only if μ_s were below about 0.35–0.38, which is not guaranteed for typical plastic or cardboard surfaces ($\mu_s \approx 0.5$). We were concerned the slope would not be steep enough, so we changed the design to a ramp mounted on the chassis. The middle plate

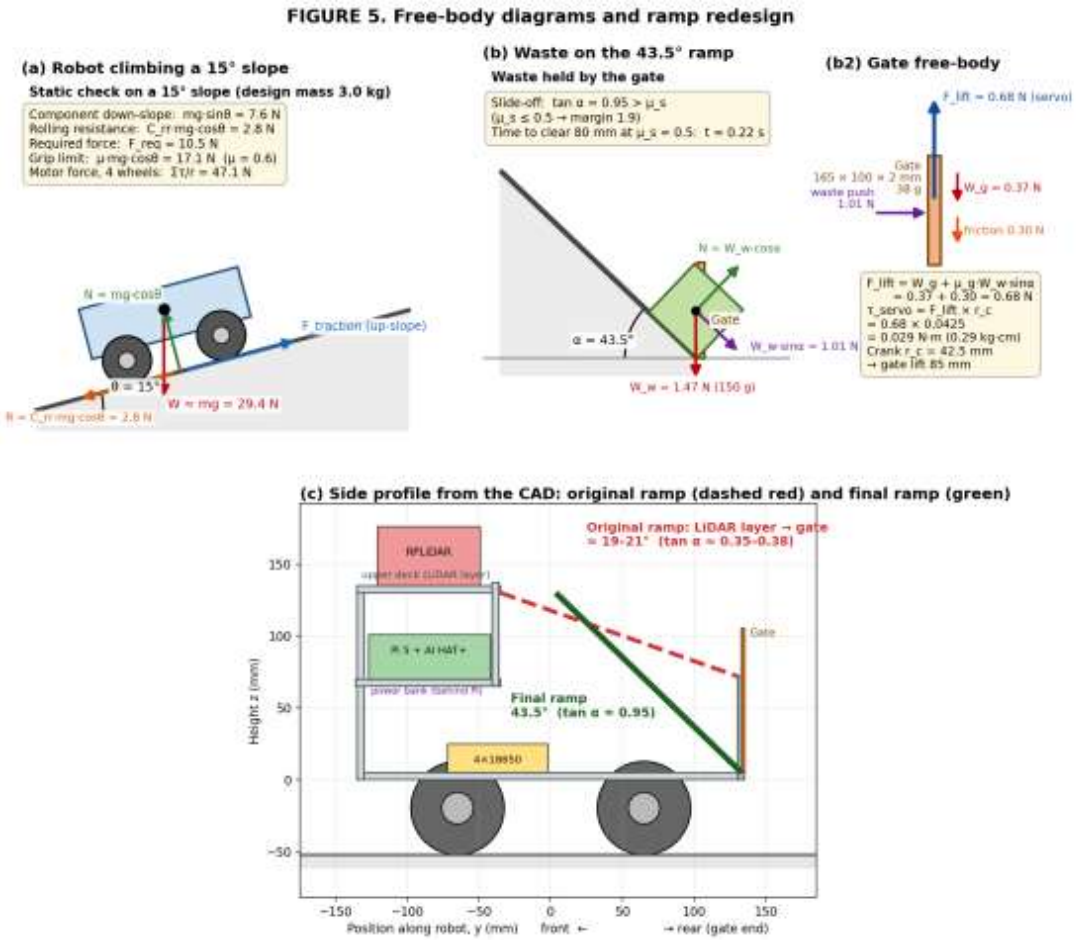


FIGURE 5. FREE-BODY DIAGRAMS FOR THE ROBOT ON A 15° SLOPE (A), THE WASTE ON THE 43.5° RAMP (B) AND THE GATE (B2), AND A CAD SIDE-PROFILE COMPARISON OF THE ORIGINAL AND FINAL RAMP (C).

now covers only the front half, which opens the rear bay. The ramp starts at deck height and ends at the base plate, giving 43.5° (Figure 5c). The lesson is to check the slide-off condition (Equation 6) before fixing the layout.

B. Electrical and Power System

Architecture. Two supplies are used. Four 18650 Li-ion cells in series [11] power the drive motors through two L298 dual H-bridge modules [9]. A 27 W power bank powers the Raspberry Pi 5 [3], with the Pi Camera [4] and RPLiDAR connected to it. The Arduino Nano [10] is powered over its USB link to the Pi, which also carries the serial data. The Nano drives the L298 modules, reads the four encoders and the MPU6050 [7], and commands the servo [12]. One L298 module drives the front pair of motors and the other drives the rear pair. Figure 3 shows the schematic and Figure 6 shows the power distribution. No buck converter is used. Table 2 lists the component ratings.

Motor-driver selection. The L298 suits the JGB37-520 motors because its ratings cover the motors' voltage, current, and heat. Table 3 sets out the margins.

- **Voltage without a regulator.** The L298's output stage drops 1.8–3.2 V at 1 A, typically 1.35 V on the source side plus 1.2 V on the sink side, or about 2.55 V [33]. From a 14.8 V pack, a motor therefore sees about $14.8 - 2.55 = 12.2$ V at full duty, which matches its 12 V rating. At full charge (16.8 V) it sees 14.3 V, so the PWM duty is limited to $12/14.3 = 0.84$. At the 12.0 V cut-off it sees 9.5 V, or 79% of rated, which is still enough for a 3–5 min run.
- **Heat.** The dissipation is $I \times V_{\text{drop}} = 1 \times 2.55 = 2.6$ W per channel, or 5.1 W for a module with both channels at rated current. This is well below the 20 W module rating [9] and the chip's 25 W limit at 75 °C case temperature

Table 2. Component ratings

Component	Supply	Current / power	Key characteristics	Ref.
JGB37-520 motor + wheel kit (×4)	12 V	0.45–1 A; 7–15 W	6–110 RPM; 10 kg·cm; Ø65 mm wheel	[8]
L298 dual H-bridge module (×2)	3–30 V	2 A peak per channel; 20 W	Bipolar bridge; TTL logic	[9], [33]
MPU6050 IMU	3–5 V	—	6-axis	[7]
18650 Li-ion cell (×4, in series)	3.7 V each	5000 mAh per cell	14.8 V nominal; 74 Wh	[11]
27 W power bank	5 V	5 A	USB-C PD	[35]
Raspberry Pi 5	5 V	up to 5 A	—	[3], [35]
Pi Camera, RPLiDAR, Arduino Nano, servo	3.3V 5V 7-12V 4.8V	0.2A, 0.75W 0.5A, 0.5W 0.030-0.035A, 0.15-0.18W 0.1-0.25A, 0.5-1.25W	---- ---- ---- ----	[4], [10], [12]

Table 3. L298 margins against the motor load

Parameter	Requirement	L298	Margin
Supply voltage	12–16.8 V	3–30 V (module) [9]; up to 46 V (chip) [33]	Covered
Current per channel, typical	0.45 A [8]	2 A peak	4.4×
Current per channel, rated	1 A [8]	2 A peak	2.0×
Dissipation, both channels at 1 A	2 × 2.6 W = 5.1 W	20 W [9]	3.9×
Logic input	5 V from Nano	TTL, $V_{IH} \geq 2.3$ V [33]	Direct drive

[33].

- **Current and stalls.** The 2 A limit is 2× the rated current and 4.4× the typical current. A stalled motor draws 2.4 A, which exceeds the L298's 2 A rating. The firmware therefore cuts the PWM when the encoders report no motion for 600 ms while the power command is 90 or more (of 255). A new command clears the cutoff.
- **Logic.** The inputs accept TTL levels, so the Nano's 5 V outputs drive them directly, with PWM on the enable pins. The logic supply is 5 V from the Nano [33].

Encoders and the Nano. Each encoder gives 11 pulses per motor revolution. The firmware counts both edges of channel A, which gives 22 counts per motor turn, and channel B gives the direction. With a 1:90 gear ratio, this is $22 \times 90 = 1980$ counts per wheel revolution, or 0.10 mm of travel per count on the 204 mm wheel circumference. At 110 RPM one wheel produces $110/60 \times 1980 = 3630$ counts per second, so 14,520 for four wheels. At the firmware's speed cap of 91 RPM the total is 12,000 counts per second. The Nano's 16 MHz processor handles this at roughly 6% load if each interrupt takes about 5 µs. The Nano has only two external-interrupt pins, D2 and D3 [34]. They serve the front-left and front-right encoders, and the rear encoders use pin-change interrupts on port B (Table 1).

Compute offload. Classification runs on a laptop rather than on the Raspberry Pi 5, which keeps the Pi's CPU free for SLAM and Nav2 and lets the model be swapped and debugged easily. The Pi captures 640 × 480 frames at 15 fps with rpicam-vid and compresses each one as a JPEG (MJPEG, quality 70) before sending it over Wi-Fi. Raw frames at that size and rate would need $640 \times 480 \times 3 \times 15 \approx 13.8$ MB/s, about 110 Mbit/s, which is more than a lab Wi-Fi link carries reliably. JPEG frames of typically 20–60 KB cut this to about 0.3–0.9 MB/s, 15 to 45 times less, with no visible effect on detection. MJPEG was chosen over H.264 for two reasons: the Pi 5 has no hardware H.264 encoder, and every JPEG frame stands alone, so a dropped frame never corrupts the frames after it.

Battery and power budget. Each 18650 cell is marked 5000 mAh and 3.7 V. Four cells in series give 14.8 V nominal, 16.8 V fully charged, and about 12.0 V at cut-off (3.0 V per cell). The pack stores $14.8 \times 5 = 74$ Wh. The L298 passes the motor current straight through, so the pack current equals the sum of the motor currents. A run lasts 3–5 minutes, so we budget for 5.

- **Typical load:** 4×0.45 A $= 1.8$ A, or $14.8 \times 1.8 = 26.6$ W. Over 5 minutes the motors use $26.6 \times 5/60 = 2.2$ Wh and 0.15 Ah, which is 3% of the pack.

Table 4. Power budget († = estimate)

Supply	Load	Typical	Peak
Li-ion pack (14.8 V)	4 × JGB37-520 motors	26.6 W (1.8 A)	59.2 W (4.0 A)
Power bank (5 V)	Raspberry Pi 5	8 W	12 W
	Pi Camera	1 W	1 W
	RPLiDAR	2.5 W	2.5 W
	Nano + IMU	0.5 W	0.5 W
	Gate servo	∼0 W	3 W
	Power bank total	12 W	19 W

- **Rated load:** 4×1 A $= 4$ A, or 59.2 W. Over 5 minutes this is 4.9 Wh and 0.33 Ah, which is 6.7% of the pack. The peak current is $4/5 = 0.8$ C. Even the L298 limit of 4×2 A $= 8$ A is only 1.6 C.

The 27 W power bank supplies a 5 V / 5 A USB-C PD profile. A supply without that profile would limit the current the Pi 5 offers to its USB devices. Table 4 shows the budget.

The peak power-bank load of 19 W is within the 27 W rating, a margin of 1.4×. Over a 5-minute run the bank supplies at most $19 \times 5/60 = 1.6$ Wh.

Wiring. The grounds of the pack, Nano, L298 modules, and Pi are connected so that all signals share one reference. Motor power cables are routed separately from encoder and sensor wires to limit noise on the signal lines.

C. Control and Software Architecture

Operating modes. The robot has two operating modes that share one interface. In teleoperation, an operator logs in to the Raspberry Pi over Wi-Fi with SSH and drives with teleop_twist_keyboard, which publishes velocity commands on /cmd_vel (0.25 m/s forward and 0.8 rad/s turning by default). This mode was used to commission the drive base and remains the manual fallback. In autonomous mode, Nav2 publishes the same /cmd_vel commands. Because both sources speak the same message, nothing below /cmd_vel needs to know who is driving, and the operator can take over at any moment. For bench work without the Pi, a Python console on the laptop talks to the Nano directly over USB, so motors, encoders, the gate, and the IMU can be tested one at a time.

Software architecture. The software is split across three machines. The Arduino Nano runs the real-time firmware as a single loop with no blocking delays: it reads serial commands, updates the IMU, runs the wheel-speed controller every 20 ms, ramps the motor power, and sends telemetry every 50 ms. The Raspberry Pi 5 runs ROS 2 Jazzy natively on Ubuntu 24.04 [13] and hosts everything that needs more memory or a wider view of the robot. The laptop is used to write, compile, and upload code, to tune the motors, and to monitor the robot. On the Pi, our package malkia_base holds the bridge node (base_node), an odometry module, a parser for the serial protocol, and a simulated Nano used for testing without hardware. The vendor driver sllidar_ros2 [6] publishes the LiDAR scan, and robot_localization runs an extended Kalman filter (EKF). Nodes communicate through topics (Table 7) and the transform tree, and Nav2 is driven through its navigation action.

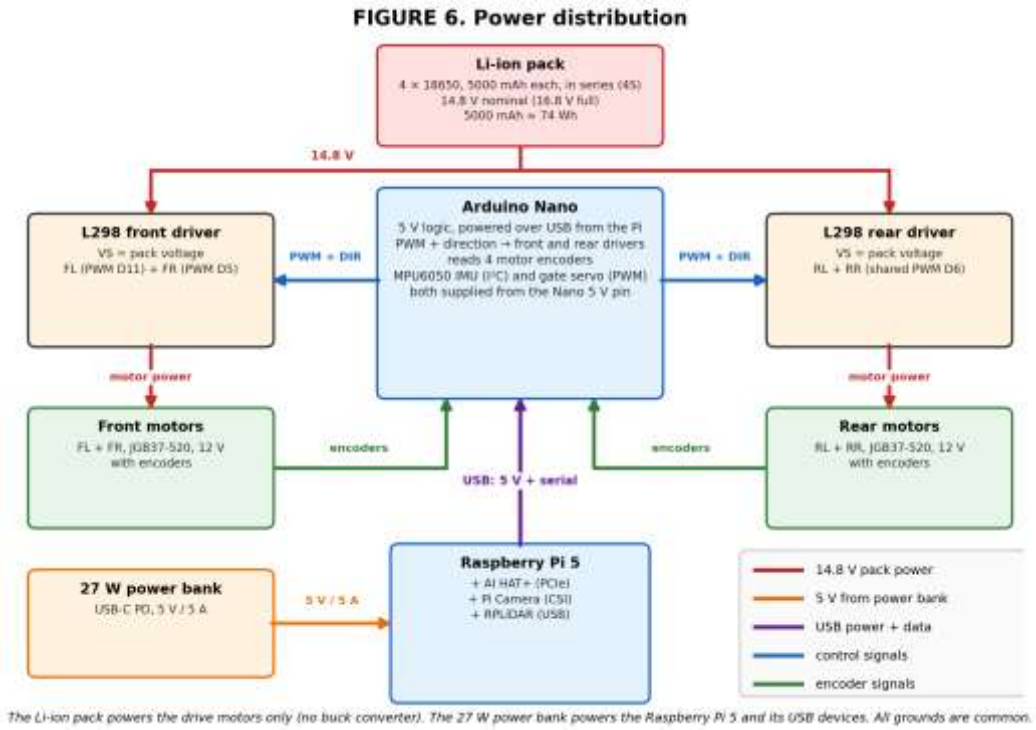


FIGURE 6. POWER DISTRIBUTION: THE LI-ION PACK FEEDS THE DRIVE MOTORS THROUGH THE FRONT AND REAR L298 DRIVERS, AND THE 27 W POWER BANK FEEDS THE RASPBERRY PI 5 AND ITS USB DEVICES.

Table 7. Main ROS 2 topics

Topic	Publisher → subscriber	Content
/cmd_vel	Teleop or Nav2 → base_node	Forward speed (m/s) and turn rate (rad/s)
/odom	base_node → EKF	Wheel odometry: pose, forward speed, turn rate
/imu/data	base_node → EKF	Orientation, gyro and accelerometer; published only while the IMU is healthy
/odometry/filtered	EKF → Nav2	Fused pose estimate
/scan	sllidar_ros2 → SLAM, Nav2	RPLiDAR range scan
/waste/image	Waste detector → RViz	Camera frame with boxes, labels, and confidence drawn on it
/waste/detections	Waste detector → task logic	Labels found in the frame as text, e.g. plastic 72%
/gate, /gate_angle	Task logic → base_node	Open or close the gate, or set a servo angle (0–180°)
/tf	EKF, base_node, SLAM	map → odom → base_link, with fixed links to imu_link and the LiDAR frame

Serial protocol. The Nano and the Pi exchange plain text lines over USB serial at 115200 baud. Telemetry goes up about 20 times per second as one line carrying the Nano's clock, the accelerometer and gyro readings, pitch, roll and yaw, and the cumulative count of each encoder. Commands go down as short lines, such as M followed by a left and a right power to drive, X to stop, and O or C to open or close the gate. We chose text over a binary format because any terminal can read it, which made debugging much faster, and 115200 baud leaves ample bandwidth.

From command to wheels. A velocity command reaches the wheels in four steps. The bridge splits the forward speed v and turn rate ω into left and right wheel speeds, $v - \omega b/2$ and $v + \omega b/2$, where b is the track width. It scales each speed to a power value out of 255, limits it to 150, and sends it to the Nano ten times per second. The Nano turns each value into a wheel-speed target in encoder ticks per second, and the feed-forward and PI controller described in Section V sets the power of each driver channel. Going the other way, the bridge integrates the encoder counts into wheel odometry and publishes the IMU data. The EKF fuses the forward speed and turn rate from the wheels with the gyro turn rate. It deliberately ignores the IMU yaw angle, which drifts, and publishes a smoother pose on /odometry/filtered.

Navigation and control. Autonomous navigation is handled by Nav2 [17]–[19], with SLAM [15], [16] providing the map and the robot's position in it. SLAM publishes the map-to-odom transform and the EKF publishes odom-to-base_link, so Nav2 always has a complete chain from the map to the robot. The global costmap is built from the SLAM map and inflates every wall so that paths keep a safe distance from it, taking the robot's footprint into account (Figure 10). Given a goal pose, the planner computes a route across this costmap. The controller then follows the route and sends velocity commands to the bridge, using a local costmap built from the latest LiDAR scans to react to obstacles that are not in the map. Nav2's own behaviour tree re-plans the route as the robot moves and, if the robot is blocked or stops making progress, runs recovery actions such as clearing the costmaps, rotating in place, backing up, or waiting before trying again. With this setup the robot navigated autonomously to goals on the game field.

Task behaviour tree. The competition task is organised as a behaviour tree [14] that sits above Nav2. Its sequence is search, detect, classify, select the zone, transport, and off-load. Detection and classification use the Pi Camera

stream, with the model running on the laptop. Once an item is classified, the tree sends the matching zone as a Nav2 goal, and at the zone it publishes on /gate to open the ramp gate and let the item slide off. Fallback branches re-scan when classification confidence is low and retry a navigation goal that fails. Each branch reports success or failure, so the tree always knows which step to retry.

Detection pipeline. The classifier is a YOLO26 nano model trained on three classes, metal, organic, and plastic, and exported to ONNX (about 9.5 MB). It takes a 640 × 640 RGB image and returns 8400 candidate boxes, each with four box values and three class scores. Each 640 × 480 camera frame is first letterboxed to 640 × 640 without stretching, scaled to the 0–1 range, and converted from BGR to RGB. For every candidate, the highest class score gives the label and its confidence. Candidates below 0.4 are discarded, the remaining boxes are mapped back to the original frame, and non-maximum suppression with an IoU threshold of 0.45 keeps one box per object. The surviving boxes are drawn with their labels and published on /waste/image, and the labels are also published as text on /waste/detections (Table 7), so the annotated feed can be watched live in RViz and the behaviour tree can act on the result. On the laptop the model runs through OpenCV's DNN module, so no deep-learning framework has to be installed. A separate thread reads the camera stream and keeps only the newest frame. Without it, frames arriving at 15 fps would queue behind a detector running at only a few frames per second, and the delay would grow every second; with it, older frames are dropped and the output stays close to real time.

Safety and fault handling. Safety is layered so that a single fault cannot leave the robot running out of control:

- **Command watchdog.** The Nano stops the motors if no command arrives for 500 ms. The bridge re-sends commands at 10 Hz, so normal operation never trips it, but a lost cable or a crashed node does.
- **Motor protection.** Power is capped at 200 of 255 and may change by at most 12 units every 20 ms. Before a wheel reverses, its power ramps down to zero, the direction pins switch, and only then does the power ramp up again, which protects the drivers.
- **Stall and encoder guards.** If a side receives power of 90 or more but its wheels do not move for 600 ms, that side is switched off and an error is reported. If an encoder stops counting while the other wheels turn, the firmware reports it and drives that wheel on its feed-forward value alone, rather than letting the controller push it to full power.
- **IMU faults.** An I^2C timeout stops a missing IMU from freezing the Nano. A missing IMU reports exact zeros, which the bridge recognises, and it then stops publishing IMU data so that no false readings reach the EKF.
- **Connection loss.** The bridge reconnects automatically if the USB link drops. It also stops the robot if /cmd_vel goes quiet for 0.5 s, and it sends an explicit stop command when the requested speed is zero.

Calibration, diagnostics, and logging. The gyro is calibrated each time the serial port opens, so the robot is kept still while the software starts. The accelerometer level is measured once on a flat surface and stored in the Nano's EEPROM, so it survives restarts. Because the Nano resets whenever the port opens, the bridge re-sends its settings, such as the heading-hold gain, at every connection. Odometry is checked with a short calibration routine: turning one wheel by hand through one revolution to confirm the counts, driving a marked 1 m line, spinning through a full turn to tune the track width, and confirming that the four wheels match within 2% at the same command. For diagnostics, a status command returns the Nano's targets, per-channel power, directions, gains, and stall flags in one line, and faults are reported as short error messages. During tuning, every run was logged to a CSV file, and a script summarised each wheel's steady speed, error, overshoot, and settling time. The odometry maths is covered by unit tests, and the whole ROS package was first run in a Docker container against the simulated Nano.

Lesson learned: Writing a simulated Nano and testing the ROS package against it in Docker meant that the bridge, odometry, and EKF ran on the Pi almost first time, long before the hardware was finished. The opposite lesson came from configuration. Settings typed into the Nano by hand were lost every time the serial port opened and reset the board, which led to confusing runs in which the robot behaved differently from the last test. Next time we would keep every tunable value in one place, the ROS parameter file, and have the host send them all at each connection from the start, and we would build the simulated hardware first for every new subsystem.

V. Experimental Results

The drive base, the Nano firmware, and the ROS 2 bridge have been built and tested on the real robot. SLAM mapping, autonomous navigation with Nav2, waste classification, and ramp off-loading have all been tested. This section reports what was measured and what failed along the way.

Test method. The firmware (about 24,500 bytes, 79% of the Nano's flash, and 855 bytes of RAM, 41%) runs one loop with no blocking delays. The wheel-speed controller and the motor ramp run every 20 ms, and telemetry goes to the host every 50 ms. Wheels were first matched on the bench with a motors-only tuning sketch that has the same wiring as the robot. It logged each wheel's speed to a file, and a script reported the steady speed, error, overshoot, and settling time. The key runs were then repeated on the floor. Before the hardware was ready, the ROS 2 package was run in a Docker container against a simulated Nano, and its odometry maths was covered by unit tests.

Wheel matching. With the same power (150 of 255) and no controller, the four wheels turned at 87.9, 82.0, 77.8, and 77.7 RPM (Figure 7a). The front-left wheel ran 13% faster than the rear-right one, which pulled the robot to one side. The front-left motor needed less power for the same speed, so it was given its own power pin and each wheel received a feed-forward trim (0.83, 0.87, 0.98, 0.98 for FL, FR, RL, RR). With a proportional gain of 0.5 and

Table 5. Steady wheel speed in RPM

Test	FL	FR	RL	RR	Spread
Open loop, power 150	87.9	82.0	77.8	77.7	10.2 RPM (12.5%)
Closed loop, 40 RPM	39.9	40.0	39.7	40.3	0.6 RPM (1.5%)
Closed loop, 70 RPM	71.7	72.1	71.2	72.5	1.3 RPM (1.9%)
Closed loop, 100 RPM	101.4	101.1	99.3	101.6	2.3 RPM (2.3%)

an integral gain of 1.5 per RPM, and a full-power speed of 138 RPM, the wheels settled within 3.6% of the target at 40, 70, and 100 RPM, and within 2.3% of each other (Figure 7b, Table 5). At 40 RPM every wheel was within 0.8% of the target.

At 100 RPM the rear pair needed 216 of the 220 power limit, so there is no headroom above this speed. The firmware therefore caps wheel speed at 91 RPM (0.31 m/s) and power at 200 of 255. On the bench, start-up showed an overshoot of about 15% for about one second, which the robot firmware's soft start mostly hides. The gains were converted to ticks per second for the robot firmware (kp 0.0152, ki 0.07), and the trims were re-tuned on the floor to 0.845, 0.85, and 0.98 for the front-left, front-right, and rear pair.

Floor tests. With equal commands to both sides and no heading correction, the distance travelled by each wheel agreed within about 1%, but the robot drifted by a few degrees, in a different direction on different runs, because the wheels scrub on the floor. An opposite command to the two sides turned the robot to the right and gave a negative yaw (between -59° and -239° in the runs), which confirmed the sign of the gyro yaw. With the gyro heading correction on at a gain of 5, the yaw stayed between -6.0° and -3.4° over about 0.8 m, with wheel distances still within 1%. A gain of 15 made the wheel speeds pull apart while the gyro showed a large drifting yaw, so the gain was lowered to 5.

IMU. The MPU6050 first returned only zeros. Measurements showed the SDA and SCL lines sitting too low for the Nano to read as high, and the fault followed the IMU board and its wiring, not the Nano. Once the board answered at address 0x68, pitch and roll stayed within about two degrees and the yaw held 0.0° while the robot was still. An I^2C timeout of 3 ms now stops a missing IMU from freezing the Nano, and a missing IMU reports exact zeros so that no false data reaches ROS.

Faults found and fixed. Table 6 lists the faults met during tuning. The front-right encoder fault was found by reading the log: the power was at its limit while the measured speed was zero.

ROS 2 integration. On the Raspberry Pi 5, the bridge node reads about 20 telemetry lines per second from the Nano and publishes wheel odometry, the IMU data, and the transforms. It turns velocity commands into motor commands at 10 Hz. The Nano stops the motors if no command arrives within 500 ms, so a lost cable cannot make the robot run away. The launch file also starts an extended Kalman filter that fuses wheel speed and gyro turn rate, but not the drifting yaw angle, into a smoother pose estimate. The odometry uses a 65 mm wheel, a 0.22 m track width, and 1980 counts per

Table 6. Faults found and fixed

Symptom	Cause	Fix
Front-left wheel ran fast and pulled the robot	It needed less power than the others and shared a power pin with the other left wheel	Own pin (D11), own speed loop, feed-forward trim; front driver replaced
A wheel read 0 RPM while turning and its power went to the limit	Front-right encoder connection dropped out	Wires re-seated; firmware warns after 0.5 s and holds the wheel on its feed-forward
Front-right encoder under-read	Pin D13 has the board LED and shares a pin-change group	Encoder moved to D3 (INT1); servo moved to D13
All wheels at 0 RPM while power climbed, then overshoot	Motor battery switched off; the integral term wound up	Switch checked; integral stops growing at the power limit
Robot shook in place at start	First version let the power go negative	Forward-only power, smoothing, conditional integration

wheel turn, and ROS limits the power to 150 of 255 (about 0.28 m/s). On the real robot the bridge connected, the IMU was detected, and the filter ran. A keyboard command of 0.25 m/s becomes a power of 136 and a wheel target of about 74 RPM, which is below the 91 RPM cap. The robot has since been driven on the floor both from the keyboard and, through the same velocity interface, by Nav2.

Robot model. Before the chassis was built, the robot was described as a URDF using the meshes from the CAD model and loaded in RViz (Figure 8). This early model carries the body panels, the four treaded wheels, the RPLiDAR on the upper deck above the front half, and the inclined ramp across the open rear bay. RViz loaded it with no transform errors, which confirmed that every link was attached to the base frame and gave us the frames that the bridge node and the LiDAR driver publish against. The same description was used for the first Gazebo runs on the model of the game field.

SLAM mapping. The RPLiDAR was then added to the launch with its transform to the robot frame, and SLAM was run on the game field during test runs. Figures 9 and 10 show two of the resulting maps. In both, the outer boundary and the internal walls appear as continuous, straight lines that meet at clean corners, and the layout of the field can be read directly from the grid. In Figure 9 the live scan (red) lies on top of the mapped walls next to the robot (white), so the pose estimate and the map agreed when the image was taken. Grey rays reach far beyond the field in both maps. They come from beams that pass through openings or over low parts of the boundary and strike the walls of the room. The boundary itself is still mapped as occupied, so planned paths stay inside the field, but the extra area enlarges the map and costs processing time. A keep-out mask around the field would remove it.

Figure 10 shows a map from an earlier run with the navigation costmaps drawn on top. Each wall is surrounded by an inflation band, shaded from light blue to pink, which keeps planned paths at a safe distance from the walls given the size of the robot. The brighter band around the robot is the local costmap, rebuilt from the latest scans, and the blue line shows the path leading to the robot's current pose. These maps show that the LiDAR, the odometry, and SLAM work together on the real robot. Using them, Nav2 navigated the

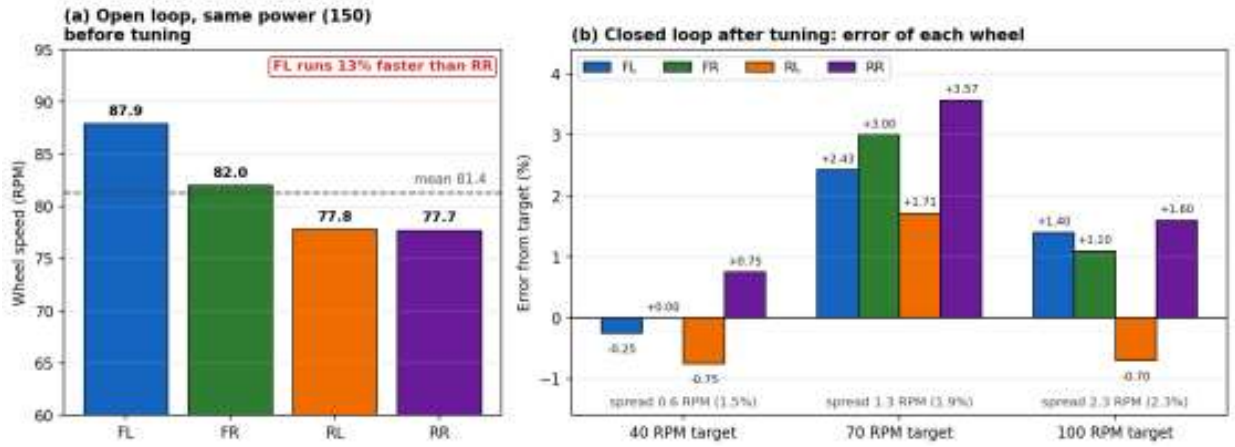


FIGURE 7. WHEEL SPEEDS BEFORE AND AFTER TUNING: (A) THE FOUR WHEELS AT THE SAME POWER WITH NO CONTROLLER, (B) THE ERROR OF EACH WHEEL FROM ITS TARGET IN CLOSED LOOP.

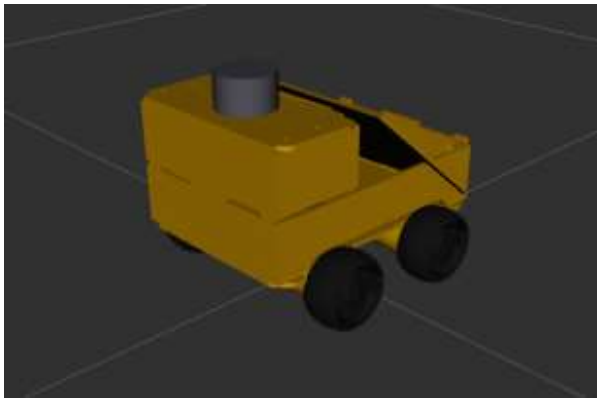


FIGURE 8. EARLY-STAGE URDF MODEL OF THE ROBOT DISPLAYED IN RVIZ.

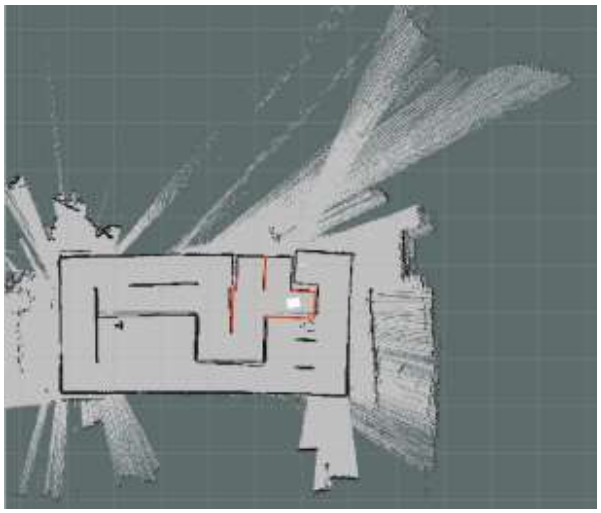


FIGURE 9. OCCUPANCY-GRID MAP OF THE GAME FIELD BUILT BY SLAM, WITH THE LIVE LIDAR SCAN IN RED AND THE ROBOT IN WHITE.

robot autonomously to goals on the game field, planning routes that kept clear of the inflated walls and following them without operator input.

Waste classification. The detector runs as a ROS 2 node that publishes each annotated frame on /waste/image, and

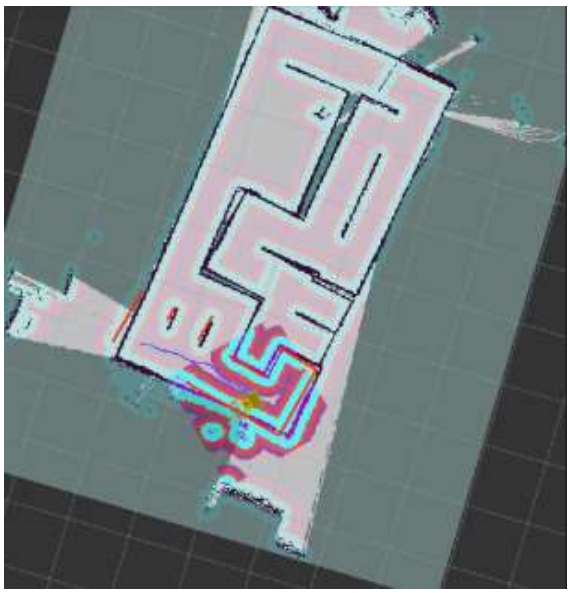


FIGURE 10. SLAM MAP FROM AN EARLIER TEST RUN WITH THE GLOBAL AND LOCAL COSTMAPS OVERLAID AND THE ROBOT FOOTPRINT SHOWN.

RViz shows the topic live in its Image display. Figure 11 shows two frames as they appeared in RViz. In Figure 11a a plastic bottle is boxed in blue and labelled plastic with 63% confidence, and in Figure 11b an organic item is boxed in green and labelled organic with 57% confidence. In both cases the model picked the correct class for the item in view, even with the item only partly in frame and under ordinary indoor lighting. Each label carries a confidence score, which the behaviour tree uses to decide whether to accept the label or re-scan the item. Running on the laptop CPU, the node processed about 3 frames per second (2.9 and 3.0 FPS in the two frames).

Off-loading. The ramp and gate were tested on the robot. With an item resting against the closed gate, the servo lifted the gate and the item slid down the 43.5° ramp and out of the robot cleanly, without sticking on the ramp or catching on the gate. This confirms in practice the slide-off condition of Equation 6 and the servo sizing of Equation 8, and shows



FIGURE 11. LIVE CLASSIFICATION OUTPUT IN RVIZ'S IMAGE DISPLAY: (A) A PLASTIC BOTTLE IN A BLUE BOX, LABELLED PLASTIC AT 63%, (B) AN ORGANIC ITEM IN A GREEN BOX, LABELLED ORGANIC AT 57%. THE BOX COLOUR MARKS THE CLASS: BLUE FOR PLASTIC, GREEN FOR ORGANIC, AND GREY FOR METAL.

that a single servo is enough to complete the off-load step.

Design trade-offs. Four choices shaped the results:

- **Heading-hold gain.** A gain of 5 kept the robot within a few degrees of straight. A gain of 15 amplified the gyro drift, so the higher gain was rejected.
- **Speed cap.** The L298 drops about 2 V and the rear pair runs out of power near 100 RPM, so the robot runs at up to 91 RPM, below the rated 110 RPM.
- **Shared rear power pin.** The Nano has six PWM pins, which are not enough for four motors, four encoders, and the servo. The rear wheels therefore share one pin and the controller uses their average speed. This is enough for straight driving and spinning, but the rear wheels cannot follow different speeds in a wide curve. A PWM board on the I^2C bus would give four independent channels.
- **Counting and control.** Counting both edges of channel A gives 0.10 mm per count with one interrupt per wheel. The speed loop uses proportional and integral terms only, because encoder speed measured over 20 ms is too noisy for a derivative term.

VI. ACKNOWLEDGEMENTS

Much of this robot rests on knowledge that others chose to share openly. The Robotics Dojo articles written for competitors gave us a practical place to start: F. Opiyo's introductions to SLAM and to the Nav2 stack [15], [17] shaped how we set up mapping and navigation, M. M. Farida's guide to URDF [27] helped us describe the robot long before it existed in hardware, and V. Rotich's article on simulation worlds [29] showed us how to model the game field in Gazebo. We also learned a great deal from the SLAM Toolbox work of S. Macenski and I. Jambrecic [16], the behaviour-tree text by M. Colledanchise and P. Ögren [14], the waste datasets published by G. Thung and M. Yang [20] and by P. F. Proença and P. Simões [21], and the paper by M. Quigley et al. that introduced ROS [22]. We are equally grateful to the maintainers of ROS 2, Nav2, Gazebo, and RViz, whose documentation we returned to at almost every step.

We thank Mr. Lenny, who guided us through the whole process, from the first design ideas to testing on the game field. His recommendations were consistently helpful and shaped many of the decisions described in this paper.

We thank Mr. Billy, who helped us acquire the components the robot is built from and helped us 3D print its parts.

Finally, we thank the Robotics Dojo organisers and trainers for the sessions that introduced us to these tools, and for a competition that let us learn by building.

VII. REFERENCES

[1] ZenRobotics (Terex), "Fast Picker: AI-powered waste sorting robot." [Online]. Available: https://www.terex.com/zenrobotics/fast-picker-artificial-intelligence-robot

[2] AMP Robotics, "AMP Robotics unveils new AI-enabled automated sortation solution, introduces integrated facility offering for new recycling infrastructure," Business Wire, May 1, 2023. [Online]. Available: https://www.businesswire.com/news/home/20230501005304/en

[3] Pixel Electric Ltd., "Raspberry Pi 5 [variant]," product page. [Online]. Available: https://www.pixelelectric.com/development-boards/raspberry-pi-5-model-4gb-ram/

[4] Pixel Electric Ltd., "Raspberry Pi Camera Module v2," product page. [Online]. Available: https://www.pixelelectric.com/search-results-page?q=Raspberry%20Pi%20Camera%20V2%208MP

[5] [Author], "RPLidar SetUp On Windows PC," Robotics Dojo, [Month Day, Year]. [Online]. Available: https://roboticsdojo.substack.com/p/rplidar-setup

[6] Slamtec, "sllidar_ros2: ROS 2 driver for Slamtec RPLIDAR," GitHub. [Online]. Available: https://github.com/Slamtec/sllidar_ros2

[7] Pixel Electric Ltd., "MPU6050 6-axis accelerometer and gyroscope module," product page. [Online]. Available: https://www.pixelelectric.com/imu-accelerometer-magnetometer-gyroscope/mpu-6050-accelerometer-gyro/

[8] Pixel Electric Ltd., "JGB37-520 12V 110RPM Encoder Motor plus wheel Kit," product page. [Online]. Available: https://www.pixelelectric.com/more-categories/robots/wheels/rc-wheels/jgb37-520-12v-110rpm-encoder-motor-plus-wheel-kit/

[9] Pixel Electric Ltd., "L298 dual H-bridge motor driver," product page. [Online]. Available: https://www.pixelelectric.com/brushed-brushless-dc-motor-driver/l298-dual-h-bridge-motor-driver/

[10] Pixel Electric Ltd., "Arduino Nano," product page. [Online]. Available: https://www.pixelelectric.com/boards-compatible-with-arduino/arduino-nano-v3-0

[11] Pixel Electric Ltd., "18650 Li-ion battery," product page. [Online]. Available: https://www.pixelelectric.com/li-ion-cylindrical-battery-cell/lithium-ion-cylindrical-battery-3-7v-18650-cell-2500mah-10c/

[12] Pixel Electric Ltd., "[Servo model]," product page. [Online]. Available: https://www.pixelelectric.com/towerpro-servo-motors/micro-servo-9g-ts360-degree-digital-servo-motor/

[13] Open Robotics, "ROS 2 Documentation: Jazzy." [Online]. Available: https://docs.ros.org/en/jazzy/

[14] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton, FL, USA: CRC Press, 2018, doi: 10.1201/9780429489105.

[15] F. Opiyo, "Introduction to Simultaneous Localization and Mapping," Robotics Dojo, Aug. 20, 2025. [Online]. Available: https://roboticsdojo.substack.com/p/introduction-to-simultaneous-localization

[16] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021, doi: 10.21105/joss.02783.

[17] F. Opiyo, "Introduction to Navigation 2 Stack," Robotics Dojo, Aug. 20, 2025. [Online]. Available: https://roboticsdojo.substack.com/p/introduction-to-navigation-2-stack

[18] Nav2 Project, "Nav2 documentation." [Online]. Available: https://docs.nav2.org/

[19] Open Robotics, "navigation2 package," ROS 2 Documentation: Jazzy. [Online]. Available: https://docs.ros.org/en/jazzy/p/navigation2

[20] G. Thung and M. Yang, "Classification of Trash for Recyclability Status," CS229 Project Report, Stanford University, 2016. [Online]. Available: https://cs229.stanford.edu/proj2016/report/ThungYang-ClassificationOfTrashForRecyclabilityStatus-report.pdf

[21] P. F. Proença and P. Simões, "TACO: Trash Annotations in Context for Litter Detection," arXiv:2003.06975, 2020.

[22] M. Quigley et al., “ROS: an open-source Robot Operating System,” ICRA Workshop on Open Source Software, Kobe, Japan, 2009.

[23] ROBOTIS, “TurtleBot3,” GitHub. [Online]. Available: https://github.com/ROBOTIS-GIT/turtlebot3

[24] 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,” *Robotics and Autonomous Systems*, vol. 168, Art. no. 104493, 2023, doi: 10.1016/j.robot.2023.104493.

[25] RoboCup Federation, “RoboCup.” [Online]. Available: https://www.robocup.org

[26] Eurobot Association, “Eurobot.” [Online]. Available: https://www.eurobot.org

[27] M. M. Farida, “Understanding URDF: Building the Blueprint of Your Robot,” Robotics Dojo, Aug. 20, 2025. [Online]. Available: https://roboticsdojo.substack.com/p/understanding-urdf-building-the-blueprint

[28] Open Robotics, “Building a visual robot model with URDF from scratch,” ROS 2 Documentation: Jazzy. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Intermediate/URDF/Building-a-Visual-Robot-Model-with-URDF-from-Scratch.html

[29] V. Rotich, “Introduction to Worlds in Robotics Simulation,” Robotics Dojo, Aug. 20, 2025. [Online]. Available: https://roboticsdojo.substack.com/p/introduction-to-worlds-in-robotics

[30] Gazebo, “SDF worlds,” Gazebo Harmonic documentation. [Online]. Available: https://gazebosim.org/docs/harmonic/sdf_worlds/

[31] Gazebo, “ros_gz: Integration between ROS and Gazebo,” GitHub. [Online]. Available: https://github.com/gazebosim/ros_gz

[32] Open Robotics, “RViz user guide,” ROS 2 Documentation: Jazzy. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Intermediate/RViz/RViz-User-Guide/RViz-User-Guide.html

[33] STMicroelectronics, “L298: Dual full-bridge driver,” datasheet. [Online]. Available: https://www.electrokit.com/upload/quick/4f/70/f1f1_40350298-tds.pdf

[34] Arduino, “Arduino Nano documentation.” [Online]. Available: https://docs.arduino.cc/hardware/nano/

[35] Kitronik, “Raspberry Pi 5 27W USB-C power supply (5.1 V, 5 A),” product page. [Online]. Available: https://kitronik.co.uk/products/5049-raspberry-pi-5-uk-27w-5a-usb-c-power-supply