Autonomous Agricultural Robot for Mapping, Crop Disease Detection and Object Recognition.

Sandra Keya, Stephen Oduor,Collins Chemweno, Faith Cherotich, Sylvia Waweru

***Abstract*—This paper describes the design and implementation of a miniature robot intended for agricultural purposes. The robot employs SLAM using a 2D lidar sensor sensor to map unknown environments, while estimating its own position and orientation within that map. It is also optimized to perform crop disease detection and to identify potato blight and other crop diseases in addition to navigation of various kinds of terrains.**

I. INTRODUCTION

There has been an increase in reliance on automation to improve efficiency within agriculture to reduce labor costs as well as address crop health challenges.There is thus a growing interest in robotics to not only perform the said tasks but also to perceive and adapt to their environment in real time, performing mapping, navigation and disease detection. The research and development of mobile autonomous robots has also substantially expanded within the last decade and continues to increase due to the rapid advancement in theory and electronic technology. Different methods of how a robot can navigate through an unknown structured environment,that is to estimate its current position and orientation, have been developed. Robot platforms for SLAM include sensors/systems such as motor encoders, optical vision, miniature radars and satellite positioning.Simultaneous Localization and Mapping (SLAM) enables robots to build a map of an unknown environment while estimating their own location within it. Differential-drive robots equipped with sensors such as LiDAR, cameras, wheel encoders, infrared or laser rangefinders can navigate in unstructured or semi-structured agricultural settings. Several prior works use vision or sensor fusion for leaf disease detection, but often rely on stationary setups, GPS, or require well-controlled lighting.

In contrast, the advent of smart robotics combined with Raspberry Pi and LiDARs provides an intelligent and efficient alternative. In this system, the Raspberry Pi, a small yet powerful single-board computer, serves as the central processing unit. It works with a camera and image processing algorithms to capture and analyze images of plant leaves to detect signs of diseases such as blights, rusts, and fungal infections. The system can be configured with machine learning models or classical image processing techniques to identify and classify diseases based on visual symptoms. Once a disease is detected, the system triggers a real-time alert, and the Raspberry Pi logs the event and can display or transmit the information via Wifi . Simultaneous Localization and Mapping enables the robots to build a map of the unknown field which is especially valuable in agriculture In agricultural robotics, SLAM supports several key navigation tasks such as: Autonomous field traversal, allowing the robot to move through uneven terrain, avoid obstacles (plants, uneven soil, debris), and follow crop rows, repeated coverage, where the robot returns to specific zones (for disease monitoring, spraying, harvesting) with good positional consistency as well as mapping for decision-making, where maps show where plants are missing, areas of disease, or where soil is rough so useful actions can be taken.Recent studies show that fusing LiDAR data with wheel odometry (and optionally inertial sensors) improves navigation reliability. For instance, research into 2D LiDAR SLAM without needing GNSS/IMU shows improved robustness in arboreal or forest-like field situations where GNSS is blocked.

II. PROBLEM STATEMENT

Despite significant progress in agricultural automation, farmers still face challenges in efficiently monitoring crop health and navigating complex, unstructured farm environments. Traditional disease detection methods often rely on manual inspection, stationary imaging setups, or require controlled conditions, making them impractical for real-world field deployment. Similarly, navigation of mobile agricultural robots is limited when dependent solely on GPS, which suffers from poor reliability under canopy cover or in semi-structured environments.

There is a need for a low-cost, mobile robotic system that integrates real-time plant disease detection with robust navigation using SLAM. Such a system should leverage affordable hardware (e.g., Raspberry Pi, cameras, LiDAR) and intelligent algorithms to autonomously traverse fields, perceive crop conditions under natural lighting, and adapt to uneven terrain. This would reduce reliance on labor-intensive monitoring, improve accuracy in detecting crop diseases, and enable more efficient and data-driven agricultural management.

III. PAPER CONTENTS

A. Design Strategy

The competition required us to map the game field, detect crop diseases, identify cube colors, and reliably unload a payload, while maintaining robustness throughout navigation. To achieve this, we began with prototyping the core subsystems: differential drive with encoder feedback controlled via Arduino, SLAM mapping with LiDAR, disease detection through the camera, IR-based payload detection, and the unloading mechanism. After validating each subsystem in controlled settings, we gradually integrated them and tested them in more realistic field environments. Between tests we refined algorithms, and adjusted mechanical hardware based on observed failures. Our strategy emphasizes reliability over adding extra features: only once core functions are stable do we tune for efficiency or performance improvements.

B. Vehicle Design

Based on our competition requirements, we decided a miniature robot design would be advantageous because smaller systems tend to have fewer failure points (less mechanical complexity, fewer parts to break). We chose a circular footprint initially: it is compact and, we believed, would allow the robot to navigate obstacles and turns in the game field more easily.

Our first mechanical architecture was: two powered (driven) wheels on the sides, and two castor wheels (free-spinning) at front and rear to stabilize the robot. This layout allowed differential drive motion (turning by differential wheel speeds) and stable support.

However, when we tested the robot attempting to climb a slight ramp (a feature in the field), we discovered a critical limitation: the unpowered castor wheel would often lose traction or slip, preventing the robot from climbing. Because the castor wheel could not drive, at some orientations its contact or load wasn't favorable, leading to wheel lift or slippage.

In response, we revised our design: the two powered wheels were moved to the front, and one castor wheel placed at the back. This redesign ensures that when climbing a ramp, the powered wheels carry the load and maintain traction, while the castor wheel acts purely as a support without having to pull. This layout improved ramp climbing ability while still keeping the benefits of a compact design.

1. Chassis & Structure

We 3D-printed the chassis using polylactic acid (PLA) filament due to its low cost, ease of fabrication, and suitability for rapid iteration. The body's external dimensions are 29 cm (length) × 21 cm (width) × 24 cm (height). The chassis houses all key components in a compact layout, with the center of mass positioned near the geometric center to promote balance and stability.

2. Hardware Components

Wheels & Castor

Drive wheels: We used two off-road wheels (rubber tread) sized at 85mm diameter, 31mm width, chosen to negotiate terrain such as sawdust and grass. The tread and soft compound provide grip on loose surfaces.

Fig. 1. 85 mm rubber wheel

Castor wheel: Rather than using a standard castor, which may struggle on uneven terrain, we designed a custom castor wheel holder and selected a rubber wheel for the castor. This design improves rolling and stability on rough terrain and avoids binding in soft ground.

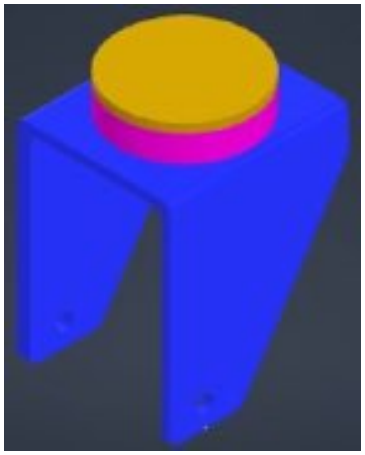


Fig. 2. Castor wheel CAD design

Motors & Drivers

We selected the JGB37-520 motor series. These motors run on 12 V and deliver a torque of ~1.5 kg·cm in some variants. They come with dual-phase Hall encoders, enabling closed-loop control via Arduino.

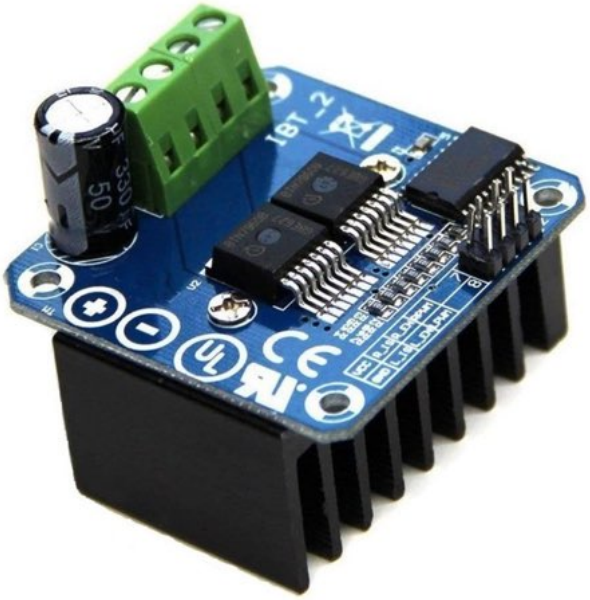


Fig. 3. Double BTS7960 Motor Driver Module

Fig. 4. JGB37-520 motor

The Double BTS7960 43A H-Bridge High-Power Stepper Motor Driver Module is; a fully integrated high current H bridge for motor drive applications using the BTS7960 high current half bridge. The BTS7960 is part of the NovalithICTM family. This contains one p-channel high side MOSFET and one n-channel low side MOSFET; with an integrated driver IC in one package. Due to the p-channel high side switch; the need for a charge pump is eliminated thus minimizing EMI.

Interfacing to a microcontroller is made easy by the integrated driver IC which features logic level inputs; diagnosis with current sense, slew rate adjustment; dead time generation and protection against over temperature, overvoltage, under voltage; overcurrent, and short circuit.

3. Control Architecture

Low-level / Medium-level controller

Arduino Mega 2560

The Arduino Mega 2560 (ATmega2560) was selected for its abundant I/O (54 digital pins, 15 PWM outputs, 16 analog inputs, 4 hardware UARTs, etc.). It handles the motor driver control (PWM, direction), encoder feedback, servo control, and IR sensor input.

Fig. 5. Arduino Mega with USB cable

High-level controller

Raspberry Pi 4 Model B

The Raspberry Pi 4 is used for high-level tasks: camera image capture, color detection, SLAM, path planning, and issuing high-level commands to the Arduino. It features a quad-core 1.8 GHz CPU, up to 8 GB RAM, dual micro-HDMI, CSI camera interface, and full Linux stack

Fig. 6. Raspberry Pi 4

4. Sensors & Perception

Raspberry Pi Camera v2

This camera features an 8 MP Sony IMX219 sensor, connected via a 15 cm ribbon cable to the Pi's CSI port. It is used for color detection, disease identification, and object (cube) color classification.

Fig 7. Raspberry Pi Camera Model 2

LiDAR

A 2D rotating LiDAR is mounted at a suitable height and orientation (on a dedicated holder) to scan the environment and support SLAM and obstacle detection.

Fig 8. LiDAR with connecting wires

5. Component Placement

On the base platform we mounted:

- Arduino Mega 2560 – Handles low-level sensor control, motor commands, and communicates with the Raspberry Pi.
- Battery pack – Three Li-ion cells in series (nominal 12.6 V).
- Payload / unloading box assembly – Includes the micro-servo for actuating the unloading mechanism.

Below the base, adjacent to each drive wheel, we placed:

- JGB37-520 12 V DC geared / encoder motor

- BTS7960 high-power motor driver for each motor
- An XL4015 buck converter to step down voltage for sensors / logic circuits

Above the base, we added a secondary platform supporting:

- LiDAR in a dedicated holder, at a height and orientation optimized for scanning the environment
- Raspberry Pi 4, the main processing unit
- Pi Camera v2 in a holder with appropriate field of view for crop disease / color detection
- A power bank beneath the Pi to provide stable 5 V power to the Pi and camera module

Additionally, we mounted an infrared (IR) object detection sensor on the loading platform to sense when a payload cube is placed. Its digital output signals the control system to proceed with the unloading routine.

6. Power & Voltage Regulation

The main battery delivers ~12.6 V fully charged; this rail feeds the motor drivers directly. For lower-voltage components (sensors, logic), the XL4015 buck converter steps down the voltage. This ensures stable and safe supply levels for sensitive electronics. The wiring is done to minimize voltage drop and uses proper gauge wires.

7. Mounting & Mechanical Interfaces

Each motor is mounted securely to the chassis using brackets and couplers to minimize misalignment and vibration. The LiDAR, camera, and sensors are fixed using holders that maintain orientation and line-of-sight. The unloading servo is attached to the payload box and is mechanically robust to avoid binding.

8. Mass Distribution & Stability

We measured the weight of all components (PLA body, motors, electronics, battery, sensors). We balanced the layout such that no side is overloaded, and the chassis height is kept moderate to reduce torque about the ground (less tipping). This configuration helps maintain traction, especially when ascending ramps.

9. Safety, Thermal, and Robustness

The motor driver and buck converter are mounted with thermal dissipation in mind.
We included simple protective measures (e.g., fuses) to prevent damage from overcurrent or short circuits.
Structural rigidity is verified so that under vibration or slight impacts, nothing loosens.
We sealed or protected open components from dust and debris to maintain performance in field conditions.

10. Design Tradeoffs & Alternatives Considered

We considered using acrylic material for higher strength, but decided on PLA to allow rapid prototyping and weight savings. We also considered more powerful motors or higher current drivers, but these would increase cost, weight, and power consumption. In early tests, some heavier motor-driver combinations led to overheating; thus we settled on the current configuration which balances power and reliability.

Schematic Diagram

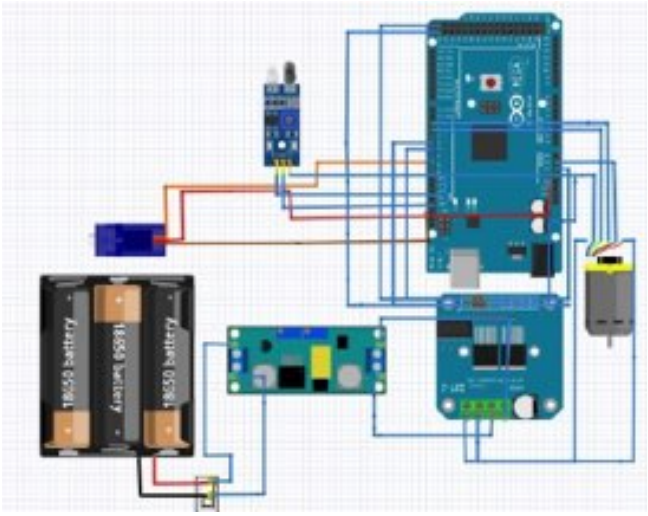


Fig 9. Schematic Diagram showing connection of components

Above is a schematic diagram demonstrating the low -level control via the Arduino Mega and all the sensors it is controlling.
It demonstrates connection of one of the motors to the Arduino Mega.

C. The software

This chapter presents the design and implementation of the software architecture for an autonomous mobile robot system developed on ROS 2 Humble Hawksbill. The system adopts a modular architecture inspired by the workflow popularized by Articulated Robotics, emphasizing simulation-first validation and seamless transition to hardware deployment. The robot employs a 2D LiDAR sensor for environmental perception, enabling incremental map construction and real-time obstacle detection. Navigation is managed by the Nav2 stack, which provides global path planning, local trajectory following, and dynamic obstacle avoidance.
System validation and parameter tuning were initially conducted in Gazebo simulation, where a URDF model was defined, controller interfaces were configured, and ros2_control parameters were verified prior to hardware integration. By following a staged methodology simulation, parameter optimization, and hardware deployment the development process minimizes integration risks, ensures consistency between virtual and physical environments, and allows rigorous testing of navigation modules.

1. Software Architecture

The software architecture is organized around a ROS 2 node graph that separates sensing, planning, control, and actuation, with a bridging layer connecting ROS 2 to the Arduino microcontroller. YAML configuration files load the diff_drive_controller and joint_state_broadcaster, while the

gazebo_ros2_control plugin instantiates the controller manager and links simulated joints to ROS 2 controllers. LiDAR data is exposed to ROS 2 via Gazebo plugins, publishing LaserScan messages that feed directly into the navigation stack. The Nav2 framework subscribes to sensor topics and publishes velocity commands, which are relayed to the Arduino bridge. The bridge transmits these commands to the motor hardware and publishes odometry feedback back into ROS 2. The system is built on ROS 2 Humble Hawksbill , chosen for its long-term support, compatibility with Ubuntu 22.04, and widespread adoption in both research and industry. ROS 2 provides modern features such as improved middleware, enhanced quality-of-service policies, and distributed system support, making it well suited for complex autonomous robotic systems.

In the developed robot, the LiDAR node publishes /scan data, which is consumed by both the SLAM and Nav2 nodes. Nav2 generates velocity commands (/cmd_vel), which are transmitted to the motor drivers via the ros_arduino_bridge. The Arduino firmware handles low-level motor control and sensor feedback, while the bridge publishes odometry back into ROS 2. During testing, teleoperation and the joint_state_publisher were employed to validate odometry and verify system behavior. This bridging strategy ensures that the same navigation and perception stack validated in simulation can be deployed to hardware with minimal modification.

2. *Software Implementation and Validation*

The robot was modeled using URDF/Xacro, which defined its geometry, joints, and sensors for both simulation and deployment. Python-based launch files were developed to configure Gazebo simulation, RViz visualization, and hardware execution. Controller parameters such as wheel separation and PID gains were stored in YAML files, while the ros_arduino_bridge was implemented in C++.
Validation followed a simulation-first approach. In Gazebo and RViz, the accuracy of the model, transforms, sensor data publishing, and navigation performance were verified using Nav2 for mapping, planning, and obstacle avoidance. Once validated in simulation, the same codebase was deployed to the physical robot, where trajectory tracking, odometry accuracy, and navigation reliability were tested. Minor discrepancies, primarily caused by serial latency and parameter mismatches, were iteratively refined until the system achieved stable performance.

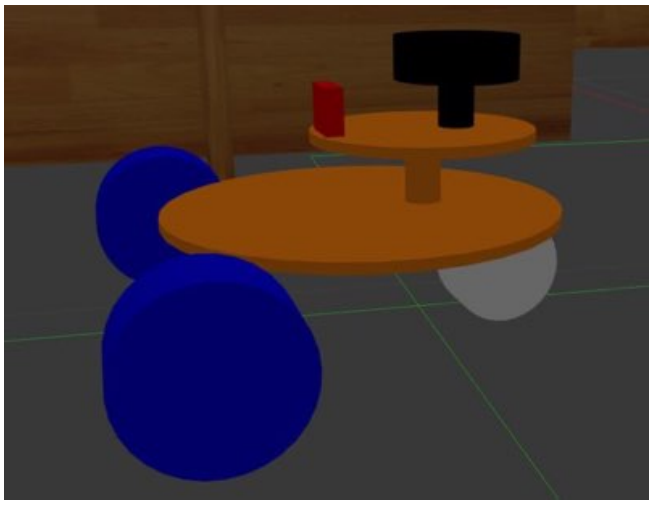


Fig. 10. Diagram showing the URDF spawned in gazebo

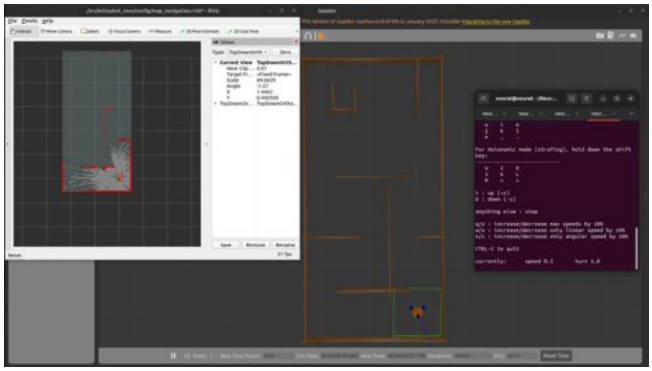


Fig. 11. Diagram showing the robot navigating a gamefield in gazebo

3. *Results*

The results demonstrated low communication latency between ROS 2 nodes, the Arduino bridge, and the firmware. Navigation commands were executed with minimal delay, enabling smooth motion even during frequent updates from the local planner. Several iterations of parameter tuning were required to achieve consistent navigation accuracy. Adjusting the robot radius parameter was critical to ensure that Nav2's planners respected the robot's footprint, while tuning the wheel separation parameter aligned simulated kinematics with real-world behavior.
After refinement, the robot achieved reliable path following, stable obstacle avoidance, and accurate odometry. In simulation, path tracking error remained within a small margin of truth, while hardware tests confirmed sufficient accuracy for SLAM and navigation in structured indoor environments.

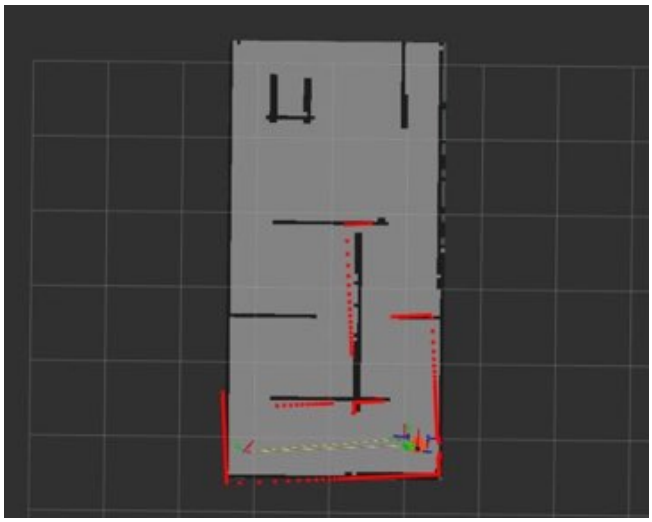


Fig. 12. Diagram showing the map generated during a practise run in gazebo

4. Challenges and Solutions

Several challenges were encountered during development. Early attempts to use WSL on Windows introduced high communication latency and frequent build errors, which were resolved by migrating to a native Ubuntu installation for full ROS 2 compatibility. Cross-compilation on the Raspberry Pi also posed difficulties, as many packages compiled successfully on x86 but failed on ARM64. This was addressed by excluding non-essential packages, allowing navigation and control components to compile successfully.

Another challenge involved integrating the ros_arduino_bridge with the URDF and navigation stack, where mismatches between simulation and hardware behavior arose from synchronization issues. These were resolved through iterative parameter tuning, particularly of wheel separation and robot radius, and by leveraging knowledge sharing with peers. Through these solutions, the software stack was stabilized across both simulation and hardware platforms.

5. Conclusions

The development of the robot's software architecture demonstrates the effectiveness of a simulation-first methodology combined with a modular ROS 2 framework. By validating the system in Gazebo prior to hardware deployment, integration risks were minimized, and discrepancies between simulated and real-world performance were addressed. The use of ROS 2 Humble, together with the Nav2 navigation stack and the ros_arduino_bridge, enabled a clean separation between high-level autonomy and low-level control, ensuring portability and reproducibility.

The results confirm that careful parameter tuning, particularly of robot radius and wheel separation, is essential for achieving accurate navigation and odometry.

D. Disease Detection Model using OpenCV

Once we had the navigation done, we proceeded to experiment with training a simple disease detection model using frames of plant leaves. The model was written using **python** and various libraries like **TensorFlow** and **NumPy** for the training. We used a simple data set of about 10 healthy plants and 10 diseased plants to train the model. On application the model was fairly accurate and was able to correctly classify a test image which was diseased.

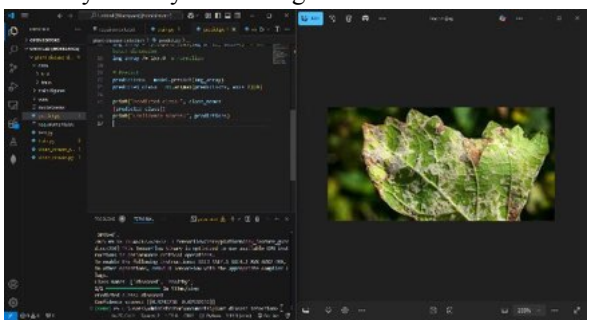


Fig. 13. Sample CV model to classify plant leaves as diseased or healthy

We later obtained a pre-trained model that classified the frames of the potato leaves into *healthy, early blight* and *late blight.* We tested the model and integrated it with the Pi camera feed, which is talked about in the sections that follow.

1. Camera Integration

For the competition we were required to use the Raspberry Pi camera to run computer vision models for detecting disease in potato plants. The model was trained to detect whether the potato plant was **healthy,** had **Early blight** or **Late blight** based on frames taken of the leaves of the plant. The camera was also to be used for color detection which was to be integrated with the picking up and dropping of the cube mechanism. In summary the camera was required to detect the presence of the cube on our carriage mechanism as well as its color and match it with the color code of the dropping area. If the colors matched the camera would publish a boolean true which would initiate the actuation of the dropping mechanism.

2. Camera Testing

Once we connected the camera, we SSH'd into the Pi and checked that the camera was detected by running the command; **ls /dev/video***. This command lists the available video devices, the first (index0) of which being the camera and the rest used to generate metadata for the photos and videos generated by the camera. We then run commands to take pictures which were saved on the Pi.

3. Video Stream

For the video stream we had two options;

- Using the built-in rQt GUI interface to see the video stream.
- Using a webpage to access the video stream and include functionalities such as taking snapshots and recording video frames.

Both options were feasible, however we decided to go with the second one, specifically using Flask to design a simple webpage that would run on a localhost port to access the video stream. This allowed us to access the stream from any device as well as to customize the page for functions we would need in the future.

4. Potato Disease Detection Model

Once the video stream was running, we linked the python video stream script with the ROS disease detection model by publishing streams to the **/image** node of the disease detection model. On clicking record stream, the stream starts publishing frames to the node which listens for frames that are then run through the model and the result streamed in logs on the raspberry pi ros workspace. The model detects whether the leaf is **healthy, has early blight** or **late blight.**

5. Color and Object Detection

We used a separate script for color detection of the frames. The script runs on Open CV and checks the pixels of the frame against a presaved set of colors using the **HSV** model. The following pictures show the model in action showing results for the colors; yellow, green and red.

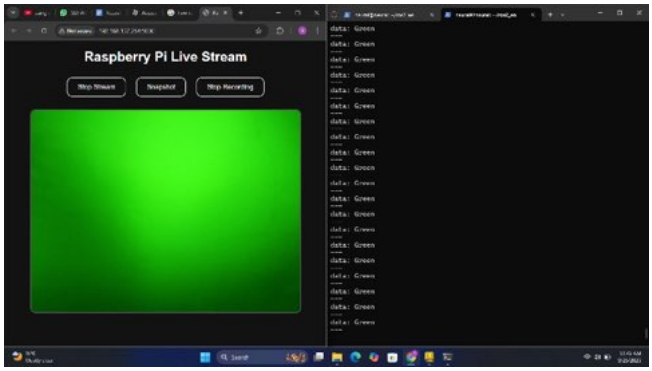


Fig. 14. Color detection model using the Raspberry Pi Camera

We were also able to run a simple cube detection model which detects the presence of cube by analyzing sharp edges that contrast with the background as well as running the previous color detection model on the detected cube. This model will be improved and applied to detect once the cube has been placed on our carriage mechanism and the color of the cube to be able to compare it with the color code of the dropping area. The following pictures show the model in action.

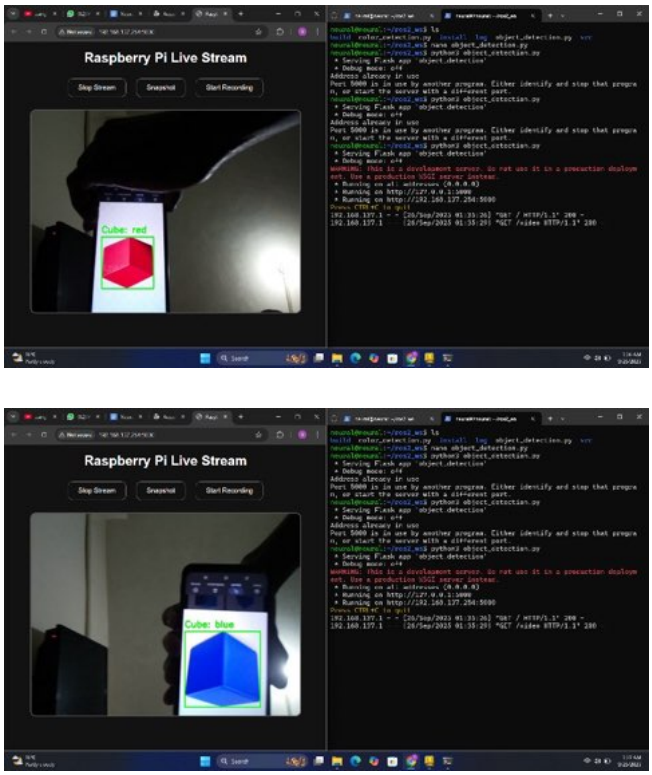


Fig. 15. Testing cube and color detection model using OpenCV

6. Results from Camera and Computer Vision Model Tests and Planned Integration

Potato Disease Detection Model

The potato disease detection model had positive results on testing with the model being fairly accurate in detecting healthy potato leaves. For leaves with early blight that had few spots, the accuracy of the model was a bit low but for leaves with late blight or leaves with pronounced spots that are still in early blight, the model was able to classify them accurately. Further training of the model with a greater data-set is planned to improve the accuracy of the model.

Color and Cube Detection Model

The color detection model showed high accuracy especially since the range of colors was limited to the primary colors and colors like *green, purple* etc which are easily detectable especially if converted to the HSV format.

The cube detection model was also fairly accurate with just a few instances where areas with noise in the frame were detected as cubes. We improved the model to ignore sections with small pixels that seem to be noise and focus on large sections having pixels that are similar in color. This greatly improved the accuracy of the model. In terms of detecting the color of the cube the model had exceptional accuracy.

Planned Integration

With the individual potato-disease detection model and the object and color detection model working well, the next plan is integrating the systems with the SLAM and the dropping mechanism using the servo-motor and IR sensor

E. Experimental setup

We have tested and tried various aspects of the robot including the design of the body, the material for the robot body, the architecture of the dropping mechanism and use of a custom castor wheel. These are discussed below;

1. The body

Initially our robot design was circular, having two motor driven wheels along the diameter of the bottom platform and two custom designed rubber castor wheels. The motion of the design was mostly smooth however on testing the robot on a ramp we realized the robot couldn't climb a ramp.

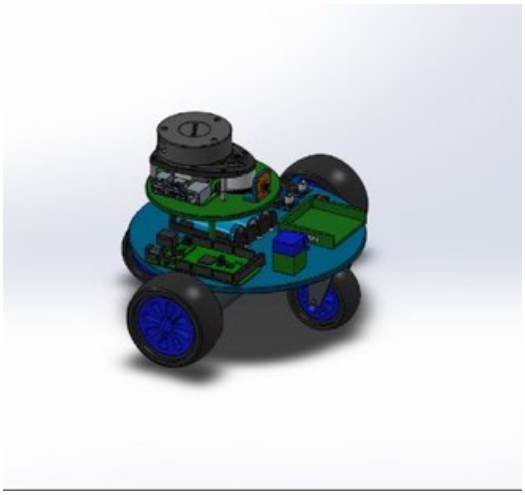
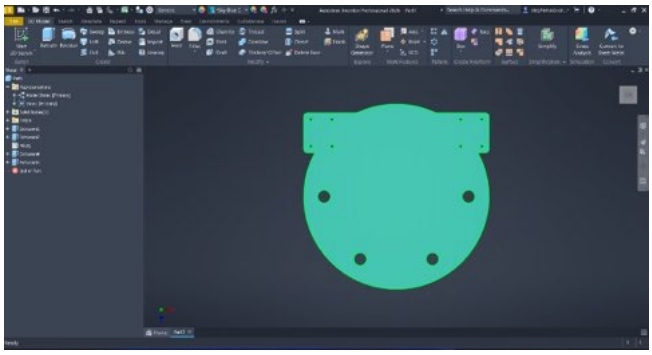


Fig. 16. First CAD design of robot

We decided to change our design from a circular one with the driven wheels along the diameter and pushed the driven wheels further behind. This enabled the wheels to keep in contact with the ground even while the robot was climbing the ramp.

The following pictures show the new design and the redesign of the base to enable the robot to easily climb the ramp. We also added more holes on the base platform to improve our cable management.



- *Fig. 17. Redesigned robot base with wheels placed further from the center*

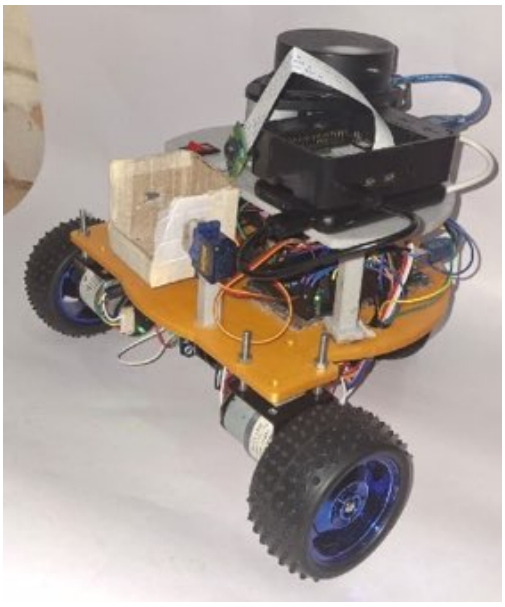


Fig. 18. New Robot Design

2. Material Selection for the Robot

For the material of the robot platforms we had two options;

1. Using 3D printed PLA (Polylactic Acid)
2. Using Acrylic cut using LASER cutting

We experimented with Acrylic platforms and found that they were relatively strong and had great structural integrity. A downside we noticed was the pillars were their stability. Since the acrylic was just 5mm thin the pillars were relatively unstable compared to 3D printed pillars. We also realized 3D printed parts adhered better with each other when using hot glue as compared with acrylic. These considerations made us choose 3D printing instead of using acrylic.

3. Custom Castor Wheel Design

As part of the agricultural application of the robot project, the game plan of the robot has different terrains including

sawdust and grass. These different terrains require rubber tyres since they have enough friction to be able to navigate the terrain. This consideration led us to designing a custom castor wheel design that would use a rubber wheel. The castor wheel design was 3D printed having three parts, the castor wheel pin, the castor wheel holder and the wheel holder. A 6001 bearing was used to enable the smooth circular motion and the 3D printed parts were designed around it.

.

Fig. 19. 6001 ball bearing

The following pictures show the castor wheel design, its assembly and the 3D printing process.

Fig. 20. Castor Wheel Design

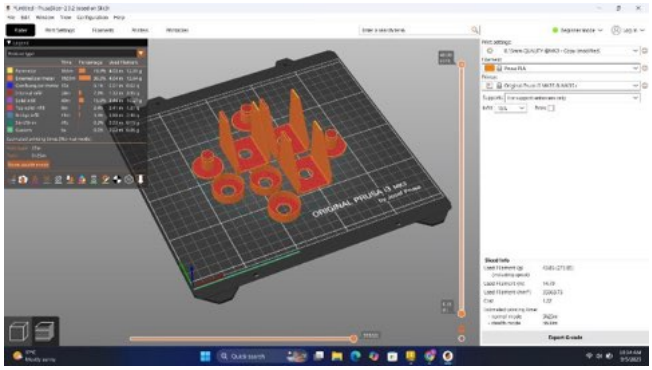


Fig. 21. Castor wheel design slicing for 3D printing using Prusa Slicer

F. Challenges faced

- First time experience. Since this was our first time experience in the Robotics Dojo competition for our group, we had to learn most of the concepts from scratch which was a slight challenge.

- Limited funds. We had to buy components within the budget we were allocated which forced us to design some components and also use 3D printed parts for the platforms as well rather than acrylic.

- Limited time allocated. The allocated time of one month to complete the project proved to be quite a challenge especially with all the concepts that required to be integrated with the robot.

G. Acknowledgement

We would like to express our gratitude to the Japan International Corporation Agency (JICA) for sponsoring this program and providing all the essential components and for the Jomo Kenyatta University of Agriculture and Technology for providing a venue for the competition. We thank the Robotics Dojo intern team and Lenny Nganga, for their support and mentorship. We are also grateful for the blogs that they provided regarding various topics on their substack. We would also like to appreciate Articulated Robotics for all his YouTube videos and blogs that have been instrumental in our journey. We are also grateful for Brian Fundi and Nathan Machira ,fellow competitors, for their unwavering assistance, support and feedback

H. References

[1] Articulated Robotics, ROS 2 Tutorials and Guides [Online]. Available: https://articulatedrobotics.xyz/

[2] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The Nav2 project: ROS 2 navigation system," Proceedings of ROSCon, 2020.

[3] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2149–2154, 2004.

[4] ROS 2 Control Working Group, ros2_control Documentation [Online]. Available: https://control.ros.org/

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

[6] M. Ferguson, ros_arduino_bridge Package. [Online]. Available: https://wiki.ros.org/ros_arduino_bridge

[7] Robotics Dojo, "Robotics Dojo," Substack.. Available: https://roboticsdojo.substack.com/.

[8] Danilo Gervasio, (2022 October). Design and development of an Autonomous Mobile Robot (AMR) based on a ROS controller, [Online], Available: https://thesis.unipd.it/retrieve/dfe158de-ec8a-41d3-a9b9-6854970fbf38/Gervasio_Danilo.pdf

[9] B. Satish Chandra, M. Shiva Kumar, P. Vaishnavi, P. Rahul, P. Dayakar, (2025 June). Smart Robot for Plant Disease,Detection,[Online],Available: https://restpublisher.com/wp-content/uploads/2025/06/18.-Smart-Robot-for-Plant-Disease-Detection.pdf?utm

[10] Computer Vision Tutorial - GeeksforGeeks [Online]. Available; https://www.geeksforgeeks.org/computer-vision/computer-vision/

[11] OpenCV contour-based color detection in Python — GeeksforGeeks,[Online].Available: https://www.geeksforgeeks.org/computer-vision/computer-vision/

[12] ROS 2 rqt plugin development — ROS 2 Documentation,[Online],Available: https://docs.ros.org/en/rolling/Tutorials/GUI/Tutorials/rqt.html

[13] Smith, J.; Kumar, SLAM for agricultural robotics: fusing LiDAR and vision sensors — A. Journal of Field Robotics, 2023.