Jabari Robotics Dojo Technical Design Paper Robotics Dojo 2025

Japheth Libese, Jerry Kiche, Clinton Sabi, Nathan Mwaura, Ian Ndirangu (Jabari)

***Abstract*—This paper presents the design and implementation of an autonomous rover developed for the JKUAT Robotics Dojo competition, aimed at navigating a structured game field with object transport capabilities. The rover integrates multiple sensing modalities: including LIDAR, camera vision, and ultrasonic sensors under the ROS 2 (Robot Operating System) framework to enable simultaneous localization and mapping (SLAM), obstacle avoidance, and goal-oriented navigation. The robot is capable of autonomously mapping its environment, detecting and classifying objects via computer vision, transporting a payload across the field, and offloading it at a designated location. Emphasis is placed on sensor fusion, robust path planning, and real-time autonomy in dynamic conditions. This work demonstrates a practical implementation of autonomous mobile robotics for logistics-style tasks in a controlled competition environment.**

I. Introduction

Autonomous mobile robots have become an essential component in modern robotics research and practical applications, ranging from industrial automation to exploration, logistics, and smart environments. The demand for robots capable of intelligent navigation, real-time decision-making, and efficient task execution continues to grow, driven by advancements in sensing, computation, and open-source robotic frameworks.

This paper presents the development of an autonomous rover designed for participation in the JKUAT Robotics Dojo competition. The primary objective of the project is to build a mobile robot capable of navigating a structured game field, mapping the environment in real time, transporting a physical object onboard, and accurately offloading it at a designated target location all without human intervention.

To achieve full autonomy, the robot utilizes the Robot Operating System 2 (ROS 2), which provides a modular and scalable middleware for integrating various components such as LIDAR for environmental mapping, a camera for visual perception and object detection, and ultrasonic sensors for obstacle avoidance. The system leverages SLAM (Simultaneous Localization and Mapping) to dynamically construct a map of the game field while planning optimal paths to reach goals efficiently and safely.

This paper details the hardware integration, software architecture, navigation algorithms, and perception strategies implemented to achieve autonomous operation. The project aims not only to address the specific challenge of the competition field but also to demonstrate a scalable approach to autonomous robotics that can be adapted to real-world applications in logistics, surveillance, and delivery

II. Paper Contents

A. Mechanical Design Strategy

The main frame of the rover is constructed from *laser-cut acrylic sheets*, chosen for their affordability, ease of prototyping, and sufficient rigidity for indoor robotics applications. Acrylic panels serve as the base platform for mounting all mechanical and electronic components, including motors, sensors, battery, and control boards.

3D printed brackets and mounts were used to hold components such as the LIDAR, camera, and servo mechanism. These custom parts were designed using CAD software (Fusion) and printed using PLA filament, allowing for fast iterations and design customization.

The chassis designed is modular allowing for quick access to internal components, making maintenance and upgrades more efficient.

The rover uses a *4-wheel drive system*, with each wheel independently powered by a *DC geared motor*. This configuration provides enhanced traction and torque, especially useful when carrying payloads or navigating ramps and textured surfaces.

Each motor is rigidly mounted onto the acrylic chassis using brackets and fastened by bolts and nuts, ensuring alignment and minimizing vibration. The differential drive control is implemented in software, allowing the rover to perform smooth turns, pivots, and straight-line movements using speed and direction control of individual motors.

The 4WD configuration contributes to:

1. Improved load distribution
2. Better grip during acceleration or braking
3. Enhanced stability, especially when the payload center of mass shifts

A key mechanical feature of the rover is its *object handling and offloading mechanism*, which enables it to transport and release a payload at the target zone.

The mechanism consists of a *servo-actuated lowering platform*, designed to secure the payload during transit and release it by tilting or lowering at the destination. The platform is hinged at the back and attached to a *standard servo motor*, which is controlled via the microcontroller through a ROS node to execute pre-programmed movements.

Design considerations for this mechanism included:

1. Low weight to avoid overloading the servo
2. Sufficient clearance from ground and sensors
3. Secure locking during motion to prevent payload shift

3D printed parts were used to build the lowering tray and linkages, allowing easy customization and adjustments during the testing phase.

Mounting points for sensors were carefully designed to optimize field of view and minimize interference:

1. *LIDAR* was placed centrally and elevated to provide an unobstructed 360-degree scan
2. *Camera* was mounted at the front with a slight downward tilt to assist in object recognition and path alignment
3. *Ultrasonic sensors* were positioned on the front and sides to detect close-range obstacles

Protective housings were added around fragile components to shield them from impacts during field operation.

B. Electrical Design Strategy

Power Supply:

A pack of four 18650 lithium-ion cells (nominal 14.8V) powers the high-current components, including the DC motors (via L298N drivers) and the servo motor. This pack is connected directly to the motor driver input terminals and a 5V step-down voltage regulator for peripherals requiring lower voltages.

Raspberry Pi Power Supply: A 20000mah USB power bank supplies stable 5V directly to the Raspberry Pi via its USB-C input. This ensures the Pi receives uninterrupted and clean power, independent of the motor power system. This was decided after realizing it was difficult to get a stable 5v from the stepped down voltage.

This dual-source architecture provides:

1. Electrical isolation between motors and control electronics
2. Reduced risk of Pi brownouts during motor spikes
3. Better battery management and modular debugging

A *common ground connection* is established between both systems to ensure signal consistency for serial communication between the Arduino and Raspberry Pi

An onboard power distribution board or wiring hub is used to split power lines to all components while keeping the system compact and manageable.

Each pair of DC motors (left and right) is controlled by an *L298N dual H-bridge motor driver*. The L298N modules are responsible for supplying current to the motors and allowing directional control (forward/reverse) and speed control via PWM.

To reduce wiring complexity and save microcontroller pins, the *Enable (EN) pins of each motor pair are looped together, For example:*

- *EN_A* for both left-side motors (looped)
- *EN_B* for both right-side motors (looped)

This setup allows for synchronized control of each side using a single PWM signal per side, which simplifies software control while preserving differential drive behavior.

Motor Driver Wiring:

1. *IN1, IN2* → Arduino digital pins (Left motor direction)
2. *IN3, IN4* → Arduino digital pins (Right motor direction)
3. *EN_A, EN_B* → Arduino PWM pins
4. *12V/VCC* → Battery positive
5. *GND* → Common ground
6. *OUT1–OUT4* → Connected to motors

Encoder Feedback

The motors are equipped with *encoders* to provide feedback for closed-loop control. Each encoder has two channels (A and B), producing quadrature signals which the *Arduino* reads to determine:

1. Wheel rotation direction
2. Speed (by measuring pulse frequency)
3. Distance (by counting pulses)

The encoder data is processed by the Arduino to enable features like:

1. Velocity control (PID)
2. Distance-based movement
3. Odometry for dead-reckoning (if needed by the ROS stack)

A *standard servo motor* is used to operate the payload lowering mechanism. The servo is powered by the 5V regulated line and is controlled by the *Arduino* via a single PWM pin.

1. *Signal* → Arduino digital PWM pin
2. *VCC* → 5V regulator output
3. *GND* → Common ground

The system uses a dual-controller architecture:

- Arduino Uno: Handles low-level motor control, encoder feedback, and servo actuation
- Raspberry Pi (4/3): Runs ROS 2, manages sensor processing, navigation algorithms, and decision making

The Raspberry Pi and Arduino communicate via USB serial or UART, with the Arduino exposing a custom serial interface for commands like:

- MOTOR_SPEED(left, right)
- GET_ENCODER_DATA()
- LOWER_PAYLOAD()

ROS 2 nodes on the Pi publish motor commands and subscribe to encoder data for navigation and SLAM integration.

C. Software Strategy

The software architecture for the autonomous rover was designed around the principles of *modularity***,** *real-time performance*, and *scalability,* with a clear separation between *high-level decision-making* and *low-level hardware control*. The system leverages the capabilities of *ROS 2 (Robot Operating System 2***)** running on a *Raspberry Pi*, in conjunction with an *Arduino* handling real-time motor and actuator control.

This division allows the system to perform complex navigation, mapping, and perception tasks while ensuring fast and deterministic control of the motors and sensors.

Layered Software Architecture

The software stack is divided into three main layers:

1. *Hardware Control Layer (Arduino)*
   Responsible for motor driving, encoder reading, and servo actuation. The Arduino exposes a simple serial interface to receive commands and send sensor feedback.
2. *Robot Operating Layer (ROS 2 on Raspberry Pi)*
   Manages all sensor inputs (LIDAR, camera, ultrasonic), mapping, localization, path planning, and autonomous decision-making using ROS 2 nodes and standard packages.
3. *Communication Layer (Serial Interface)*
   Facilitates robust data exchange between Raspberry Pi and Arduino, converting ROS 2 motion commands into low-level motor control signals and relaying encoder feedback for localization

The Arduino runs a *lightweight firmware loop*, written in C++, to perform the following functions:

- *Read encoder pulses* using interrupts and compute speed or position
- *Control motor speed and direction* via PWM and digital signals to L298N
- *Actuate the servo* when commanded for payload lowering
- *Parse serial commands* received from the Raspberry Pi
- *Send structured data* (e.g., encoder counts, servo status) back to the Pi at a regular rate

The rover uses LIDAR-based *Simultaneous Localization and Mapping (SLAM)* to autonomously build a map of the environment and localize itself in it. This is implemented using standard ROS 2 packages such as:

- slam_toolbox or cartographer for 2D mapping
- nav2 stack for localization and navigation
- tf2 for managing coordinate transforms between robot base, odometry, and map frames

Odometry from the Arduino (based on encoder data) is fused with LIDAR input to enhance localization accuracy.

The ROS 2 nav2 stack handles the motion planning tasks. It includes:

1. Global path planner
2. Local planner for obstacle avoidance (e.g., DWB controller)
3. Cost maps built from LIDAR and ultrasonic data
4. Goal setting from mission control (e.g., move to drop-off zone)

The *camera vision node* uses OpenCV (or a pre-trained lightweight model) to detect objects, align the rover, or verify drop-off success. Potential vision tasks include:

- Line detection or color tracking
- Object presence verification

- Zone identification using April Tags or fiducial markers

Camera data is processed onboard the Raspberry Pi, and detection results are used to trigger state transitions in the mission control logic.

On the laptop, computationally intensive tasks such as SLAM, the Nav2 navigation stack, and deep learning–based disease detection are executed. SLAM Toolbox generates a 2D map from LiDAR and odometry data, while Nav2 handles localization, global path planning, and obstacle avoidance. RViz2 provides a graphical interface for visualization and goal setting, enabling operators to monitor the robot in real time. A PyTorch-based vision node processes camera images to identify diseased plants, publishing results for further analysis. This distributed architecture balances processing loads across the devices and enables the robot to autonomously explore environments while contributing to precision agriculture through plant health monitoring.

D. Experimental Results

To validate the performance, reliability, and integration of the autonomous rover, a structured testing approach was adopted. Testing was conducted progressively throughout development, beginning with individual component verification (unit testing), followed by system-level integration and limited field trials. As of the technical design paper submission, the system has undergone multiple iterations of hardware-software validation both in simulation and on physical test environments modeled after the Robotics Dojo game field.

Unit and Integration Testing

Each major subsystem was first tested independently before being integrated into the complete system:

1. *Motor Control Testing*: The Arduino and L298N motor driver configuration was tested using simple serial commands to validate direction, speed, and braking behavior. Encoders were monitored for real-time feedback accuracy.
2. *Sensor Testing*: The LIDAR was tested in static and dynamic environments using ROS 2's rviz2 to ensure proper scan generation and data publishing. The ultrasonic sensors were tested for obstacle detection thresholds. The camera vision pipeline was validated with controlled image datasets and real-world lighting conditions.
3. *Servo Actuation*: The payload-lowering mechanism was actuated repeatedly to evaluate response time, position accuracy, and torque sufficiency under load.
4. *Communication Link*: Serial communication between the Raspberry Pi and Arduino was tested under load conditions, with consistent command execution and feedback confirmed over extended operation periods.

The team utilized *ROS 2 simulation tools (Gazebo)* to test core navigation and perception algorithms in a virtual environment before deployment. This allowed testing of:

1. SLAM performance in synthetic environments
2. Path planning and obstacle avoidance
3. Mission control state machine transitions
4. Safety overrides based on sensor input

Simulation accelerated development by allowing risk-free testing of high-level logic and edge cases that would be difficult to replicate physically.

To date, the rover has undergone *over 15 combined hours of physical testing*, spread across motion control tuning, SLAM trials, and object handling tests. Key milestones achieved include:

1. Successful 2D mapping of a 3x3 meter test field using LIDAR and SLAM toolbox
2. Obstacle detection and avoidance using ultrasonic sensors and costmap integration
3. Reliable point-to-point navigation using ROS 2 `nav2` stack
4. Payload pickup and offloading with repeatable servo actuation

Testing has also helped identify mechanical issues, such as servo overloading and wheel slippage, which are currently being addressed in hardware iterations.

Although a full quantitative reliability model has not been implemented yet, the team has performed several qualitative assessments and stress tests:

1. *Servo Life Testing*: The payload mechanism was cycled over 50 times continuously to check for mechanical wear or servo overheat. No significant degradation was observed.
2. *Battery Runtime Estimation:* Under full operation, the 4x18650 battery pack provides approximately 40–50 minutes of continuous drive and control, depending on terrain and load.
3. *Encoder Signal Stability*: The team observed occasional signal jitter at high speeds, leading to plans for interrupt signal filtering in future firmware versions.
4. *Structural Robustness:* While acrylic was chosen for prototyping, minor cracks developed under repeated mechanical stress—indicating a need for additional reinforcement or material substitution (e.g., aluminum or ABS) in future revisions.

The team is also exploring simple *failure mode scenarios*, such as motor stall, sensor dropout, and communication loss, to implement recovery strategies (e.g., timeouts,

E. Referees

[1] M. Quigley et al., "ROS: an open-source Robot Operating System," in ICRA Workshop on Open Source Software, vol. 3, no. 3.2, 2009.

[2] Open Robotics, "ROS 2 Documentation," [Online]. Available: https://docs.ros.org/en/ros2_documentation/index.html. [Accessed: Sep. 25, 2025].

[3] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *Proc. IEEE Int. Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2011, pp. 155–160.

[4] Google Cartographer, "Cartographer ROS Integration," [Online]. Available: https://google-cartographer.readthedocs.io. [Accessed: Sep. 25, 2025].

[5] R. Smits, "Control of a Servo Motor Using PWM on Arduino," *Arduino Project Hub*, 2020. [Online]. Available: https://create.arduino.cc/projecthub. [Accessed: Sep. 25, 2025].

[6] Texas Instruments, "L298 Dual Full-Bridge Driver Datasheet," *Texas Instruments*, 2005. [Online]. Available: https://www.ti.com/lit/ds/symlink/l298.pdf. [Accessed: Sep. 25, 2025].

[7] OpenCV.org, "OpenCV: Open Source Computer Vision Library," [Online]. Available: https://opencv.org/. [Accessed: Sep. 25, 2025].

[8] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The Office Marathon: Robust Navigation in an Indoor Office Environment," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 300–307.

[9] Arduino.cc, "Arduino UNO Technical Specs," [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. [Accessed: Sep. 25, 2025].

[10] Raspberry Pi Foundation, "Raspberry Pi 4 Model B Datasheet," [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html. [Accessed: Sep. 25, 2025].

[11] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Trans. Robotics*, vol. 23, no. 1, pp. 34–46, Feb. 2007.

[12] J. Borenstein and Y. Koren, "The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots," *IEEE Trans. Robotics and Automation*, vol. 7, no. 3, pp. 278–288, Jun. 1991.

[13] Intel RealSense, "Intel RealSense Depth Camera D435i," [Online]. Available: https://www.intelrealsense.com/depth-camera-d435i/. [Accessed: Sep. 25, 2025]