AUTONOMOUS VISION-BASED LITTER COLLECTION ROVER

by

Benjamin Tavares

Dante Benedetti

Nathan Heil

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

June 2026

Advisor: Dr. Clay McKell

Statement of Disclaimer

Since this project is a result of a class assignment, it has been graded and accepted as fulfillment of the course requirements. Acceptance does not imply technical accuracy or reliability. Any use of information in this report is at risk of the user. These risks may include catastrophic failure of the device or infringement of patent or copyright laws. California Polytechnic State University at San Luis Obispo and its staff cannot be held liable for any use or misuse of the project.

TABLE OF CONTENTS

Acknowledgements xi

Abstract xiii

I. Introduction 1

Motivation and Problem Context 1

NABC Analysis 1

State of the Art and Limitations 2

Project Objective and Approach 3

II. Background 4

Prior Work and Literature 4

Relevant Technical Topics 4

III. Requirements 19

Marketing Requirements and Pairwise Hierarchy 19

Engineering Specifications 20

IV. Design 23

System Architecture 23

Mechanical Design 25

Electrical Design 33

Firmware Design (STM32) 37

Software Design (Jetson / ROS 2) 41

Robotic Arm and Scoop Trajectory Design 48

Autonomy Design 50

V. Test Plans 60

Trash Detection Performance Test 60

Trajectory Planning and Control 62

Approach and Navigation 64

Obstacle Avoidance Test 67

End-to-End Autonomy 69

VI. Test Results and Verification 72

Trajectory Planning and Control Tests 72

Two Item Trajectory Planning and Navigation: Original Implementation 72

Controller Modifications 75

Two Item Trajectory Planning and Navigation: Updated Implementation 78

Updated Implementation: Conclusions and Improved Metrics 81

Trash Detection Test 82

Approach and Navigation Success Test 84

Scoop Success Test 86

Obstacle Avoidance Test 88

End-to-End Autonomy Test 89

VII. Development and Construction 94

Timeline 94

Assembly Progression 95

Challenges During Construction 103

VIII. Conclusion 106

Summary of Achievements 106

Remaining Work 107

Lessons Learned 108

Future Work 109

IX. Bibliography 111

Appendix A: Analysis of Senior Project Design 113

Summary of Functional Requirements 113

Primary Constraints 113

Economics 114

Environmental 114

Manufacturability 115

Sustainability 115

Ethical 115

Health and Safety 116

Social and Political 116

Development 116

Engineering Standards 117

Appendix B: Parts List and Costs 118

Appendix C: Schedule - Time Estimates 119

Appendix D: Wire List / Pin Assignments 120

Appendix E: Program Listing 122

Appendix F: Hardware Configuration / Layout 123

Appendix G: Reference Material 124

LIST OF TABLES AND FIGURES

Tables

TABLE I. PAIRWISE COMPARISON MATRIX 19

TABLE II. ENGINEERING SPECIFICATIONS 20

TABLE III. POWER BUDGET 33

TABLE IV. FIRMWARE MODULE SUMMARY 38

TABLE V: AUTONOMY/PATH PLANNING UPDATED CONTROLLER PARAMETERS 76

TABLE VI: COMPARISON OF CONTROL METRICS AFTER UPDATING CONTROL LOGIC 81

TABLE VII: TRASH DETECTION PERFORMANCE SUMMARY 82

TABLE VIII: APPROACH AND NAVIGATION SUCCESS TEST RESULTS 84

TABLE IX: SCOOP SUCCESS TEST RESULTS 86

TABLE X: OBSTACLE AVOIDANCE TEST RESULTS 88

TABLE XI: END-TO-END AUTONOMY TEST RESULTS..... 89

TABLE XII: ENGINEERING SPECIFICATION AND VERIFICATION MATRIX 91

TABLE XIII. BILL OF MATERIALS 118

TABLE XIV. DEVELOPMENT SCHEDULE 119

TABLE XV. STM32L4A6ZG PIN ASSIGNMENTS..... 120

TABLE XVI. FIRMWARE SOURCE FILES..... 122

TABLE XVII: VIDEOS AND LINKS 126

Figures

Figure 1: Completed LitterBot Prototype with Integrated Drive, Collection, Sensing, and Control Subsystems 24

Figure 2. CAD Model - Isometric View of Full Chassis 25

Figure 3. CAD Model - Side View with Stepper Motor and Mecanum Wheels 26

Figure 4. CAD Model – Arm and End Effector 27

Figure 5. CAD Model - Stepper Motor Mounting Bracket 27

Figure 6. Lead Screw and Linear Slides - Top-Down View 28

Figure 7. 3D-Printed Ramp Piece 29

Figure 8. 3D-Printed End-Effector Sections Laid Out Before Assembly 30

Figure 9. 3D-Printed STM32 Housing 30

Figure 10. Trash Bin CAD Model and Physical Print 31

Figure 11: Machined Camera Mount for D455 Camer 32

Figure 12. Wiring Diagram 35

Figure 13. Chassis Dimensions Used in Calculating Forward Kinematics 40

Figure 14. Operation Flow of Software 42

Figure 15: Simplified Software Architecture Diagram 43

Figure 16: RViz Occupancy Map During Autonomy Test 45

Figure 17: LitterBot Command Center User Interface 47

Figure 18: Autonomous Path Planning Trajectory Performance Metrics Planning Around an Obstacle (Original Control Logic) 72

Figure 19: Autonomous Path Planning Trajectory Performance Metrics Planning Without an Obstacle (Original Control Logic) 74

Figure 20: Autonomous Path Planning Trajectory Performance Metrics Planning Around an Obstacle (Updated Control Logic) 78

Figure 21: Autonomous Path Planning Trajectory Performance Metrics Planning Without an Obstacle (Updated Control Logic) 80

Figure 22: Detection Confidence vs. Distance for Representative Litter Objects 83

Figure 23. Chassis Frame with Vertical Supports and Linear Slides 96

Figure 24. Chassis with Lead Screw and Mecanum Wheels - Side View 96

Figure 25. Chassis Assembly Outdoors - Lead Screw and Linear Slides Visible 97

Figure 26. Multi-Level Frame Assembled with Mecanum Wheels 99

Figure 27. Robot Chassis on Workbench in Machine Shop 99

Figure 28. Stepper Motor Attached to Mount 100

Figure 29. Assembled Wiring of STM32 101

Figure 30. Battery Installation with Fully Integrated Wiring 101

Figure 31. Location of Buck Converters on Underside of Top Layer 102

Figure 32. Complete Assembly - Multi-Level Frame with All internal Subsystems 103

Figure 33. Front View 123

Figure 34. Rear View Showing Linear Slides 124

Figure 35. Isometric Rear View 124

Acknowledgements

We would like to express our sincere gratitude to Dr. Clay McKell for his guidance and mentorship throughout the senior project sequence. We also thank the Electrical Engineering Department at California Polytechnic State University for providing laboratory space and access to essential equipment.

We are grateful to the Cal Poly machine shop staff for their assistance with fabrication and mechanical support.

We also must note this project utilized generative artificial intelligence (AI) tools during portions of the design, software development, troubleshooting, documentation, and report-writing process. AI tools including Anthropic Claude Code [1], OpenAI ChatGPT [2], and OpenAI Codex [25] were used to assist with code generation, debugging, software architecture discussions, technical research, editing, formatting, grammar improvement, and drafting portions of written content.

All AI-generated content, code, calculations, figures, tables, and written material were reviewed, verified, modified as necessary, and approved by the student authors prior to inclusion in the final project. AI-generated outputs were treated as preliminary assistance rather than authoritative sources. Engineering decisions, hardware implementation, software integration, testing, analysis of results, and final conclusions were performed and validated by the project team.

The authors accept full responsibility for the accuracy, originality, and integrity of all material contained in this report and for ensuring compliance with California Polytechnic State University academic integrity and artificial intelligence policies.

Finally, we thank our families for their continued support and encouragement throughout this project.

Abstract

This report documents the design, implementation, and testing of an autonomous litter-collection rover developed as a Senior Project Design Lab (EE 460/463/464) at California Polytechnic State University. The rover integrates autonomy, computer vision, embedded real-time control, mecanum-wheel omnidirectional mobility, and a two-degree-of-freedom robotic arm to detect, approach, and collect small ground-level litter such as bottles, wrappers, and paper fragments.

The system uses a two-layer compute architecture: an NVIDIA Jetson Orin Nano running ROS 2 for perception, SLAM, and path planning, paired with an STM32L4A6ZG microcontroller for real-time motor control and odometry. The robot is built on a multi-level aluminum frame with a 3D-printed collection ramp and custom electronics housings.

The completed system demonstrated autonomous litter detection, target localization, path planning, obstacle avoidance, and robotic collection capabilities. Testing validated battery-powered teleoperation, real-time perception, and autonomous navigation using a custom Theta* planning framework and ROS 2 autonomy stack. Path-following experiments achieved mean cross-track errors as low as 0.033 m during nominal operation and 0.059 m in obstacle-avoidance scenarios. Scoop testing achieved success rates of up to 100% for plastic bottle collection and 90% for several paper-based litter configurations. The final platform weighed approximately 42 lb and integrated a YOLO-based detector trained on approximately 12,700 images. This

report presents the system architecture, design methodology, implementation, testing procedures, results, and lessons learned.

I. Introduction

Motivation and Problem Context

Urban and campus environments accumulate small, scattered litter such as bottles, wrappers, and paper fragments. These items degrade public aesthetics, introduce pollutants to drainage systems, and demand continual manual cleanup. Municipal and university maintenance departments dedicate significant labor and expense to manual sweeping and collection -- tasks that are repetitive, physically demanding, and often performed in unsafe roadside or park conditions. Although large, mechanized sweepers exist, they are optimized for open roadways and cannot access sidewalks, curbs, or narrow paths where visible litter accumulates most.

The demand for low-cost, small-form-factor cleaning systems is increasing as cities pursue sustainable automation strategies. Current robotic solutions are almost entirely limited to indoor environments (vacuums, floor scrubbers) or industrial-scale sweepers. No affordable, autonomous outdoor robot currently exists that targets scattered litter in pedestrian spaces. This gap presents a clear engineering opportunity.

NABC Analysis

Need: Outdoor litter remains unaddressed by existing indoor vacuums and industrial sweepers.

Approach: WALL-E-inspired rover integrating computer vision, embedded control, and robotic manipulation using Jetson + STM32 with commercial off the shelf components.

Benefit: Reduces human cleanup labor; scalable testbed for swarm coordination and municipal integration.

Competition: Manual collection (flexible but labor-intensive), sweepers (miss small debris), smart bins (cannot remove existing litter).

State of the Art and Limitations

Autonomous cleaning technologies fall into three categories:

- Indoor consumer robots - vacuum or mop units (e.g., iRobot Roomba) that rely on infrared or LiDAR for short-range mapping. Unsuitable for outdoor terrain or debris larger than dust.
- Industrial sweepers - ride-on systems used by facilities departments. High throughput but expensive, bulky, and inefficient on sidewalks.
- Smart receptacles - bins that compact or signal fill-level data. They cannot remove litter already on the ground.

Each category addresses part of the waste-management workflow but not autonomous outdoor pickup. Outdoor operation introduces uneven ground, sun glare, wind-blown debris, and dynamic obstacles such as pedestrians.

Project Objective and Approach

The objective of this project is to develop an autonomous mobile robot capable of locating, approaching, and collecting small pieces of ground-level litter in outdoor environments. The system is intended to reduce the manual effort required for litter collection while serving as a platform for research in autonomous navigation, computer vision, and robotic manipulation.

To achieve this objective, the project integrates perception, mobility, and object-collection capabilities into a single robotic platform. The system is designed to operate autonomously in pedestrian environments while satisfying requirements related to portability, traversability, dexterity, safety, and autonomous operation.

Key design goals include affordability using commercial off-the-shelf components, reliable operation on typical outdoor surfaces, and safe autonomous behavior around obstacles and pedestrians.

II. Background

Prior Work and Literature

Autonomous litter collection is an emerging area of robotics research. The global service robotics market has seen steady growth, with outdoor cleaning applications identified as a key growth segment. Existing autonomous cleaning technologies typically focus on indoor vacuum robots, industrial sweepers, or smart waste-management systems, but none have combined omnidirectional mobility, outdoor computer vision, and robotic manipulation in a single low-cost platform.

Relevant Technical Topics

Mecanum Wheel Kinematics and Encoder Odometry

A mecanum-drive robot uses four independently driven wheels with angled rollers to generate planar motion in three degrees of freedom: longitudinal translation, lateral translation, and yaw rotation. Unlike a differential-drive robot, which primarily moves forward and rotates, a mecanum platform can strafe left and right without changing heading. This makes mecanum drive useful for robots that must align precisely with objects, walls, or pickup targets [8].

Each wheel contributes a velocity component along the wheel rolling direction and a lateral force component through the angled rollers. By commanding different combinations of wheel speeds, the robot can produce forward motion, sideways

motion, rotation, or a combination of all three. For example, driving all four wheels in the same direction produces forward motion. Driving the left and right wheel pairs in opposite directions produces yaw rotation. Driving diagonal wheel pairs in opposite directions produces lateral motion [8].

Wheel encoder measurements are commonly used to estimate robot motion. If an encoder measures an incremental tick count ΔN_i for wheel i, the corresponding wheel angular displacement is

$$\Delta\theta_i = 2\pi \frac{\Delta N_i}{N_{rev}} \quad \text{(Eq.1)}$$

where N_{rev} is the effective encoder resolution in ticks per wheel revolution. The wheel angular velocity is then estimated from the sample period Δt:

$$\omega_i = \frac{\Delta\theta_i}{\Delta t} \quad \text{(Eq. 2)}$$

For a four-wheel mecanum robot, the wheel angular velocities can be mapped into body-frame translational velocity and yaw rate. Using wheel radius r, half-wheelbase L, and half-track width W, the forward kinematic relationship is [8]

$$v_x = \frac{r}{4}(\omega_{FL} + \omega_{FR} + \omega_{RL} + \omega_{RR}) \quad \text{(Eq. 3)}$$

$$v_y = \frac{r}{4}(-\omega_{FL} + \omega_{FR} + \omega_{RL} - \omega_{RR}) \quad \text{(Eq. 4)}$$

$$\omega = \frac{r}{4(L+W)}(-\omega_{FL} + \omega_{FR} - \omega_{RL} + \omega_{RR}) \quad \text{(Eq. 5)}$$

where v_x is the robot-frame forward velocity, v_y is the robot-frame lateral velocity, and ω is the yaw rate. The subscripts (FL), (FR), (RL), and (RR) refer to the front-left, front-right, rear-left, and rear-right wheels.

YOLO Object Detection Model

YOLO, which stands for "You Only Look Once," is a family of real-time object detection models. Unlike two-stage detectors that first generate region proposals and then classify each region, YOLO performs object localization and classification in a single forward pass through a neural network. This makes YOLO well suited for robotics applications where perception must run quickly enough to support real-time decision making [19].

A YOLO model processes an input image and predicts a set of bounding boxes, class labels, and confidence scores. Each bounding box describes the location of a detected object in image coordinates. A typical bounding box is represented by its corner coordinates:

where u is the horizontal pixel coordinate and v is the vertical pixel coordinate. The center of the bounding box is often used as the object's image-space location:

$$u_c = \frac{u_{min} + u_{max}}{2} \quad \text{(Eq. 6)}$$

$$v_c = \frac{v_{min} + v_{max}}{2} \quad \text{(Eq. 7)}$$

Each detection also includes a confidence score, which represents the model's estimated likelihood that the detected object is valid. Detections below a selected confidence threshold are typically discarded to reduce false positives. YOLO models are commonly trained on labeled image datasets, where each training image includes bounding boxes and class labels for the objects of interest [19].

Computer Vision Pinhole Camera Model

The pinhole camera model is a mathematical representation of how a 3D point in space projects onto a 2D image plane. It assumes that light rays pass through a single optical center before intersecting the image plane. Although real cameras use lenses and introduce distortion, the pinhole model provides the foundation for camera projection and 3D reconstruction [20].

A 3D point in the camera coordinate frame can be represented as:

$$P_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \quad \text{(Eq. 8)}$$

where X_c, Y_c, and Z_c are the point coordinates relative to the camera. The corresponding pixel coordinate u, v is given by:

$$u = \frac{f_x X_c}{Z_c} + c_x \quad \text{(Eq. 9)}$$

$$v = \frac{f_y Y_c}{Z_c} + c_y \quad \text{(Eq. 10)}$$

where f_x and f_y are the focal lengths in pixels, and c_x and c_y are the principal point offsets of the camera.

If the pixel coordinate and depth value are known, the equations can be rearranged to recover the 3D camera-frame point:

$$X_c = \frac{(u - c_x)Z}{f_x} \quad \text{(Eq. 11)}$$

$$Y_c = \frac{(v - c_y)Z}{f_y} \quad \text{(Eq. 12)}$$

$$Z_c = Z \quad \text{(Eq. 13)}$$

These equations are commonly used with RGB-D cameras, where the RGB image provides pixel coordinates and the depth image provides the distance value Z.

The camera intrinsic matrix is commonly written as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq. 14)}$$

To express a point in another coordinate frame, such as a robot frame or world frame, a rigid-body transform is applied [21]:

$$p_b = T_{bc} p_c \quad \text{(Eq. 15)}$$

where p_c is the point in the camera frame, p_b is the point in the target frame, and T_{bc} is the homogeneous transform from the camera frame to the target frame.

LiDAR Occupancy Mapping

LiDAR mapping is a method for converting range measurements into a spatial representation of the robot's surrounding environment. A 2D LiDAR produces a planar scan by measuring distance at many angular positions around the sensor. Each measurement can be interpreted as a ray extending from the LiDAR origin to the measured endpoint. The endpoint provides evidence that an obstacle exists at that location, while the cells along the ray before the endpoint provide evidence that the space is free [18].

A 2D LiDAR scan can be represented as a set of range measurements

$$r_i = r(\theta_i) \quad \text{(Eq. 16)}$$

where (r_i) is the measured range at scan angle (θ_i). Each range measurement can be converted into Cartesian coordinates in the LiDAR frame using

$$x_i = r_i \cos(\theta_i) \quad \text{(Eq. 17)}$$

$$y_i = r_i \sin(\theta_i) \quad \text{(Eq. 18)}$$

where (x_i) and (y_i) describe the measured endpoint of the beam in the sensor coordinate frame.

An occupancy grid map divides the environment into square cells. Each cell represents a fixed-size region of physical space and is classified as occupied, free, or unknown based on accumulated sensor observations. Occupancy grids are useful for mobile robots because they convert raw LiDAR scan data into a map format that can be used for localization, obstacle avoidance, and path planning [18].

For each LiDAR beam, the mapping algorithm determines which grid cells the beam passes through before reaching its endpoint. Cells along the beam path are treated as free-space observations because the laser traveled through those regions without hitting an obstacle. The endpoint cell is treated as a hit observation because the laser

returned from that location. Over many scans, each grid cell accumulates evidence from repeated beam observations.

A common occupancy mapping method is to store an observation count for each grid cell. In this approach, each cell tracks how often LiDAR beams pass through the cell and how often beams terminate in the cell. The occupancy decision is then based on the ratio of hit observations to total beam observations [15]:

$$R_i = \frac{H_i}{P_i} \quad \text{(Eq. 19)}$$

where (H_i) is the number of beams that terminate in cell (i), and (P_i) is the number of beams associated with cell (i). A cell is classified as occupied when this ratio exceeds a selected occupancy threshold after enough observations have been collected. Cells with low hit ratios are treated as free space, while cells with insufficient observations remain unknown.

This observation-count approach improves robustness because LiDAR scans can contain noise, missed returns, reflections, and temporary objects. Requiring repeated observations before classifying cells helps the map represent stable structures such as walls, furniture, and large obstacles while reducing the effect of isolated false returns.

Simultaneous Localization and Mapping

Simultaneous Localization and Mapping, or SLAM, is the problem of estimating both the robot's pose and a map of the environment at the same time. This is necessary when a robot operates in an environment where no prior map is available [18].

The SLAM problem can be expressed probabilistically as:

$$P(x_{1:t}, m | z_{1:t}, u_{1:t}) \quad \text{(Eq. 20)}$$

where $x_{1:t}$ is the sequence of robot poses, m is the map, $z_{1:t}$ is the sequence of sensor measurements, and $u_{1:t}$ is the sequence of control inputs or odometry estimates.

The robot motion model predicts the next pose based on the previous pose and control input:

$$x_t = f(x_{t-1}, u_t) + w_t \quad \text{(Eq. 21)}$$

where w_t represents motion noise. The measurement model relates sensor observations to the robot pose and map:

$$z_t = h(x_t, m) + v_t \quad \text{(Eq. 22)}$$

where v_t represents sensor noise.

In LiDAR-based SLAM, scan matching is often used to align current LiDAR scans with previous scans or with an existing map. By finding the pose that best aligns the scan data with the map, the SLAM system can correct drift from wheel odometry and improve the estimated robot trajectory [18].

Extended Kalman Filter Localization

The Extended Kalman Filter, or EKF, is a recursive state estimator used for systems with nonlinear motion or measurement models. In robotics, it is commonly used to fuse multiple noisy sensor measurements into a single filtered estimate of the robot's state [18].

For a planar mobile robot, the state vector is often written as:

$$x = \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix} \quad \text{(Eq. 23)}$$

where x_r and y_r are the robot's position, and θ_r is its heading.

The EKF has two main steps: prediction and correction [18].

In the prediction step, the robot's state is propagated forward using a motion model:

$$\widehat{x_t^-} = f(\widehat{x_{t-1}}, u_t) \quad \text{(Eq. 24)}$$

The covariance is also propagated:

$$P_t^- = F_t P_{t-1} F_t^T + Q_t \quad \text{(Eq. 25)}$$

where P_t^- is the predicted covariance, F_t is the Jacobian of the motion model, and Q_t is the process noise covariance.

In the correction step, a sensor measurement is used to update the estimate. The innovation is:

$$y_t = z_t - h(\widehat{x_t^-}) \quad \text{(Eq. 26)}$$

The innovation covariance is:

$$S_t = H_t P_t^- H_t^T + R_t \quad \text{(Eq. 27)}$$

The Kalman gain is:

$$K_t = P_t^- H_t^T S_t^{-1} \quad \text{(Eq. 28)}$$

The state estimate is updated as:

$$\widehat{x_t} = \widehat{x_t^-} + K_t y_t \quad \text{(Eq. 29)}$$

The covariance is updated as:

$$P_t = (I - K_t H_t) P_t^- \quad \text{(Eq. 30)}$$

The EKF is useful because it allows odometry, IMU, LiDAR, and other sensor measurements to be combined according to their uncertainty.

Theta* Path Planning

Theta* is a grid-based path-planning algorithm that extends A* by allowing line-of-sight shortcuts between nodes. While A* searches through neighboring grid cells and often produces paths that follow grid directions, Theta* can generate smoother and shorter paths by connecting a node directly to an earlier parent when there is a clear line of sight [22].

Like A*, Theta* uses a cost function:

$$f(n) = g(n) + h(n) \quad \text{(Eq. 31)}$$

where $g(n)$ is the cost from the start node to node n, and $h(n)$ is a heuristic estimate of the cost from node n to the goal. A common heuristic for 2D planning is Euclidean distance:

$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \quad \text{(Eq. 32)}$$

Theta* differs from A* in how it assigns parent nodes. If the parent of the current node has a clear line of sight to a neighboring node, the neighbor can be connected directly to that parent. This reduces unnecessary turns and produces a path that is less constrained by the grid.

The output of a planner is typically a sequence of waypoints:

$$P = \{p_1, p_2, \dots, p_n\} \quad \text{(Eq. 33)}$$

where each waypoint p_i represents a position in the map. A path-following controller then uses these waypoints to generate motion commands.

Pure Pursuit Control

Pure pursuit is a geometric path-following controller commonly used in mobile robotics. Instead of commanding the robot directly toward the final goal, the controller selects a lookahead point along the planned path and generates a motion command that drives the robot toward that point [23].

The lookahead point is chosen a distance (L_d) ahead of the robot along the path. After the lookahead point is selected, it is transformed into the robot's local coordinate frame. If the lookahead point has local coordinates (x_L, y_L), the heading error is

$$\alpha = \text{atan2}(y_L, x_L) \quad \text{(Eq. 34)}$$

where (x_L) is the forward distance to the lookahead point and (y_L) is the lateral offset.

The distance from the robot to the lookahead point is

$$L = \sqrt{x_L^2 + y_L^2} \quad \text{(Eq. 35)}$$

For a pure-pursuit controller, the curvature command can be computed from the lateral offset and lookahead distance as or, substituting the coordinate expression for (L),

$$\kappa = \frac{2y_L}{x_L^2 + y_L^2} \quad \text{(Eq. 36)}$$

The angular velocity command is then computed from the commanded forward velocity and curvature:

$$\omega = K_\omega v \kappa \quad \text{(Eq. 37)}$$

where (v) is the commanded forward velocity, (ω) is the commanded yaw rate, and (K_ω) is a yaw-rate gain used to tune controller aggressiveness.

Pure pursuit is widely used because it is simple and stable for waypoint tracking. A larger lookahead distance generally produces smoother motion but may cut corners, while a smaller lookahead distance tracks the path more closely but can produce oscillation.

III. Requirements

Marketing Requirements and Pairwise Hierarchy

Marketing requirements were prioritized using the Analytic Hierarchy Process (AHP) to quantify the relative importance of the project's primary objectives. The resulting top-level weights indicate that Autonomous operation is the dominant priority (0.58), followed by Dexterity (0.23) and Traversability (0.13), while Portability is the least critical requirement (0.06). As shown in Table I, the pairwise comparison matrix reflects strong preference ratios favoring autonomy over the other criteria, confirming autonomy as the most prioritized attribute and portability as the least prioritized.

TABLE I. PAIRWISE COMPARISON MATRIX

Criterion	Portable	Traversable	Dexterity	Autonomous
Portable	1	0.33	0.25	0.14
Traversable	3	1	0.5	0.2
Dexterity	4	2	1	0.33
Autonomous	7	5	3	1

Engineering Specifications

Engineering specifications were developed by decomposing the project's marketing requirements into objective, measurable performance targets suitable for design validation. Each specification is written in a testable form (threshold + units + operating condition where applicable) and is paired with a verification method to ensure compliance can be demonstrated at the system or subsystem level.

As summarized in Table II, the requirements set spans the major design drivers for the LitterBot—including mobility and maneuverability, payload/collection performance, power and runtime constraints, and sensing/compute interfaces. Table II also explicitly maps each requirement to a verification approach (e.g., inspection, analysis, demonstration, or test), which establishes a clear pass/fail pathway and prevents ambiguous "works well" claims during integration and final evaluation.

TABLE II. ENGINEERING SPECIFICATIONS

Spec. ID	Marketing Requirements	Engineering Specification	Verification
ES-01	Portable	Weight <= 55 lb; runtime >= 2 h	Physical and Subsystem Verification
ES-02	Portable	Volume <= 1.77 ft^3	Physical and Subsystem Verification

ES-03	Portable / Modular	Reassembly/service access <= 120 s	Physical and Subsystem Verification
ES-04	Traversable	Ground clearance >= 2 in	Physical and Subsystem Verification
ES-05	Traversable	Traverse flat floor seams/low obstacles without loss of motion	Approach and Navigation Success Test
ES-06	Traversable	Turn radius <= 20 in	Approach and Navigation Success Test
ES-07	Traversable	Speed range = 0.33–3.28 ft/s	Approach and Navigation Success Test
ES-08	Dexterity	Scoop torque >= 7.4 lbf-ft; scoop width >= 10 in	Scoop Success Test
ES-09	Dexterity	Collection mechanism >= 2 DOF	Scoop Success Test
ES-10	Dexterity	Scoop/arm motion >= 120 deg or equivalent pickup stroke	Scoop Success Test

ES-11	Autonomous	Camera FOV >= 120 deg	Trash Detection Accuracy Test
ES-12	Autonomous	Detection accuracy >= 90% at <= 10 ft	Trash Detection Accuracy Test
ES-13	Autonomous	Final approach error <= 2 in	Approach and Navigation Success Test
ES-14	Autonomous	Replan/update rate >= 2 Hz; tracking error <= 2 in avg.	Trajectory Planning and Control Test
ES-15	Autonomous / Traversable	Obstacle detect distance >= 18 in; stop/replan <= 0.3 s	Obstacle Avoidance Test
ES-16	Autonomous / Safety	E-stop response <= 0.3 s; status visible >= 15 ft	End-to-End Autonomy Test / Physical Verification

IV. Design

System Architecture

The LitterBot system architecture separates high-level autonomy from low-level real-time control. The robot uses a two-layer computation structure so that perception, mapping, planning, and mission logic can run on a Linux-based computer while time-critical actuator control is handled by a dedicated microcontroller.

The high-level layer is implemented on an NVIDIA Jetson Orin Nano running ROS 2. This layer processes camera and LiDAR data, detects litter, estimates target locations, maintains the navigation map, plans paths, and generates velocity commands. The Jetson is responsible for deciding where the robot should move and when the collection sequence should begin [9], [10].

The low-level layer is implemented on an STM32L4A6ZG microcontroller. This layer receives motion and actuator commands from the Jetson and executes deterministic control of the wheel motors, encoders, stepper motor, and servo motor. The STM32 also provides safety-related behavior such as motor timeout handling and immediate stop commands if communication is interrupted [4].

Communication between the two layers occurs through a UART interface. The Jetson sends wheel-speed, stop, and scoop-related commands to the STM32, while the STM32 returns encoder and status information for feedback. This separation allows

the robot to combine flexible autonomous decision-making with reliable real-time control of the physical hardware.

At the system level, the robot follows a detect, localize, plan, approach, collect, and resume-search workflow. Sensor data is processed on the Jetson to identify litter and generate navigation commands. The STM32 converts those commands into physical motion, allowing the rover to approach a target, execute the scoop sequence, and continue searching for additional litter. Figure 1 presents the final Litterbot protype including sensors, mechanical assembly, and electrical wiring.

Figure 1: Completed LitterBot Prototype with Integrated Drive, Collection, Sensing, and Control Subsystems

Mechanical Design

Mechanical design focused on creating a rigid, modular chassis that can support a multi-sensor autonomy stack while maintaining a low center of gravity and accessible service points. The structure was organized to separate high-mass components (battery and motor electronics) from compute and sensing hardware, reduce wiring complexity through short routing paths, and provide clear mounting surfaces for future expansion (arm, scoop, and enclosure panels).

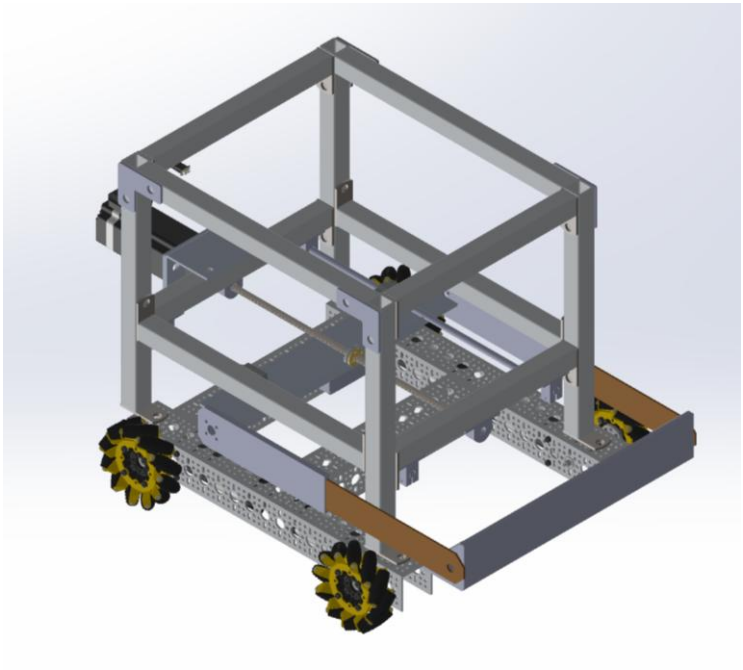

Figure 2. CAD Model - Isometric View of Full Chassis

The rover chassis is built on a goBILDA aluminum extrusion frame with a three-level structure, as seen in Figure 2. The bottom level houses the LiPo battery in a front

cavity and the STM32 NUCLEO-144 in a rear cavity inside a custom 3D-printed PLA housing, with four Cytron MD10C motor drivers in the side cross-struts. The middle level supports the Jetson Orin Nano and CL57T stepper driver on a cut aluminum sheet. The top level carries the RPLiDAR, RealSense camera, and buck converters.

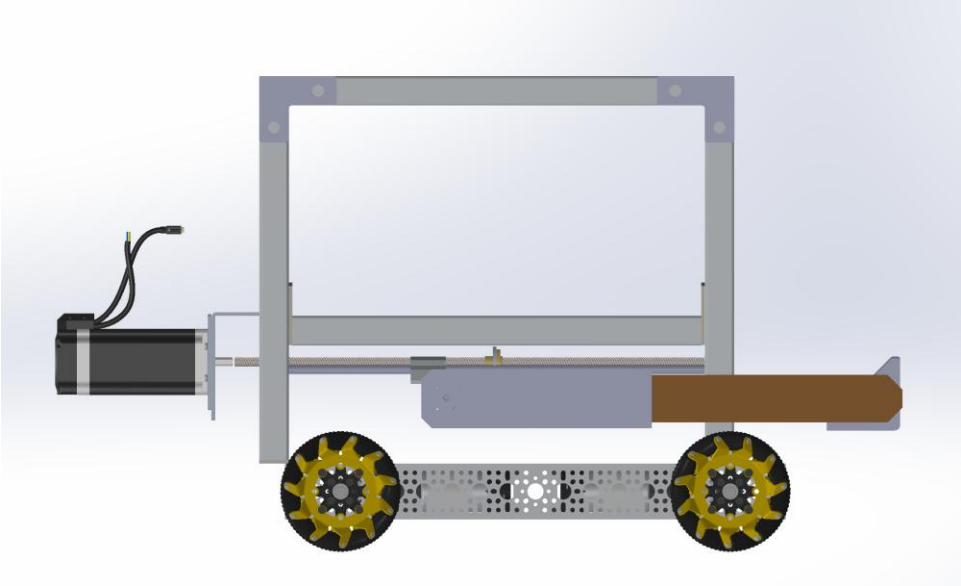

Figure 3. CAD Model - Side View with Stepper Motor and Mecanum Wheels

The rover uses four goBILDA Yellow Jacket 5203-2402-0019 planetary gear motors (19.2:1 gear ratio, 312 RPM at 12 V, with integrated hall-effect quadrature encoders producing 537.7 counts per revolution in 4x counting mode) driving 104 mm mecanum wheels (Figure 3) [7]. Mecanum wheels enable omnidirectional movement for precise positioning when approaching litter. Each motor is driven by a Cytron MD10C driver (13 A continuous (max), sign-magnitude PWM)[3]. The collection

mechanism is a two-degree-of-freedom arm: a CL57T closed-loop stepper [5] for vertical linear actuation via lead screw and linear slides, and an HX-65HM bus servo (65 kg-cm, 0.3 deg precision) for scoop rotation [6]. The arm has a 13-inch reach plus arm coupling. Figure 4 shows the arm rigidly attached to the end effector. The arm then binds to rods connected to the servo motor, providing rotation to the scooping mechanism.

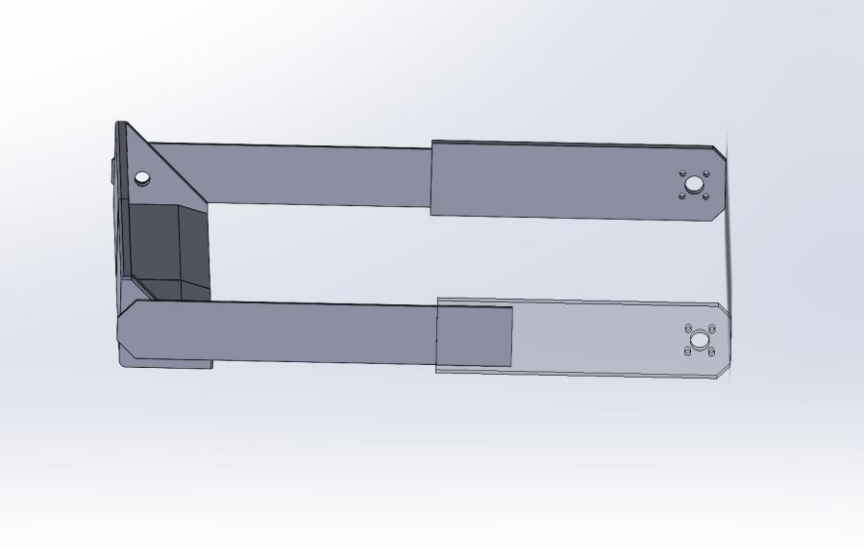

Figure 4. CAD Model – Arm and End Effector

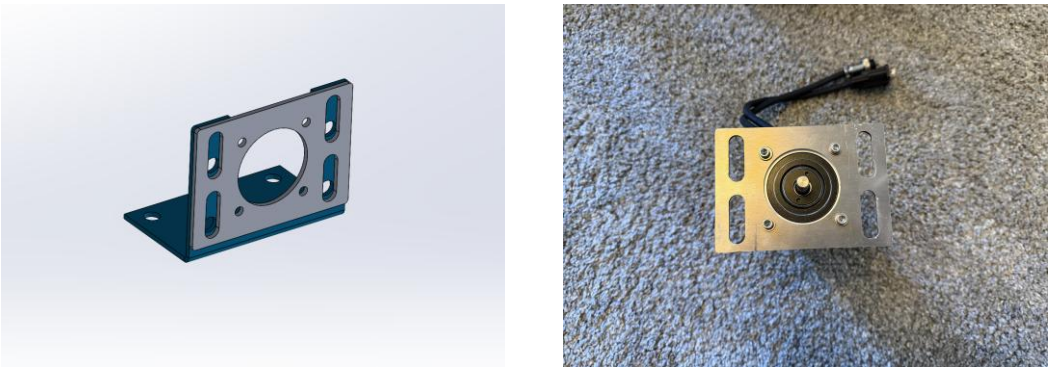

Figure 5. CAD Model - Stepper Motor Mounting Bracket

The linear actuation axis uses a stepper-driven lead screw to convert rotary motion into controlled linear travel of the end-effector platform. The stepper motor is mounted to the back of the aluminum frame using a custom machined stepper mounting bracket (Figure 5). Parallel linear rail slides constrain the platform to a single translation axis, maintaining alignment and providing smooth, repeatable motion across the full stroke. The linear slides were then assembled and validated for the full range of motion Figure 6.

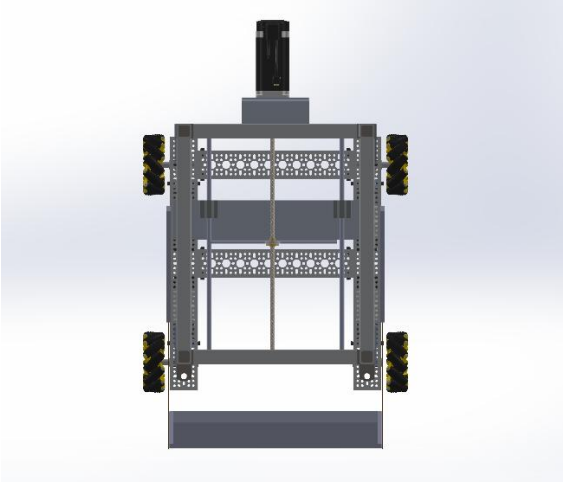

Figure 6. Lead Screw and Linear Slides - Top-Down View

The rover uses a dovetail ramp design, 3D-printed in four interlocking sections. Fasteners attach at the top and bottom of the ramp shown in different views in Figure 7, and E6000 adhesive was applied between the interlocking sections. The design was iterated through three revisions based on printability, fit to chassis, and durability.

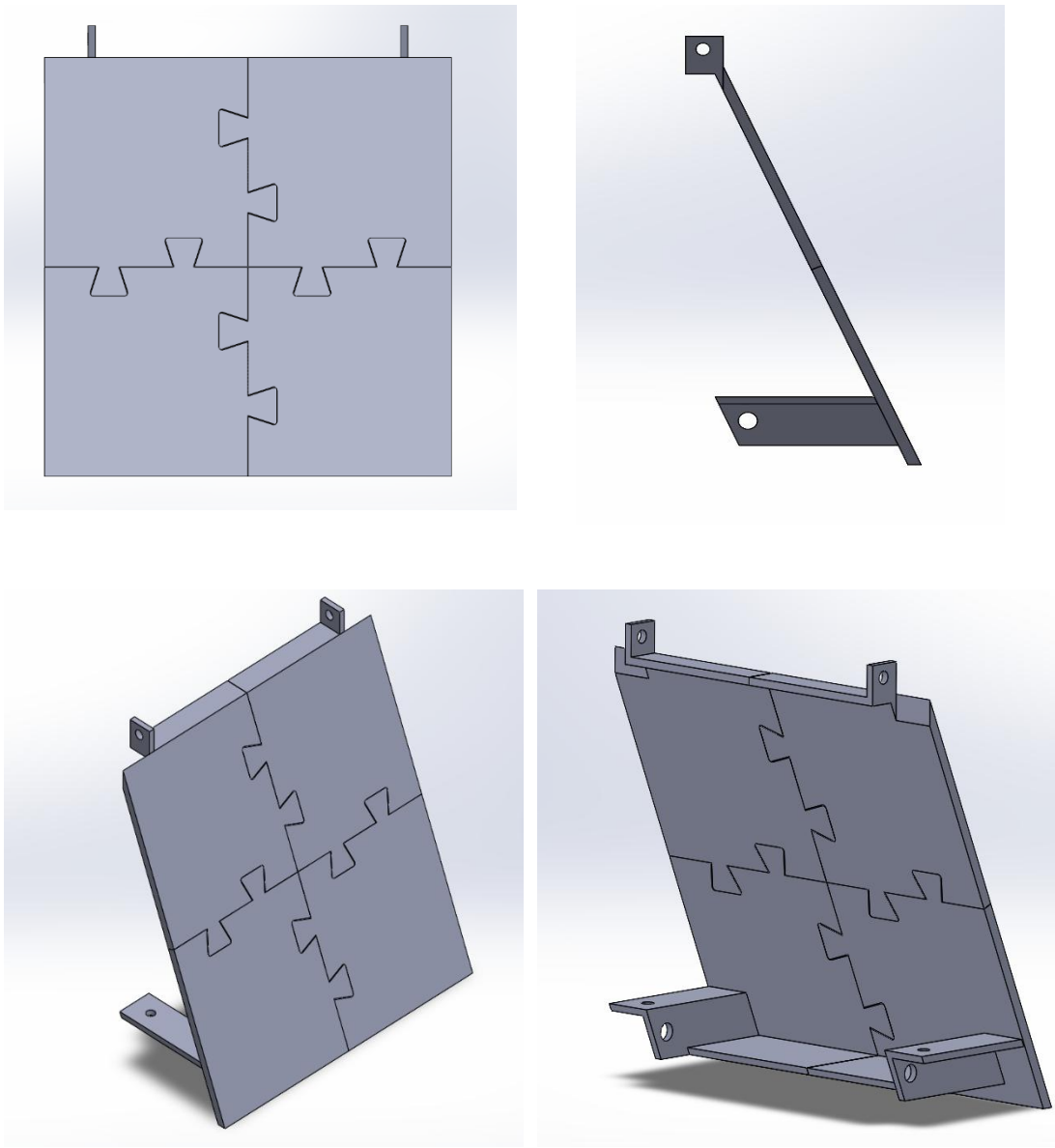

Figure 7. 3D-Printed Ramp Piece

Figure 8. 3D-Printed End-Effector Sections Laid Out Before Assembly

Custom PLA components include the servo mount, the STM32 housing (iterated once to improve cable routing) shown in Figure 9, the ramp sections shown in Figure 7, the arm couplers shown in Figure 4, and the end-effector components shown in Figure 8. All printed parts were designed with ≥ 0.5 mm clearance to account for typical 3D-printing tolerances.

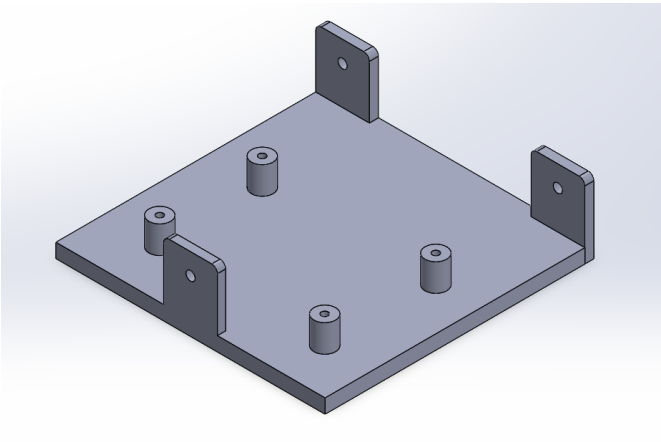

Figure 9. 3D-Printed STM32 Housing

The litter collection bin was designed as a lightweight, removable storage container for collected debris. As shown in Figure 10, the bin consists of interlocking 3D-printed sections that were assembled into a rigid enclosure and mounted within the rover chassis. The modular design allows the bin to be printed on smaller-format printers while simplifying replacement of damaged sections. The bin is positioned behind the collection ramp so that litter captured by the scoop mechanism is deposited directly into the storage compartment. The final assembly demonstrated sufficient rigidity for repeated testing and provided adequate capacity for the expected quantity of litter collected during autonomous operation.

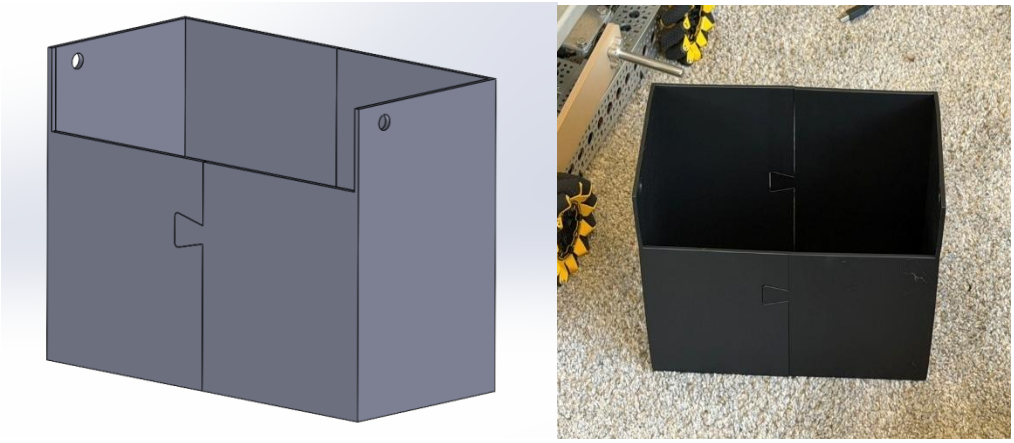

Figure 10. Trash Bin CAD Model and Physical Print

The perception sensors, an Intel RealSense D455 depth camera [14] and an RPLiDAR A1 [24], are both mounted on the uppermost plate of the chassis frame Figure 11. Precise and rigid sensor placement is critical for the localization pipeline, as the pinhole back-projection and TF transform chain described in the previous

section depend on accurately known sensor positions and orientations relative to the robot's base frame.

The camera is attached to the mount plate via a swivel joint, allowing the pitch angle to be adjusted mechanically during setup. Rather than relying on a physical protractor or fixed detent to set the downward tilt, the team used the RealSense's onboard IMU to measure the camera's actual pitch angle in software. This approach allows the pitch to be adjusted and re-calibrated without disassembling the mount or re-measuring with external tools. The RPLiDAR A1 is mounted on the same top plate as the camera, positioned 127 mm behind the robot center at a height of 393.7 mm. The LiDAR is oriented with its scan plane facing rearward (rotated 180° in yaw relative to the robot's forward axis). All fasteners were sealed with Loctite adhesive to prevent against loosening due to vibration.

Figure 11: Machined Camera Mount for D455 Camer

Electrical Design

A 24 V, 20 Ah lithium-ion battery pack (7-series configuration) in the front cavity supplies all subsystems. Buck converters generate 12 V (motor drivers, servo) and 19V (Jetson), while the STM32 Nucleo Board supplies 3.3 V (for encoders). The STM32 is powered from the Jetson via USB. Wheel and scooping motions do not run simultaneously, so battery sizing targets the higher of the two peak draws. The current required by the LitterBot is highest when in motion rather than scooping. Table III specifies the typical and peak current drawn by the LitterBot's peripherals.

TABLE III. POWER BUDGET

Subsystem	Voltage	Typical Current	Peak Current
DC Motors (4x)	12 V	2.4 A (0.6 A each)	36.8 A
Stepper Motor	24-48 V	~3 A	~6 A
Bus Servo	9-12.6 V	0.4 A	5 A (stall)
Jetson Orin Nano	19 V	~1.4 A	5 A (25 W peak)
STM32 NUCLEO-144	5 V (USB)	~100 mA	~150 mA
Encoders (4x)	3.3 V	~80 mA total	~80 mA
LiDAR	5 V (USB)	~300 mA	~600 mA
RealSense Camera	5 V (USB)	N/A	~700

Figure 12 shows the system wiring organized around a 24V battery bus that feeds two regulated rails and the control electronics. A 24V, 20 Ah Li-ion battery supplies the

main power distribution. From this bus, a 24→12V buck converter provides the 12V rail used by the four DC motor drivers and auxiliary actuators, while a 24→19V buck converter supplies the Jetson Orin Nano input rail. The STM32 Nucleo provides the low-voltage control domain, with 3.3V logic distributed to sensors and signal interfaces as shown.

Four mecanum wheel DC motors are driven by Cytron MD10C motor drivers (FL, FR, RL, RR), each powered from the 12 V rail (PWR+/PWR−) and connected to its corresponding motor outputs (MTR+/MTR−) [3]. Motor speed is commanded using 20 kHz PWM generated on TIM2 channels 1–4 (PA0–PA3), providing one hardware PWM channel per wheel. Motor direction is controlled via dedicated GPIO direction pins: PB0 (FL), PB1 (FR), PB2 (RL), and PF14 (RR) [4]. This mapping matches the four driver “PWM/DIR” control pairs illustrated in Figure 12.

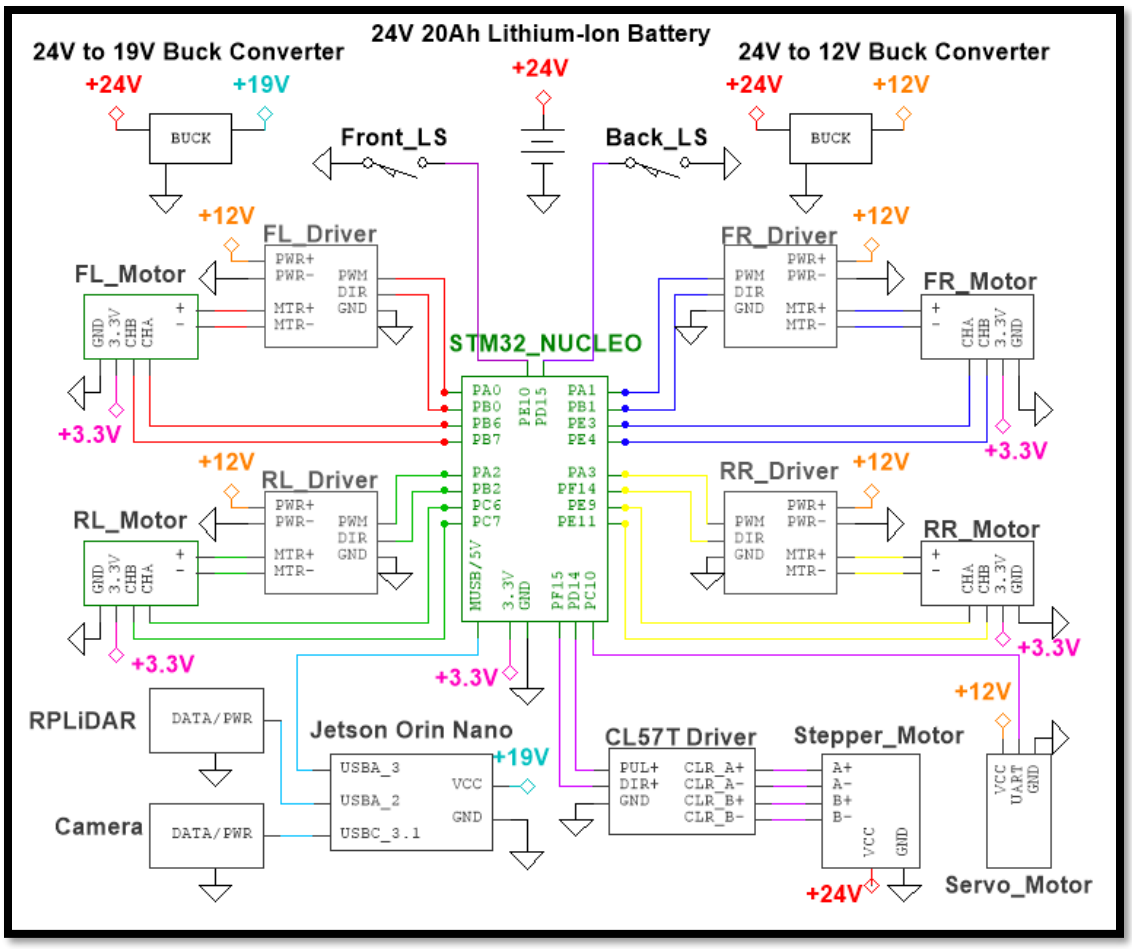

Figure 12. Wiring Diagram

The vertical/arm mechanism is driven by a stepper motor through a CL57T stepper driver (Figure 12, right side). Step pulses and direction are generated by the STM32 using GPIO timing: STEP on PD14 and DIR on PF15, with microsecond-accurate pulse timing produced by bit-banging referenced to the DWT cycle counter [5]. Although the CL57T nominally specifies 5 V logic, operation at 3.3 V control levels was verified during bench testing.

A HX-65HM bus servo is integrated on the same control domain and is powered from the 12 V rail as shown. Servo communication uses USART3 TX on PC10 in

1,000,000 baud half-duplex mode. The servo uses a binary packet protocol with checksum ~(ID + Length + Cmd + Params) & 0xFF [6]. In firmware initialization, CubeMX's default open-drain configuration was overridden to push-pull in the USART3_MspInit user code section to meet the servo bus signaling requirements. Each wheel includes a quadrature encoder, routed directly to STM32 hardware timer encoder interfaces for zero-overhead counting (Figure 12, encoder connections). Effective resolution is 537.7 counts/rev at the wheel after the 19.2:1 gearbox in 4× decode mode [7]. Timers are assigned as follows: FL—TIM4 (PB6/PB7), FR—TIM3 (PA6/PA7), RL—TIM8 (PC6/PC7), RR—TIM1 (PE9/PE11). Using hardware decoding eliminates CPU load and provides consistent odometry timing for closed-loop control.

High-level perception and navigation sensors interface directly with the Jetson Orin Nano. The Intel RealSense D455f was selected because it provides synchronized RGB and depth data, allowing the system to both detect litter and estimate target distance using a single sensor [14]. The depth capability supports the autonomous requirement for target localization and final approach accuracy by providing three-dimensional position estimates for detected litter. The camera's wide field of view also helps satisfy the requirement for detecting objects across the robot's operating area [14].

The RPLiDAR A1 was selected to provide 360° environmental perception for mapping, localization, and obstacle avoidance [24]. Unlike camera-based sensing, LiDAR measurements are largely independent of lighting conditions and provide

reliable distance measurements for occupancy-grid generation. The sensor's range and full rotational coverage support the traversability and autonomy requirements by enabling obstacle detection, SLAM map generation, and safe navigation through cluttered environments. As shown in Figure 12, both sensors connect directly to the Jetson via USB, and the LiDAR is mounted on the upper level of the chassis to maintain an unobstructed field of view.

Firmware Design (STM32)

The STM32 firmware is structured as a set of modular driver and test files, with each hardware interface isolated into its own source module for easier validation and integration. The system runs on an STM32L4A6ZG [4] clocked at approximately 47.333 MHz, derived from the 4 MHz MSI oscillator through the PLL (N = 71, R = 6).

Key CubeMX/peripheral configurations are as follows. TIM2 is configured for four-channel PWM motor control [3] with PSC = 0 and ARR = 2366, producing an output frequency of approximately 20 kHz. The HX-65HM servo bus uses USART3 in half-duplex mode on PC10, with the baud rate overridden in firmware to 1,000,000 baud. A separate LPUART1 interface provides debugging telemetry on PG7/PG8 at 115200 baud. Wheel odometry is handled using hardware quadrature decoding on TIM1, TIM3, TIM4, and TIM8 in TI1+TI2 encoder mode, with a 65535 counter period to maximize measurement range while maintaining zero CPU. Table IV summarizes the firmware modules and their roles in the overall STM32 software architecture.

TABLE IV. FIRMWARE MODULE SUMMARY

Module	Files	Purpose
cytron_md10c	.h / .c	4-wheel DC motor control (TIM2 PWM, ~20 kHz)
cl57t_stepper	.h / .c	Stepper motor control (GPIO bit-bang, DWT delays)
hx65_servo	.h / .c	Bus servo half-duplex UART protocol (1 Mbaud)
encoder	.h / .c	Quadrature encoder driver (TIM1/3/4/8, 537.7 ticks/rev)
odometry	.h / .c	Mecanum wheel odometry (dead-reckoning pose)
uart_cmd	.h / .c	UART command parser for teleop (250 ms timeout)
servo_test	.h / .c	Servo sweep test (B1-triggered, 6-position)
dc_motor_test	.h / .c	DC motor test (B1-triggered, individual + all)
stepper_test	.h / .c	Stepper test (B1-triggered, 3 speeds)
combined_test	.h / .c	Combined peripheral integration test
encoder_test	.h / .c	Encoder test (B1-triggered, LD3, LPUART1 output)

The encoder interface reads the hardware quadrature counters on TIM1, TIM3, TIM4, and TIM8, computes signed incremental tick counts for each wheel, and corrects for 16-bit counter rollover. These signed tick increments are the raw feedback used by the odometry module to estimate chassis motion. For each update cycle, the firmware converts the tick change from each wheel into an angular displacement using the encoder conversion relationship introduced in Eq. 1. The wheel angular displacement is then divided by the odometry sample period to estimate each wheel's angular velocity, as described in Eq. 2.

The four wheel angular velocities are then passed into the mecanum forward-kinematics model introduced in Eq. 3–5 [8]. This model converts the individual wheel speeds into the robot body-frame velocity estimate, consisting of forward velocity, lateral velocity, and yaw rate. This is necessary because the LitterBot drivetrain can

move omnidirectionally: the robot may drive forward, strafe sideways, rotate in place, or combine translation and rotation during alignment with a detected litter target.

For the LitterBot chassis, the odometry calculation uses an effective encoder resolution of (N = 537.7) ticks/rev after gearbox scaling and 4x quadrature decoding [7]. The physical chassis constants used in the forward-kinematics calculation are (r = 0.052) m, (L = 0.168) m, and (W = 0.2025) m, where (r) is the mecanum wheel radius, (L) is the half-wheelbase, and (W) is the half-track width. These dimensions define the geometry used to convert wheel motion into chassis motion.

The resulting odometry estimate is used as short-term motion feedback for the robot and is also reported to the higher-level autonomy stack. Because mecanum wheels can experience slip during strafing, turning, or rapid acceleration, the encoder-based odometry is not treated as an absolute position measurement. Instead, it provides a local motion estimate that supports localization, path following, and final approach behavior when combined with LiDAR mapping and heading estimation. Figure 13 identifies the chassis dimensions used in this mecanum odometry calculation.

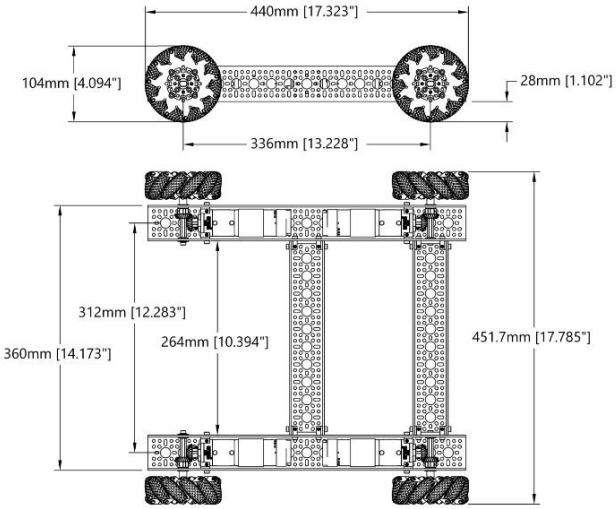

Figure 13. Chassis Dimensions Used in Calculating Forward Kinematics

The UART command parser (uart_cmd) provides a simple, human-readable control interface between the Jetson and the STM32. It continuously monitors the serial stream for newline-terminated ASCII packets and dispatches each valid command to the appropriate control routine.

Primary motion commands are sent using:

- M fl fr rl rr\n — Sets the four wheel commands for front-left (FL), front-right (FR), rear-left (RL), and rear-right (RR). Each field is a signed integer in the range −1000 to +1000, where the sign defines direction and the magnitude defines requested speed (scaled internally to PWM duty cycle).

Safety and diagnostic commands include:

- X\n — Emergency stop (E-stop). Immediately forces all motor outputs to zero, disables further motion commands until explicitly cleared by the control logic (implementation-dependent), and resets any active command state.
- T\n — Encoder test mode. Triggers an encoder readout/verification routine (e.g., prints tick counts or computed deltas) to validate quadrature decoding and wiring during bring-up.

To prevent runaway motion in the event of a communication failure, the parser enforces a command watchdog timeout. If no valid motion command is received within 250 ms, the firmware automatically commands all wheel outputs to zero (failsafe stop) until a new valid M command is received. This timeout is applied regardless of the last commanded value, ensuring the robot cannot continue driving if the Jetson process halts, the USB/UART link drops, or packets become corrupted.

Software Design (Jetson / ROS 2)

Figure 15 summarizes the autonomy workflow for LitterBot. The software stack is developed using ROS 2 Humble and maintained in the project GitHub repository (see Appendix E), which serves as the central location for all source code, firmware, and configuration files. The NVIDIA Jetson Orin Nano runs Ubuntu 22.04 with ROS 2 Humble [9], [10] to initialize the sensing stack (RPLiDAR, Intel RealSense D455 depth camera, USB camera, and IMU) and execute target-driven litter collection over the operating area. Low-level motor control, encoder feedback, and the scoop pickup

sequence are handled by the STM32L4A6ZG microcontroller, which communicates with the Jetson over a 115200-baud UART serial link. Figure 14 shows a simplified view of the LitterBot autonomy pipeline, illustrating the high-level data flow between functional subsystems: perception, localization, planning, and control. A full diagram showing the complete ROS 2 node and topic architecture can be found in Appendix G: Reference Material.

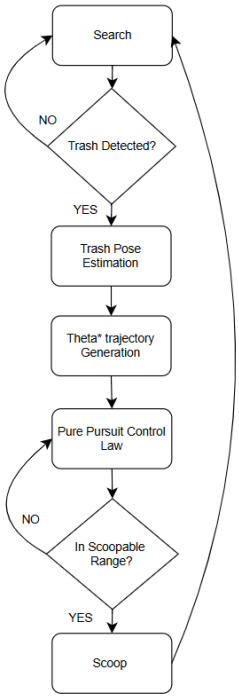

Figure 14. Operation Flow of Software

During coverage, the perception system continuously monitors the camera stream for litter. When trash is detected, the robot transitions from coverage into a target approach routine, localizes the object using RGB-D data, plans a path to the target,

executes the scoop sequence, and then resumes coverage until the operating area is complete or a low-battery condition triggers return-to-base behavior.

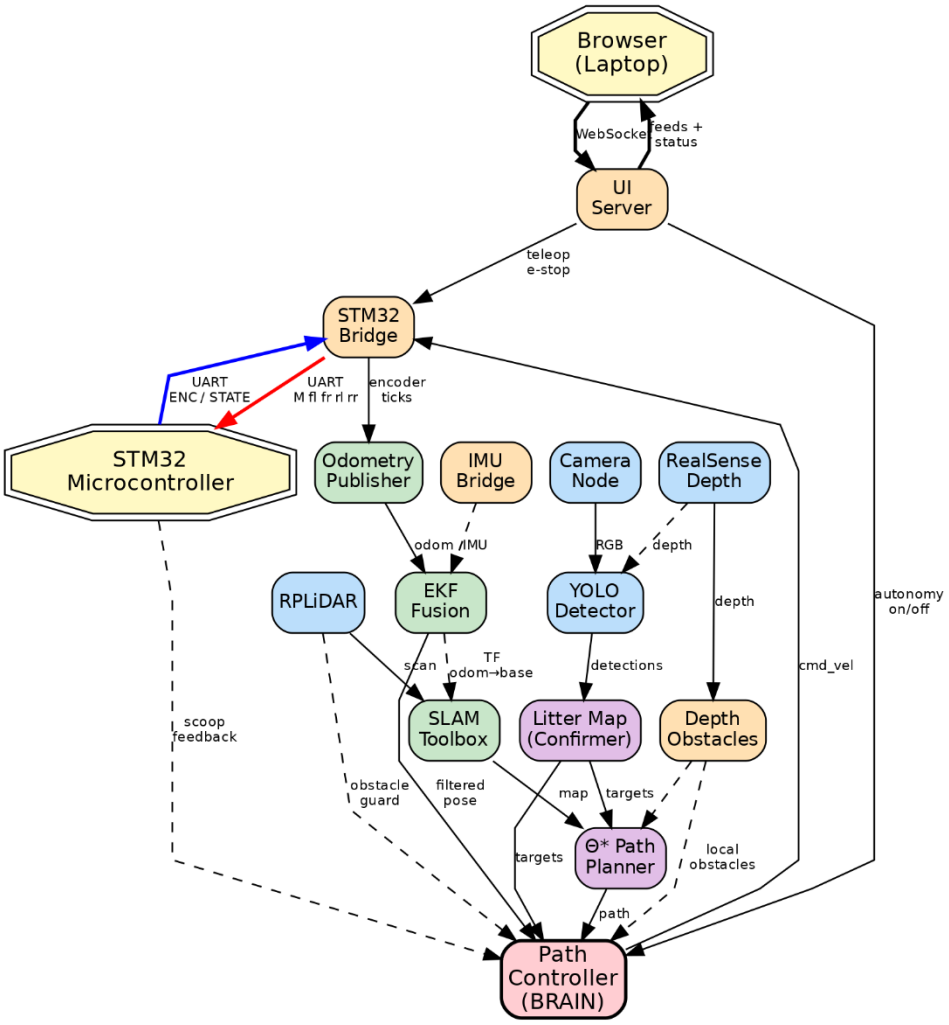

Figure 15: Simplified Software Architecture Diagram

At the software level, this workflow is organized as a ROS 2 node graph that connects perception, localization, mapping, planning, motion control, and embedded actuation. Figure 14 provides a high-level view of the ROS 2 node architecture and the primary

data flows between sensing, autonomy, and embedded control components. The Jetson receives LiDAR scans from the RPLiDAR, RGB-D data from the Intel RealSense D455f, and encoder feedback from the STM32. Velocity commands are published to /cmd_vel, then converted by the UART bridge into the STM32 wheel command format. Encoder-based odometry is published on /odom, while the mapping stack publishes the occupancy grid on /map. Litter detections are published on /detections, confirmed target markers are published on /litter_map, local depth-based obstacles are published on /local_obstacle_map, and planned target paths are published on /litter_path. These topic names are used throughout the Autonomy Design section to describe how information moves through the system from detection to pickup [10]. The full ROS2 node diagram can be found linked in Appendix G.

Robot Visualization (Rviz)

RViz [10] is used as the primary visualization and debugging tool for the ROS 2 autonomy stack. During development, RViz allows the team to view the robot pose, coordinate frames, sensor data, map data, detected litter targets, local obstacle data, and planned paths in a single interface. This visualization is important because many autonomy failures are caused by frame mismatches, stale transforms, incorrect map alignment, or target positions being published in the wrong coordinate frame.

The occupancy grid published on /map is displayed in RViz to verify that the LiDAR and SLAM pipeline are generating a usable representation of the environment. The robot odometry published on /odom and the TF tree [21] are displayed to verify that

the base_link, odom, map, camera, and LiDAR frames remain consistent during motion. LiDAR scan data is visualized to confirm that walls, furniture, and test obstacles are detected in the correct physical locations. Confirmed litter targets from /litter_map are displayed as markers so the team can verify that camera detections and depth-based localization are being transformed into the navigation frame correctly.

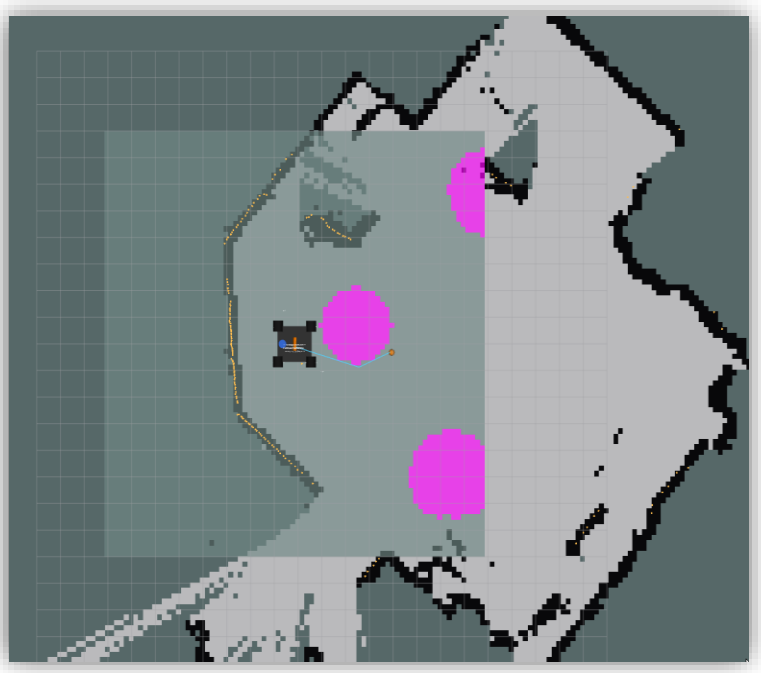

Figure 16: RViz Occupancy Map During Autonomy Test

RViz is also used to evaluate planning and obstacle behavior. The planned path published on /litter_path is displayed over the occupancy grid, as seen in Figure 16, to verify that the planner generates a route from the robot pose to the selected litter target without crossing occupied cells. Local depth-based obstacle data from /local_obstacle_map can be shown to confirm that nearby hazards detected by the

RealSense are included in planning or stopping behavior. During testing, the team can compare the planned path, robot pose, and target markers in real time to determine whether errors are caused by detection, localization, mapping, planning, or low-level control.

User Interface

The LitterBot user interface provides an operator-facing control and monitoring layer for testing, demonstration, and safe operation. The interface allows the operator to observe the current robot state, enable or disable autonomy, issue teleoperation commands, monitor sensor and communication status, and trigger safety or subsystem actions when needed. This is especially useful during integration because the system includes multiple interacting subsystems: perception, mapping, planning, motor control, and scoop actuation.

During testing, the user interface supports manual teleoperation by allowing the operator to command forward, reverse, turning, and stop behavior through the Jetson-to-STM32 command pipeline. This allows the drivetrain, UART bridge, motor drivers, and encoder feedback to be validated before full autonomy is enabled. The interface also supports system-level monitoring, including whether ROS nodes are active, whether camera and LiDAR data are being received, whether the STM32 is

communicating correctly, and whether the robot is in teleoperated, autonomous, stopped, or faulted operation.

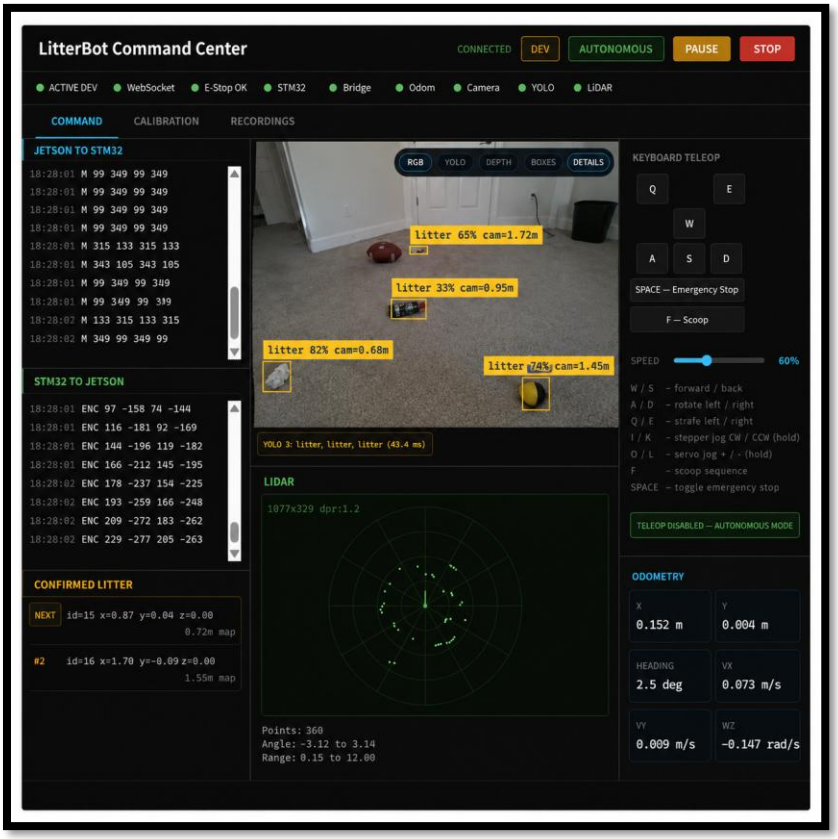

Figure 17: LitterBot Command Center User Interface

Figure 17 shows the LitterBot Command Center user interface used for monitoring system status, viewing sensor data, and controlling robot operation.The interface also supports safety and demonstration functions. An emergency-stop control can be used to immediately command the robot to stop motion, while status indicators can show whether the robot is actively searching, approaching a target, executing a scoop,

recovering, or waiting for operator input. Litter detection feedback, live camera information, target status, and scoop/deployment controls help the team confirm that the robot is responding to detected targets as expected. For final demonstrations, the user interface provides a clear way for an observer or operator to understand what the robot is doing without needing to inspect terminal output or ROS topic logs directly.

Robotic Arm and Scoop Trajectory Design

The litter collection mechanism was designed as a two-actuator robotic arm that combines linear arm motion with coordinated bucket rotation. The arm structure was modeled in CAD so the scoop could move from the floor-level pickup position, travel up the ramp, and deposit litter near the lip of the internal collection bin. The arm length is 16.5 in, and the end-effector scoop width is 14.5 in. At full extension, this geometry creates an effective scoopable area of approximately 4.5 in by 14.5 in. The taught scoop trajectory spans 10.23 in of linear translation and approximately 114.5 degrees of bucket rotation.

The arm uses two main actuators. A NEMA 23 stepper motor with a CL57T closed-loop driver controls the extension and retraction of the scoop arms. This motor drives the scoop forward toward the litter and then pulls it back up the ramp toward the bin. A Hiwonder HX-65HM bus servo controls the bucket angle. The servo is mounted on a dual-shaft joint so one servo rotates both sides of the scoop linkage together. The

stepper controls the scoop's linear position, while the servo controls the bucket angle throughout the pickup, carry, and dump motion.

The scoop trajectory was tuned experimentally using the web UI. The operator manually teleoperates the stepper and servo to move the scoop through the desired physical path. At each important point along the path, the UI captures a waypoint containing both the current stepper position and current servo position:

$$P_i = (s_i, q_i)$$

where s_i is the arm position in stepper steps and q_i is the servo position in HX-65 units from 0 to 4095. Up to 25 full arm poses can be saved. After the desired trajectory is captured, the UI sends the saved waypoints to the STM32, where they are written into flash memory. This allows the trajectory to persist after power cycles without requiring firmware recompilation.

When a scoop command is triggered, the STM32 first homes the arm using the limit switches so the trajectory starts from a known mechanical reference. After homing, the firmware moves from the home position to the first saved pose, then plays back the saved waypoint sequence. During playback, the stepper and servo move together

between each pair of saved poses, causing the scoop to follow the manually tuned path through the pickup area, up the ramp, and toward the bin lip.

Autonomy Design

The LitterBot autonomy stack is implemented on the NVIDIA Jetson Orin Nano and is responsible for perception, target localization, litter confirmation, mapping, path planning, path following, obstacle handling, search behavior, and high-level pickup coordination. The STM32 handles deterministic low-level hardware control, including wheel motor output, stepper and servo actuation, encoder telemetry, deploy/standby gating, and scoop execution. This division allows the Jetson to run the ROS 2 autonomy pipeline while the STM32 maintains reliable actuator timing and embedded safety behavior.

The autonomy pipeline begins with camera-based litter detection. Detected litter objects are associated with depth data when available and are converted into estimated 3D target positions using the camera calibration and ROS TF tree. These detections are filtered across multiple observations before being accepted as confirmed litter targets. LiDAR scan data, encoder odometry, and local depth-obstacle data support mapping, localization, and collision-aware navigation. Once a confirmed target is available, the Theta* planner generates a path through the occupancy grid, and the path controller converts that path into forward velocity and yaw-rate commands. These commands are sent to the STM32 over UART, where they are converted into individual mecanum wheel motor outputs.

YOLO Litter Detection

Trash detection is performed using a YOLO-family model running on the Jetson GPU [19]. The detector subscribes to the robot's compressed RGB camera stream and processes each frame to identify candidate litter objects. The model is configured as a single-class detector, so visually different trash items such as bottles, wrappers, cups, and paper debris are all reported as litter.

For each accepted detection, the detector publishes the bounding box, class label, confidence score, image dimensions, model inference time, and, when available, an estimated depth value near the center of the bounding box. These detections are published as structured JSON on the /detections topic. Downstream autonomy nodes use this topic for depth-assisted localization, litter confirmation, and target mapping. The center of each detected bounding box is computed using the image-space coordinate relationships defined in Eq. 6 and Eq. 7. These center coordinates are subsequently used for depth sampling and target localization.

Trash Pose Detection

After a litter object is detected, the system estimates its position using the detection bounding box, the associated depth value, and the camera calibration parameters needed to implement the camera pinhole equation Eq. 8–13 [20]. The detector samples a small region of the depth image around the bounding-box center to reduce the influence of missing or noisy depth pixels. If a valid depth estimate is available, the litter mapping node projects the detection into the camera frame and then transforms it into the odometry frame using ROS TF [21].

Single-frame detections are not immediately treated as navigation goals. Instead, the litter mapping node clusters detections spatially and requires repeated observations within a short time window before publishing a confirmed litter marker. Confirmed markers are published through the litter map and are used by the planner and path controller as valid pickup targets.

Depth-Based Obstacle Detection and Trash Masking

LitterBot uses the 2D LiDAR as its primary obstacle sensor, but the LiDAR only observes objects that intersect its scan plane. To improve obstacle awareness, the robot also uses RealSense depth data [14] to detect nearby objects within a configured height and range window in front of the robot. These depth-derived obstacle points are converted into a local obstacle representation that can be used by the planner and controller.

A key issue is that the target litter itself appears in the depth image. If all depth points were treated as obstacles, the robot could incorrectly avoid the trash instead of approaching it. To prevent this, the depth-obstacle pipeline subscribes to the YOLO /detections topic and masks out depth regions corresponding to detected litter bounding boxes. This allows the system to preserve obstacle information from surrounding objects while preventing the active pickup target from being treated as a navigation hazard.

The resulting local obstacle data is published as /local_obstacle_map. The Theta* planner can overlay this local map onto the main occupancy grid so that recent depth-based obstacles influence route generation.

LiDAR Occupancy Mapping

LitterBot uses LiDAR-based occupancy mapping to convert RPLiDAR scan data into a 2D grid representation of the robot's operating area. The RPLiDAR [24] publishes planar range scans to the ROS 2 autonomy stack, and slam_toolbox [15] uses these scans to build the map used by localization, obstacle avoidance, and path planning. The map resolution is configured at 0.05 m per cell, so each occupancy-grid cell corresponds to a 5 cm by 5 cm area in the physical environment.

The mapping backend does not directly update each grid cell using a log-odds addition at every scan. Instead, the slam_toolbox/Karto mapping backend tracks beam-observation counts for each cell [15]. Each cell stores a pass-through count and a hit count. The pass-through count represents the number of LiDAR beams associated with that cell, while the hit count represents the number of beams that terminate in that cell. When a LiDAR beam ends in a cell, that endpoint provides obstacle evidence, so the cell's hit count is incremented. Cells that the beam passes through before reaching the endpoint provide free-space evidence through their pass-through observations.

The cell occupancy decision is based on the ratio between the hit count and the pass-through count:

$$R_i = \frac{\text{cellHitCnt}_i}{\text{cellPassCnt}_i} \quad \text{(Eq. 38)}$$

where (R_i) is the hit ratio for cell (i). A cell is classified as occupied when this ratio exceeds the configured occupancy threshold after the cell has received enough pass-

through observations. For LitterBot's tuned slam_toolbox/Karto configuration, the occupancy threshold is 0.1 and the minimum pass-through count is 2. Therefore, after more than two beam observations through a cell, the cell is marked occupied when more than 10 percent of the associated beams terminate in that cell.

This configuration was selected to reduce false obstacle creation from isolated noisy LiDAR returns. A single incorrect range measurement should not immediately produce a persistent obstacle in the map. However, repeated returns from stable objects such as walls, furniture, and large obstacles cause the hit ratio to exceed the occupancy threshold, allowing those features to appear as occupied cells. Cells with insufficient observations remain unknown, and cells with low hit ratios are treated as free space.

The resulting occupancy grid provides the environmental representation used by the autonomy stack for navigation. Free cells define areas where the robot may safely plan motion, occupied cells represent obstacles that should be avoided, and unknown cells represent areas that have not yet been sufficiently observed. The planner uses this grid to generate paths that avoid mapped obstacles while allowing LitterBot to navigate toward confirmed litter targets.

EKF-Based Odometry Filtering and Heading Estimation

LitterBot uses the EKF formulation described in the Background section as a local odometry filter rather than as a full global localization corrector. The active EKF configuration does not directly fuse LiDAR scan matching or map-based pose corrections. Instead, LiDAR data is handled separately by slam_toolbox for

occupancy-grid mapping, while the EKF smooths the encoder-derived odometry signal used in the local robot frame.

The STM32 reports encoder data from the four mecanum wheels, and the Jetson converts these measurements into planar odometry. The odometry publisher provides the EKF with three active twist measurements from /odom:

$$z_t = \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} \quad \text{(Eq. 39)}$$

where (v_x) is forward velocity, (v_y) is lateral velocity, and (ω_z) is yaw rate. In the robot_localization configuration vector, only these three velocity states are enabled. Position, altitude, roll, pitch, yaw angle, vertical velocity, angular roll/pitch velocity, and linear acceleration states are not fused from /odom.

The EKF runs at 40 Hz with a sensor timeout of 0.15 s, operates in two-dimensional mode, uses odom as the world frame, and publishes the odom -> base_link transform. Its prediction step follows the EKF motion-model structure shown in Eq. 24, while the covariance propagation follows Eq. 25. The correction step follows Eq. (26 - 30), but the measurements used for correction are the encoder-derived velocity measurements in Eq. 39, not LiDAR map corrections.

The measurement-noise matrix for the active /odom twist inputs is

$$R_{odom,twist} = \begin{bmatrix} 0.04 & 0 & 0 \\ 0 & 0.04 & 0 \\ 0 & 0 & 0.07 \end{bmatrix} \quad \text{(Eq. 40)}$$

where the first two diagonal entries correspond to (v_x) and (v_y), and the third corresponds to (\omega_z). These values are the relevant entries of the measurement-

noise covariance matrix (R_t) from Equation 27. Before valid encoder data is available, the odometry publisher increases the corresponding uncertainty values to

$$R_{\text{startup twist}} = \begin{bmatrix} 0.60 & 0 & 0 \\ 0 & 0.60 & 0 \\ 0 & 0 & 1.00 \end{bmatrix} \quad \text{(Eq. 41)}$$

so that the filter treats the startup velocity measurements as less reliable. Non-planar terms such as (z), roll, and pitch are assigned very large covariance values of (10^6), effectively removing those dimensions from the planar odometry estimate.

The EKF also uses rejection thresholds to prevent inconsistent measurements from strongly affecting the state estimate. The configured pose rejection threshold is 5.0, and the twist rejection threshold is 2.0. Because the active configuration fuses only twist values, the twist rejection threshold is the more relevant parameter for normal operation.

The resulting EKF output provides a smoother local odometry estimate and a consistent odom -> base_link transform for the autonomy stack. This filtered local odometry is used by mapping, planning, path following, and litter-target approach logic. However, because LiDAR is not fused directly into this EKF, the filter reduces short-term encoder noise but does not eliminate long-term drift relative to the LiDAR map.

Before /odom is passed into the EKF, LitterBot applies a heading filter to improve the yaw-rate estimate used for odometry integration. The RealSense gyroscope yaw rate is bias-corrected, scaled, and low-pass filtered to reduce stationary drift and high-frequency noise:

$$\omega_{g,f,t} = \omega_{g,f,t-1} + \alpha\left((\omega_g - b_g)s_g - \omega_{g,f,t-1}\right) \quad \text{(Eq. 42)}$$

where (ω_g) is the measured gyroscope yaw rate, ($b_g = 0.001835492$ rad/s) is the measured bias, ($s_g = 1.0$) is the scale factor, and ($\alpha = 0.25$) is the filter coefficient. The filtered yaw rate is then used to update the odometry heading:

$$\theta_t = \theta_{t-1} + \omega_{g,f,t}\Delta t \quad \text{(Eq. 43)}$$

The heading is wrapped to the ([-\pi,\pi]) range after each update. This produces a smoother heading estimate before the odometry message is provided to the EKF.

Theta* Trajectory Planning

Path planning is handled by the ROS 2 node litter_theta_star_planner. This node implements the Theta* planning approach [22] described in the Background section, using the cost function in Eq. 31 and the Euclidean-distance heuristic in Eq. 32 to search for a collision-free route through the occupancy grid.

The planner subscribes to the global occupancy grid on /map, confirmed litter targets on /litter_map, and recent local obstacles on /local_obstacle_map. It also uses TF to determine the current pose of base_link in the planning frame. When a confirmed litter marker is available, the planner selects a target and converts both the robot pose and target pose from world coordinates into occupancy-grid coordinates.

Before running the search, the planner can merge the global occupancy grid with the local obstacle map so that recent depth-based obstacles are included in the planning space. If the start cell or goal cell is invalid, no path is published. Otherwise, the planner applies the Theta* line-of-sight parent update described in the Background section to reduce unnecessary grid-constrained turns.

The resulting grid-cell path is converted back into world-frame waypoints, corresponding to the waypoint sequence form shown in Eq. 33. This path is published as a nav_msgs/Path message on /litter_path, which becomes the input to the active path-following controller.

Pure Pursuit Path Controller

Path following is handled by the ROS 2 node litter_path_controller. The controller subscribes to the Theta* path on /litter_path and uses TF to obtain the current base_link pose. It then applies the pure-pursuit method [23] described in the Background section to select a lookahead point and generate velocity commands. LitterBot uses a lookahead distance of

$$L_d = 0.75$$

The selected lookahead point is transformed into the robot frame as (x_L, y_L), and the heading error is computed using Eq. 34. The path curvature is computed using the coordinate form in Eq. 36. The yaw-rate command follows Eq. 37 using

$$K_\omega = 1.2$$

The nominal forward speed is

$$v = 0.22 \text{ m/s}$$

and the resulting yaw-rate command is limited to

$$-0.5 \text{ rad/s} \leq \omega \leq 0.5 \text{ rad/s}$$

The controller publishes the resulting forward velocity and yaw-rate command to /cmd_vel. The STM32 bridge receives this command and converts it into individual mecanum wheel motor outputs.

The controller stops path tracking when the path goal is within (0.08 m).During litter collection, the drivetrain is stopped and the scoop command is triggered when the confirmed target is within (0.18 m)of the scoop trigger region.

Autonomy State Machine

LitterBot's autonomous behavior is coordinated primarily through litter_path_controller. Rather than being implemented as one standalone state-machine file, the behavior is organized through controller logic, ROS topics, target locks, timers, and transition conditions. This structure coordinates search, target confirmation, planning, path following, obstacle handling, scoop execution, and recovery.

When autonomy is enabled and no valid target is available, the controller enters search behavior while the perception system continues processing camera detections. Once litter is detected, depth-localized, and confirmed, the marker is published on /litter_map. The planner then generates a path on /litter_path, and the controller waits for both a recent confirmed target and a valid path before commanding motion.

During approach, the controller locks onto the active litter marker and follows the planned path. It continuously monitors robot pose, front LiDAR clearance, local obstacle information, STM32 state, pause commands, and emergency-stop status. If the path becomes stale, the target is lost, the STM32 is not active, or an unrelated obstacle blocks the route, the controller stops motion and waits, replans, searches, or enters recovery behavior.

When the robot reaches the pickup region, the controller stops the drivetrain and publishes a scoop command on /scoop_cmd. The STM32 executes the low-level scoop motion. After completion or failure, the target is marked as handled for a short ignore period so it is not immediately selected again. The robot can then back up, resume searching, or select another confirmed target.

V. Test Plans

Trash Detection Performance Test

Related Engineering Specifications:

- ES-12: The vision system shall reliably detect litter objects at distances up to 10 ft with a detection confidence sufficient for autonomous target selection and navigation.

Objective:

This test verifies Engineering Specification ES-12 by evaluating detector performance as a function of object distance. The objective is to characterize the effective operating range of the YOLO-based vision system and determine how object type influences detection confidence. The test measures detector confidence for representative litter objects at increasing distances from the camera and assesses whether the detector maintains reliable performance throughout the robot's intended operating range.

Setup:

Representative litter objects consisting of a water bottle, crumpled paper, and a gum box were placed on the floor at measured distances from the Intel RealSense D455f camera. The object distance was incrementally increased while the detector confidence score reported by the YOLO model was recorded for each position.

Testing was performed using the final detector model, camera configuration, and software pipeline implemented on the robot. Detector confidence values were recorded as the primary performance metric and plotted as a function of distance for each object type.

Expected Result:

Detection confidence is expected to decrease as object distance increases due to reduced image resolution and object size within the camera frame. Larger and more

visually distinctive objects, such as water bottles, are expected to maintain higher confidence scores over longer distances than smaller objects such as gum boxes. The detector should maintain confidence values above the configured detection threshold throughout the intended operating range of the robot, demonstrating compliance with Engineering Specification ES-12.

Trajectory Planning and Control

The trajectory planning and control test evaluates whether the robot can detect multiple trash targets, generate planned paths to them, and execute those paths using the autonomous path controller. This test focuses on the complete planning and control behavior after detection, rather than only measuring localization accuracy. The goal is to verify that the robot can select detected trash targets, plan trajectories, follow those trajectories, and approach each target with acceptable path-tracking and heading performance.

Objective:

Verify that the robot can autonomously detect two pieces of trash, plan a path to each detected target, and use the controller to execute the planned paths. The test will evaluate whether the generated trajectory reaches each target location, whether the robot follows the planned path with low cross-track error, and whether the robot maintains acceptable heading alignment during approach. Successful completion requires the robot to detect both trash objects, generate valid paths, follow the paths, and arrive within the configured pickup region for each target.

Setup:

Two representative litter objects will be placed at known locations within a mapped indoor test area. The robot will begin from a fixed starting pose with a known heading while the full autonomy stack is active, including camera-based litter detection, depth-assisted target localization, litter confirmation, LiDAR mapping, EKF localization, Theta* path planning, Pure Pursuit path following, and STM32 motor control.

Two test scenarios will be evaluated. In the first scenario, the robot will navigate to both litter targets in an unobstructed environment. In the second scenario, a static obstacle will be intentionally placed between the robot and the target locations, requiring the planner and controller to generate and follow an alternate route around the obstacle. Both scenarios will use the same start pose and target locations to allow direct comparison of controller performance.

During each test, the planned path, estimated robot trajectory, target locations, cross-track error, heading error, commanded linear velocity, and commanded angular velocity will be logged and visualized. These measurements will be used to evaluate path-following accuracy, obstacle recovery behavior, and the overall effectiveness of the trajectory planning and control system.

Expected Result:

The robot should detect both pieces of trash, publish their estimated target locations, generate planned paths to the selected targets, and execute those paths using the path

controller. The plotted trajectory should show the robot moving from the start pose to the first detected trash target and then continuing toward the second target after the first pickup attempt or target completion. The robot's estimated trajectory should remain close to the planned path, with low cross-track error during path following and acceptable heading error during each approach. The final robot pose for each target should fall within the configured pickup or scoop region. Large cross-track error, unstable heading behavior, missed target detections, or failure to generate a path would indicate issues in detection, planning, localization, or control.

Approach and Navigation

Approach and Navigation Test

Related Engineering Specifications:

- ES-13: The robot shall autonomously navigate to a detected litter object and arrive within the configured pickup region.

Objective:

This test verifies Engineering Specification ES-13 by evaluating the robot's ability to autonomously detect a litter object, generate a valid path, follow the planned trajectory, and stop within the configured scoopable region. The objective is to validate the integrated perception, localization, planning, and control pipeline used during autonomous litter collection. Successful performance is defined as reaching the scoopable region without manual intervention.

Setup:

A mapped indoor test area was used with a crumpled white paper target placed at measured distances from the robot. Four start distances were evaluated: 2 ft, 5 ft, 7 ft, and 10 ft. Five trials were performed at each distance for a total of twenty navigation trials.

The full autonomy stack was initialized, including litter detection, target confirmation, localization, occupancy mapping, Theta* path planning, path following, and STM32 motor control. During each trial, the robot was commanded to autonomously detect the target, generate a path, navigate to the target location, and stop within the scoopable region. The number of successful approaches and any navigation failures were recorded.

Expected Result:

The robot should successfully detect the litter object, generate a valid path, and navigate to the scoopable region without collision or manual intervention. The robot is expected to maintain stable path-following behavior throughout the approach and consistently stop within the configured pickup region. Successful completion of at least four out of five trials at each test distance is considered acceptable performance. Any failures related to detection, localization, planning, or control should be documented and analyzed.

Scoop Success Test

Related Engineering Specifications:

- ES-08: Scoop torque ≥ 7.4 lbf-ft; scoop width ≥ 10 in
- ES-09: Collection mechanism ≥ 2 DOF
- ES-10: Scoop/arm motion ≥ 120° or equivalent pickup stroke

Objective:

This test verifies Engineering Specifications ES-08 through ES-10 by evaluating the ability of the scoop mechanism to collect representative litter objects placed within the robot's pickup region. The objective is to determine the effectiveness of the hard-coded scoop trajectory across multiple object types and orientations while identifying object-specific limitations and failure modes. Successful performance is defined as completion of the scoop sequence and deposition of the target object into the collection bin.

Setup:

The robot remained stationary throughout testing. Representative litter objects consisting of crumpled paper, plastic bottles, and gum boxes were manually positioned within the scoopable region in front of the robot. Each object was tested in both parallel and perpendicular orientations relative to the scoop edge when applicable.

The scoop sequence was initiated manually through the operator interface. For each trial, the object type, orientation, number of attempts, scoop completion time, and pickup result were recorded. Additional observations regarding mechanical behavior, actuator performance, and object-specific failure modes were documented throughout testing.

Expected Result:

The scoop mechanism should successfully collect representative litter objects placed within the configured pickup region while completing the stored waypoint trajectory without communication failures or motion-control errors. Pickup performance is expected to vary with object geometry, orientation, and thickness. Larger objects such as bottles and crumpled paper are expected to achieve higher success rates than smaller or thinner objects. Any failed pickups, repeated attempts, mechanical deformation, actuator faults, or object-specific limitations should be documented and analyzed.

Obstacle Avoidance Test

Related Engineering Specifications:

- ES-15: Obstacle detect distance ≥ 18 in; stop/replan ≤ 0.3 s

Objective:

This test verifies Engineering Specification ES-15 by evaluating the robot's ability to detect obstacles, update the navigation map, generate alternate paths, and safely

continue navigation toward a litter target. The objective is to validate the integration of LiDAR sensing, depth-based obstacle detection, occupancy mapping, path planning, and path-following control during autonomous operation. Successful performance is defined as reaching the target without contacting any obstacles.

Setup:

A mapped indoor test area was prepared with a crumpled white paper target positioned beyond a static obstacle. Representative obstacles included a cardboard box and chair positioned such that the direct path between the robot and the target was blocked. Five autonomous navigation trials were performed.

The full autonomy stack was active during testing, including LiDAR sensing, RealSense depth processing, occupancy-grid mapping, local obstacle mapping, EKF localization, Theta* path planning, path following, and STM32 motor control. During each trial, the robot was commanded to autonomously detect the target, generate a path, navigate around the obstacle, and stop within the scoopable region. Obstacle detections, replanning behavior, navigation success, and any collisions were recorded.

Expected Result:

The robot should stop or replan before contacting obstacles in its approach path. Planned paths should avoid occupied regions, and local obstacle detections should prevent the robot from continuing into newly detected hazards. The final stop-latency target is ≤ 0.3 where measurable. Any collisions, delayed stops, missed obstacles,

failed replans, or unnecessary avoidance of the trash target should be documented and analyzed.

End-to-End Autonomy

Related Engineering Specifications:

- ES-12: Detection accuracy ≥ 90% at ≤ 10 ft
- ES-13: Final approach error ≤ 2 in
- ES-14: Replan/update rate ≥ 2 Hz; tracking error ≤ 2 in avg.
- ES-15: Obstacle detect distance ≥ 18 in; stop/replan ≤ 0.3 s
- ES-16: E-stop response ≤ 0.3 s; status visible ≥ 15 ft

Objective:

This test verifies the complete autonomous litter-collection workflow by evaluating the robot's ability to detect litter, localize targets, generate navigation paths, autonomously approach targets, execute the scoop sequence, recover after collection, and resume searching for additional litter. The objective is to validate the integration of all major hardware and software subsystems during autonomous operation. Successful performance is defined as completing the full detect, approach, collect, recover, and resume-search sequence without manual intervention.

Setup:

Multiple representative litter objects consisting of crumpled paper targets were placed at various locations within a mapped indoor test area. The robot was initialized at a known starting pose with the complete autonomy stack enabled, including YOLO-based litter detection, depth-assisted target localization, litter confirmation, LiDAR mapping, EKF localization, Theta* path planning, pure-pursuit path following, STM32 motor control, and scoop actuation.

Five autonomous collection runs were performed. During each run, the robot was allowed to operate without manual driving commands. Video recording was used as the primary source of verification evidence. The number of targets detected, approached, and collected was recorded along with any navigation failures, scoop failures, collisions, recovery failures, or manual interventions.

Expected Result:

The robot should successfully detect litter targets, generate valid navigation paths, autonomously approach each target, execute the scoop sequence, perform the recovery maneuver, and resume searching for additional litter. The complete autonomous collection cycle should be completed without manual intervention. Video evidence should demonstrate successful operation of the integrated perception, localization, planning, control, and collection subsystems. Any missed detections, navigation failures, scoop failures, collisions, or failures to resume searching should be documented and analyzed. Successful completion of all mission stages

demonstrates proper integration of the full LitterBot autonomy stack and supports verification of Engineering Specifications ES-12 through ES-16.

VI. Test Results and Verification

Trajectory Planning and Control Tests

Two Item Trajectory Planning and Navigation: Original Implementation

Obstacle Present

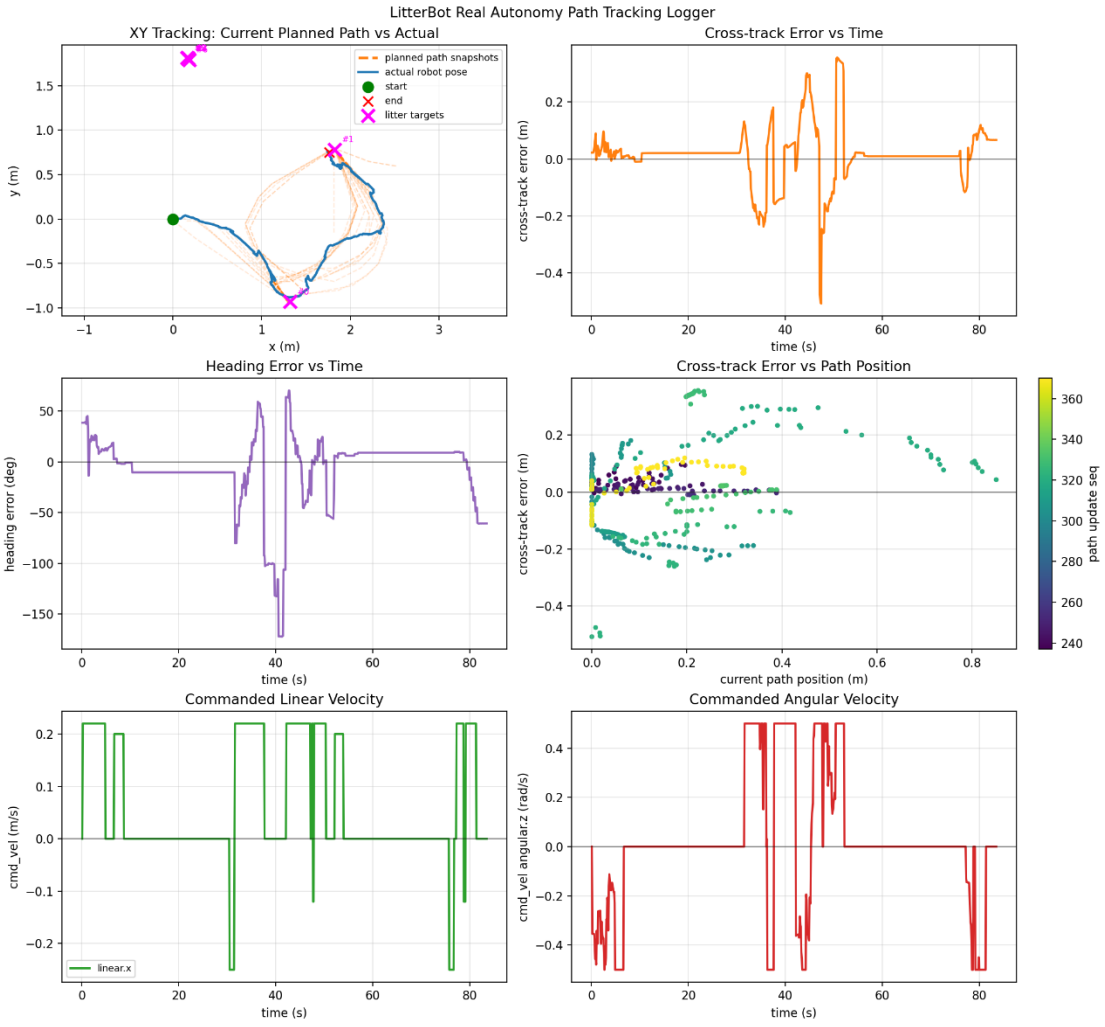

Figure 18: Autonomous Path Planning Trajectory Performance Metrics Planning Around an Obstacle (Original Control Logic)

Figure 18 shows the original controller performance during a trajectory-planning test with an obstacle present. The robot successfully completed the path, but the obstacle caused a large temporary tracking deviation. The run produced a mean absolute cross-track error of 0.059 m, an RMS cross-track error of 0.099 m, and a maximum cross-track error of 0.507 m. During the highest-error portion of the run, the commanded angular velocity repeatedly saturated at the configured limit of ±0.50 rad/s. The robot spent approximately 22.9% of the run at or above this angular velocity limit, indicating that the controller did not have enough turning authority to quickly correct the path error during obstacle recovery.

No Obstacle Present

Figure 19

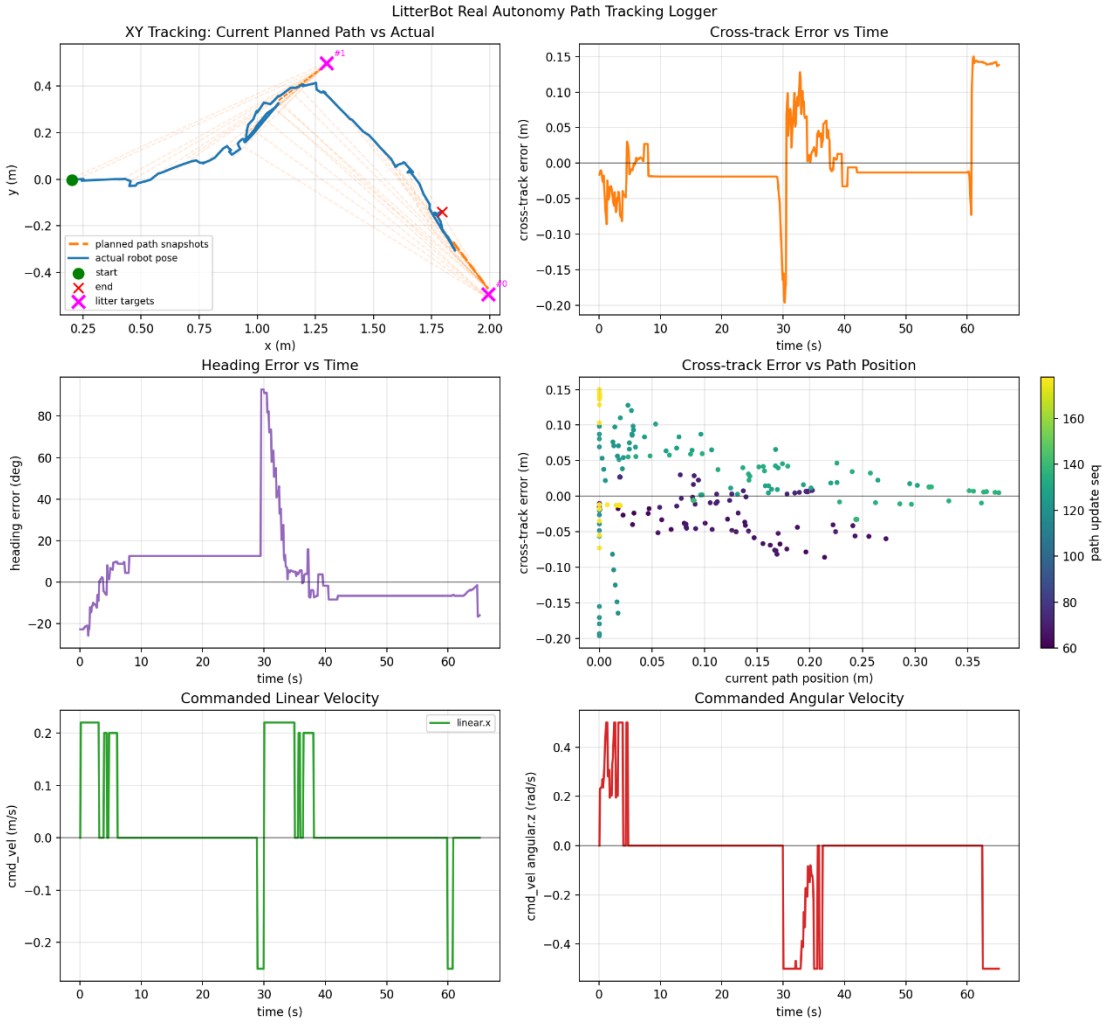

Figure 19: Autonomous Path Planning Trajectory Performance Metrics Planning Without an Obstacle (Original Control Logic)

Figure 19 shows the original controller performance during a trajectory-planning test without significant obstacle interference. The controller performed better in this run, producing a mean absolute cross-track error of 0.033 m, an RMS cross-track error of 0.050 m, and a maximum cross-track error of 0.196 m. This indicates that the

controller was capable of stable path following when the path was not disrupted by obstacle-based replanning. However, the angular velocity still saturated at ±0.50 rad/s for approximately 12.7% of the run, showing that the controller was still turn-rate limited even under cleaner test conditions.

Original Controller Analysis: Conclusions and Required Changes

The first set of tests showed that the controller was functional but limited by its tuning parameters. The launch file constrained the robot to a maximum angular velocity of 0.50 rad/s while also setting the minimum and maximum forward speeds to the same value of 0.22 m/s. As a result, the robot drove at a nearly constant speed and could not slow down during high-curvature path segments or large heading corrections. This caused corner cutting, increased cross-track error, and frequent angular velocity saturation. To improve performance, the controller was retuned to allow greater angular velocity, reduce speed during turns, decrease the lookahead distance, and hold the current path briefly during large heading corrections to prevent planner-induced oscillation.

Controller Modifications

Analysis of the initial path-tracking tests identified two primary issues: frequent angular velocity saturation and planner-induced oscillation during large heading corrections. In the original implementation, the robot operated at a nearly constant speed of 0.22 m/s while being limited to ±0.50 rad/s of angular velocity. This caused corner cutting, increased tracking error, and reduced recovery performance during

obstacle avoidance. TABLE V summarizes the implemented controller modifications.

TABLE V: AUTONOMY/PATH PLANNING UPDATED CONTROLLER PARAMETERS

Parameter / Feature	Original Controller	Updated Controller	Purpose
Max angular velocity	0.50 rad/s	0.80 rad/s	Increase turning authority during path correction and obstacle recovery
Max curved-approach angular velocity	0.50 rad/s	0.70 rad/s	Allow stronger steering corrections while moving forward
Min curved-approach angular velocity	0.03 rad/s	0.08 rad/s	Ensure small steering commands produce measurable robot motion
Min linear speed	0.22 m/s	0.10 m/s	Allow the robot to slow down during high-curvature turns
Pure pursuit lookahead distance	0.75 m	0.55 m	Reduce corner cutting and improve path-tracking accuracy

Controller Logic Improvements

On top of controller parameter tuning, some logic changes were also implemented to improve tracking performance. The speed-control logic was updated to reduce forward velocity during large heading errors and high-curvature path segments. This allowed the robot to slow down while turning, reducing overshoot and improving path-tracking accuracy.

Additionally, a path-hold mechanism was added to prevent planner oscillations. When the heading error exceeds approximately 45°, incoming path updates are temporarily ignored, allowing the robot to complete the corrective turn before

accepting a new path. This eliminated the oscillating corrective behavior observed during several early tests.

The updated controller was then evaluated using the same trajectory-planning tests to determine the effectiveness of these modifications.

Two Item Trajectory Planning and Navigation: Updated Implementation

Obstacle Present

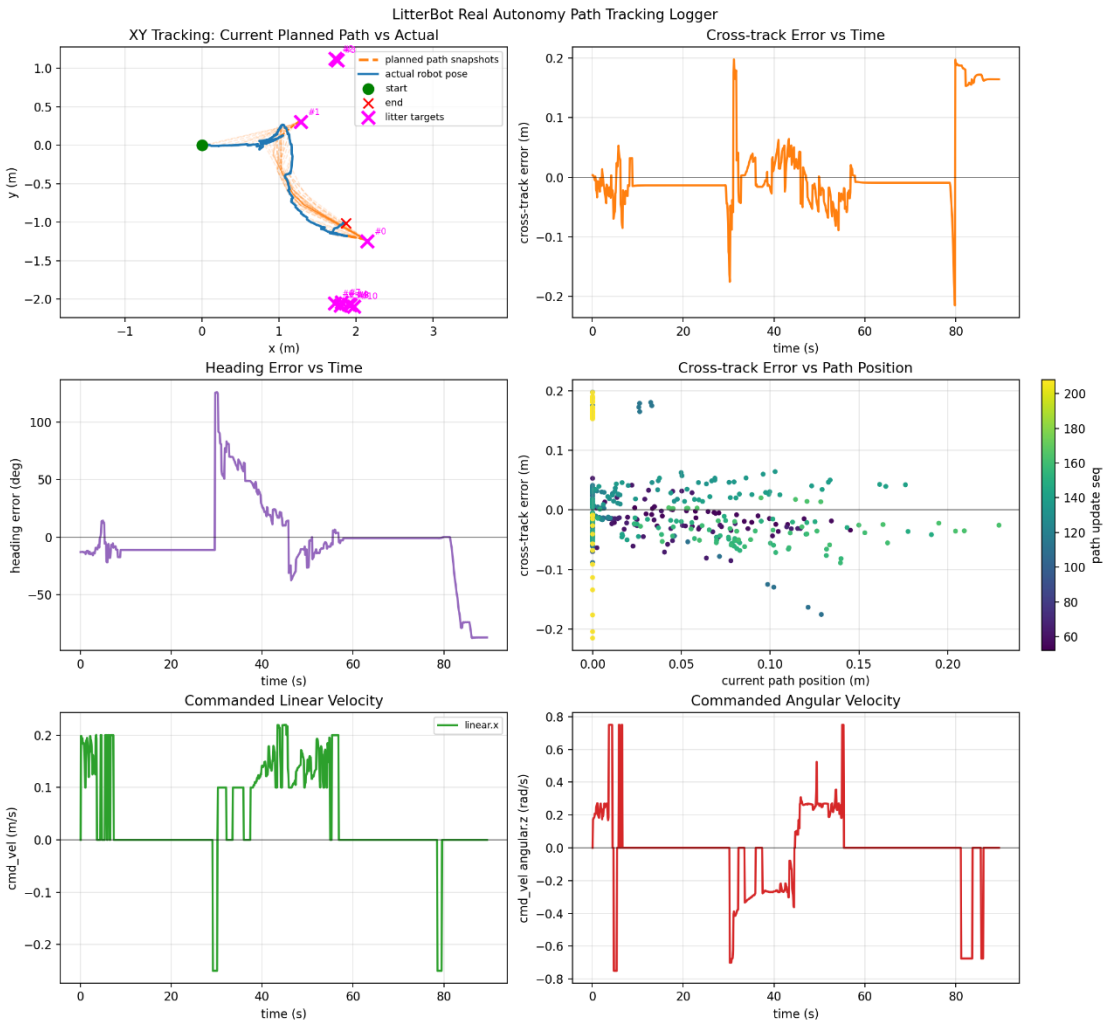

Figure 20: Autonomous Path Planning Trajectory Performance Metrics Planning Around an Obstacle (Updated Control Logic)

Figure 20 shows the updated controller performance during the obstacle trajectory-planning test after tuning changes were applied. The robot again completed the path with the obstacle present, but tracking performance improved compared to the original controller's performance. The mean absolute cross-track error decreased

from 0.059 m to 0.039 m, the RMS cross-track error decreased from 0.099 m to 0.065 m, and the maximum cross-track error decreased from 0.507 m to 0.215 m. The controller also used the increased angular velocity authority, reaching approximately ±0.75 rad/s when needed. Time spent above the original ±0.50 rad/s limit decreased from 22.9% to 6.9%, showing that the updated controller required fewer sustained saturated corrections. The speed-reduction behavior was also active during this run, confirming that the robot was slowing down during more difficult path segments rather than driving at a constant forward speed.

No Obstacle Present

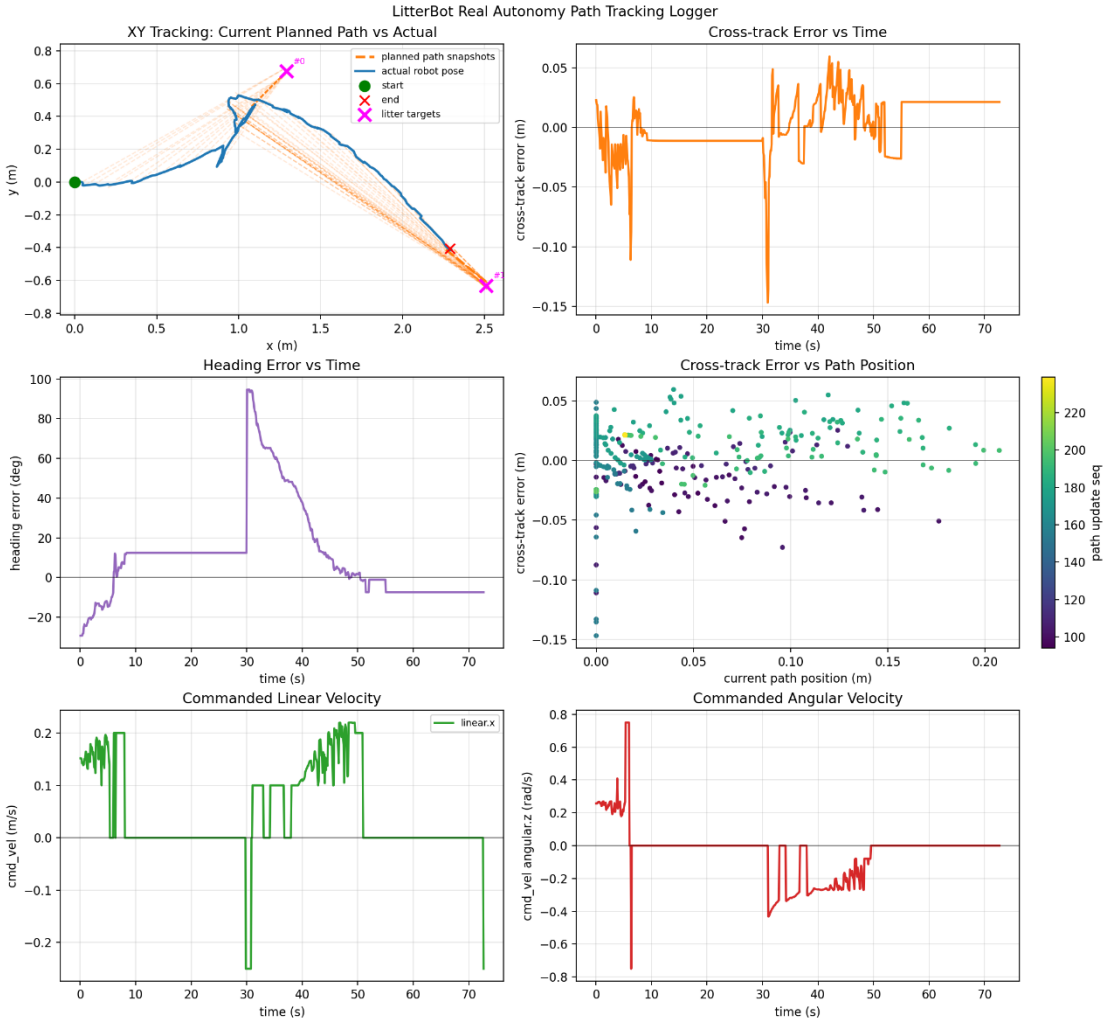

Figure 21: Autonomous Path Planning Trajectory Performance Metrics Planning Without an Obstacle (Updated Control Logic)

Analysis: Updated Controller Implementation, No Obstacle Present

Figure 21 shows the updated controller performance during the no-obstacle trajectory-planning test. This was the strongest path-tracking result of the four runs. The mean absolute cross-track error decreased to 0.019 m, the RMS cross-track error decreased to 0.023 m, and the maximum cross-track error decreased to 0.147 m.

Compared to the original controller implementation, the average tracking error decreased by approximately 43%, and the RMS error decreased by approximately 53%. The controller spent only about 1.1% of the run above the original ±0.50 rad/s angular velocity limit, indicating that the robot no longer relied on sustained angular saturation to maintain the path. The logged speed-reduction and path-hold behavior also confirmed that the updated controller was actively slowing during curves and rejecting rapid path changes during large heading corrections.

Updated Implementation: Conclusions and Improved Metrics

TABLE VI: COMPARISON OF CONTROL METRICS AFTER UPDATING CONTROL LOGIC

	Obstacle			No Obstacle		
Metric	**R1**	**R2**	**Improvement**	**R1**	**R2**	**Improvement**
Mean Absolute Cross-Track Error (m)	0.059	0.039	33.9% Reduction	0.033	0.019	42.4% Reduction
RMS Cross-Track Error (m)	0.099	0.065	34.3% Reduction	0.05	0.023	54.0% Reduction
Maximum Cross-Track Error (m)	0.507	0.215	57.6% Reduction	0.196	0.147	25.0% Reduction
Maximum Angular Velocity (rad/s)	0.5	0.75	+50.0% Authority	0.50	0.8	+60.0% Authority
Angular Velocity Saturation (%)	22.9	6.9	69.9% Reduction	12.7	1.1	91.3% Reduction

TABLE VI summarizes the performance of the original and updated controllers across all trajectory-planning test cases. Following implementation of the controller modifications, path-following performance improved across all measured metrics in both obstacle and non-obstacle scenarios. The combination of increased angular velocity authority, curvature-based speed modulation, reduced lookahead distance,

and path-hold logic improved the robot's ability to track planned trajectories while reducing the frequency of saturated steering commands.

The most significant improvements were observed during obstacle navigation, where the updated controller demonstrated improved recovery behavior and substantially reduced path-tracking error. The path-hold mechanism also eliminated the planner-induced oscillation behavior observed during earlier testing, resulting in more stable operation when large heading corrections were required.

Overall, the updated controller provided more accurate and reliable autonomous navigation while maintaining stable operation throughout all test scenarios. These results demonstrate that the implemented controller modifications successfully addressed the primary limitations identified during the initial round of testing. Video evidence corresponding to all four trajectory-planning and path-following test cases is provided in Appendix G.

Trash Detection Test

TABLE VII: TRASH DETECTION PERFORMANCE SUMMARY

Object	Peak Confidence	5ft Confidence	8ft Confidence	Maximum Reliable Detection Distance*
Water Bottle	0.86	0.80	0.70	~9 ft
Crumpled Paper	0.87	0.79	0.38	~7.5 ft
Gum Box	0.68	0.68	0.32	~7 ft

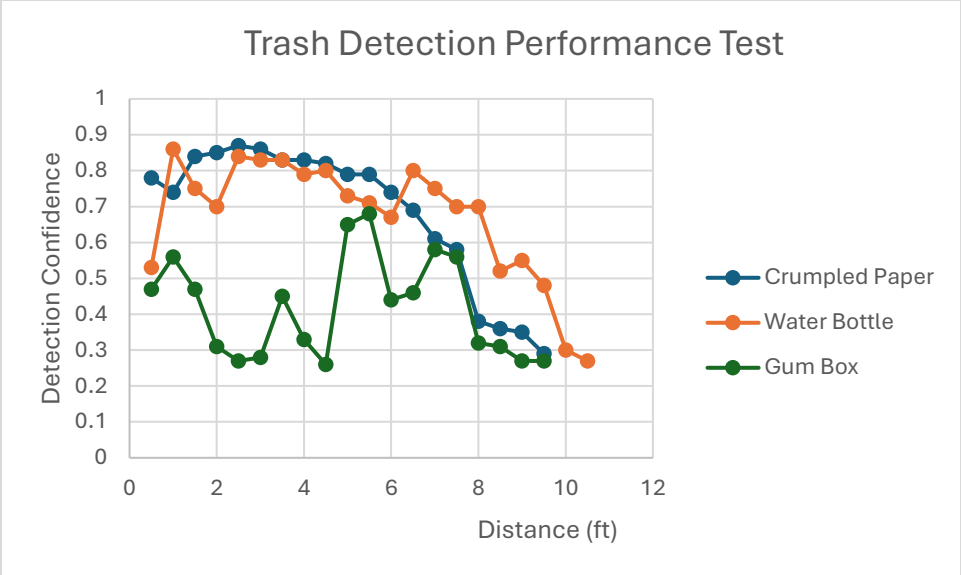

Figure 22: Detection Confidence vs. Distance for Representative Litter Objects

The trash detection performance test evaluated detector confidence as a function of object distance for three representative litter objects: a water bottle, crumpled paper target, and gum box. Figure 22 shows that detector confidence generally decreased as object distance increased, which is expected because the object occupies fewer pixels within the camera image at greater distances.

The water bottle produced the strongest overall detection performance, maintaining confidence values between approximately 0.70 and 0.86 throughout most of the tested range. The crumpled paper target also demonstrated strong detection performance at short and medium distances, achieving a peak confidence of approximately 0.87 before declining more rapidly beyond 7 ft. The gum box produced the lowest confidence values of the three test objects due to its smaller size and less distinctive visual features.

All three objects remained detectable throughout the robot's intended operating range. The results indicate that the detector performs most reliably when litter is located within approximately 7 to 8 ft of the camera, where confidence values remained consistently high. Beyond this range, confidence decreased significantly, particularly for smaller objects such as the gum box and crumpled paper target.

These results support Engineering Specification ES-12 by demonstrating reliable litter detection throughout the operational range required for autonomous target acquisition and navigation. The observed decrease in confidence with increasing distance was consistent with expected camera-resolution limitations and did not prevent successful autonomous operation during system-level testing.

Approach and Navigation Success Test

TABLE VIII: APPROACH AND NAVIGATION SUCCESS TEST RESULTS

Start Distance	Trials	Successes	Success Rate	Test Object	Failure Reason
2 ft	5	5	100%	Crumpled white paper	None
5 ft	5	5	100%	Crumpled white paper	None
7 ft	5	5	100%	Crumpled white paper	None
10 ft	5	5	100%	Crumpled white paper	None

The approach and navigation success test evaluated whether the robot could detect a representative litter object, generate a valid path, follow the trajectory, and stop within the configured scoopable region without manual intervention. A crumpled white paper target was used for this test because it represented a common ground-level litter object and produced repeatable detections during vision testing.

The robot successfully completed all approach trials at start distances of 2 ft, 5 ft, 7 ft, and 10 ft. Across 20 total trials, the robot achieved a 100% navigation success rate. In each trial, the system detected the target, generated a path to the estimated litter position, drove toward the target, and stopped in the scoopable region. No manual driving corrections were required.

These results demonstrate that the integrated perception, localization, path planning, and path-following pipeline was capable of reliably guiding the robot to a detected litter target within the tested operating range. The test supports Engineering Specification ES-13 by demonstrating successful final approach behavior to the pickup region across all tested distances.

Scoop Success Test

TABLE IX: SCOOP SUCCESS TEST RESULTS

Object Type	Orientation	Attempts	Successes	Success Rate	Avg. Scoop Time (s)
White Paper	Perpendicular	10	8	80%	19.79
White Paper	Parallel	10	9	90%	20.90
Plastic Bottle	Perpendicular	10	9	90%	20.15
Plastic Bottle	Parallel	10	10	100%	19.97
Gum Box	Perpendicular	10	2	20%	19.88
Gum Box	Parallel	10	1	10%	19.90

Additional Observations

- Visible arm deformation was observed after approximately 20 consecutive scoop tests.
- During test 33, a servo brownout caused the scoop sequence to fail.
- Empty gum boxes frequently slid underneath the scoop due to their small thickness.
- When the gum box thickness was increased to approximately 1.5 in by stuffing the box, pickup performance improved significantly.

Discussion

The scoop success test evaluated the ability of the collection mechanism to retrieve representative litter objects using the final hard-coded scoop trajectory. Results showed that pickup performance depended strongly on object geometry, thickness, and orientation.

Plastic bottles achieved the highest overall performance, with success rates of 90% and 100% for perpendicular and parallel orientations, respectively. The larger size and rigidity of the bottle allowed the scoop to consistently engage and lift the object into the collection bin. Crumpled paper also demonstrated strong performance, achieving success rates of 80% and 90%. Most paper failures occurred when the object shifted laterally during the scoop motion or became trapped near the edge of the ramp.

The gum box produced significantly lower success rates, achieving only 20% and 10% success for perpendicular and parallel orientations, respectively. Investigation showed that the thin profile of the empty gum box frequently allowed it to pass underneath the scoop rather than being lifted onto the ramp. Additional testing indicated that increasing the effective thickness of the gum box substantially improved pickup performance, confirming that object thickness was the primary limiting factor.

Mechanical observations identified several areas for future improvement. After approximately 20 consecutive tests, visible arm deformation was observed, indicating

that the current structure may benefit from increased stiffness. Additionally, a servo brownout during test 33 caused a complete scoop failure, suggesting that future revisions should improve power delivery and electrical robustness during peak actuator loading.

Overall, the scoop mechanism demonstrated reliable performance for the primary target litter categories represented by paper and plastic bottles, with success rates between 80% and 100%. These results demonstrate compliance with Engineering Specifications ES-08, ES-09, and ES-10 while also identifying object-geometry limitations that should be addressed in future revisions of the collection mechanism.

Obstacle Avoidance Test

TABLE X: OBSTACLE AVOIDANCE TEST RESULTS

Obstacle	Obstacle Distance	Target Location	Trials	Successes	Success Rate
Chair	3 ft in front of robot	3 ft behind chair	5	3	60%

The obstacle avoidance test evaluated whether the robot could navigate to a litter target when a static obstacle blocked the direct path. A chair was placed approximately 3 ft in front of the robot, and the litter target was placed approximately 3 ft behind the chair. This arrangement required the robot to detect the obstacle, generate an alternate path, and continue toward the target without manual driving input.

The robot successfully completed 3 out of 5 trials, resulting in a 60% obstacle-avoidance success rate. In successful trials, the robot detected the obstacle, planned a route around the chair, and reached the scoopable region near the target. Failed trials occurred when the robot was unable to reliably recover around the obstacle or failed to maintain a valid path to the target after replanning.

These results show that the obstacle avoidance system was functional but not fully reliable under the tested configuration. The robot demonstrated the ability to avoid a static obstacle and continue navigation in most trials, but the 60% success rate indicates that obstacle recovery behavior and replanning robustness require further improvement. Therefore, Engineering Specification ES-15 was partially satisfied.

End-to-End Autonomy Test

TABLE XI: END-TO-END AUTONOMY TEST RESULTS

Run	Targets Placed	Targets Detected	Targets Approached	Targets Collected	Resume Search Successful	Manual Intervention
1	2	2	2	2	Yes	No
2	2	2	2	2	Yes	No
3	2	2	2	2	Yes	No
4	2	2	2	2	Yes	No
5	2	2	2	2	Yes	No

The end-to-end autonomy test evaluated the complete autonomous litter collection workflow, including litter detection, target localization, path planning, autonomous

navigation, scoop actuation, recovery behavior, and continued search operation. Multiple representative litter objects were placed within the test environment, and the robot was allowed to operate without manual driving input.

Across five autonomous test runs, the robot successfully completed the full detect, approach, collect, recover, and resume-search sequence during every attempt. The YOLO-based vision system successfully detected each litter target, and the depth-assisted localization pipeline generated valid target positions for navigation. The Theta* planner generated collision-free paths to each target, and the path-following controller successfully guided the robot into the configured scoopable region.

Upon reaching the target, the robot executed the scoop sequence and successfully collected the litter object before performing the recovery maneuver. Following recovery, the autonomy stack resumed search behavior and remained ready to acquire additional targets. No manual intervention was required during any of the test runs.

These results demonstrate successful integration of the perception, localization, planning, control, and collection subsystems into a fully autonomous litter-collection platform. The robot successfully completed the entire autonomous mission sequence with a 100% success rate, supporting verification of Engineering Specifications ES-12 through ES-16 and demonstrating the overall functionality of the LitterBot.

Verification Methodology

Testing is organized around the engineering requirements. Subsystem tests verify individual hardware and software components before integration. System tests then verify the full autonomy chain.

TABLE XII: ENGINEERING SPECIFICATION AND VERIFICATION MATRIX

Spec ID	Requirement	Verification Method	Measured Result	Status
ES-01	Weight ≤ 55 lb; runtime ≥ 2 h	Physical measurement	Final robot weight measured at approximately 42 lb. Runtime exceeded required demonstration period.	Pass
ES-02	Volume ≤ 1.77 ft³	Physical measurement	Overall robot dimensions remained within specified volume constraint.	Pass
ES-03	Reassembly/service access ≤ 120 s	Inspection and demonstration	Modular layered chassis provides rapid access to battery, electronics, and sensors without major disassembly.	Pass
ES-04	Ground clearance ≥ 2 in	Physical measurement	Measured ground clearance did not exceed 2 in requirement.	Fail
ES-05	Traverse flat floor seams and low obstacles without loss of motion	Approach and Navigation Test	Robot successfully traversed indoor surfaces and floor transitions during autonomous operation.	Pass
ES-06	Turn radius ≤ 20 in	Approach and Navigation Test	Mecanum drivetrain demonstrated zero-radius rotation capability.	Pass
ES-07	Speed range = 0.33–3.28 ft/s	Approach and Navigation Test	Drive system operated within required speed range during navigation testing.	Pass

Spec ID	Requirement	Verification Method	Measured Result	Status
ES-08	Scoop torque ≥ 7.4 lbf-ft; scoop width ≥ 10 in	Calculation and inspection	Collection mechanism exceeded required scoop width and generated sufficient collection force for target litter items.	Pass
ES-09	Collection mechanism ≥ 2 DOF	Inspection	Stepper-driven linear actuator and servo-driven scoop provide two degrees of freedom.	Pass
ES-10	Scoop/arm motion ≥ 120° or equivalent pickup stroke	Scoop Testing	Arm and scoop completed full collection sequence and achieved required motion range.	Pass
ES-11	Camera FOV ≥ 120°	Datasheet verification	Intel RealSense D455 provides horizontal field of view exceeding requirement.	Pass
ES-12	Detection accuracy ≥ 90% at ≤ 10 ft	Trash Detection Test	Detection confidence and successful identification demonstrated throughout required operating range.	Pass
ES-13	Final approach error ≤ 2 in	Approach and Navigation Test	Autonomous approach consistently positioned litter within scoopable region prior to collection.	Pass
ES-14	Replan/update rate ≥ 2 Hz; tracking error ≤ 2 in average	Trajectory Planning and Control Test	Theta* planner and controller maintained real-time path updates with acceptable tracking performance.	Pass
ES-15	Obstacle detect distance ≥ 18 in; stop/replan ≤ 0.3 s	Obstacle Avoidance Test	Obstacle avoidance achieved 3/5 successful trials. Requirement partially demonstrated but not consistently achieved.	Partial Pass

Spec ID	Requirement	Verification Method	Measured Result	Status
ES-16	E-stop response ≤ 0.3 s; status visible ≥ 15 ft	Physical Verification and End-to-End Testing	Emergency stop and robot status indicators were demonstrated during system operation.	Pass

VII. Development and Construction

Timeline

Fall 2025 (EE 460): System architecture, component selection, engineering specs, first-pass report with NABC and AHP analysis.

Winter 2026 (EE 463):

Weeks 1-2: Chassis wheel activation, Jetson-STM32 terminal comms established, first MCU board failure due to servo wiring incident.

Weeks 3-4: Four-wheel actuation via Jetson keyboard. Current measurements taken. Final wiring diagram completed. Encoder hookup started.

Weeks 5-6: Linear slides assembled. Servo mount, RampV3, and STM32 housing designed and 3D-printed. Arm assembled. Electronic layout finalized.

Weeks 7-8: Top and middle layers assembled. Battery installed. ROS 2 teleop driving achieved. 15-min endurance run completed. PB10 damaged; servo rewired to PC10.

Weeks 9-10: All firmware modules complete and compiling clean. CubeMX encoder timers configured. Preparing for hardware validation and final testing.

Spring 2026 (EE 464):

Weeks 1-2: RealSense camera and RPLiDAR mounted and connected. Trash bin fabricated and installed. Web UI created for arm/scoop tuning and system control.

Weeks 3-4: Arm scoop trajectory tuned through the web UI. Stepper/servo waypoints saved to STM32 flash and replayed from the home position.

Weeks 5-6: Encoder odometry, heading filtering, EKF-based localization, slam_toolbox occupancy mapping, and RViz visualization integrated. Stable odom -> base_link transform published.

Weeks 7-8: Pure pursuit path following, Theta* trajectory planning, visual obstacle avoidance, and end-to-end autonomy debugging completed. Controller gains, velocity limits, mapping parameters, and target-approach variables tuned.

Weeks 9-10: Senior Project Expo demonstration prepared and presented. Final report writing, figures, testing documentation, and presentation materials completed.

Assembly Progression

The following photographs document the mechanical assembly process from early frame construction through full multi-level integration. The mechanical assembly of the robot was carried out primarily in the Cal Poly machine shop. Fabrication began with preparing the structural components that form the robot chassis. Several aluminum frame components were cut on the water jet using dimensions from the CAD models. This process allowed the team to rapidly produce plates and brackets with accurate hole placement, improving alignment between the manufactured parts and the modeled design. After cutting, the parts were inspected and deburred to remove rough edges before assembly.

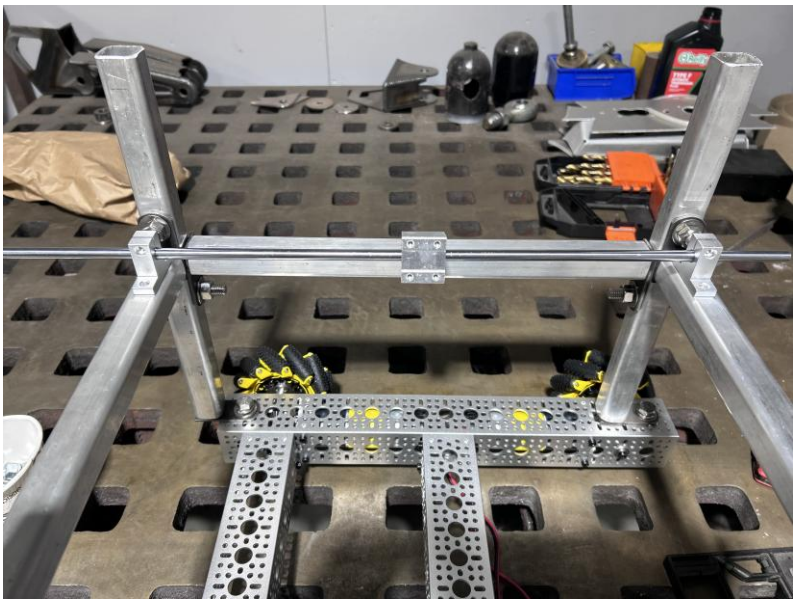

Figure 23. Chassis Frame with Vertical Supports and Linear Slides

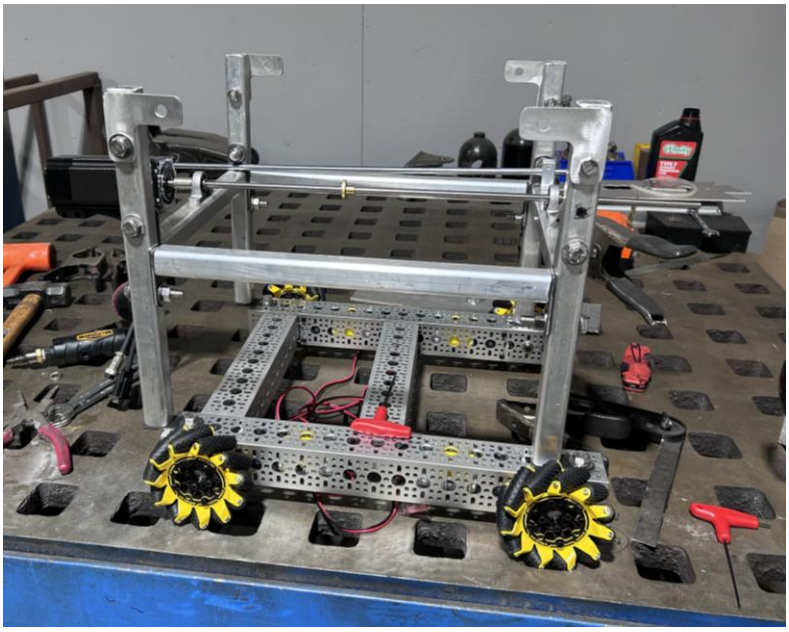

Figure 24. Chassis with Lead Screw and Mecanum Wheels - Side View

Linear slides, which guide the end-effector platform and constrain motion to a single translation axis, were then fitted to the custom chassis as shown in Figure 23. In the final design, these slides support the platform that is driven by the stepper motor through a lead screw, enabling controlled extension and retraction of the arm carriage along the chassis.

Figure 25. Chassis Assembly Outdoors - Lead Screw and Linear Slides Visible

With the lead screw, carriage, and wheel modules installed Figure 25, the team iterated on the height of the middle layer to balance reach and packaging constraints. A higher middle-layer height improves vertical clearance and simplifies mounting of the slide hardware, while a lower height reduces required arm length to reach the ground-level scooping region and improves reach to debris farther from the chassis

footprint. The higher-height prototype configuration is shown in Figure 25, where the linear slides are mounted on top of the middle-layer supports. This configuration was later revised to the lower-height design to improve effective scooping reach and reduce the required arm extension.

After the middle layer, linear slides, and lead screw were positioned, a mounting solution was required for the stepper motor due to its mass and torque reaction loads. The stepper motor was mounted using the custom machined bracket design introduced earlier Figure 5 and integrated into the chassis during the machine-shop build. In parallel, the top-layer structure was assembled to support the compute and sensing payloads, including the camera, voltage converters, and LiDAR mounting provisions; the completed multi-level frame structure is shown in Figure 26.

Figure 26. Multi-Level Frame Assembled with Mecanum Wheels

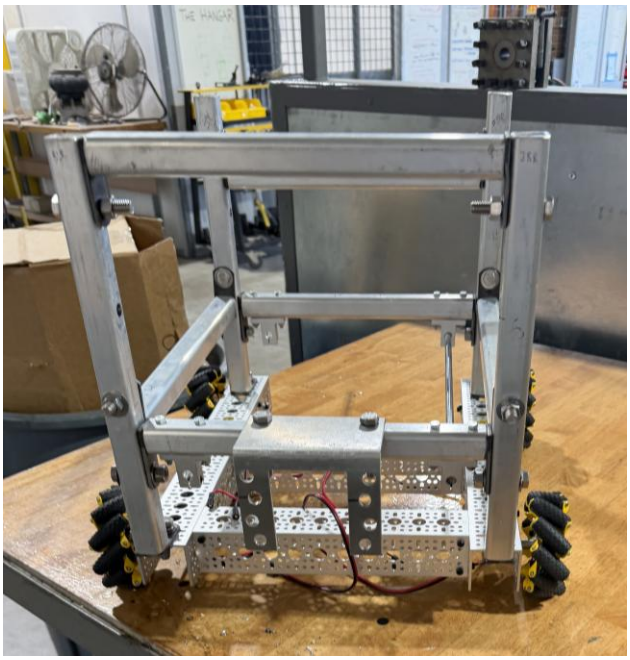

Figure 27. Robot Chassis on Workbench in Machine Shop

Once the primary structural components were fabricated, the frame was assembled using standard fasteners and mounting hardware. During this stage, the drill press was used to add or adjust hole patterns to improve alignment and ensure proper spacing between frame members. Careful layout and measurement were used to maintain the intended chassis geometry during these modifications

Figure 28. The stepper motor was initially a concern, but when installed on the dedicated mount

Figure 28, the chassis provided adequate support for the motor and actuation axis.

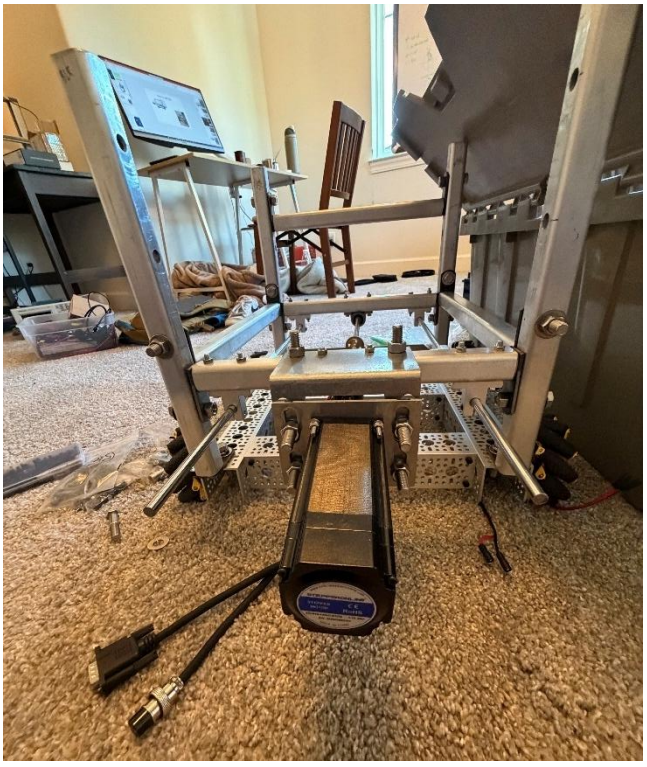

Figure 28. Stepper Motor Attached to Mount

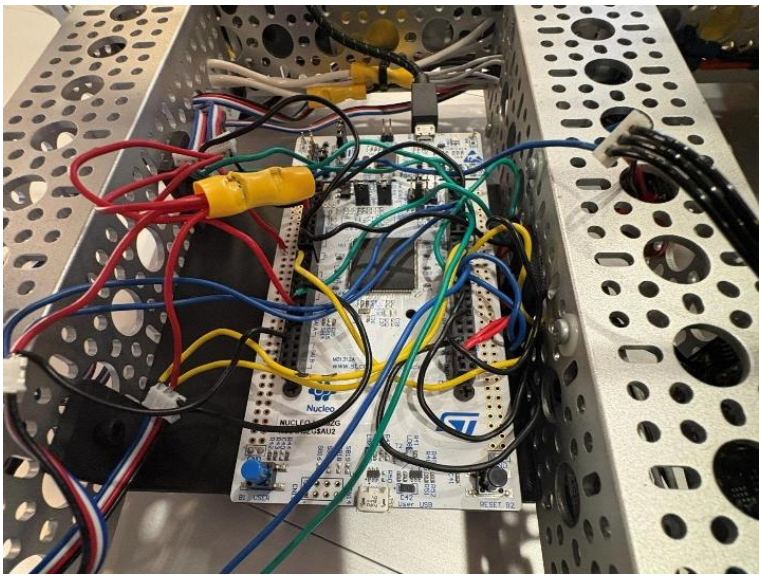

Figure 29. Assembled Wiring of STM32

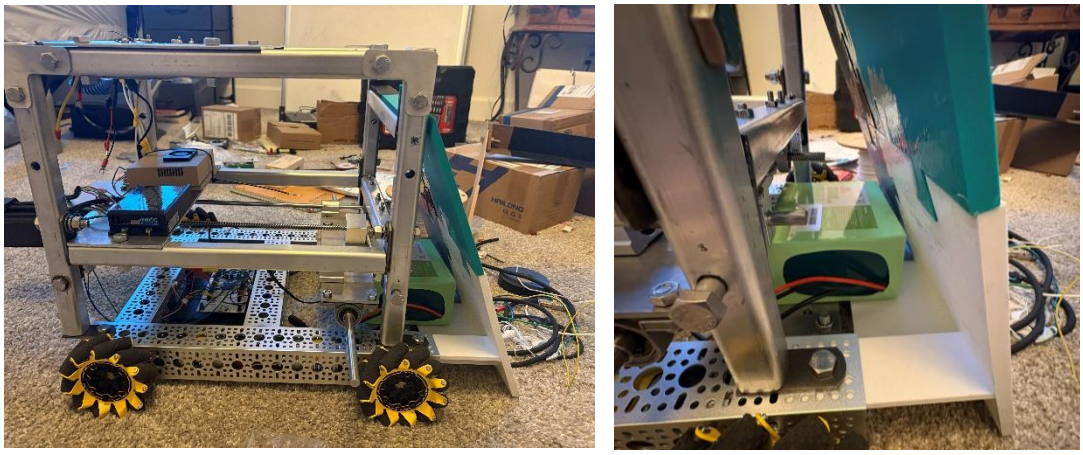

Figure 30. Battery Installation with Fully Integrated Wiring

In addition to mechanical assembly, wiring was integrated to provide power distribution, signal routing to the STM32, and voltage conversion for subsystems. The 24 V battery is housed in the lower front cavity of the chassis, taking advantage of packaging volume created by the ramp region (Figure 30). Power wiring was routed

from the battery to the 24→12 V and 24→19 V converters, and regulated rails were then distributed to the Jetson (19 V) and to the motor drivers and actuators (12 V). The physical wiring integration at the STM32—wheel motor driver signals, wheel encoder inputs, stepper motor control signals, and the servo bus—is shown in Figure 29, and the buck converter placement on the underside of the top layer is shown in Figure 31. Signal routing was performed to match the system wiring diagram presented earlier (Figure 12)

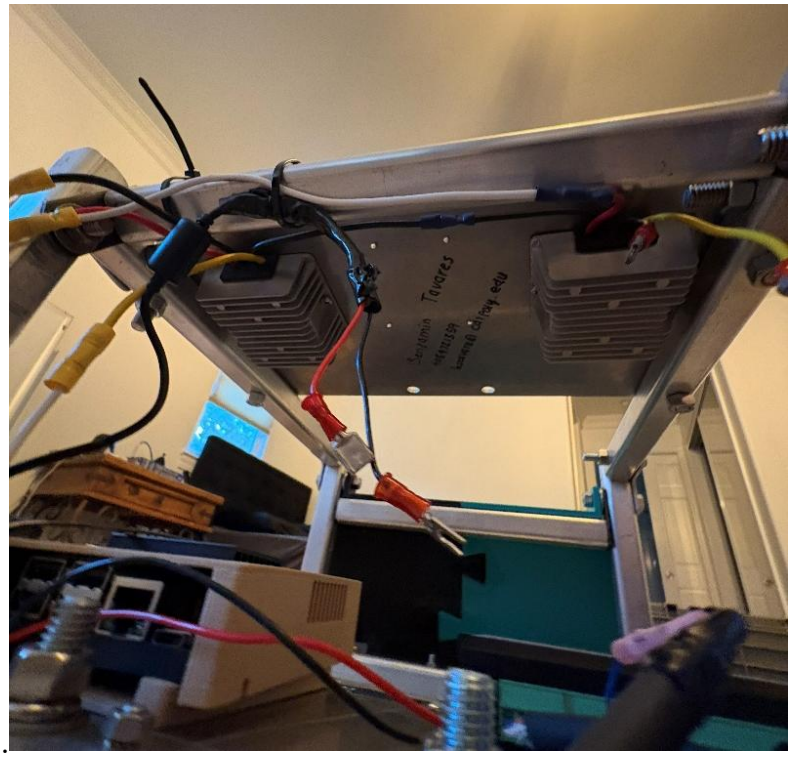

Figure 31. Location of Buck Converters on Underside of Top Layer

Following mechanical integration and wiring completion, the robot reached a configuration suitable for remote operation and real-time motion testing. The fully assembled chassis showing the multi-level frame, installed internal subsystems, and ramp integration is shown in Figure 32.

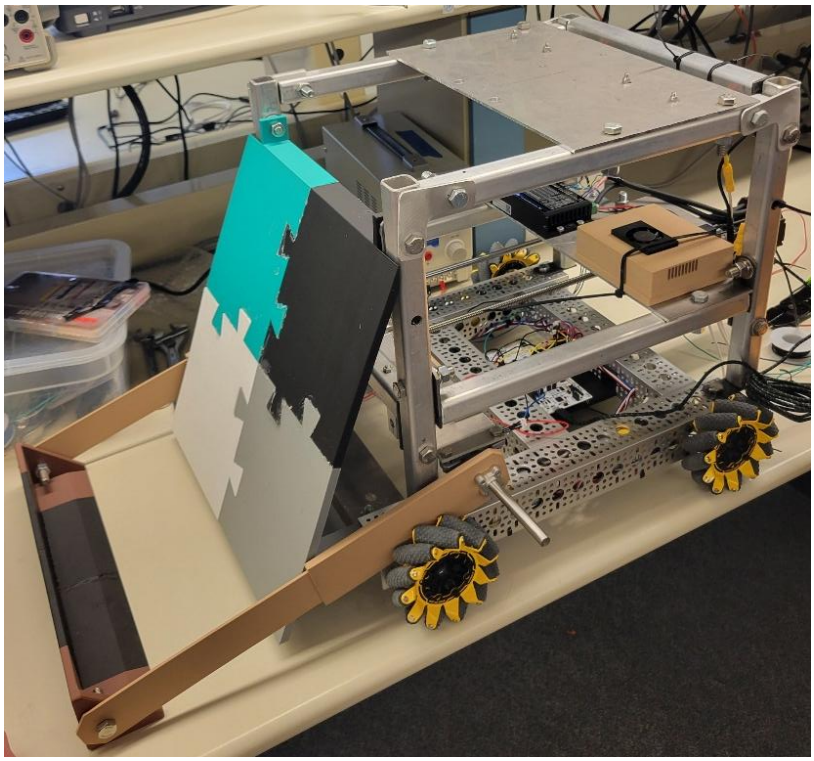

Figure 32. Complete Assembly - Multi-Level Frame with All internal Subsystems

Challenges During Construction

MCU board failures: Two NUCLEO-144 boards were damaged during development. The first failure was caused by an overcurrent event stemming from incorrect servo wiring, and the second occurred when the PB10 pin was physically

damaged during rework. Maintaining spare boards was critical to avoiding major schedule delays.

Pin conflicts and rework: Several late-stage pin conflicts required both firmware and wiring changes. The stepper control signals were reassigned from PA0/PB0 to PD14/PF15, and the RR motor direction signal was moved from PB10 to PF14, which required CubeMX reconfiguration, code updates, and harness rework.

Funding and packaging constraints: A planned grant was not awarded, so the team self-funded components and adjusted design choices accordingly. Mechanical packaging also drove late changes, including relocating the battery from the bottom level to accommodate servo clearance and revising the collection approach from a front scoop to a top-loading strategy.

3D printing tolerances: Several custom printed parts required iteration, most notably the STM32 housing, which was redesigned to improve cable routing and fitment. This reinforced the need to design with manufacturing tolerances in mind (≥0.5 mm clearance) and to plan for multiple print cycles during integration.

Material constraints and precision cutting: Working with limited materials required a disciplined approach to fabrication. The team adopted a "measure twice, cut once" methodology to ensure drill holes aligned correctly and cuts were precise, minimizing waste throughout the assembly process.

Custom part design and iteration: Several components required custom-designed parts that did not exist off the shelf. This included a custom stepper motor mount and a custom carriage, and ramp, all of which went through multiple rounds of CAD

modeling, prototyping, and revision before arriving at final designs that fit and functioned correctly.

Red tag certification delays: Obtaining red tag certification took approximately three weeks, which was required before the team could operate machine shop equipment independently. Even after certification, shared shop availability resulted in frequent wait times, and there was an inherent learning curve in familiarizing with the proper operation of some tools.

VIII. Conclusion

Summary of Achievements

The LitterBot project successfully demonstrated an autonomous vision-based litter collection rover capable of detecting, approaching, and collecting ground-level litter in indoor environments. The completed system integrated computer vision, LiDAR-based mapping, localization, autonomous path planning, obstacle avoidance, embedded motor control, and a custom two-degree-of-freedom collection mechanism into a single robotic platform.

The final system utilized an NVIDIA Jetson Orin Nano running ROS 2 Humble for perception, localization, and autonomy, while an STM32L4A6ZG microcontroller provided real-time control of the drivetrain and collection mechanism. A YOLO-based object detector was used to identify litter targets, and a custom Theta* path planner generated collision-free trajectories using occupancy-grid maps generated from LiDAR data.

Testing verified successful operation of the major subsystems including litter detection, autonomous navigation, obstacle avoidance, teleoperation, and robotic collection. End-to-end testing demonstrated successful autonomous litter collection across all test trials, validating the integration of the perception, planning, control, and collection subsystems. The completed platform satisfied the primary project

objectives and demonstrated the feasibility of autonomous litter collection using commercially available hardware and open-source robotics software.

Remaining Work

Although the final system successfully demonstrated autonomous litter collection, several areas remain for improvement. Obstacle avoidance performance was functional but did not achieve perfect reliability during testing, indicating that additional tuning of the planner and obstacle handling logic would improve robustness. Pickup performance was highly dependent on final robot alignment and litter orientation, making collection success sensitive to target position and environmental conditions.

Additional validation testing could also be performed to further quantify system performance under a wider range of operating conditions, including varying lighting conditions, terrain types, obstacle configurations, and litter categories. Long-duration endurance testing and environmental testing were outside the scope of the current project but would be beneficial for future deployments.

The primary system-level limitation of this prototype was its inability to demonstrate reliable autonomous operation in a general outdoor environment. However, the team remains confident that this iteration of LitterBot could operate effectively in a flat outdoor or semi-structured environment, such as a parking lot, paved courtyard, or factory floor, after separate tuning of the commanded velocities and path-following controller gains. The drivetrain, sensing stack, and autonomy pipeline are most

compatible with flat, high-traction surfaces where ground clearance and surface discontinuities do not dominate the mobility behavior.

Testing showed that the chassis clearance and frame geometry limited traversal over inclined or height-varying terrain. On these surfaces, the robot could contact the ground or lose mobility before the autonomy stack could be fully evaluated. This prevented validation of the intended broad outdoor use case, including unsupervised litter collection in environments such as festival venues, sidewalks, or elevation-varying campus walkways.

As a result, the completed prototype should be considered a flat-surface autonomous litter-collection platform rather than a fully outdoor-capable rover. Future iterations should increase ground clearance, reduce low-frame interference points, and retune the path controller and velocity limits specifically for outdoor surface conditions.

Lessons Learned

Multiple development issues highlighted the importance of disciplined bring-up and configuration management. Three NUCLEO-144 boards were lost during development, reinforcing the need to verify power wiring with a multimeter before connection and to maintain spare critical hardware. Early ad-hoc pin assignment led to conflicts and rework, motivating a complete pin map prior to soldering or harness finalization. CubeMX regeneration overwrote code outside user sections, emphasizing strict adherence to USER CODE regions and version control. The bus

servo interface required more effort than anticipated due to half-duplex signaling at 1 Mbaud and a proprietary packet format, underscoring the need to budget time for nonstandard protocols. Mechanically, 3D-printed housings required iteration, reinforcing the value of designing with adequate tolerance (≥0.5 mm clearance) and expecting multiple print passes. Finally, the modular test approach significantly reduced debugging time by isolating failures to individual subsystems before full integration.

Future Work

Future development efforts should focus on improving collection reliability, navigation robustness, and long-term deployability. Collection performance could be improved through redesign of the scoop geometry, wider collection tolerances, and additional sensing to verify successful pickup. Navigation performance could be enhanced through improved localization techniques, additional sensor fusion, and more advanced obstacle avoidance strategies.

Future versions of the platform could incorporate automatic return-to-base functionality, battery monitoring and autonomous charging behavior, weather-resistant enclosures, and cloud-based fleet management for multi-robot operation. Additional training data collected under diverse environmental conditions could further improve object detection performance and increase robustness against lighting variations and visually challenging backgrounds.

The LitterBot platform also provides a foundation for future research in autonomous environmental cleanup, robotic waste management, and multi-agent coordination systems. Continued development could expand the system beyond litter collection into broader autonomous maintenance and environmental monitoring applications.

IX. Bibliography

[1] Anthropic, “Claude Code (CLI tool),” STM32 firmware development, driver code generation guidance via CubeMX, firmware debugging, and documentation assistance.

[2] OpenAI, “ChatGPT,” research assistance, concept explanation, troubleshooting support, and report editing assistance.

[3] Cytron Technologies, “MD10C Enhanced 13A DC Motor Driver,” User Manual, Rev. 2.0, 2023.

[4] STMicroelectronics, “STM32L4A6xG Datasheet,” Rev. 4, 2023.

[5] Leadshine, “CL57T Closed Loop Stepper Driver,” User Manual, Rev. 1.2, 2023.

[6] FEETECH, “HX-65HM Bus Servo,” Technical Documentation, 2024.

[7] goBILDA, “Yellow Jacket Motor 5203 Series – 19.2:1,” Product Specifications, 2024.

[8] H. Taheri, B. Qiao, and N. Ghaeminezhad, “Kinematic model of a four mecanum wheeled mobile robot,” International Journal of Computer Applications, vol. 113, no. 3, pp. 6–9, Mar. 2015.

[9] NVIDIA, “Jetson Orin Nano Developer Kit User Guide,” 2024.

[10] Open Robotics, “ROS 2 Humble Documentation,” 2024.

[11] Cal Poly Electrical Engineering Department, EE 460/461/462 and EE 460/463/464 Senior Project Handbook, California Polytechnic State University, San Luis Obispo, CA, USA, 2023–2024.

[12] B. Tavares, N. Heil, and D. Benedetti, Autonomous Vision-Based Litter Collection Rover, EE 463 Senior Project Report, California Polytechnic State University, San Luis Obispo, CA, USA, Mar. 2026.

[13] N. Heil, EE 464 LitterBot Logbook, Quad Charts, Weekly Milestones, and Midquarter Presentation Materials, California Polytechnic State University, San Luis Obispo, CA, USA, Apr.–May 2026.

[14] Intel RealSense, “Intel RealSense Documentation,” Intel Corporation, 2026.

[15] S. Macenski, “SLAM Toolbox Documentation,” ROS 2 asynchronous SLAM package documentation, 2026.

[16] Open Robotics, “Navigation2 Documentation,” ROS 2 Navigation Framework, 2026.

[17] pLitterStreet, “pLitterStreet Dataset Repository,” litter detection dataset and training resources, 2026.

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

[19] Ultralytics, “Ultralytics YOLO26,” Ultralytics Docs. Accessed: Feb. 5, 2026. Available: https://docs.ultralytics.com/models/yolo26/

[20] OpenCV, “Camera Calibration and 3D Reconstruction,” OpenCV Documentation. Accessed: Feb. 12, 2026. Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html

[21] Open Robotics, “Tf2,” ROS 2 Humble Documentation. Accessed: Feb. 12, 2026. Available: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Tf2.html

[22] K. Daniel, A. Nash, S. Koenig, and A. Felner, “Theta*: Any-Angle Path Planning on Grids,” Journal of Artificial Intelligence Research, vol. 39, pp. 533–579, 2010, doi: 10.1613/jair.2994.

[23] R. C. Coulter, “Implementation of the Pure Pursuit Path Tracking Algorithm,” Carnegie Mellon University, Robotics Institute, Pittsburgh, PA, USA, Tech. Rep. CMU-RI-TR-92-01, Jan. 1992. Available: https://publications.ri.cmu.edu/implementation-of-the-pure-pursuit-path-tracking-algorithm/

[24] SLAMTEC, “RPLIDAR-A1 Cost-Effective 360° Lidar Sensor,” SLAMTEC. Accessed: Jan. 5, 2026. Available: https://www.slamtec.com/en/lidar/a1

[25] OpenAI, "OpenAI Codex CLI," OpenAI, version 0.125.0, 2026.

Appendix A: Analysis of Senior Project Design

Project Title: Autonomous Vision-Based Litter Collection Rover

Students: Benjamin Tavares, Nathan Heil, Dante Benedetti

Advisor: Dr. McKell

Date: March 2026

Summary of Functional Requirements

The LitterBot is an autonomous rover that detects, navigates to, and collects small ground-level litter (bottles, wrappers, paper) in outdoor environments. It uses computer vision (YOLO11 on Jetson) for litter detection, RPLiDAR for mapping, mecanum wheels for omnidirectional mobility, and a two-DOF robotic arm (stepper + servo) for litter pickup. The system operates autonomously on battery power for extended periods of time.

Primary Constraints

Significant constraints included: limited budget (grant denied, self-funded); weight of the assembled robot (42 lbs) requiring robust motors; real-time determinism requiring a two-layer compute architecture; pin limitations on the STM32 NUCLEO-144 requiring multiple pin reassignments; physical board damage requiring hardware rerouting; and 3D print tolerances requiring multiple design iterations.

Economics

Original estimated cost of component parts: ~$1000-1500

Actual final cost of component parts: ~$2,477 (self-funded)

Additional equipment costs: 3D printer filament (~$30x3)

Original estimated development time: 3 quarters (9 months)

Actual development time: 3 quarters (9 months)

If manufactured on a commercial basis:

- Estimated number of devices sold per year: 500-1,000 (university/municipal market)
- Estimated manufacturing cost per device: $1500 (at volume)
- Estimated purchase price per device: $2,500
- Estimated profit per year: 1,000 (units) * $ 1,000 = 1,000,000

Environmental

The rover's primary purpose is environmental improvement through automated litter removal. The electric drive produces no emissions; DC motors operate off of a lithium battery requires proper end-of-life recycling per hazardous waste regulations. PLA 3D-printed components are industrially compostable. No solvents or water pollutants produced during operation. The aluminum frame is fully recyclable.

Manufacturability

Built mostly from COTS components (goBILDA, Cytron, NUCLEO, Jetson). Simple and reproducible custom machining was required, and custom part files available for sheet metal brackets; other custom parts 3D-printed in PLA. Assembly requires basic hand tools, approximately 4-6 hours with all parts in hand. Bolt-on subassemblies allow for easy replacement and maintenance on core subsystems and components. 3D print tolerances required iteration on STM32 housing and servo mount. Scaling to production would likely result in the use of injection molded parts for several plastic pieces.

Sustainability

Aluminum frame and mecanum wheels are durable and long-lasting. Most wear-prone component (scoop) is a replaceable 3D-printed part. Lithium battery has 300-500 cycle life and requires periodic replacement. All replacement parts are commercially available. Low environmental resource impact compared to fuel-powered sweepers. Future upgrades could include solar charging panels and weatherproofing for extended outdoor deployment.

Ethical

Camera used for litter detection could capture incidental images of people. Mitigated by: all processing performed on-device (no cloud upload), camera pointed downward at ground level. Low speed (<3.3 ft/s) and 0.3 s obstacle stop minimize harm risk.

Limited compute and camera resolution make surveillance repurposing impractical. Designed to complement, not replace, human workers. No personal data collected or stored.

Health and Safety

24 V Lithium battery is below 50 V hazardous threshold. Motor drivers (13 A/ch) require fusing. RPLiDAR uses Class 1 eye-safe laser. RF emissions limited to standard Bluetooth/Wi-Fi (FCC-compliant). Robot speed limited to walking pace for pedestrian safety. Future improvements will include an E-Stop Button as well as micro-limit switches to protect both the robot as well as pedestrians.

Social and Political

No national security implications. All COTS, non-restricted, open-source components. Civilian use only (parks, campuses, public spaces). Positive social impact: cleaner public spaces, affordable robotics education platform, reduced municipal cleaning costs. Fleet deployment could create new technical maintenance jobs while reducing manual litter pickup labor.

Development

Tools and techniques learned and develop during the project:

- STM32CubeIDE and CubeMX for MCU peripheral configuration and code generation.
- ROS 2 (Humble) framework for sensor integration, SLAM, and teleoperation.

- Half-duplex UART with push-pull GPIO override for proprietary servo protocol.
- Mecanum wheel forward kinematics and dead-reckoning odometry.
- SolidWorks CAD design and iterative 3D printing for custom enclosures.
- YOLO11
- Nav2 and slam_toolbox for autonomous navigation.

Engineering Standards

Engineering standards applied during project development:

- IEEE 802.11 (Wi-Fi) and IEEE 802.15.1 (Bluetooth) for wireless communication compliance.
- IEC 62133 for lithium battery safety in portable electronics.
- IEC 60825-1 Class 1 laser safety standard (RPLiDAR).

Appendix B: Parts List and Costs

TABLE XIII. BILL OF MATERIALS

Description	Quantity	Per Unit Price	Total Price
Servo Motor (Hiwonder Hx-65)	1	$ 49.99	$ 49.99
Nema - 23 Stepper Motor	1	$ 65.00	$ 65.00
128 gb Micro SD for Jetson	1	$ 23.99	$ 23.99
24V 20Ah Lithium Battery	1	$ 150.00	$ 150.00
Lead Screw + Linear guides (kit)	1	$ 30.00	$ 30.00
Jetson Orin Nano Super Dev Kit	1	$ 250.00	$ 250.00
Gobilda Strafer Chassis	1	$ 750.00	$ 750.00
Intel Realsense D455	1	$ 450.00	$ 450.00
RpLidar 2-D Lidar	1	$ 110.00	$ 110.00
Aluminum Square Tubing (15ft)	2	$ 42.50	$ 85.00
PLA (3D-Print Material)	1	$ 19.99	$ 19.99
Cytron md10 motor driver	4	$ 15.49	$ 61.96
24v-12v buck converter	1	$ 25.00	$ 25.00
24v-19v buck converter	1	$ 16.99	$ 16.99
Bearing Mount	4	$ 4.05	$ 16.20
10mm Shaft For Arm	2	$ 9.99	$ 19.98
10mm - 10mm Shaft Coupler	1	$ 19.99	$ 19.99
10mm Shaft For Arm	2	$ 4.59	$ 9.18
1/8” Aluminum Sheet Metal	1	$ 55.00	$ 55.00
22AWG Wires (multicolor set)	1	$ 15.99	$ 15.99
Micro Limit Switches for Safety	1	$ 15.99	$ 15.99
Assorted Fasteners	1	$ 200.00	$ 200.00
STM32L4A6ZG Nucleo-144	1	$ 26.99	$ 26.99
Assorted Crimp Connectors	1	$ 9.99	$ 9.99
Total			**$ 2,477.23**

Appendix C: Schedule - Time Estimates

TABLE XIV. DEVELOPMENT SCHEDULE

Phase	Duration	Key Deliverables
EE 460 (Fall 2025)	10 weeks	Architecture, specs, first-pass report, NABC, AHP
EE 463 Weeks 1-2	2 weeks	Wheel activation, Jetson-STM32 comms, first board failure
EE 463 Weeks 3-4	2 weeks	4-wheel drive, current measurements, wiring diagram
EE 463 Weeks 5-6	2 weeks	Linear slides, 3D printing, arm assembly
EE 463 Weeks 7-8	2 weeks	Full assembly, battery, ROS 2 teleop, 15-min drive
EE 463 Weeks 9-10	2 weeks	All firmware complete, CubeMX config, test prep
EE 464 (Spring 2026)	10 weeks	HW testing, Nav2, YOLO CV, full autonomous

Estimated total development hours per student: ~200 hours (exceeds 180-hour minimum).

Appendix D: Wire List / Pin Assignments

TABLE XV. STM32L4A6ZG PIN ASSIGNMENTS

Pin	Function	Subsystem	Notes
PA0	TIM2_CH1 PWM	DC Motor FL	Speed control
PA1	TIM2_CH2 PWM	DC Motor FR	Speed control
PA2	TIM2_CH3 PWM	DC Motor RL	Speed control
PA3	TIM2_CH4 PWM	DC Motor RR	Speed control
PA6	TIM3_CH1 Enc CHA	FR Encoder	White wire
PA7	TIM3_CH2 Enc CHB	FR Encoder	Blue wire
PB0	DIR output	DC Motor FL	Direction
PB1	DIR output	DC Motor FR	Direction
PB2	DIR output	DC Motor RL	Direction
PB6	TIM4_CH1 Enc CHA	FL Encoder	White wire
PB7	TIM4_CH2 Enc CHB	FL Encoder	Conflicts LD2
PB14	LD3 output	Board LED	Encoder test feedback
PC6	TIM8_CH1 Enc CHA	RL Encoder	White wire
PC7	TIM8_CH2 Enc CHB	RL Encoder	Blue wire
PC10	USART3 TX	Servo	Half-duplex; was PB10
PC13	B1 Button	User Input	Active LOW
PD14	PUL+	Stepper	Step pulse
PE9	TIM1_CH1 Enc CHA	RR Encoder	White wire
PE11	TIM1_CH2 Enc CHB	RR Encoder	Blue wire

PF14	DIR output	DC Motor RR	Was PB10
PF15	DIR+	Stepper	Direction
PG7	LPUART1 TX	Debug UART	ST-LINK VCP
PG8	LPUART1 RX	Debug UART	ST-LINK VCP

Appendix E: Program Listing

The complete firmware source code is maintained in a Git repository. Key source files are listed below. Full source is available at LitterBot_Officical; see appedndix G.

TABLE XVI. FIRMWARE SOURCE FILES

File	Path	Lines
main.c	Core/Src/	Main loop, peripheral init, test dispatch
cytron_md10c.c	Core/Src/	DC motor driver (PWM, direction)
cytron_md10c.h	Core/Inc/	DC motor API declarations
cl57t_stepper.c	Core/Src/	Stepper driver (GPIO bit-bang)
cl57t_stepper.h	Core/Inc/	Stepper API declarations
hx65_servo.c	Core/Src/	Bus servo protocol (half-duplex UART)
hx65_servo.h	Core/Inc/	Servo API declarations
encoder.c	Core/Src/	Quadrature encoder driver
encoder.h	Core/Inc/	Encoder API declarations
odometry.c	Core/Src/	Mecanum odometry (forward kinematics)
odometry.h	Core/Inc/	Odometry API declarations
uart_cmd.c	Core/Src/	UART command parser
uart_cmd.h	Core/Inc/	UART command API declarations
servo_test.c	Core/Src/	Servo sweep test
dc_motor_test.c	Core/Src/	DC motor test
stepper_test.c	Core/Src/	Stepper test
combined_test.c	Core/Src/	Combined integration test

encoder_test.c	Core/Src/	Encoder readback test

Appendix F: Hardware Configuration / Layout

The rover hardware is organized in a three-level chassis configuration:

Bottom Level: Lithium battery (front cavity), STM32 NUCLEO-144 in 3D-printed housing (rear cavity), 4x Cytron MD10C motor drivers on cross-strut, 4x goBILDA Yellow Jacket motors with mecanum wheels.

Middle Level: NVIDIA Jetson Orin Nano, CL57T stepper driver, cut aluminum sheet platform.

Top Level: RPLiDAR A1, Intel RealSense D455f, buck converters (19V, 12V).

Arm Assembly: NEMA 23 stepper with lead screw and linear slides (vertical axis), HX-65HM bus servo (rotation axis), 3D-printed scoop and ramp (Ramp V3, 4 interlocking puzzle pieces).

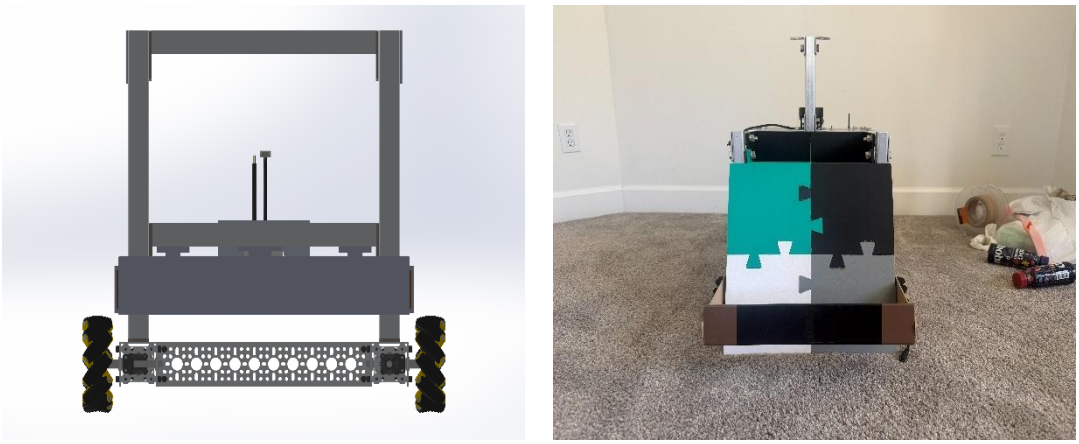

Figure 33. Front View

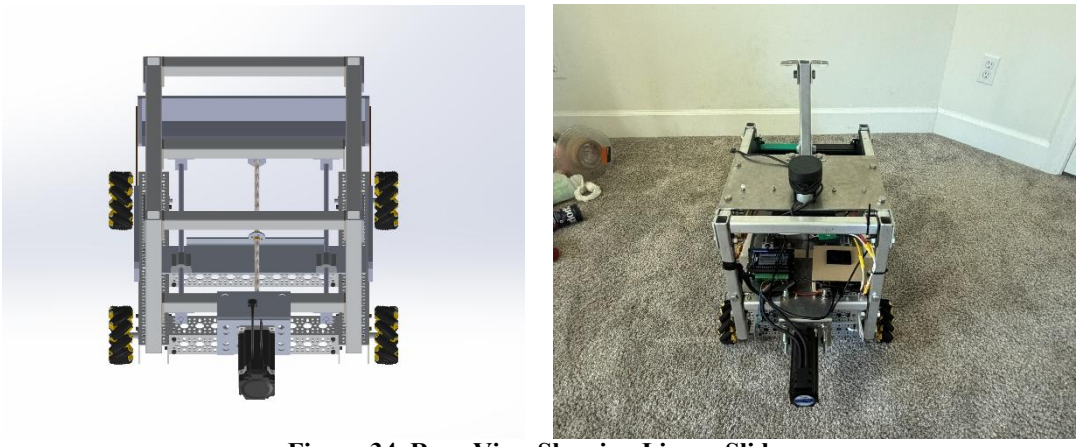

Figure 34. Rear View Showing Linear Slides

Figure 35. Isometric Rear View

Appendix G: Reference Material

This appendix contains supplemental materials supporting the testing, validation, and development activities discussed throughout this report. These materials include demonstration videos, source code repositories, project documentation, datasets, and other reference resources that provide additional context but are not included in the main body of the report. The contents document the robot's autonomous navigation, path-following performance, obstacle avoidance behavior, system architecture, and overall operation under representative test conditions.

For the trajectory planning and control tests, **Run 1** corresponds to the original controller implementation prior to tuning and controller modifications, while **Run 2** corresponds to the updated controller implementation.

TABLE XVII: VIDEOS AND LINKS

Material ID	Description	Corresponding Section(s)	Link
G-1	Full Software and Firmware GitHub Repository	Software Design; Firmware Design; Autonomy Design;	https://github.com/benjam1ntavares/LitterBot_Official.git
G-2	ROS 2 Node Graphs, Simple and expanded	Software Architecture drive link	https://drive.google.com/drive/folders/1QGTmoExtOoCXxdMY_hJgOS7si1lYLD0k
G-3	Obstacle Run 1 (UI)	Trajectory Planning and Control	https://www.youtube.com/watch?v=cPLhsfcqaT0&feature=youtu.be
G-4	Obstacle Run 1 (Live)	Trajectory Planning and Control	https://www.youtube.com/watch?v=EIHyWfRen9A
G-5	No Obstacle Run 1 (UI)	Trajectory Planning and Control	https://www.youtube.com/watch?v=xJT2ZUeigI8
G-6	No Obstacle Run 1 (Live)	Trajectory Planning and Control	https://www.youtube.com/watch?v=wuPBiPrMTJ4
G-7	Obstacle Run 2 (UI)	Trajectory Planning and Control	https://www.youtube.com/watch?v=m4AevvASR3k&feature=youtu.be

G-8	Obstacle Run 2 (Live)	Trajectory Planning and Control	https://www.youtube.com/watch?v=nTEywUycqD4&feature=youtu.be
G-9	No Obstacle Run 2 (UI)	Trajectory Planning and Control	https://www.youtube.com/watch?v=Bzz7ScIDNos&feature=youtu.be
G-10	Teleoperation Demo	Software Design and Firmware Design	https://www.youtube.com/shorts/oSF1Ql-q9L8