# Development of Autonomous Mobile Robot for Robotics Dojo 2025 Competition

*Victoria Rotich, Maryanne Mutwa, Ruth Olumo, Susan Kimani, Caleb Wambua, Fiona Opiyo, Joshua Njau*

*(The Obsidian Order)*

**Abstract - This paper presents the design and implementation process of the Obsidian Order team for a mobile robot to be used in the Robotics Dojo 2025 Competition. It contains the chief considerations used in the design process and the successes, challenges and solutions that were present along the development of the mobile robot.**

**Key words - robotics, ROS2, Raspberry Pi, map, detection, autonomous**

## I. Introduction

Robotics Dojo Competition is an annual competition that challenges students to build fully autonomous mobile robots capable of navigation through a game field and execution of given tasks within the game field.

The 2025 competition presented challenges such as climbing and descending ramps, moving through different terrain which included grass, gravel, and sawdust, a dynamic barrier, real-time detection of plant disease using a camera, and loading and offloading a cube within the gamefield.

This paper describes the process taken by The Obsidian Order in developing a mobile robot to solve the challenges presented by the Robotics Dojo Competition 2025.

## II. Design Strategy

### *A. Core Approach*

The design approach used to solve the competition was to build a mobile robot that was robust enough to ease through the challenge while still maintaining simplicity in order to avoid hitches that arise from overcomplication.

CAD tools specifically OnShape, Inventor and Fusion were used for rapid visualisation, and prototyping was done through 3D printing in order to come up with the optimal design.

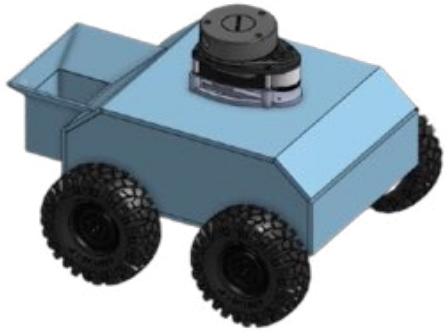

Figure 1: CAD design of the robot

### *B. Game Field Considerations*

The different parts of the game field were thoroughly considered while coming up with the design, and informed the type of components that were used.

This includes:

- Terrain. In light of the terrain, large wheels were selected to be able to coast through them. A four-wheel drive system was also favoured as sending power to all four wheels would reduce the chances of getting stuck within the terrain.
- Ascending and descending ramps. The presence of the ramps magnified the importance of having a balanced weight distribution. This was so as to keep the resultant weight component of the robot always within the base of the robot, minimising the amount of effort needed to climb the ascending ramp and maintaining

control on the speed of going down the descending ramp.
- Cube dimensions. The dimensions of the cube that was to be loaded onto the robot greatly dictated the size of the boot mechanism that was designed.

### *C. Cost Considerations*

The cost of components was also a key consideration in coming up with the design. A balance had to be struck between high-quality but pricey components and cheap but low-quality components in order to both stay within the budget and not compromise on the reliability of the mobile robot.

### *D. Time Constraints*

Due to the time allocated for the development of the robot, priority was given to rigorous and iterative testing as opposed to addition of functionality to the robot.

## III. Vehicle Design

The mobile robot composed of the following components:

- Raspberry Pi
- Raspberry Pi camera
- LiDAR
- Power System
- Motors and Wheels
- Motor Driver
- Arduino Mega
- Joystick
- Frame
- Boot
- Software

### *A. Raspberry Pi*

The Raspberry Pi used is the Raspberry Pi 4 Model B with 4GB of RAM. The Raspberry Pi has 40 General Purpose Input/Output pins (GPIO pins), 2 USB 2.0 ports, 2 USB 3.0 USB ports, 1 ethernet port, and a microSD card slot.

It is booted by a 32GB memory card loaded with Ubuntu Server 22.04.5 LTS and Robot Operating System 2 (ROS2), which is what is used to control and run the robot.


*Figure 2: Raspberry Pi 4 Model B*

It is powered by a 5V output.

The Raspberry Pi is the main brain of the robot.

### *B. Raspberry Pi Camera*

The Raspberry Pi Camera Module V2 is used. It has an 8-megapixel sensor and can support up to 1080p video mode.

It attaches via a ribbon to the CSI port of the Raspberry Pi.

It is used for detection and classification of plant disease.

*Figure 3: Pi Camera Module 2*

### C. LiDAR

The LiDAR used in this robot is the RPLiDAR A1. It has a range of 0.15 – 12m, a sample rate of 2000Hz and a scan rate of 5.5Hz.

It is connected via USB to the Raspberry Pi.

The LiDAR serves to scan the game field and, in conjunction with the encoders of the motors, create a map through a process called Simultaneous Localisation and Mapping (SLAM).

*Figure 4: RPLidar A1*

### D. Power System

A 12V Li-Po battery and a power bank are used to supply power to the robot.

The Li-Po battery sends power to the motors and motor driver. It sends the full 12V to the motors, and is also connected to a buck converter that drops the voltage down to 5V, which is then sent to the motor driver.


*Figure 5: Lipo Battery*

The power bank powers the Raspberry Pi, which consequently powers the LiDAR, Raspberry Pi camera and the Arduino Mega.

*Figure 6: Power Bank*

### E. Motors

2 JGB37-520 12V 110RPM motors are used. They have a maximum torque of 10KG.CM and a power rating of 7 – 15W. They are connected to an encoder.

The motors receive directional and velocity commands from the motor driver and execute them. Data concerning the revolution of the motors is collected by the encoder and sent to the motor driver.

Wheels used were the off-road 85 by 38mm type, chosen as they would be the wheels most capable of navigating through the terrain.

*Figure 7 DC Motor with Encoder*

### *F. Motor Driver*

The L298N Dual H-Bridge Motor Driver is used.

It is powered by a 5V input. It also receives 12V which it sends to the motors along with enable signals.

The L298N Dual H-Bridge Motor Driver is used to control the speed and direction of the motors, and receives data from the motor encoders.

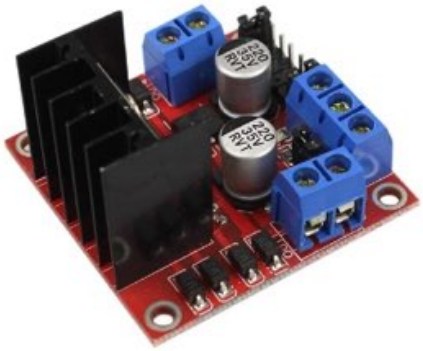

*Figure 8: Motor Driver*

### *G. Belt System*

A belt system is set up to transfer the torque of the 2 motors to the 4 wheels, effectively making it a four-wheel drive system.

### *H. Arduino Mega*

An Arduino Mega 2560 is used. It has 54 digital input/output pins, 16 analogue input pins, and a USB connection. It is connected to the Raspberry Pi through a USB port.

The Arduino Mega is an intermediary between the Raspberry Pi and the motor driver. It receives commands from the former and sends them to the motor driver, and receives encoder data from the motor driver and sends it to the Raspberry Pi.

*Figure 9: Arduino Mega*

### *I. Joystick*

A joystick is used to control the movement of the robot while it is performing SLAM.

The joystick is connected to a host computer through Bluetooth.

### *J. Frame*

The frame of the robot was fabricated through 3D printing.

*Figure 10: Mobile Robot*

### *K. Boot (Unloading) Mechanism*

The unloading boot is a detachable, funnel-shaped receptac**l**e mounted at the rear of the robot chassis.

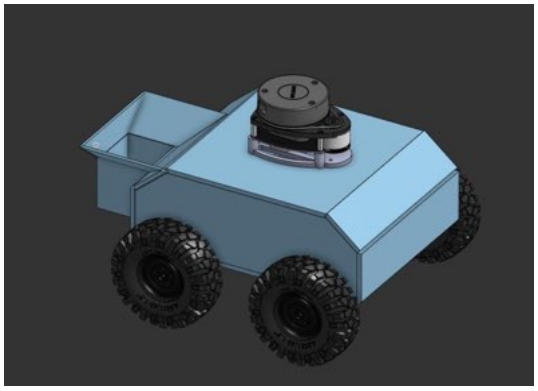

*Figure 11: CAD design with boot at the rear of the robot chassis*

*TABLE 1: BOOT DIMENSIONS*

| Feature | Dimension |
|---|---|
| Top rectangular opening | 15cm by 11.8cm |
| Internal drop chute | 5.8cm by 5.8cm by 5cm |
| Cube size (competition spec) | 3cm by 3cm by 3cm |

- The wide top lip ensures that a cube released from the loading station is captured even if the robot is a few centimetres off-center.
- The tapered funnel guides the cube toward a rectangular trapdoo**r** at the base.

The trapdoor is a 3D printed 3mm thick sheet and hinged on a servo bracket.

The boot mechanism is powered and controlled entirely by the Raspberry Pi. The MG996R servo motor is wired directly to the Raspberry Pi, taking 5 volts and ground from the board while its signal line connects to GPIO 18, which can output the precise PWM pulses the servo needs. The servo is mounted beneath the detachable funnel-shaped boot at the back of the robot. A flat rectangular trapdoor, sized to hold the 3 cm cube, is fixed to the servo horn so that rotating the servo swings the door open and closed.

Control is handled by a ROS 2 node that uses the pigpio library to generate the PWM signals for the servo. When the node starts it positions the servo at about 100 degrees so the trapdoor remains closed. The same node subscribes to the */joy* topic, which carries button data from the paired PS3 controller.


*Figure 12: PS3 (joystick) controller*

Each time the square button on the joystick is pressed, the node receives the message and commands the servo to rotate to an open position, holds it there briefly, and then returns it to the original 100-degree angle to re-seal the boot. In manual testing this allows a simple press of the square button to release the cube.

For the autonomous round, the same node is launched alongside the navigation and perception nodes. Instead of a human pressing the button, the higher-level mission controller sends the equivalent trigger at the unloading station after color detection and navigation have confirmed the correct drop-off point. The servo then follows the same open–pause–close motion, ensuring the cube is reliably released without any manual input while drawing all its power and control signals directly from the Raspberry Pi.

### *L. Software*

The robot's software was built on ROS2.

Launch files were built for the LiDAR, Raspberry Pi Camera, SLAM, joystick, teleoperation, robot state publisher etc.

Control files were built for ROS2 control of the motor, the joystick, mapping, simulation etc.

A simplified Unified Robot Description Format (URDF) file for the robot was also created for simulation and use in mapping and autonomous navigation.

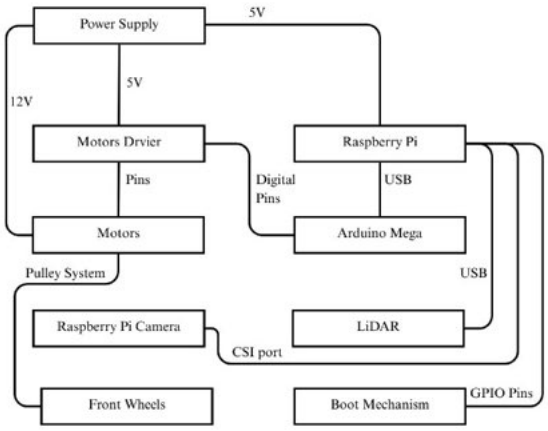


Figure 13: Hardware Structure of Mobile Robot

## IV. Experimental Results

### 1. *Unit Tests*

Unit tests were carried out for the following functionalities:

- Movement of the robot
- Generation of scan by the LiDAR
- Opening and closing of the boot mechanism
- Colour detection by the Raspberry Pi Camera.
- Plant disease detection by the Raspberry Pi Camera.

#### a. *Movement of Robot*

After assembly, the robot's linear motion was tested. This was through creating a node that would send */cmdvel* instructions to the Arduino Mega via serial. The instruction contained an RPM and directional components to it. The Arduino would then interpret the command and send the appropriate signal to the motors.

On testing linear movement the robot kept turning left, meaning that the right motor, despite getting the same pulse width modulation (PWM) signal as the left motor, was rotating faster. It was, therefore, necessary to create a proportional integral differential (PID) controller.

The PID controller was implemented on the Arduino Code. Error was calculated by subtracting the target number of ticks per frame from the actual value of ticks per frame. For the robot to move in a straight line, the following were the constants that were empirically arrived on: kp = 50, ki = 5, kd = 1.

Initial testing was done using the ROS2 teleoperation package that ran and controlled on the host computer. The joy stick was eventually set up to give the */cmdvel* instructions and thus control the robot.

#### b. *Generation of scan by LiDAR*

The LiDAR was set up by connecting it to the Raspberry Pi through a micro USB cable. A repository by SLAMTec, the company that manufactures the RPLiDAR A1, was cloned into the Raspberry Pi. The LiDAR was then launched using its respective launch file, and generated a 2D scan of the environs, which was visualised on RViz.

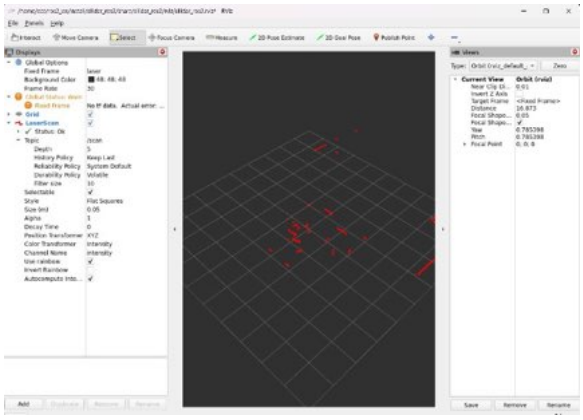

*Figure 14: LiDAR laser scans visualized in Rviz*

Occasionally, the LiDAR would fail to launch. On troubleshooting, it was discovered that the LiDAR was not receiving enough power from the Raspberry Pi. This arose due to the fact that the initial setup had the same Li-Po battery powering every single component of the robot. The issue was initially solved by slightly increasing the voltage sent to the Raspberry Pi from 5V to 5.7V. However, it was eventually agreed upon to use a powerbank instead to

power the Raspberry Pi, which supplied sufficient power for the Raspberry Pi and all connected devices.

*c. Opening and closing of boot mechanism*

The boot mechanism was unit-tested by sending test */joy* messages and verifying that the servo opened to release a cube and then returned to the closed position

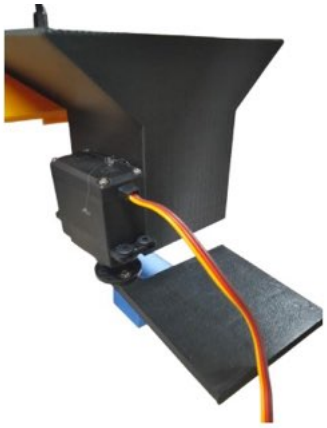

*Figure 15: Boot mechanism unloading the cube*

*d. Colour detection by Raspberry Pi Camera*

The colour detection system serves the critical function of identifying the colour of cubes placed on the robot's loading mechanism by analyzing the LED illumination displayed in the loading area. This capability is essential for the robot's autonomous decision-making process, as it determines whether to deposit the cube in the blue or orange zone during the final unloading phase.

The system operates using a Raspberry Pi Camera Module v2 mounted on the robot, which captures real-time video feed of the loading area where cubes are dispensed with coloured LED indicators.

The colour detection node operates as a ROS 2 package that continuously monitors the camera feed and employs computer vision algorithms to detect and classify colours.

When a cube is loaded onto the robot, the system analyzes the predominant colour visible in the camera's field of view using OpenCV (cv2) for image processing. The algorithm converts the captured BGR images to HSV colour space, which provides better colour segmentation consistency under varying lighting conditions compared to standard RGB.

The system creates binary masks for specific colour ranges—primarily focusing on blue and orange detection—by applying threshold values that define the lower and upper bounds of each target colour in the HSV spectrum.

The detection process involves calculating the percentage of pixels falling within the predefined colour ranges for both blue and orange. The system compares the relative proportions of detected blue versus orange pixels against a confidence threshold to determine the dominant colour. This approach ensures reliable classification even when environmental lighting conditions fluctuate during competition. The node publishes two main output topics: one indicating the detected colour as a string value ("blue", "orange", or "none") and another providing a confidence score between 0.0 and 1.0 representing the certainty of the classification.


*Figure 16: Detecting color blue in an image*

This colour detection node is functionally dependent on several other system components. It requires the camera driver node to provide a stable image feed through the */camera/image_raw* topic. The system interfaces with the task coordinator node, which triggers colour detection at the appropriate moment using a service during the loading sequence and utilizes the colour classification results to make navigation decisions. The node interacts with the navigation system to determine the correct deposit zone based on the identified cube colour.

*e. Plant disease detection by Raspberry Pi Camera*

The potato disease detection subsystem was designed as a modular vision pipeline built around the Raspberry Pi camera and implemented in ROS 2. The main objective was to acquire a continuous image stream from the camera, process the frames through machine learning, and provide disease classification results in real time or on demand. The architecture emphasizes modularity, low computational overhead, and flexibility, allowing smooth integration into the broader robotics framework for competition tasks.

Initially, a custom model was trained using TensorFlow, focusing on distinguishing between healthy and unhealthy potato leaves. However, as development progressed, a more complete PyTorch-based model was provided, trained to classify three classes: *Early Blight*, *Late Blight*, and *Healthy*. To align with this improved model, it was decided upon to integrate it into our pipeline instead of continuing with the TensorFlow approach. This ensured consistency, allowed more accurate classification, and streamlined the system design.

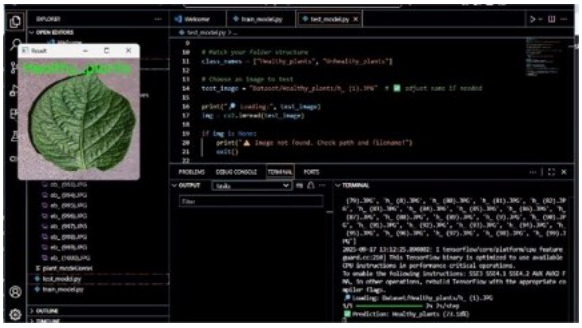

*Figure 17: Potato Plant disease detection model detecting a healthy plant*

The system consists of four key ROS 2 components:

- Headless Camera Node – acquires and publishes images, provides image saving service.
- Potato Disease Detection Node – subscribes to camera feed, performs inference, publishes results.
- Inference Engine – wraps a fine-tuned ResNet-18 model for disease classification.
- Service Node – provides on-demand, averaged predictions for stable outputs.

Detailed Description of the Nodes

Headless Camera Node

This node is responsible for live image acquisition from the Raspberry Pi camera. Captured frames are converted into ROS‑compatible *sensor_msgs/Image* messages and published on the */camera/image_raw* topic. It runs in a lightweight, headless mode without GUI dependencies to minimize resource usage on the embedded platform.

Additionally, the node offers a ROS service (/save_image) that allows an external request to capture and save the current frame with a timestamp. This function is useful both for dataset generation and for verifying the camera pipeline during testing.

Potato Disease Detection Node

The detection node, implemented within the package *rdj2025_potato_disease_detection*, subscribes to the incoming image topic. Frames are converted from ROS → OpenCV → PIL format to prepare them for inference. The node then passes the images to the inference engine and publishes the classification result as a std_msgs/String message on the /inference_result topic.

This node therefore serves as the bridge between raw visual data and decision-level output, making results available to other modules in the system.

Inference Engine

The inference engine is defined in a dedicated module (*inference_engine.py*). It encapsulates the trained machine learning model and its preprocessing pipeline. Specifically, a ResNet-18 architecture is employed, with its final fully connected layer modified to handle three target classes. Pretrained weights were fine-tuned for potato leaf disease classification, and the model is loaded at runtime from stored .pth weights.

Input images are preprocessed using resizing, cropping, and normalization before inference. The engine supports both CPU and GPU execution, automatically selecting GPU if available. The output

is the predicted disease label (Early Blight, Late Blight, or Healthy).

Service Node

In order to balance continuous live feed availability with efficient inference, an additional service node was introduced. This node subscribes to the headless camera stream but only processes frames on request. When called, it captures three consecutive frames, performs inference on each, and averages the results to output a single classification response.

This design reduces the redundancy of running inference on every frame (~0.1 s intervals), lowers computational load on the Raspberry Pi, and improves robustness by mitigating noise from motion or lighting variations.

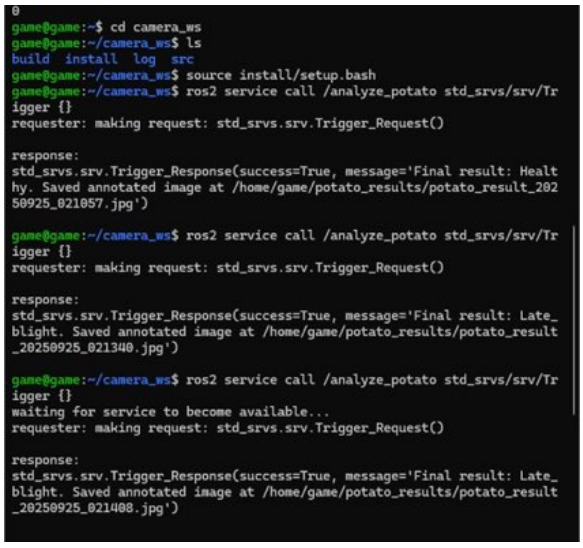


*Figure 18: Screenshot of a terminal during a service call.*

Several challenges arose during implementation. First, running inference continuously on every frame led to high system load and repetitive outputs. The service-based approach was introduced to address this limitation, but additional optimization of model deployment (e.g., quantization or pruning) may further improve efficiency.

Another difficulty was with visualization: although inference results are accessible as ROS messages, overlaying predictions directly on images in real time proved problematic due to GUI compatibility issues on the headless Raspberry Pi setup. Future work will explore lightweight visualization tools or web-based dashboards to display annotated outputs.

Model performance varied across test conditions. While the current ResNet-18 achieves reasonable accuracy, retraining or fine-tuning with additional potato leaf datasets may be necessary to improve reliability under field conditions.

### *2. Integration Tests*

Integration tests were carried out for the following functionalities:

- Generation of map
- Autonomous navigation of the robot

#### *a. Generation of map*

The prerequisites to generating a good map are:

- A working URDF with correct transforms.
- LiDAR scans publishing to */scan* topic
- Wheel encoder odometry publishing to */odom* topic.
- SLAM launch file.
- Teleoperation for moving the robot around.

After getting the requirements a map was generated by moving the robot around with the joystick in space. SLAM worked properly, but there were some small distortions in the map generated due difficulties that arose from the use of a four-wheel drive system.

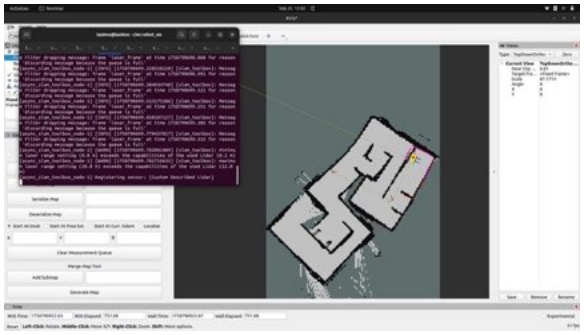

*Figure 19: Map generated from the gamefield*

Parameters were adjusted to solve the issue by setting up the robot to chiefly depend on laser scan from the LiDAR in order to rectify itself when the odometry of the URDF were off. This caused a significant improvement in the map generated.

*b. Autonomous navigation of the robot*

For autonomous navigation, the main prerequisites were:

- A previously generated and saved map.
- Adaptive Monte Carlo Localisation (AMCL) for localization on the map.
- The Nav2 stack with planner, controller, and recovery behaviors.
- Correct transforms in the URDF.

Once the map was ready, we placed the robot in the mapped space, launched AMCL for localization, and started Nav2. Using RViz, we set different goal poses for the robot. The robot was able to localize itself and move towards the goals on its own.

There were a few deviations in the path, mostly due to the four-wheel drive setup and small differences in the motors, but overall the navigation worked as expected. The robot combined localization, planning, and motion control into a working pipeline that allowed it to move without any manual control.

## V. Acknowledgement

The Obsidian Order team would wish to acknowledge the invaluable help and support offered by Mr. Lenny Ng'ang'a and Mr. Billy Isaac throughout the design, fabrication and testing of the mobile robot.

## VI. References

[1] Raspberry Pi Foundation, "Raspberry Pi Camera Module 2," Raspberry Pi Documentation, 2023. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html

[2] OpenCV Foundation. *OpenCV Documentation*. [Online]. Available: https://docs.opencv.org/4.x/

[3] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[4] Open Robotics. *ROS 2 Humble Hawksbill Documentation*. [Online]. Available: https://docs.ros.org/en/humble/

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, May 2022. DOI: 10.1126/scirobotics.abm6074

[6] Raspberry Pi Foundation. *Raspberry Pi OS Documentation*. [Online]. Available: https://www.raspberrypi.com/documentation/computers/os.html

[7] Slamtec. 2019. RPLIDAR 360 Degree Laser Range Scanner Interface Protocol and Application Notes. Shanghain Slamtec. Co.

[8] A. N. Meena and M. Pradhan, "Potato leaf disease detection using deep learning," in *2020 International Conference on Communication and Signal Processing (ICCSP)*, Chennai, India, 2020, pp. 1229–1233.