JOURNAL OF ADVANCED MECHANICAL ENGINEERING APPLICATIONS
e-ISSN: 2716-6201

JAMEA

Vol. 6 No. 1 (2025) 22-39
https://publisher.uthm.edu.my/ojs/index.php/jamea

Development and Testing of an Autonomous Mobile Robot for Material Handling Using SLAM and Nav2

Prince Panta[1], Nirmal Prasad Panta[1], Pawan Shrestha[1], Saki Basnet[1], Surya Prasad Adhikari[1]*

[1] *Department of Mechanical and Aerospace Engineering, Pulchowk Campus, Institute of Engineering, Tribhuvan University, NEPAL*

*Corresponding Author: spadhikari@pcampus.edu.np
DOI: https://doi.org/10.30880/jamea.2025.06.01.003

Article Info

Received: 3 February 2025
Accepted: 15 April 2025
Available online: 30 June 2025

Keywords

Autonomous mobile robots, simultaneous localization and mapping, navigation2, OpenCV, computer vision

Abstract

The adoption of Autonomous Mobile Robots (AMRs) for material handling has grown significantly across manufacturing, healthcare, and the service sector. To stay competitive in the age of automation, businesses are increasingly shifting from manual labor to AMRs to enhance the efficiency and reliability of transportation and material handling tasks. This study outlines the development of an AMR that integrates Simultaneous Localization and Mapping (SLAM), Navigation2 (Nav2), and Computer Vision (CV) to enhance material handling efficiency. The development process is structured into three distinct phases to ensure clear tracking and milestone management. In the first phase, the mobile robot is designed and developed. The second phase focuses on integrating SLAM and Nav2, enabling precise and efficient navigation within complex, dynamic environments. In the final phase, OpenCV is implemented for ArUco tag detection, enhancing the AMR's capability to perform material handling operations. This study presents a structured approach to developing Autonomous Mobile Robots (AMRs) equipped with real-time obstacle detection, avoidance, path planning, and material handling capabilities, streamlining deployment in real-world applications.

1. Introduction

In a rapidly evolving world, agile corporations are best positioned to sustain growth and thrive, leveraging their adaptability and capacity to deliver diverse products swiftly [1]. Robotics, particularly AMRs, play a vital role in fostering agility by enabling flexible, efficient production processes. As a result, substantial investments in robotics are expected to continue rising.

Global spending on military robotics is projected to reach $16.5 billion by 2025 [2]. Additionally, 88% of businesses worldwide are planning to adopt robotic automation, and robotics orders saw a 67% increase in the second quarter of 2021 [3-4]. Collaborative robots (cobots), such as Amazon's 'Kiva' system, have significantly boosted efficiency in large-scale warehouse operations. Without cobots, up to 80% of workers' time is spent navigating the warehouse, leaving only 20% for productive tasks like order fulfillment. With cobots, this ratio is reversed—allowing workers to dedicate 80% of their time to fulfilling orders. This dramatic improvement in operational efficiency, alongside effective inventory management, is a cornerstone of Amazon's logistical success [2].

AMRs enhance manufacturing operations by navigating complex factory layouts, adapting to shifting production demands, and managing material replenishment using systems such as Kanban signals. By minimizing labor dependency and maximizing operational efficiency, AMRs contribute to cost reduction, increased productivity,

and long-term profitability. Their ability to provide a flexible, 24/7 material flow system helps mitigate disruptions caused by labor shortages, health risks, and safety concerns [5]. The COVID-19 pandemic underscored the critical need for such automation technologies to strengthen supply chain resilience, address workforce limitations, and maintain continuous operations.

The primary challenge lies in developing an AMR-based automation system that enhances manufacturing agility, material handling, productivity, cost-efficiency, and operational resilience. Such a system must ensure efficient material flow and support just-in-time delivery by leveraging advanced navigation and computer vision algorithms to meet the evolving demands of modern manufacturing environments.

The evolution of Automated Guided Vehicle Systems (AGVSs) from basic track-guided systems in the 1950s to advanced, sensor-driven, and wireless-controlled technologies by 2010 has significantly improved their reliability and versatility [6]. This transformation has been driven by the increasing complexity of manufacturing processes, including modular product designs and flexible manufacturing systems (FMSs), which require efficient transport of raw materials, semi-finished products, and finished goods across inventory, job stations, and assembly lines to remain competitive in global markets [7]. However, despite their routing flexibility, AGVs remain complex and costly, with inefficiencies potentially impairing system performance, such as causing machine starvation [8].

Modern AGVs have evolved from traditional fixed-route systems, which relied on tapes or lines, to utilizing virtual maps created by laser sensors or GPS. This shift has enhanced flexibility, precision in maneuvering, and improved obstacle-avoidance capabilities [9]. Lynch's review underscores the vital role of sensors in advancing AGV navigation and enhancing automated manufacturing processes [10]. The paper compares various AGV types—including Laser Guidance, Line Following Guidance, Magnetic Spot Guidance, and Barcode Guidance, evaluating them based on factors such as cost, complexity, flexibility, efficiency, and ease of expansion. It also highlights the pivotal role of Laser Scanners and Magnetic Guide Sensors in improving AGV navigation. The integration of Proportional-Integral-Derivative (PID) controllers in robotic navigation has proven effective in achieving high-precision 2D movements, as demonstrated by experimental prototypes designed for autonomous unmanned systems [11].

The emergence of AMRs has overcome the limitations of traditional AGVs by utilizing advanced control systems for self-navigation and obstacle avoidance in dynamic environments, significantly enhancing operational flexibility and efficiency in manufacturing settings. LiDAR plays a critical role in enabling precise 3D perception for AMRs, providing real-time distance measurements regardless of lighting conditions. It supports navigation, obstacle detection, and collision avoidance, and when combined with camera data, enhances scene understanding and path planning. Its ability to perform reliably in low-light environments makes LiDAR indispensable for autonomous systems [12]. Furthermore, the integration of machine learning and AI has led to the development of Autonomous Intelligent Vehicles (AIVs), which enhance flexible manufacturing systems by optimizing software interfacing, wireless communication, navigation, route planning, and payload handling. These advancements rely on sophisticated algorithms for dynamic decision-making and fleet management, maximizing efficiency and system visibility [13].

A significant advancement in this field is the emergence of graph-based SLAM, as detailed in Grisette et al.'s tutorial. This approach revolutionizes mobile robotics by framing SLAM as a graph problem, enabling enhanced error minimization techniques and probabilistic formulations for both 2D and 3D mapping [14]. The integration of the SLAM Toolbox, an open-source ROS package, into ROS2 Navigation2 further enhances real-time mapping capabilities, allowing for the mapping of spaces up to 24,000 m^2. This advancement surpasses previous SLAM libraries, addressing the needs of both industrial and research applications [15].

In addition to autonomous navigation, AMRs have seen significant feature enhancements, expanding their applicability across various sectors such as manufacturing, agriculture, horticulture, healthcare, and logistics [16-20]. Many of these applications involve handling dynamic objects, where accurate pose estimation is essential. To facilitate this, various computer vision tools have been developed, with fiducial markers being a widely used solution. Fiducial markers serve as reference points or measurement tools in images, often encoded with specific IDs or messages. Common fiducial marker systems include ARToolkit, ARTag, ArUco, and Pi-Tag. Among these, ArUco markers stand out due to their customizable libraries, making them particularly suitable for applications such as UAV landing systems and augmented reality [21-22]. Despite these advancements, existing studies often concentrate on individual components of AMRs. There is a growing need for a standardized framework that seamlessly integrates ROS2 packages, such as ros2_control, Slam_toolbox, and Navigation2, with computer vision libraries like OpenCV to enable efficient autonomous navigation and material handling [14, 23-24]. This integration remains particularly underexplored in the context of two-wheeled differential drive robots.

Building on the advancements and gaps identified, this study aims to develop a comprehensive framework for two-wheeled differential drive robots. By leveraging ROS2 and its core packages, and integrating them with OpenCV, the focus is on enhancing autonomous navigation and material handling capabilities. The following sections outline the systematic design approach used to achieve these objectives.

2. Mechanical Design and Structural Analysis

A differential (non-holonomic) drive with a rectangular frame was chosen for locomotion due to its ability to maneuver in tight spaces, ease of positioning, accessibility of internal components, simplicity, and cost-effectiveness. Additionally, a unit load material handling system was selected for its compatibility with flexible production line environments [10, 26].

2.1 Chassis Design

A rectangular frame was designed and analyzed, resulting in overall robot dimensions of 900 mm by 622.80 mm in ground projection. The chassis was engineered to support a load of 100 kg, which was verified through a static structural study of the weldments. The analysis yielded a factor of safety of 1.9 for von Mises stresses and a maximum displacement of just 0.2993 mm (Fig. 1). Two hub motors (24 volts, 350 watts, with a 1:16 gear reduction) were mounted at the center of each side of the frame's width, paired with Segway scooter tires and two caster wheels positioned centrally along the frame's length. Rotary encoders were attached to the rear of each motor. Additionally, mounts were incorporated for a camera (positioned 65 mm above the base/frame, 510 mm in front of the center, and inclined at 75 degrees) at the front, along with an RP-LiDAR placed at the top center.

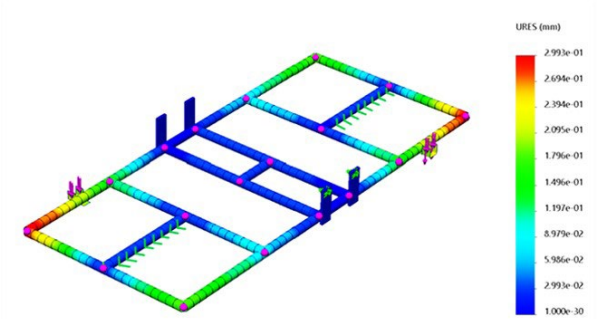

Fig. 1 *Displacement plot of the chassis frame generated under a 100-kilogram load applied at the linear actuator mountings (for material handling), with the motor and caster wheel mountings fixed*

2.2 Construction of References Load Lifting Actuation Mechanism

Linear actuators with a 1500N load capacity have been utilized in the system. Typically, vertical actuation in material handling requires at least three or four linear actuators for stability; however, due to cost and availability constraints, the AMR uses only two actuators positioned diagonally. This setup presents challenges when the top platform is asymmetrically loaded, as the hinge joints of the linear actuators allow the platform to rotate. To mitigate this, telescopic channels are placed at the remaining corners to restrict movement to the vertical axis, while struts are installed between the linear slider and the platform to prevent deformations in the pins and other parts, which could cause platform instability under asymmetric loads. These struts deform under axial force, compressing by approximately 0.707 times the displacement of the pin joint. Reducing this displacement can be achieved by increasing the height (h) or decreasing the distance (y) from the hinge to the pin joint, as shown in Fig. 2. However, limitations in attachment points on the telescopic channel (i.e., fixed h) and the increased load leverage (with smaller y) restrict these adjustments. The resistance force comes from the strength of the struts, which is directly proportional to their displacement, helping to stabilize the platform.

2.3 Kinematic and Dynamic Analysis

A differential drive mechanism was chosen for the robot due to its simplicity and ease of maneuverability. This system utilizes two drive motors connected to two wheels, referred to as power wheels. For stability, two additional caster wheels are incorporated (Fig. 3). While three-wheel systems are typically preferred for rough terrain due to their continuous ground contact, this limitation can be addressed in the future by adding springs or suspension to the casters. Currently, the design assumes that the robot will operate on smooth indoor floors.

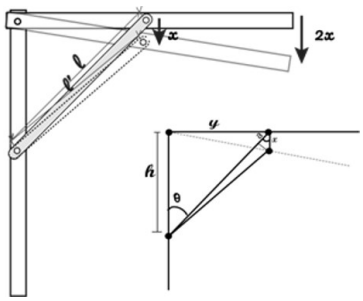

Fig. 2 *Geometry of strut; where x is displacement at pin joint, l is original strut length, ′ is deformed strut length, h is the pin distance from top frame, y is the top pin distance from far end, θ is the angle made by strut with vertical column*

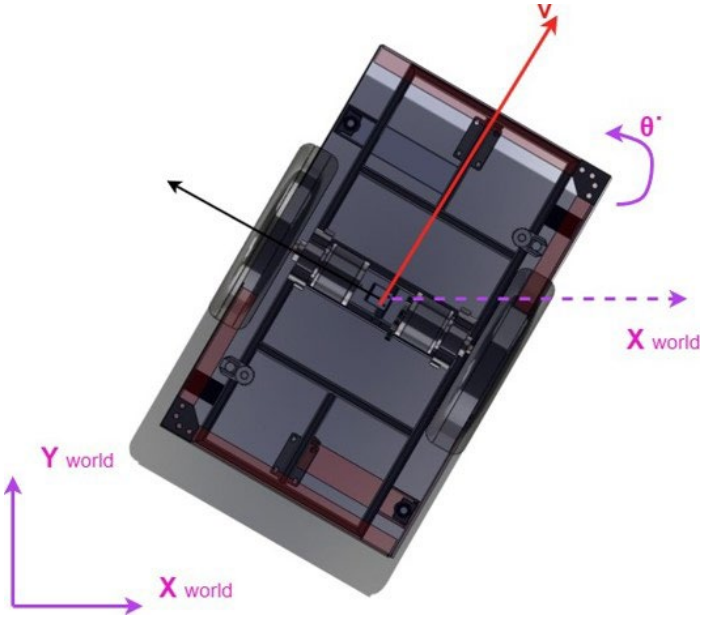

Fig. 3 *Steering direction for differential drive AMR, which is at a certain angle with a global frame X_{world} and Y_{world} v represents the linear forward velocity in the robot's local frame and θ represents the angular velocity in an anticlockwise direction*

The robot's kinematics are modeled in two dimensions, requiring two motion variables: linear velocity (v_x) and angular velocity (ω_z). The robot's motion can be described as follows:

- The instantaneous linear velocity in the body frame is:

$$v_{\text{body, linear}} = \begin{pmatrix} v_x \\ 0 \end{pmatrix} \tag{1}$$

- The linear velocity in the world frame is:

$$v_{\text{world, linear}} = \begin{pmatrix} v_x \cos\theta \\ v_x \sin\theta \end{pmatrix} \tag{2}$$

- The angular velocity is defined as:

$$\omega = \dot{\theta} \tag{3}$$

- Combining linear and angular velocities into a single vector:

$$v_{\text{world}} = \begin{pmatrix} v_x \cos\theta \\ v_x \sin\theta \\ \dot{\theta} \end{pmatrix} \tag{4}$$

Given linear velocity (v_x) and angular velocity (ω_z), the left (V_l) and right (V_r) wheel linear velocities can be calculated as:

- Left wheel velocity:

$$V_l = \frac{v_x - b\omega_z}{2} \tag{5}$$

- Right wheel velocity:

$$V_l = \frac{v_x - b\omega_z}{2} \tag{6}$$

Where (b) is the width between the two wheels.
The required angular velocities of the left (ω_l) and right (ω_r) wheels can be then obtained as:

- Left wheel angular velocity:

$$\omega_l = \frac{V_l}{r} \tag{7}$$

- Right wheel angular velocity:

$$\omega_r = \frac{V_r}{r} \tag{8}$$

Where r is the radius of the wheels.
These equations 2.5 to 2.8 are later used in the hardware interface of ROS system to control the wheel velocity using a PID loop as per the desired command velocity.

Then, to calculate the maximum payload capacity, the force exerted by the wheels (F_r) is used against the friction force of the tire(F_{tire}), air resistance (F_{aero}), inertial force ($F_{inertial}$), and force against slope (F_{slope}) as:

$$F_r = F_{\text{tire}} + F_{\text{aero}} + F_{\text{inertial}} + F_{\text{slope}} \tag{9}$$

Here, the air resistance (F_{aero}), is negligible for low velocity applications and with a flat indoor environment, the AMR has zero resistance by slope (F_{slope}). The remaining friction resistance of the tire (F_{tire}), and inertial force ($F_{inertial}$) can be calculated as:

$$F_{\text{tire}} = c_{\text{tire}} \cdot mg \tag{10}$$

$$F_{\text{inertial}} = \text{m} \cdot \ddot{x} \tag{11}$$

where (c_{tire}) is the coefficient of friction for rolling, (m) is the mass of the vehicle, ($\ddot{x}$) is acceleration, and (g) is acceleration due to gravity. Since this is provided by two motors, we obtain:

$$F_r \cdot r = 2 \cdot \tau_{\text{motor}} \tag{12}$$

where (r) is the radius of the wheel and (τ_{motor})is the torque of each motor.
Given, c_{tire} = 0.005, g = 9.81m/s^2, r = 0.205m, maximum acceleration ($\ddot{x}$) = 0.328 m/s^2 and torque of each motor (τ_{motor})= 125 kg.cm, we can find the mass m as:

$$\text{m} = \frac{2 \cdot \tau_{\text{motor}}}{r \cdot (c_{\text{tire}} \cdot g + \ddot{x})} \tag{13}$$

Reducing the robot's mass of 30.88 kg, the maximum payload capacity can be found as:
Maximum Payload = m − 30.88 kg (Eq. 2.13)
Maximum Payload = 127.7 kg
This aligns with our target load of 100 kg.

3. Control System Design and Development

ROS2 Humble was chosen as the middleware for controlling the Autonomous Mobile Robot (AMR) due to its extensive software package library and strong community support, making it more favorable than newer versions. The ros2_control framework of ROS2 is used with diff_drive_controller package for transportation control. Then, for autonomous navigation, the slam_toolbox is used to create workspace maps, while the Nav2 package handles navigation within these maps. Additionally, OpenCV is utilized to detect ArUco tags placed on the racks. The

transforms of these tags are calculated relative to the map frame and provided to Nav2 as goal pose for accurate docking. This is followed by a sequence of commands enabling autonomous material handling. These can be divided into three distinct phases as mentioned below.

3.1 Phase 1: Transportation Control

Effective control of an AMR relies on the precise regulation of wheel velocities, which is crucial for accurate navigation and obstacle avoidance. The core of this control system is the ability to independently adjust the speeds of the robot's wheels, especially in differential drive robots, allowing for smooth and precise movement. A key element of this velocity regulation is the Proportional-Integral-Derivative (PID) controller, which is tuned to optimize three main parameters: proportional gain (Kp), integral gain (Ki) and derivative gain (Kd). These parameters can be empirically tuned without requiring a mathematical model, using a microcontroller in conjunction with encoders to ensure that the robot responds effectively to control inputs, minimizing errors and achieving stable control. The resulting PWM values from the PID loop are then passed to the motor driver, regulating the wheel velocities. Fig. 4 shows the Raspberry Pi running ROS2 communicates with an Arduino Uno micro-controller.

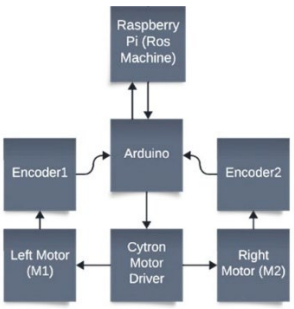

Fig. 4 *Raspberry Pi running ROS2 communicates with an Arduino Uno micro-controller. The Arduino interfaces with encoders on the left and right motors and signals a Cytron Motor Driver, enabling precise differential drive control for transportation*

To develop and manage control systems effectively, ROS2 can be employed as middleware, offering a robust framework for integrating various control components and managing their communication. Specifically, ros2_control serves as the backbone for handling all control components of the AMR. Key elements include the diff_drive_controller package, hardware interface plugin, joint state broadcaster plugin, and controller manager. The diff_drive_controller translates command velocities into abstract wheel velocities, while the hardware interface plugin converts these into signals for motor controllers, using the ros_arduino_bridge to facilitate communication between the Arduino and the package diff_drive_controller. The controller manager, within the ros2_control_node, coordinates all components, and the joint state broadcaster reads motor encoder positions from the hardware interface, publishing them to the /joint_states topic to enable the robot state publisher to generate wheel transforms.

This architecture allows the AMR to be controlled remotely using the teleop_twist_keyboard package. This package contains a node of the same name that publishes the desired velocity, entered through the developer's computer keyboard, to the /cmd_vel topic. This topic is mapped to /diff_cont/cmd_vel_unstamped, which the controller listens to. Additionally, a joystick can serve as a user-friendly input device for manual control of the AMR by using the joy package, which converts raw joystick data to the /joy topic and then to a Twist message via the teleop_twist_joy package. The joy_node receives raw joystick data and publishes button presses and their corresponding values to the /joy topic. Using this topic, the teleop_node publishes messages to the /cmd_vel topic based on the parameters in the configuration file. Again, the /cmd_vel topic is remapped to /diff_cont/cmd_vel_unstamped as required by the controller manager to achieve the desired control for transportation (Fig. 5).

J. of Advanced Mechanical Engineering Applications Vol. 6 No. 1 (2025) p. 22-39

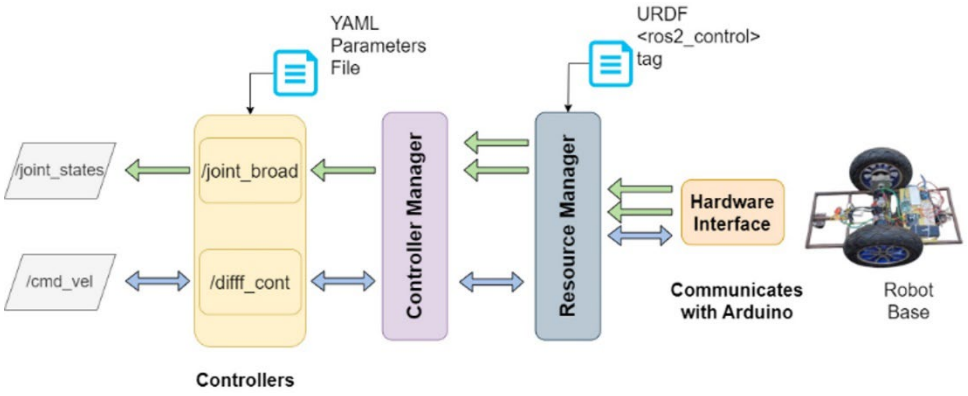

Fig. 5 *The ROS2-based control system for an AMR uses topics like /cmd_vel and /joint_states, managed by the Controller Manager. The diff_drive_controller converts velocity commands, while the joint state broadcaster ensures accurate encoder data, interfacing with hardware for coordinated movement control*

3.2 Phase 2: Autonomous Navigation Enhancement

Please The effectiveness of efficiently building large maps reliably and easily makes the slam_toolbox an appropriate choice for the AMR's autonomous navigation [13]. It provides the necessary tools and utilities to simultaneously build the map and localize the robot within it using sensors like LIDAR or cameras, and devices like IMU or odometry. For 2D point clouds generated by LIDAR, the data can be converted into ROS LaserScan messages using the relevant LIDAR package. This message is then published to the /scan topic, which is used to generate point clouds in visualization tools like RViZ from the laser frame defined in the URDF of the robot model. Once the LIDAR is set up, the slam_toolbox can be launched to create a map of the workspace. To obtain a live map while processing only the latest scans for better performance, the online asynchronous mode in slam_toolbox is available. RViZ provides visualization capabilities by adding the map to the display and setting the topic to /map. The fixed frame should be set to "map" to ensure stability as the map gradually forms while the robot navigates the workspace. Various services provided by the slam_toolbox, such as /serialize_map and /save_map, can be utilized for map management. Additionally, the SlamToolbox plugin can be accessed in RViZ for these services using a graphical user interface. Options for saving maps in different formats such as .pgm and .yaml (old format), and .data and .posegraph (new format) are available. These saved maps support lifelong mapping, allowing them to be refined and updated as the AMR continues to interact with its environment.

The creation of a map and the localization of the AMR within it facilitate the use of the nav2 package for autonomous navigation, assisting the AMR in reaching its goals while avoiding live obstacles. The nav2 package not only enables movement from Point A to Point B, but also supports intermediary poses and various tasks such as object following and complete coverage navigation. To complete a user-defined task of reaching a goal pose, the planner, controller, smoother, and recovery servers, known as navigation servers, work in coordination with the BT Navigator Server. The BT Navigator Server, utilizing behavior trees, orchestrates the logical communication between these servers to reach the goal pose. The planner server receives the map of the environment and LIDAR data, computing the required path to reach the goal pose and creating a global costmap. Next, the controller server comes into action, guiding the AMR along the planned path and updating the path's status at each step by creating a local costmap while providing the desired velocity of the AMR through the /cmd_vel topic. The AMR's size and shape are relayed to the Behavior Server to ensure the costmaps accurately represent its footprint and to be prepared for recovery actions if any faults arise. While following the path to the goal, the smoother server prevents the AMR from making unnecessary rotations and helps avoid high-cost areas on the map. However, this process is not strictly followed as the BT Navigator can send the call to a specific task server whenever required. Additionally, a custom behavior tree can be implemented to be used by the BT Navigator using XML files [24].

3.3 Phase 3: Material Handling Functionality

Material Handling Functionality is enabled through the use of OpenCV and ArUco tags. This is then integrated with Nav2 for navigation purposes. This task starts out with a simulation phase done in Gazebo. This enables writing and testing of code in a simulation environment and finally on the real robot in a fast manner. For this, it is essential to set up OpenCV package along with cv_bridge package to facilitate communication between OpenCV and ROS2. Basically, with the help of this package image can be converted to type supported by OpenCV to ROS2 and vice-versa. Meanwhile, the image detected by the camera – either in simulation or real-time – will be published to a topic called /image_raw

Now, subscribing to this topic, the detection of ArUco tags is done with the help of detectMarkers method. After doing necessary camera calibration, the distance to the tag can be found using estimate Pose Single Marker method, which gives translational (tvecs) and rotational vectors (rvecs) respectively. When implemented in Gazebo, the distance to the tag can be found.
The above-mentioned transform is between camera to aruco tag or camera_aruco. However, for proper navigation the transformation map_aruco is required. For that, quaternions for camera_aruco are changed by defining roll, pitch and yaw angles as -90,0,90 degrees respectively. This will be used to change the initial rotational matrix being calculated from rvecs. Then, the changed rotational matrix is converted to updated quaternion. Finally, this map_aruco is broad- cast. Meanwhile, to navigate towards a location near to the racks, CommanderAPI from Nav2 is used. For this to run, it is necessary to ensure the running of Nav2 server. Then, a basic navigator is instantiated using BasicNavigator() constructor. This enables us to use methods like goToPose(), cancelTask(), getResult(), spin() etc.

To send the robot to a particular pose, a goal pose needs to be setup which is done using PoseStamped(). It takes parameters like frame, translation and rotation coordinates. Finally, this pose is fed to goToPose() method, enabling the robot to reach its destination. This particular node – which navigates the robot to its destination – also subscribes to a topic /binary where values of 1 or 0 are being published. '1' gets published if the camera of the robot detects the ArUco tag. Meanwhile, as long as '0' is detected, the spin functionality is executed, where the robot spins. After the tag is detected, the next phase starts enabling the robot to stop spinning and move towards the tag. This phase also involves the use of CommanderAPI and aforementioned methods. The goal pose, however, is the map_aruco broadcast, which is converted into PoseStamped type.

When the robot completes this phase, now is the turn of the actuators to activate and move up for a certain amount of time. After this happens, the robot spins and CommanderAPI takes over again, leading it to its new destination.

4. Implementation, Testing, and Results

In this section, the development and integration of the Autonomous Mobile Robot (AMR) are detailed across mechanical assembly, circuit design, and operational testing. These efforts aimed to validate the robot's functionality under various operational scenarios, ensuring robust performance across simulation, tele-operation, and autonomous modes.

4.1 Mechanical Assembly

The CAD model developed and analyzed using SOLIDWORKS served as the reference for fabricating the AMR with locally available resources and techniques. The fabrication process was carried out in two steps: first, the chassis was constructed for transportation purposes, which was used during phases 1 and 2. Subsequently, the material handling base load features were integrated. The final design and assembly are shown in Fig. 6.
The placement of motors, coupling of encoders, and positioning of sensors such as LiDAR and cameras were carried out with precision due to their critical roles. The developed AMR's dimensions deviated slightly from the original specifications, with a precision variance of about 10mm. These changes, along with the required weights, were incorporated into the simplified URDF model of the AMR, which is used in ROS2 for transforms and simulation in Gazebo, as well as visualization in RViz2.

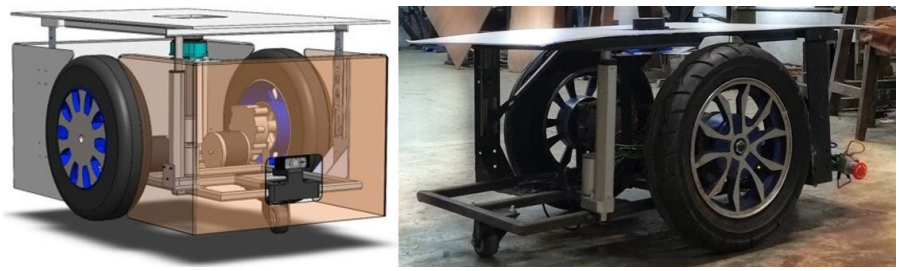

Fig. 6 *AMR final mechanical assembly; (a) AMR CAD model front 3D view with linear actuator in actuated position (as lifting the load), (b) Actual prototype - 3D view with linear actuator in normal position with no payload (no actuation)*

4.2 Mechanical Assembly

The Raspberry Pi serves as the brain of the AMR and communicates with the development computer via Wi-Fi for command transmission. The computer is responsible for publishing command velocities (for navigation), whether from user input (joysticks) or Nav2 (for autonomous navigation), and logic messages (for the linear actuator). It

is connected to the joystick via a USB port as part of the AMR system. The Raspberry Pi runs the ros2_control framework, which processes the command velocities from the cmd_vel topic and sends corresponding signals to the Arduino Uno via a serial port. The Arduino Uno executes a PID loop, sending PWM values to the motor driver to achieve the desired robot motion. It also receives encoder signals for feedback. Additionally, Raspberry Pi runs a ROS node to send corresponding serial messages to the Arduino Mega based on the received logic messages. The motors are connected to the Cytron motor driver, which receives PWM and direction signals from the Arduino and powers the motors accordingly. Raspberry also receives signals from camera and LiDAR and publishes them into respective ROS topics. Furthermore, it publishes robot states in joint_states topic.

Initially, breadboards and jumper wires were used to test the circuits, which were later replaced by a single PCB and a Matrix Broad circuit, with signal wires connected using Japan Solderless Terminal (JST) connectors. The circuits were straightforward, as no separate power circuit was required for the microcontrollers; they were powered and communicated with the Raspberry Pi via USB cables. Wires from the motor drivers and encoders were connected to 6-pin and 4-pin JST connectors, respectively, which were then linked to their corresponding connectors on the circuit boards to interface with the Arduino serial pins. The motor drivers were connected in parallel to the battery via a power cable passing through an emergency switch. The hub motors and linear actuators were subsequently connected to the respective outputs of the motor drivers. For power, the Raspberry Pi was connected to a power bank via a USB Type-C cable. The final circuit is depicted in the schematic in Fig. 7. The actual developed circuit is as shown in Fig. 8.

4.3 Mechanical Assembly

The functionality of Autonomous Mobile Robots (AMRs) was rigorously tested and evaluated through multiple stages of development, encompassing a variety of operational scenarios, including simulation, teleoperation, and autonomous operation. The testing process was structured into seven distinct steps, each focusing on specific aspects of the robot's functionality and performance. This comprehensive approach ensured a thorough evaluation of the AMRs' capabilities across diverse environments and operational modes.

4.3.1 Mechanical Assembly

Before deploying code to physical hardware, we created a simulation to replicate the robot's motion in a virtual environment using Gazebo. Gazebo allows physics-based simulations, crucial for testing and refining robot functionalities. The first step involved creating a detailed robot model with integrated sensors using URDF and Xacro files. These files describe the robot's physical and kinematic properties, ensuring the virtual model accurately represents the real AMR. Various components, including the chassis, wheels, LIDAR, and camera, were modeled and integrated into the simulation environment. Gazebo plugins were utilized to simulate sensor functionalities and robot control, enabling realistic behavior of the AMR in the virtual world

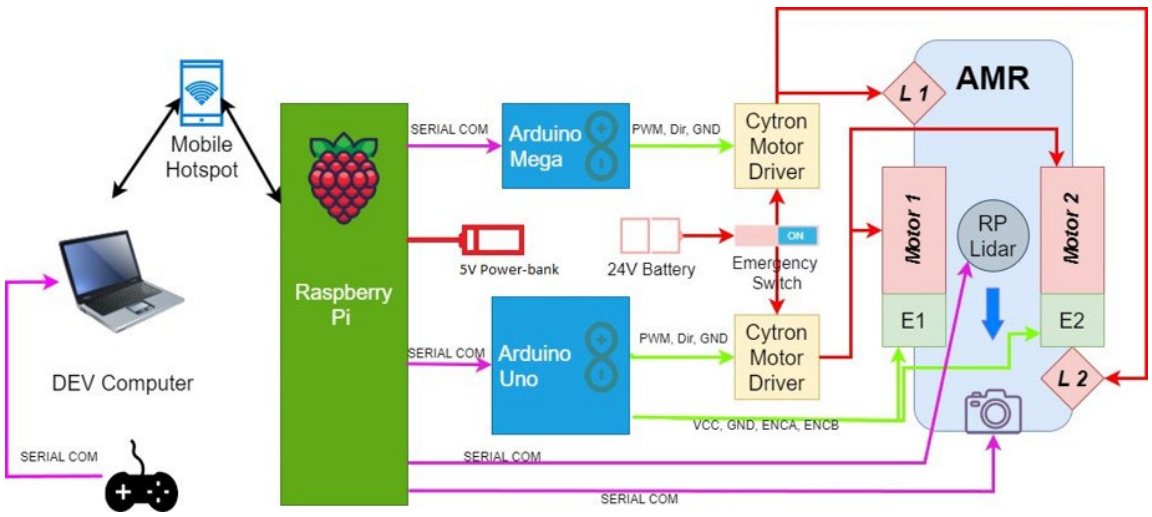

Fig. 7 *Schematic of circuit used in AMR: Arduino Uno is connected to motor driver for wheels and encoder, Arduino Mega is connected to the motor driver of linear actuators, 24V battery powers motor driver, another 5V power-bank powers Raspberry Pi both Arduino along with RP-LiDAR and camera are connected to Raspberry by USB ports, Raspberry Pi communicated to the DEV computer (server) through Wi-Fi and joystick is connected to DEV computer via USB port*

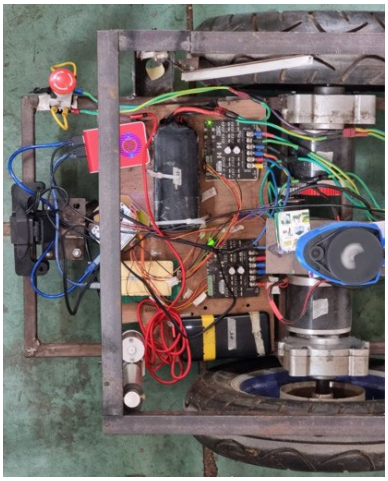

Fig. 8 *The actual circuit developed as in the schematic of fig. 7*

To manage the simulation efficiently, a ROS2 package named "project_amr" was created, containing all necessary launch files, configuration files, and the world file for the Gazebo simulation. The simulation setup included defining the robot's links and joints, specifying dimensions and mass properties, and integrating sensors with respective Gazebo plugins. ROS2_control was implemented to simulate real-world control mechanisms. The simulation environment was launched using custom launch files, allowing seamless testing and visualization in RViz. The virtual robot could be then driven using the teleop_twist_keyboard in the virtual environment within Gazebo.

4.3.2 Tele-operation of Real Robot Navigation

For autonomous navigation, the robot must accurately sense both its environment and its own motion. Motion sensing is achieved through odometry data collected from rotary encoders on both motors, which is processed through an interrupt routine in Arduino to count wheel rotations. This data is fed into a PID loop that adjusts motor speed by comparing the target and actual encoder counts, with finely tuned PID coefficients ensuring stability. Control and computation are managed by a Raspberry Pi 400 running ROS2, which communicates with the Arduino via USB. Velocity commands are converted into encoder counts within the PID loop and sent to the Arduino for precise motor control.

Following the simulation, AMR utilizes ROS2 control with key components including the diff_drive_controller, a custom hardware interface plugin, joint state broadcaster plugin, and controller manager. The diff_drive_controller translates command velocities into wheel velocities, which our custom hardware interface plugin converts into motor controller signals. The controller manager, within the ros2_control_node, manages all components, while the joint state broadcaster reads motor encoder positions and publishes them to the /joint_states topic, enabling the robot state publisher to generate wheel transforms. This setup ensures coordinated control and real-time feedback for accurate navigation.

The hardware interface exposes velocity command interfaces for each motor, along with velocity and position state interfaces. It communicates with the Arduino using "m" and "e" commands over serial communication. The "m" command controls motor velocity through PWM signals based on PID loop calculations, while the "e" command retrieves encoder positions to determine wheel speed and position. The URDF file, ros2_control.xacro, includes parameters for the hardware interface, such as loop rate, device path, baud rate, timeout, and encoder count per revolution. These parameters ensure precise control and communication with the hardware.

The robot was driven in the workshop using a joystick and visualized in RViz2, where its odometry was tuned. Encoder accuracy was enhanced by adjusting the encoder counts per revolution based on loop closure in RViz. To achieve precise motion, the wheel radius was updated for linear accuracy and wheel separation for rotational accuracy. This ensured seamless and accurate motion in both the real environment and RViz, essential for proper odometry and URDF representation in ROS2 control.

4.3.3 Autonomous Navigation in Gazebo

First, the parameter files for the Nav2 package and SLAM Toolbox, based on Nav2 tutorials and formatted in .yaml, were placed in the config folder. Using the SLAM Toolbox plugin, a map of the environment was created as the robot navigated via teleoperation. This mapping process was visualized in RViz, with the generated map reflecting the factory model constructed in Gazebo. The map's white regions represented the mapped portions of the

environment, grey regions indicated unmapped areas, and black lines marked obstacles. Once the mapping was complete, the map was saved for future navigation tasks.

The map was serialized to ensure compatibility with navigation processes, and the location of the serialized map data was included in the SLAM configuration file within the config folder. During the navigation launch, the previously saved map was referenced in the parameter file. In RViz, a goal pose was set on the map, represented by a green arrow. The Nav2 package then calculated the optimal path to the goal, enabling the robot to autonomously navigate to the designated location without human intervention.

4.3.4 Autonomous Navigation in Real Robot

The same parameter files used in the simulation were employed in real-world navigation. First, the slam_toolbox was launched in online asynchronous mode from the terminal. Then, RViZ was launched with the previous configuration used in simulation to instantly add the map to the display and set the topic to /map. The AMR was navigated in the workspace to form the map while visualizing it in RviZ (Fig. 9). Once the map was created, it was serialized and saved using the SLAM Toolbox plugin available in the RViZ panel. This serialized map was then referenced in the config folder for subsequent navigation. Once the map of the workspace was created, the nav2_bringup package was sourced with the launch file from the terminal to utilize the features available in Nav2 (Fig. 10). Initially, the RViZ interface was used to give a goal pose to the AMR directly on the generated map by pointing to a specific location. The path planned by the planner server to reach the goal pose can be visualized in RViZ by adding "Path" to the display. The global costmap used by the AMR to understand the workspace can also be visualized by adding another map and setting the topic to /global_costmap/costmap. This visualization provides information about the new path planned by the navigation servers when a live obstacle is encountered.

Fig. 9 *The SLAM Toolbox plugin is used to serialize and save the map created of the workspace. The grey region represents obstacle free zone, and the black lines outline the presence of obstacle*

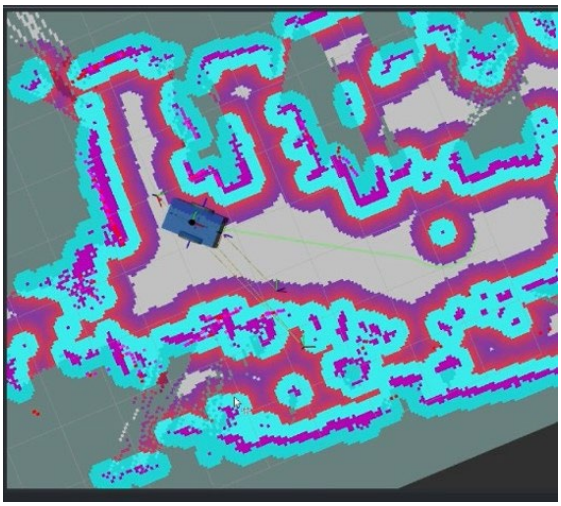

Fig. 10 *The nav2 stack provides real time visualization of path being planned and the corresponding global costmap. The green line represents the path planned by the planner server to reach the goal. The costmap with regions of shades of blue and red represents the obstacle and the white regions are the free path*

4.3.5 Tele-operation of Material handling

We utilized a base load material handling system with two linear actuators controlled by an Arduino Mega 2560 in an open-loop manner. Each actuator received a specific PWM signal for a predetermined duration, determined experimentally to achieve approximately 9 cm displacement under initial loads caused by design constraints. Due to non-uniform resisting forces from imprecise prototyping, distinct PWM values of 150/255 and 140/255 were required for the actuators. The lifting and lowering durations were set to 10.5 seconds and 10.7 seconds, respectively, with additional time allocated for lowering to account for potential battery depletion affecting the PWM signal. The Arduino program utilized an if statement to control actuator movements based on received serial messages. It included logic to prevent repeated actuation, using the Millis () function to track time and stop PWM signals once the duration was exceeded.

A Raspberry Pi, functioning as the brain of the Autonomous Mobile Robot (AMR), sends command signals to the Arduino via serial communication. The Raspberry Pi ran a ROS node subscribing to the /joy topic, reading button messages from a connected joystick. The program ensured actions were performed only when either button 1 or 3 (with indices 0 and 2) was pressed exclusively. The respective logic value was published to the logic_message topic and sent to the Arduino over the serial port at a baud rate of 115200, which would cause the actuation. The joystick was connected to a PC, and messages were published to the /joy topic using a previously configured launch file. This setup combined with remote controlled navigation enabled tele-operation of a fully operational material handling robot.

4.3.6 Docking in Gazebo

For this, a tag is placed in front of the table in a simulated workspace using gazebo. The task is for the robot to detect this tag using OpenCV and to achieve this it will rotate until detection occurs. Either through RviZ interface or via a script, the robot is made to navigate to a location near the racks which contain the desired products. Using a script, it is necessary to provide this location in terms of position (x,y) and orientation (w). The orientation can be arbitrary as the robot has to rotate after-all as will be shown later. A simple code snippet shown below will do:

goal_pose = PoseStamped()
goal_pose.header.frame_id = 'map'
goal_pose.header.stamp = navigator.ge_clock().now().to_msg()
goal_pose.pose.position.x = x
goal_pose.pose.position.y = y goal_pose.pose.orientation.w = w

Similarly, the robot will be prompted to navigate to said location via goToPose(goal_pose) method. When the goal succeeds, the spinning operation commences. This is to ensure the robot is correctly aligned to the rack that needs to be transported. This occurs via the following snippet:

navigator.spin(spin_dist=0.26 time_allowance=10)
navigator.cancelTask()

The robot spins about 0.26 radians and this process should start within 10ms as stated by time allowance parameter. Then the task is cancelled and run again until and unless the tag is detected by the camera of the robot. To facilitate this, the navigation node subscribes to a topic which publishes either 1 or 0, depending on whether the tag was in the field of view or not.

self.subscription = self.create_subscription(Int16, 'binary', self.navigation, 10)
num = msg.data

Here, a /binary topic is subscribed to and the data, either '1' or '0', is stored by variable num.

This is what the tag looks like as shown in Fig. 11 from the camera positioned in the robot at gazebo. Here, the coordinate of the tag is being published, which is done by the node subscriber.py. Likewise, with the help of map_aruco.py, the coordinate to the above tag with respect to map frame will be published as well. Now, to move forward, another node, nav_aruco.py subscribes to this topic. However, this node only runs when the topic it subscribes to publishes '1'. Then, as mentioned above, a function listens to the transform broadcast between map to aruco or "map_aruco".

transform = self.tf_buffer.lookup_transform("map", "aruco", rclpy.time.Time())

From this transform, it is possible to get the translation and rotational components of the aruco tag and to maneuver the robot towards the tag. The robot is made to navigate to a location near the tag by providing this location in terms of position (x,y) and orientation (w), which can be gotten by subscribing to the map_aruco transform. The same goal_pose snippet above can be used.

Finally, the robot will be prompted to navigate to said location via goToPose(goal_pose) method. In Fig. 12, the robot is seen going below the table and docking. The image on the right shows the cost map being developed in RviZ. The robot is near one of the legs and is about to dock inside. This concludes the docking process in gazebo, enabling the use on the real robot.

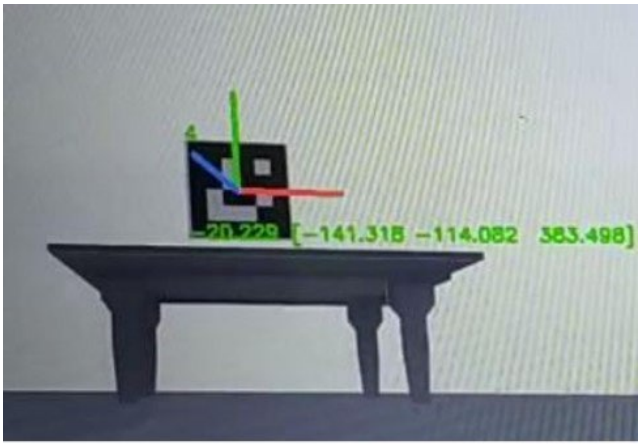

Fig. 11 *Aruco tag in gazebo showing the coordinates (x, y, z) of the tag with respect to robot's camera, along with ID number of tag and angle (z/x)*

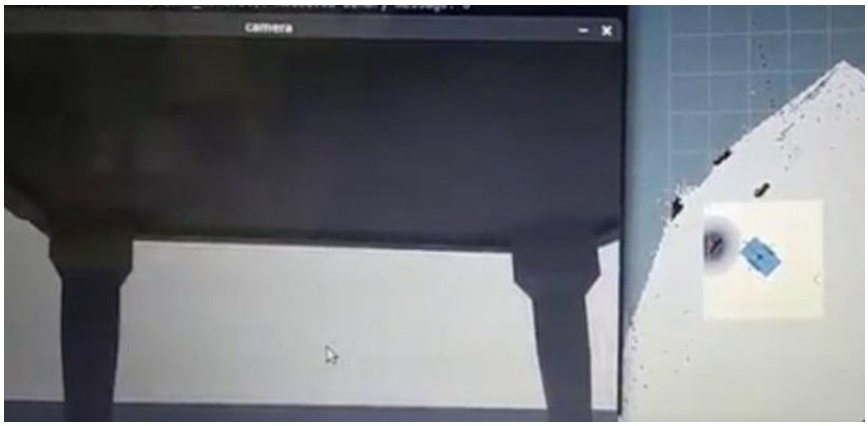

Fig. 12 *Robot moving inside the table in gazebo*

4.3.7 Complete Material Handling in Real Robot

Before implementing the previously described procedure, the camera was calibrated, and ArUco marker detection was verified using OpenCV libraries. A slight variation was observed between the readings obtained from the OpenCV library and the checked values, as shown in Fig. 13. An empirical relation was developed from 50 such readings and was used in the program to update the obtained ArUco transforms. Here, docking beneath the table or rack (as in Gazebo) is just the first phase. The second phase involves picking up the rack and moving it to the required destination.

This first phase is completed after obtaining the result "TaskResult.SUCCEEDED". Now, the nodes joy_to_msg and msg_to_arduino, are utilized. The former solely publishes to the logic message without conducting any serial communication. Meanwhile, the msg_to_arduino node runs on Raspberry Pi subscribing to this message and performing the corresponding serial communication.

To actuate the actuators, a logic message of 0 is sent. Additionally, a TimeAction of 12 seconds is initiated to pause the program. After this pause, the final goal pose is established using the goToPose function, setting the desired location to (x, y, w) on the map. Once the robot reaches its loading destination successfully, the logic message is set to "1" to lower the payload.

This process represents the full functionality of the Autonomous Mobile Robot (AMR) in material handling, ranging from carrying an entire table or rack to autonomous navigation with active obstacle avoidance, as demonstrated in the workshop shown in Fig. 14.

The entire AMR's autonomous material handling system is summarized in Fig. 15. The AMR can avoid obstacles using the cost maps and planner of Nav2 and maneuver through tight passages of approximately 900 millimeters. It performs lifting operations smoothly and can reach the designated location on the map with precision within centimeters. However, the docking process could be further refined, as the ArUco marker transforms vary occasionally.

Fig. 13 *Measuring the distance from camera to ArUco tag to compare with the value obtained by using OpenCV library*

Fig. 14 *AMR final test environment inside the workshop of Robotics Club, Pulchowk Campus with robot in its map's origin position and table arbitrarily placed at the other end*

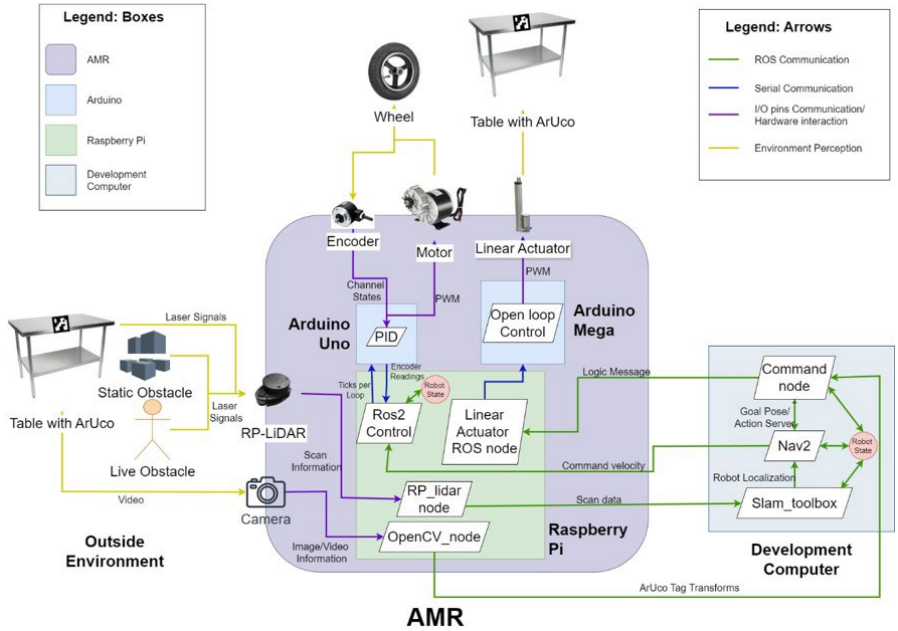

Fig. 15 *Entire AMR's autonomous material handling system*

The RP-LiDAR collects laser scan data, which is read by its node and published on the Scan topic, subscribed by SLAM_toolbox and Nav2. SLAM_toolbox manages localization, while Nav2 handles planning and navigation to the final target, avoiding both live and stable obstacles. Nav2 receives goal poses and action commands from the Command Node. The Command Node obtains ArUco marker transformations from OpenCV_node (which processes camera image data), converts them into rack locations on the map, and forwards the final goal pose to Nav2. Nav2 then computes the required command velocity and passes it through Ros2_control, which ensures the robot achieves the desired velocity by computing and sending PID ticks per second to an Arduino. The Arduino maintains the requested velocity via a PID loop, driving the wheels. Encoder data is used in the PID loop and by Ros2_control for the robot state publisher. Once Nav2 confirms the goal attainment to the Command Node, it signals the Linear Actuator node to lift the load via open-loop control on an Arduino Mega. After lifting, a new destination is set, and the robot navigates to drop the rack.

4.3.8 Performance Assessment

The autonomous mobile robot was tested in a dynamic indoor environment to evaluate the performance and robustness of the proposed controller, which integrates PID velocity regulation with LiDAR-based SLAM and ROS2 Nav2 path planning. Fig. 16 shows a sequence of frames from the test video (timestamp T8–T12), where a person enters the robot's path during navigation.

At T8–T10, the robot detects the obstacle using its LiDAR scan and begins to decelerate. Then, the robot performs a cautious counterclockwise yaw adjustment, interpreting the obstacle as dynamic. It instantly generates a new trajectory around the obstacle. By T12, the robot completes a right turn successfully navigating through the obstacle.

.

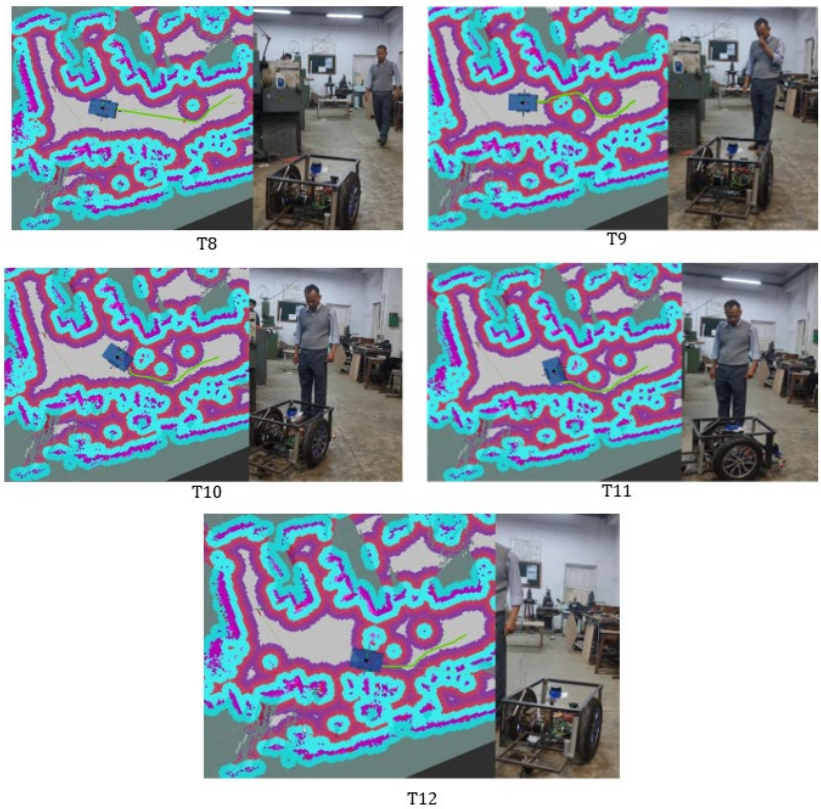

Fig. 16 *Frame timestamps*

The **performance assessment** of the autonomous mobile robot demonstrated reliable trajectory, tracking, speed control and turning, and stable localization.

1. Trajectory Tracking: The global path generated by Nav2 was followed without visible overshoot or significant deviation. Smooth movement and a clean trajectory was observed even in narrow corners and passages, which suggests that the PID controller was effectively tuned to balance responsiveness with stability as shown in Fig. 17.

2. Speed Control and Turning: Deceleration was observed before sharp turns and smooth acceleration after completion of turn. No abrupt jerking or oscillation was observed during transitions, indicating stable proportional and derivative terms in the PID loop. The velocity commands appeared to be well-synchronized with feedback from the SLAM system.

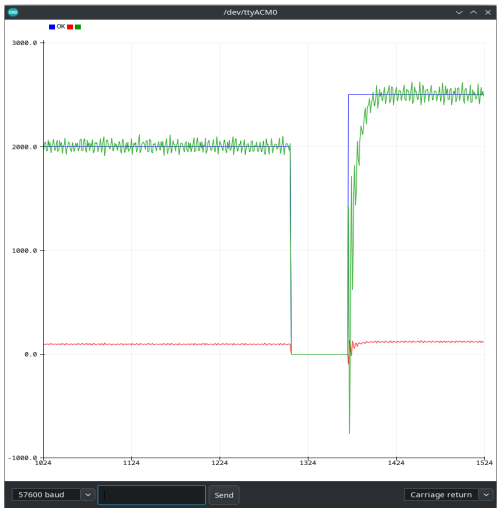

Fig. 17 *Tuned PID response:* K_p*=0.75,* K_i*=0.0001,* K_d*=4and* K_o*=50*

3. Localization Support from LiDAR: A confident trajectory was maintained by the robot even in cluttered or visually repetitive areas. No noticeable "drift" or hesitation was observed indicating that LiDAR-based pose estimation from SLAM remained consistent throughout the navigation. As a result, precise corrections were made possible by the controller, and a clean trajectory was maintained.

The **robustness evaluation** of the autonomous mobile robot highlighted its ability to navigate effectively in semi-constrained indoor environments while maintaining stable performance under varying conditions.

1. Environmental Interaction: Navigation was carried out through semi-constrained indoor spaces with furniture and obstacles. Despite proximity to objects and tight turns, no collision-prone behavior or frequent replanning was exhibited. This was interpreted as evidence that the local planner and the controller were robust under moderate real-world constraints.

2. Sensor Reliability: Parts of the LiDAR's field of view were temporarily occluded (e.g., due to furniture or nearby objects). Nonetheless, uninterrupted and stable navigation was observed. Such behavior was considered indicative of fault-tolerant operation within the perception-control loop.

3. Stability During Plan Adjustments: Smooth transitions into new trajectories were observed when global paths were updated or mid-path adjustments were triggered, potentially due to dynamic obstacles or changes in the map. When a dynamic obstacle appeared in close proximity along the planned trajectory, rapid deceleration was initiated, and the robot was brought to a pause. Reorientation was performed through controlled in-place rotations, and navigation was resumed without collisions. This response confirmed that robustness was maintained by the controller during reactive maneuvers and recovery scenarios.

4.4 Results

The implementation and testing of the AMR, developed through the systematic integration of transportation control, autonomous navigation, and material handling functionality, yielded several key observations. The most significant improvement of this AMR over conventional material handling robots is its live obstacle avoidance capability. Moreover, the robot's localization, achieved through LiDAR, SLAM, and odometry-based corrections, ensured robust positioning, which is crucial in dynamic environments for stable and accurate trajectory planning. The integration of computer vision with ArUco tags enabled the detection of payloads for transport, offering a promising and relatively unexplored approach in material handling. The successful combination of these technologies, which relied on basic encoder-based odometry without complex sensor fusion (IMUs and GPS), demonstrates the potential for enhanced autonomous material handling. This paves the way for further optimization in accuracy, efficiency, and scalability of AMRs in industrial applications.

5. Conclusion

The development of an AMR equipped with SLAM, Nav2 navigation, and CV marks a significant advancement in agile and efficient manufacturing processes. By integrating ROS2 tools and packages—including ROS2 Control, SLAM, Nav2, and OpenCV, the AMR was able to autonomously navigate, map its environment, and handle materials with precision. SLAM and Nav2 facilitated accurate mapping, localization, and obstacle avoidance, while computer vision enhanced the robot's material handling capabilities, enabling effective payload identification and manipulation.

This paper addresses the need for flexible automation in rapidly evolving industries, tackling challenges such as hardware limitations, debugging issues, and calibration discrepancies. Key achievements included chassis design and fabrication, simulation in Gazebo and RViz, and the transition from teleoperation to autonomous navigation and material handling. Each phase contributed to the overall success of the project, showcasing the potential of autonomous robots in modern manufacturing. Future research could further refine and optimize the AMR's functionality, focusing on advanced control algorithms and additional sensor integrations.

Acknowledgement

We express our gratitude to the Department of Mechanical and Aerospace Engineering, Pulchowk Campus for generously providing the essential laboratory facilities.

Conflict of Interest

The research provided in this paper was not impacted by any personal relationships or competing financial interests, as confirmed by the authors

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Prince Panta, Nirmal Prasad Panta, Pawan Shrestha, Saki Basnet, Surya Prasad Adhikari;* ***engineering design and development:*** *Prince Panta, Nirmal Prasad Panta, Pawan Shrestha, Saki Basnet;* ***testing and analysis:*** *Prince Panta, Nirmal Prasad Panta, Pawan Shrestha, Saki Basnet;* ***draft manuscript preparation:*** *Prince Panta, Nirmal Prasad Panta, Pawan Shrestha, Saki Basnet, Surya Prasad Adhikari. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Amir Hormozi. Agile manufacturing: The next logical step. Benchmarking: An International Journal, 8:132–143, 05 2001.

[2] Josh Wardini. 27+ astonishing robotics industry statistics you should know in 2023, January 2023.

[3] M. Teulieres, J. Tilley, L. Bolz, P. M. Ludwig-Dehm, and S. Wägner. Industrial robotics: Insights into the sector's future growth dynamics. Technical report, McKinsey Company, 2019.

[4] Association for Advancing Automation (A3). Robot orders increase 67

[5] Matthew Urwin. 21 robotics companies and startups on the forefront of innovation, May 25 2023.

[6] Günter Ullrich. The History of Automated Guided Vehicle Systems, pages 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[7] W. Grzechca. Manufacturing in flow shop and assembly line structure. International Journal of Materials, Mechanics and Manufacturing, 4:25–30, 01 2015.

[8] MengChu Zhou and Kurapati Venkatesh. Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach. World Scientific, 1999.

[9] Gabriel Fedorko, Stanislav Honus, and Roland Salai. Comparison of the traditional and autonomous agv systems. In MATEC Web of Conferences, volume 134, page 00013. EDP Sciences, 2017.

[10] Liam Lynch, Thomas Newe, John Clifford, Joseph Coleman, Joseph Walsh, and Daniel Toal. Automated ground vehicle (agv) and sensor technologies-a review. In 2018 12th International Conference on Sensing Technology (ICST), pages 347–352. IEEE, 2018.

[11] Lekkala, Kiran & Mittal, Vinay. (2014). PID controlled 2D precision robot. 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies, ICCICCT 2014. 1141-1145. 10.1109/ICCICCT.2014.6993133

[12] Li, Y., & Ibanez-Guzman, J. (2020). Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. IEEE Signal Processing Magazine, 37(4), 50-61.

[13] Liam Lynch, Fintan McGuinness, John Clifford, Muzaffar Rao, Joseph Walsh, Daniel Toal, and Thomas Newe. Integration of autonomous intelligent vehicles into manufacturing environments: Challenges. Procedia Manufacturing, 38:1683–1690, 2019.

[14] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. IEEE Intelligent Transportation Systems Magazine, 2(4):31–43, 2010.

[15] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. Journal of Open Source Software, 6(61):2783, 2021.

[16] R.C. Arkin and R.R. Murphy. Autonomous navigation in a manufacturing environment. IEEE Transactions on Robotics and Automation, 6(4):445–454, 1990.

[17] MAO Jiandong, NIU Wenqi, WANG Hongyan, Bai ZHANG, CAO Zhen, GUO Zhen, Hu ZhAO, ZHOU Chunyan, and GONG Xin. A agricultural spraying and fertilization robot based on visual navigation. In 2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA), pages 586–591. IEEE, 2020.

[18] Tony Hague, John A Marchant, and ND Tillett. Autonomous robot navigation for precision horticulture. In Proceedings of International Conference on Robotics and Automation, volume 3, pages 1880–1885. IEEE, 1997.

[19] Chao Wang, Andrey V. Savkin, Ray Clout, and Hung T. Nguyen. An intelligent robotic hospital bed for safe transportation of critical neurosurgery patients along crowded hospital corridors. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 23(5):744–754, 2015.

[20] Rhaian J. F. Barros, Jorge L. P. Silva Filho, João V. S. Neto, and Tiago P. Nascimento. An open-design warehouse mobile robot. In 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE), pages 1–6, 2020.

[21] A. Marut, K. Wojtowicz and K. Falkowski, "ArUco markers pose estimation in UAV landing aid system," 2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace), Turin, Italy, 2019, pp. 261-266, doi: 10.1109/MetroAeroSpace.2019.8869572.

[22] D. Avola, L. Cinque, G. L. Foresti, C. Mercuri, and D. Pannone, 'A practical framework for the development of augmented reality applications by using ArUco markers', in International Conference on Pattern Recognition Applications and Methods, 2016, vol. 2, pp. 645–654.

[23] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics, vol. 7, no. 66, May 2022.

[24] Steven Macenski, Francisco Martin, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[25] Itseez. Open-source computer vision library. https://github.com/itseez/opencv, 2015.

[26] AV Chavan and JL Minase. Design of a differential drive mobile robot platform or use in constrained environments. International Journal of Innovations in Engineering Research and Technology, 2(6):1–10, 2015.