Warehouse Robot guidance System

Mechatronic Project 478
Final Report

Author: Hauke Aakmann-Visher
24660051

Supervisor: Mr. Trinchero

October 24, 2024

Department of Mechanical and Mechatronic Engineering
Stellenbosch University
Private Bag X1, Matieland 7602, South Africa.

Plagiarism declaration

I have read and understand the Stellenbosch University Policy on Plagiarism and the definitions of plagiarism and self-plagiarism contained in the Policy [Plagiarism: The use of the ideas or material of others without acknowledgement, or the re-use of one's own previously evaluated or published material without acknowledgement or indication thereof (self-plagiarism or text-recycling)].

I also understand that direct translations are plagiarism, unless accompanied by an appropriate acknowledgement of the source. I also know that verbatim copy that has not been explicitly indicated as such, is plagiarism.

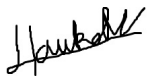
I know that plagiarism is a punishable offence and may be referred to the University's Central Disciplinary Committee (CDC) who has the authority to expel me for such an offence.

I know that plagiarism is harmful for the academic environment and that it has a negative impact on any profession.

Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully (acknowledged); further, all verbatim copies have been expressly indicated as such (e.g. through quotation marks) and the sources are cited fully.

I declare that, except where a source has been cited, the work contained in this assignment is my own work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment. I declare that have not allowed, and will not allow, anyone to use my work (in paper, graphics, electronic, verbal or any other format) with the intention of passing it off as his/her own work.

I know that a mark of zero may be awarded to assignments with plagiarism and also that no opportunity be given to submit an improved assignment.

Signature: 

Name: Hauke Aakmann-Visher Student no: 24660051

Date: October 24, 2024

AI Use Declaration

1. I am convinced and can support my claim that my assessment product is an indication of my own learning, knowledge, skills, and understanding.

2. Where I have used AI tools for enhancing my own creation of ideas and words, I acknowledge that I have to declare it.

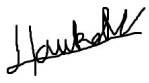
3. Where I have used AI tools for generating new ideas, words, code, image-prompts, or other AI image-generating tools, or structure and even presentations (or other AI tools that can be used as assistants to the knowledge building and representing process), I have declared and documented the use of such tools and I am prepared to talk about the process I used and what it contributed to my learning and insights.

4. I am aware that the lecturer can ask me to demonstrate my learning, for example through explaining the choices I made in terms of approach, content used, literature selected, conclusions drawn, etc. through an additional assessment like an oral (for example).

5. I understand that if I am not able to agree to the above points, there is a chance that my academic behavior will be deemed unethical and might lead to a disciplinary case being brought against me on the grounds of cheating or plagiarism and that the standard procedures for such behavior will be followed.

6. As per the Disciplinary Code of SU (par. 10.2.1 and 10.2.2) I understand that I take responsibility for the integrity of my work, which includes the obligation to ask for clarification from an academic member of staff if I am unsure of anything, and that I strictly adhered to all instructions received in the course of the academic assessment by relevant and authorized staff (whether the instruction is in oral or written format).

7. I understand that when I am not able to document and declare my use of AI tools, this behavior will be deemed as cheating in examinations and assessments (Disciplinary Code 1.1 c.) as I referred to "unauthorized notes, books, electronic devices or other reference material."

Signature: 

Name: Hauke Aakmann-Visher Student no: 24660051

Date: October 24, 2024

Executive Summary

Title of Project
Warehouse Robot Guidance System
Objectives
Design, implement, and evaluate a robot guidance system for optimizing pathfinding, obstacle avoidance, and WMS integration.
Current Practice and Limitations
Current AGVs rely on pre-installed infrastructure and struggle in dynamic environments, with limited navigation accuracy and reactive obstacle avoidance.
Innovations in This Project
This project introduced advanced pathfinding (A*), robust obstacle avoidance, grid-based localization, and a web-based interface for real-time updates. Motor compensation techniques were also developed to address hardware issues.
Potential Impact
This system enhances warehouse efficiency by reducing travel time, improving navigation, and preventing collisions. It is scalable for more robots and larger environments.
Risks and Success Factors
Challenges included sensor integration, motor control, and algorithm complexity, which were mitigated by testing. Key success factors were seamless communication and overcoming hardware limitations.
Contributions of Other Students
This is a novel project; no work from previous students was used. All aspects were developed by the author.
Continuation Aspects
The project is scalable and modular, allowing for future development of advanced algorithms, such as SLAM, and features like load carrying and package scanning.
Arrangements for Continuation
Comprehensive documentation, codebases, and test results are provided to support future work, ensuring new features can be added without redesigning the system.

Self-Assessment of ECSA Graduate Attributes

GA 1: Problem Solving

Self-assessment: The project addressed hardware limitations with structured problem-solving, from identifying issues (Section 4.4) to implementing A* pathfinding (Subsection 4.3.3) and motor compensation (Subsection 4.4.3). Creative solutions were applied to overcome challenges.

GA 2: Application of Scientific and Engineering Knowledge

Self-assessment: The project effectively applied mathematical principles like A* for pathfinding (Subsection 4.3.3), and engineering concepts such as sensor integration and motor control to solve robotics problems (Chapter 3).

GA 3: Engineering Design

Self-assessment: The navigation system, pathfinding, and web interface were all designed by the author (Subsection 4.3.3). Motor driver issues (Section 4.4) were resolved with software-based solutions like motor compensation, demonstrating flexibility.

GA 4: Engineering Methods, Skills, and Tools, Including IT

Self-assessment: Pathfinding algorithms and IT tools such as Python, JavaScript, and HTML (all self-learned) were used (Section 4.3). AI tools assisted the process but did not replace personal work.

GA 5: Professional and Technical Communication

Self-assessment: The project's design choices and solutions were clearly communicated in the report and through presentations.

GA 6: Individual, Team, and Multidisciplinary Working

Self-assessment: The project was completed individually with supervisor collaboration. The ability to meet deadlines showcased effective independent work.

GA 7: Independent Learning Ability

Self-assessment: Independent learning was crucial, as the author self-learned Python, JavaScript, and HTML, attended a Python workshop, and applied these skills to the project.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Mr. Trinchero, for his valuable guidance, support, and assistance throughout the entire project. His advice and feedback were crucial to the successful completion of this work.

I would also like to thank my family for their financial and moral support during this project. Their encouragement helped me stay motivated and focused throughout the process.

Lastly, I extend my thanks to my friends for their help in proofreading and for their constant moral support, which made a significant difference during challenging times.

Table of contents

List of figures

List of tables

List of Symbols

Variables

$f(n)$	Cost function for A* algorithm	[ ]
$g(n)$	Cost from start node to node n	[ ]
$h(n)$	Heuristic estimate of cost from node n to goal	[ ]
F	Path cost in A* algorithm	[ ]
G	Movement cost from start point	[ ]
H	Estimated movement cost to target	[ ]
$u(t)$	Control output	[ ]
$e(t)$	Error	[ ]
K_p	Proportional gain	[ ]
K_i	Integral gain	[ ]
K_d	Derivative gain	[ ]

Abbreviations

AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robot
API	Application Programming Interface
BFS	Best-First Search
GPS	Global Positioning System
LiDAR	Light Detection and Ranging
PC	Personal Computer
PID	Proportional-Integral-Derivative
RFID	Radio Frequency Identification
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
URL	Uniform Resource Locator
WMS	Warehouse Management System

Chapter 1
Introduction

1.1 Background

The fast growth of e-commerce has increased the demand for automation in warehouse environments. Current robotic systems face major challenges in navigation and collision avoidance which calls for new solutions. This project focuses on designing and building a reliable guidance system for a model warehouse robot to address these issues.

1.2 Problem Statement, Aims, and Objectives

Problem Statement: Existing warehouse robots often rely on pre-installed infrastructure and struggle with dynamic environments, limiting their flexibility and efficiency. There is a need for a more adaptable guidance system that can dynamically navigate and avoid obstacles without relying on fixed paths.

Aims and Objectives: The primary aim of this project is to design and test a reliable robot guidance system that optimizes pathfinding and obstacle avoidance in a warehouse setting. The key objectives are:

- Developing a scalable navigation algorithm.
- Integrating effective obstacle detection and avoidance systems.

The system will be tested on a small-scale robot to simulate real-world warehouse conditions.

1.3 Motivation

Improving the navigation abilities of warehouse robots addresses several key issues, such as reducing travel time and distance, preventing collisions, and increasing workflow efficiency. By successfully completing this project, the system will contribute to boosting warehouse productivity and meeting the rising demands of logistics and storage industries.

1.4 AI Tools Usage Declaration

During the development and writing of this report, several AI tools were used to assist with different aspects of the project. The specific tools and their uses are outlined below:

- **ChatGPT:** ChatGPT was used in different stages of the project:
  - **Coding and Troubleshooting:** ChatGPT was used to suggest coding ideas, help debug errors, and assist with troubleshooting code.
  - **Literature Review:** ChatGPT helped in finding relevant academic papers, and highlighting important information to avoid needing to read the entire texts.
  - **Writing Assistance:** ChatGPT provided outlines to structure ideas and assisted with making the writing clearer. It was also used for grammar and spelling checks.
  - **Brainstorming:** ChatGPT was used to brainstorm new ideas when challenges or obstacles were encountered during the project.
  - **Data Visualization:** ChatGPT was used to help create graphs from manually input data, making it easier to present data visually.
- **Quillbot:** Quillbot was used alongside ChatGPT for additional grammar and spelling checks, improving the overall clarity of the report.

These AI tools were used to assist and enhance the work. They did not replace personal effort but supported the author's learning and productivity. Full responsibility for the project and its results remains with the author.

Chapter 2
Literature review

This literature review will explore the main technologies and methods that enable autonomous navigation, localization, and multi-sensor fusion in warehouse robots. It will provide a thorough overview of the current state of research and development in this rapidly evolving field.

2.1 Objectives

The primary objectives of this literature review are to:

- Provide an overview of navigation and localization technologies in autonomous warehouse robots.
- Discuss path planning algorithms, including A*, Dijkstra, and Best-First Search, and their applications in warehouse environments.
- Explore multi-sensor fusion methods and their role in improving perception and decision-making capabilities.
- Review the software systems and frameworks used to control and optimize autonomous warehouse robots.
- Highlight the integration of autonomous robots with Warehouse Management Systems (WMS) for improved operational efficiency.

2.2 Background

The rapid expansion of e-commerce has had a large impact on the development of warehouse robotics. As e-commerce keeps growing, with sales exceeding $1 trillion in 2012 and continuing to increase at an annual rate of 15% (Bogue, 2016), the demand for more efficient and cost-effective warehouse operations has also risen. This has pushed advancements in autonomous warehouse robots, which are now becoming a vital part of modern logistics and supply chain management.

Autonomous Mobile Robots (AMRs) have become a key solution to addressing the challenges of inefficiencies and high labour costs that are common in traditional warehouse setups. Unlike Automated Guided Vehicles (AGVs), which depend on pre-installed infrastructure for navigation, AMRs can make independent decisions and navigate changing environments without needing predefined paths (Keith and La, 2024). This flexibility makes them ideal for the constantly shifting conditions found in today's warehouses.

AMRs are equipped with various advanced technologies, such as path planning and localization, to allow them to navigate and carry out tasks on their own. Algorithms like A*, Dijkstra, and grid-based methods help these robots figure out the best routes within the warehouse (Yang *et al.*, 2020; Huang *et al.*,

2022). Techniques such as SLAM (Simultaneous Localization and Mapping), GPS, and sensor fusion enable the robots to accurately determine their positions and move effectively in real-time (Alatise and Hancke, 2020; Lu *et al.*, 2021).

The use of multi-sensor fusion is especially important for improving the robots' ability to perceive their environment and make informed decisions. By merging data from LiDAR, ultrasonic sensors, cameras, and other devices, multi-sensor fusion gives the robots a better understanding of their surroundings. This, in turn, makes their navigation and obstacle avoidance more accurate and reliable (Qu *et al.*, 2021; Ayala and Mohd, 2021).

Beyond hardware, there has also been significant progress in software development for controlling and optimizing the performance of autonomous warehouse robots. These software systems oversee everything from real-time control and coordination to integration with Warehouse Management Systems (WMS), boosting overall warehouse efficiency and productivity (Hercik *et al.*, 2022).

2.3 Navigation and Localization Technologies

Overview For autonomous robots to work efficiently in a warehouse setting, navigation and localization technologies are crucial. These systems allow the robots to figure out their location, plan their routes, and move accurately throughout the environment.

2.3.1 Path Planning

2.3.1.1 Algorithms and Methods

Autonomous robots rely on various path planning algorithms to determine the best routes. Some commonly used algorithms include A*, Dijkstra, and grid-based methods, each offering different advantages depending on the scenario.

The **A* algorithm** is a popular choice due to its combination of heuristic search and traditional methods, which helps in finding the shortest path efficiently. The A* algorithm operates by evaluating the cost function, as shown in Equation 2.1:

$$f(n) = g(n) + h(n) \tag{2.1}$$

$$F = G + H \tag{2.2}$$

where $g(n)$ is the cost from the start node to the node n, and $h(n)$ is the heuristic estimate of the cost from n to the goal (Huang *et al.*, 2022). This algorithm guarantees finding the shortest path if the heuristic is admissible (never overestimates the true cost).

The A* algorithm works as follows:

- **Initialization:** Begin at the starting point A and add it to an "open list" of nodes to be considered. The open list contains nodes that need to be checked.

- **Path Scoring:** Evaluate each node using Equation 2.1, where G is the movement cost from the starting point to the current node, and H is the estimated movement cost from the current node to the target.
- **Search Process:** Choose the node with the lowest F score from the open list. Move this node to the "closed list" (nodes that have been checked).
- **Neighbour Evaluation:** For each neighbouring node, if it is not in the closed list and is not an obstacle, calculate its F score. If the node is already in the open list but the new path is better (lower G score), update the node's parent to the current node and recalculate its F and G scores.
- **Path Completion:** Repeat the process until the target node is added to the closed list, indicating that the shortest path has been found.

The **Dijkstra's algorithm** is another widely used method that finds the shortest path between nodes in a graph. It works by iteratively selecting the node with the smallest tentative distance, updating the distances of its neighbours, and marking it as visited until the destination is reached. This algorithm does not use a heuristic and explores all possible paths, ensuring optimality (Permana *et al.*, 2018).

The **Best-First Search (BFS)** algorithm evaluates the cost from any node to the target point, prioritizing the most promising nodes first. However, it does not guarantee finding the shortest path (Permana *et al.*, 2018).

2.3.1.2 Performance Comparison

Each path planning algorithm comes with its own set of strengths and weaknesses. Dijkstra's algorithm is known for guarantee of finding the shortest path and is relatively simple to implement since it doesn't require a heuristic function. However, it can be less efficient in terms of data usage, as it explores every possible path without direction, leading to higher computational demands.

On the other hand, the A* algorithm tends to be more efficient. It uses a heuristic function to guide it's search, allowing it to focus on the most promising routes and reducing the number of nodes it needs to explore. This makes A* more suitable for large, complex environments where performance or computational demands are the key concerns.

The Best-First Search algorithm can be faster than both Dijkstra's and A* in certain situations, as it prioritizes the exploration of the most promising nodes first. However, a major drawback is that it doesn't always guarantee the shortest path, which can be a serious limitation for applications where finding the shortest route is critical.

2.3.2 Localization

Localization techniques are crucial for autonomous robots to accurately understand their position within a warehouse. Some of the key methods used for

this purpose include odometry, GPS, LiDAR, sensor fusion and QR-code-based localisation.

Odometry is one of the simplest and most widely used methods for estimating a robot's position. It works by collecting data from motion sensors to calculate how far the robot has travelled and how much it has turned. However, it inevitably suffer from cumulative errors, especially if there is wheel slippage or minor inaccuracies in measuring distances and angles (Cho *et al.*, 2013).

GPS (Global Positioning System) offers absolute positioning by receiving signals from satellites. While it's very effective for outdoor environments, GPS tends to be much less reliable indoors due to issues like signal loss and interference. As a result, it's not commonly used on its own for indoor warehouse localisation (Alatise and Hancke, 2020).

LiDAR (Light Detection and Ranging) uses lasers to measure the distance to objects around the robot, allowing it to build a detailed map of its surroundings. LiDAR is very accurate and works well under different lighting conditions, making it ideal for use in structured environments like warehouses. However, LiDAR systems are expensive and may struggle when encountering transparent or reflective surfaces (Alatise and Hancke, 2020).

Sensor Fusion combines data from multiple sensors to create a more accurate and reliable estimate of the robot's position. Techniques such as the Kalman filter, extended Kalman filter (EKF), and particle filter are often used to merge information from sources like odometry, GPS, and LiDAR. This allows the system to compensate for the limitations of individual sensors by drawing on their combined strengths (Cho *et al.*, 2013; Alatise and Hancke, 2020).

QR-code Based Localisation is a practical method for localising robots in indoor environments. QR-codes are placed at strategic points, usually on the ceiling, and the robot uses these as reference points. A visual sensor (camera device) mounted on the robot can detect these QR-codes, processing the information to help calculate the robot's approximate location. The pixel coordinates of the QR-code within the image help to refine the robot's positioning, while the QR-code's orientation in the image is used to determine the robot's direction (Lee *et al.*, 2015). As shown in Figure 2.1, the block diagram illustrates the proposed QR-code based localization system architecture.

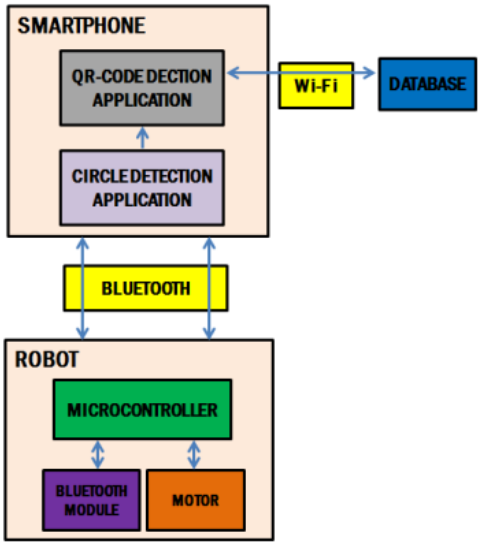


Figure 2.1: Block diagram illustrating the proposed QR-code based localization system architecture (Lee *et al.*, 2015).

Line Tracking is a common technique used by autonomous robots for indoor navigation. It relies on sensors, such as photo interrupt sensors, to follow lines marked on the floor. These sensors detect the lines by emitting a light beam, which is interrupted when a line is encountered, allowing the robot to adjust its direction accordingly. This method is both cost-effective and highly suitable for environments with fixed routes, like warehouses. Research by (Zhao *et al.*, 2022) demonstrated how combining line tracking with SLAM algorithms significantly improved the accuracy and efficiency of navigation. (Liu *et al.*, 2024) also highlighted the role of multi-sensor fusion in improving the reliability of navigation, emphasising the importance of photo interrupt sensors.

Line Tracking and Counting for Localisation is an approach that integrates line tracking with the counting of lines crossed by the robot to estimate its position within a grid. This method is especially useful in environments where GPS signals are unreliable, such as warehouses. Each time the robot crosses a line, its position is updated relative to a reference point, providing a practical means of localisation. When this technique is combined with other methods like odometry, it enhances the precision of navigation. (Farkh and Aljaloud, 2023) described how computer vision and PID controllers can improve line tracking, while line counting can further enhance localisation in complex indoor environments.

2.4 Multi-Sensor Fusion in Warehouse Robotics

Multi-sensor fusion combines the data collected from multiple sensor modules to improve navigation and decision making strategies. Although the guidance

system designed in this project will only use simple sensors, like infrared and/or ultrasonic distance sensors for obstacle detection, it's important to discuss the fact that advanced warehouse robotic systems use sensor fusion for increased accuracy.

This complexity is added to a system by fusing data from different sensors, which reduces noise and uncertainties that individual sensors might produce. For example, when a single sensor's readings are affected by interference or inaccuracies, the combined data from other sensors or sensor systems compensate for those errors, ensuring more reliable readings and smarter decisions are being made.

2.4.1 Most popular Sensor Fusion Techniques

Kalman Filter: The Kalman Filter is often used in robotics to combine sensor data collected over time, helping to create more accurate estimates of the robot's position and movement. It works by making predictions about the robot's current state and then updating those predictions using new sensor data. This process helps to smooth out any errors or noise in the measurements (Alatise and Hancke, 2020). Although this method is highly effective for reducing sensor noise and improving accuracy, it goes beyond the simpler needs of the system developed in this project, which relies on basic sensors.

Particle Filter: Particle filters handle non-linear sensor data by representing the robot's possible states using a set of particles. Each particle acts as a potential estimate of the robot's position, and as the robot moves, the filter adjusts the particles based on how closely they match the real environment. Over time, particles that better reflect the robot's true state are weighted more heavily, while less accurate particles are discarded (Keith and La, 2024). This technique is useful in scenarios where sensor data is noisy or uncertain, allowing for more robust state estimation. However, it is more complex than the basic techniques required for this project.

2.5 Software Systems and Frameworks

2.5.1 Control Systems

Control systems play a crucial role in autonomous robots, as they manage the robot's movements and ensure accurate navigation and task handling. These systems often rely on PID controllers, but more advanced control techniques can also be used for greater precision. For this project though, only PID control is relevant as the platform for the guidance system makes use of PID Control.

PID Controllers (Proportional-Integral-Derivative) are the most commonly used systems due to their simplicity and effectiveness in controlling continuous

processes. These controllers continuously calculate the error values as the difference between a desired setpoint and a measured process variable, by trying to minimize this error with adjustments to the process control inputs (Farkh and Aljaloud, 2023). Equation 2.3 displays the basic PID control equation.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.3}$$

where $u(t)$ is the control output, $e(t)$ is the error, K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain.

Advanced control techniques, such as **Model Predictive Control (MPC)**, are also employed in more complex applications. MPC uses a model of the system to predict the future behaviour and optimize the control inputs accordingly.

2.5.2 Integration with Warehouse Management Systems (WMS)

Integrating autonomous robots with Warehouse Management Systems (WMS) plays a vital role in streamlining warehouse operations. This connection enables real-time data exchange, helping to optimise processes like inventory management, order fulfilment, and overall efficiency across the warehouse (Grover and Ashraf, 2024; Mecalux, 2023).

- **Inventory Tracking:** Autonomous robots constantly update the WMS with the location and status of inventory items. This allows for precise tracking of stock levels and item locations, making it easier to manage inventory efficiently (TechBriefs, 2023).
- **Order Fulfilment:** The WMS assigns robots the task of picking and transporting items to specific locations, speeding up the order fulfilment process. This automation helps to reduce human errors and improve accuracy (Grover and Ashraf, 2024).
- **Data Analytics:** The data collected from robots provides valuable insights that help optimise the warehouse layout, improve task allocation, and identify bottlenecks. By analysing this data, warehouses can continuously improve their operations (Mecalux, 2023).
- **Real-Time Location Updates:** Robots send real-time updates on their location as they move through the warehouse. This ensures that the WMS can track their position accurately, helping with task coordination and monitoring (Grover and Ashraf, 2024).
- **Task Execution:** The WMS provides robots with detailed task lists, outlining specific actions they need to take to complete their jobs. This ensures that each robot follows a structured plan to complete tasks efficiently (TechBriefs, 2023).
- **Obstacle Detection and Recalculation:** If a robot encounters an obstacle, it can pause and send a request to the WMS for a new path.

This allows the system to adjust the robot's route dynamically in real time, ensuring smooth operations (Grover and Ashraf, 2024).

- **Position Verification:** Robots use sensors, such as QR code scanners, to confirm their location within the warehouse. This verification ensures that they stay on track and complete their tasks accurately (Mecalux, 2023).

2.5.3 Simulations and Software Used

Simulations are an essential part of developing and testing autonomous warehouse robots. They provide a virtual environment where researchers can model and assess robot behaviour before real-world deployment, saving both time and resources.

Researchers use a range of simulation platforms and software to achieve this:

- **Gazebo:** Gazebo is a powerful simulation tool that works well with ROS (Robot Operating System). It offers a realistic environment that includes physics simulation, sensor data, and 3D visualisation, making it ideal for testing robot algorithms and hardware designs (gaz, 2023).
- **V-REP (CoppeliaSim):** Formerly known as V-REP, CoppeliaSim is a flexible platform for simulating robotic systems. It supports integration with various programming languages and frameworks, making it suitable for diverse research needs (cop, 2023).
- **MATLAB/Simulink:** MATLAB and Simulink are widely used for modelling, simulation, and analysis of dynamic systems. These tools are particularly useful for developing and testing control algorithms in a simulated environment before applying them in real-world scenarios (mat, 2023).
- **Webots:** Webots is a professional robot simulation software that offers a rapid prototyping environment. It allows researchers to model, program, and simulate robots efficiently, making it ideal for testing robotic systems (web, 2023).
- **Python with Pygame:** Python, combined with the Pygame library, can be used to simulate a 2D warehouse floor. This setup is useful for testing pathfinding algorithms. Pygame provides a simple framework for graphical simulations, helping researchers visualise and refine their algorithms (pyg, 2023).

Chapter 3
Concept Development

3.1 Existing Platform

3.1.1 Description of the Existing Platform

The existing platform for the autonomous warehouse robot guidance system is the AlphaBot2 - Mobile Robot Development Platform. The AlphaBot is a versatile robotic development platform that is controlled by a Raspberry Pi Microcontroller. It includes the AlphaBot mainboard, mobile chassis, and various sensors and components necessary for movement and basic navigation. A picture of the AlphaBot2 is shown in Figure3.1 below, followed by a description of it's main components.

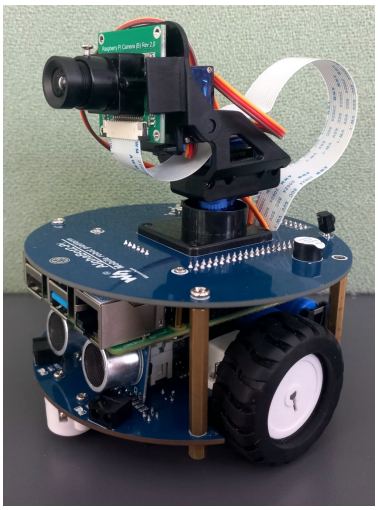


Figure 3.1: Alphabot 2 with the camera module.

- **AlphaBot mainboard:** The central processing unit of the platform, compatible with Raspberry Pi and Arduino.
- **Tracker Sensors:** Used for line tracking capabilities.
- **Infrared Proximity Sensors:** Enables obstacle detection and avoidance.
- **Two Motors with gearboxes:** Provides mobility with adjustable speed.
- **AlphaBot wheel:** The wheels for robot movement.
- **Omni-directional wheels:** Assists with balancing the robot and smoothen its movement.
- **IR remote controller:** Allows remote control operation.
- **Pi-camera:** Allows the Alphabot to stream its point of view.

3.1.2 Built-in functionalities

The AlphaBot platform, combined with a Raspberry Pi 4B, provides several built-in functionalities that are crucial for the development of the autonomous guidance system. These functionalities include:

- **Line Tracking:** Utilizing the tracker sensor for following predetermined paths within the warehouse.
- **Obstacle Avoidance:** Using infrared proximity sensors to detect and drive around or away from obstacles.
- **Basic Mobility:** Controlled via the motors and wheels, with support for omni-directional movement.
- **Remote Control:** Enabled by the IR remote controller, providing manual override capabilities.

3.1.3 Current Capabilities and Limitations

The current AlphaBot platform offers a robust foundation for the development of the warehouse robot guidance system. Its capabilities include:

- **Basic Navigation:** Basic line tracking and obstacle avoidance are supported.
- **Mobility:** The platform provides reliable movement with the ability to navigate in multiple directions.
- **Integration:** Compatibility with Raspberry Pi allows for flexible programming and expansion.

However, there are several limitations that need to be addressed to meet the project's requirements:

- **Advanced Navigation Algorithms:** While basic navigation is supported, advanced path planning and optimization algorithms need to be developed.
- **Enhanced Obstacle Detection:** The current proximity sensors are limited in range and precision; additional sensors or sensor fusion techniques may be required.
- **Intersection detection:** Although the robot is capable of following a line on the floor its incapable of detecting an intersection.
- **Integration with WMS:** Developing a system for real-time updates and interaction with the Warehouse Management System (WMS) via API (Application Programming Interface).

3.2 Requirements Analysis

3.2.1 Stakeholder Requirements

Table 3.1: Stakeholder Requirements

SR	Stakeholder	Description	Priority
SR1	End-user	User-Friendly Interface (Max 2 hours of training)	4
SR2	End-user	Cost Efficiency (Total cost $\leq$ R100000)	1
SR3	End-user	Reliability (99% reliability, downtime $\leq$ 1%)	2
SR4	End-user	Customization (Configurable within 24 hours)	3
SR5	End-user	Integration Capabilities (Comes with WMS)	6
SR6	End-user	Scalability (Support up to 50 robots simultaneously)	5

3.2.2 Engineering Requirements

Table 3.2: Engineering Requirements

ER	Description	Related SR
ER1	Navigation Accuracy (Margin of error within 5 cm)	SR1, SR3
ER2	Obstacle Detection and Avoidance (Response time $<$ 100 ms)	SR3
ER3	Path Planning Algorithm (Reduce travel time by 20%)	SR3
ER4	Speed (Up to 7 km/h)	SR3
ER5	Scalability (Handle up to 10000 m^2 warehouse)	SR6
ER6	Battery Life (Operate for 2 hours on a single charge)	SR3
ER7	Integration with WMS (Comes with WMS)	SR5
ER8	Environmental Robustness (Operate between -10°C to 40°C, up to 85% humidity)	SR3
ER9	Localization Accuracy (Within 3 cm)	SR1, SR3
ER10	Load Capacity (Carry loads up to 5 kg)	SR3
ER11	Communication Protocol (Utilize Wi-Fi and Bluetooth)	SR5
ER12	Real-Time Processing (Real-time navigation and obstacle avoidance calculations)	SR3
ER13	Sensor Fusion (Integrate data from multiple sensors)	SR3

3.2.3 Performance Requirements

Table 3.3: Performance Requirements

PR	Description	Target	Range	Unit
PR1	Navigation Accuracy	< 5	n/a	cm
PR2	Obstacle Detection Response Time	< 200	n/a	ms
PR3	Path Planning Efficiency	Reduce travel time by 20%	n/a	n/a
PR4	Maximum Speed	7	n/a	km/h
PR5	Scalability	10000	2000	m^2
PR6	Battery Life	2	0.5	hours
PR7	Integration with WMS	Real-time	n/a	n/a
PR8	Environmental Operating Range	-10 to 40	n/a	°C
PR9	Localization Accuracy	< 3	n/a	cm
PR10	Load Capacity	5	n/a	kg
PR11	Communication Protocol	Wi-Fi, Bluetooth	n/a	n/a
PR12	Real-Time Processing	Real-time	n/a	n/a
PR13	Sensor Fusion	Integrated data	n/a	n/a

3.3 Concept 1: Basic Navigation and Cost-Effective System

3.3.1 Overview

This concept focuses on creating a cost-effective warehouse robot system using essential components and basic algorithms for path planning and localisation while meeting the project's requirements.

3.3.2 Key Features

Table 3.4: Concept 1: Key Features

Feature	Details
Path Planning	A* algorithm for path planning, reducing the computational load on the Raspberry Pi. It includes modifications to avoid obstacles.
Localization	Line following combined with QR-code localisation to estimate position and reset the robot's position, minimising cumulative errors.

Obstacle Detection	Uses infrared sensors for close-range detection and ultrasonic sensors for better range, compensating for the limits of infrared.

3.3.3 Components

Table 3.5: Concept 1: Components

Component	**Details**
AlphaBot Platform	Includes basic sensors, motors, and chassis.
Raspberry Pi 4B	Handles navigation, runs algorithms, and connects with external components.
Infrared Sensors	Detects nearby obstacles.
Ultrasonic Sensor	Provides extra obstacle detection range.
QR-codes	Helps with loop-closure in localisation.
Camera	Reads QR-codes and aids video analysis.

3.3.4 Advantages and Disadvantages

Table 3.6: Concept 1: Advantages and Disadvantages

Advantages	**Disadvantages**
Low cost and minimal infrastructure changes.	Limited accuracy in navigation and localisation.
Simple to implement and maintain.	Not ideal for large or dynamic environments.
Suitable for smaller, grid-like warehouses.	Performance decreases in more complex layouts.

3.3.5 Additional Details

Table 3.7: Concept 1: Additional Details

Aspect	**Details**
Scalability	The system can scale by following floor lines, allowing autonomous operation based on instructions from the WMS.
Estimated Cost	AlphaBot: R1398 Ultrasonic sensor: R24 Electrical tape: R100 QR codes: R100 **Total: R1622**
Integration	WMS integration and testing are crucial for proper task execution.

3.4 Concept 2: Advanced Navigation and Localization

3.4.1 Overview

This concept focuses on improving navigation and localisation using advanced algorithms and multi-sensor fusion for higher efficiency and accuracy.

3.4.2 Key Features

Table 3.8: Concept 2: Key Features

Feature	**Details**
Path Planning	Dijkstra's algorithm ensures the shortest path is found by exploring all possible nodes, but it is more computationally intensive.
Localization	Multi-sensor fusion combining LiDAR, cameras, and odometry through an Extended Kalman Filter (EKF) for improved accuracy. LiDAR offers precise distance measurements while cameras provide visual data for object detection.
WMS Integration	Real-time data exchange with WMS ensures precise inventory tracking and order fulfilment.

3.4.3 Components

Table 3.9: Concept 2: Components

Component	**Details**
AlphaBot Platform	Enhanced with additional sensors for better accuracy.
Raspberry Pi 4B	Handles complex algorithms and processes data from multiple sensors.
LiDAR and Cameras	Provide high-precision obstacle detection and environmental Localization.
QR-codes	Assist in robust localisation and tracking.

3.4.4 Advantages and Disadvantages

Table 3.10: Concept 2: Advantages and Disadvantages

Advantages	**Disadvantages**
High accuracy for complex environments.	Higher cost due to additional sensors and processing needs.

Capable of handling dynamic operations.	More complex to implement and maintain.
Improved safety and efficiency.	Cumulative errors in odometry can affect EKF reliability.

3.4.5 Additional Details

Table 3.11: Concept 2: Additional Details

Aspect	**Details**
Scalability	The system can be scaled to larger warehouses by adding more sensors and processing power.
Estimated Cost	AlphaBot: R1398 LiDAR Sensor: R2038 Encoders: R300 **Total: R3736**

3.5 Concept Evaluation

3.5.1 Pathfinding Algorithm Simulation for Testing and Evaluation

3.5.1.1 Simulation Program: Python and Pygame

To evaluate different pathfinding algorithms before hardware implementation, a simulation environment is created using Python and Pygame to find the preferred pathfinding algorithm.

3.5.1.2 Warehouse Model

A detailed warehouse model is built in Python using Pygame, replicating the environment where the autonomous robot will operate. This model includes:

- Aisles and shelves to simulate the layout of a typical warehouse.
- Obstacles such as shelves or just areas the robot shouldn't navigate in.
- Possible package or home locations.

The different numbers in Figures 3.2 and 3.3 the grid represent various elements of the warehouse:

- 1's are obstacles/shelves (black).
- 2's are highways (yellow).
- 3's are possible positions of boxes/packages (green).
- 0's are the spaces between the highway and the boxes/packages (white).
- 4's are a pickup area (brown).
- 5's are the charging station of the robot/home (orange).

- 6's are the drop-off area (cyan).

```
# Define the grid with obstacles, highways, and possible goal cells
grid = [
    [1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1],  # A
    [1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1],  # B
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # C
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # D
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # E
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # F
    [1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1],  # G
    [1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1],  # H
    [1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1],  # I
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # J
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # K
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # L
    [1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1, 3, 0, 2, 0, 3, 1],  # M
    [1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1],  # N
    [4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 6, 6],  # O
    [4, 4, 4, 2, 1, 3, 3, 1, 5, 2, 5, 1, 3, 3, 1, 2, 6, 6, 6],  # P
    [3, 2, 2, 2, 1, 1, 1, 1, 5, 3, 5, 1, 1, 1, 1, 2, 2, 2, 3]   # Q
]#   1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
```

Figure 3.2: Grid with obstacles, highways, and possible goal cells

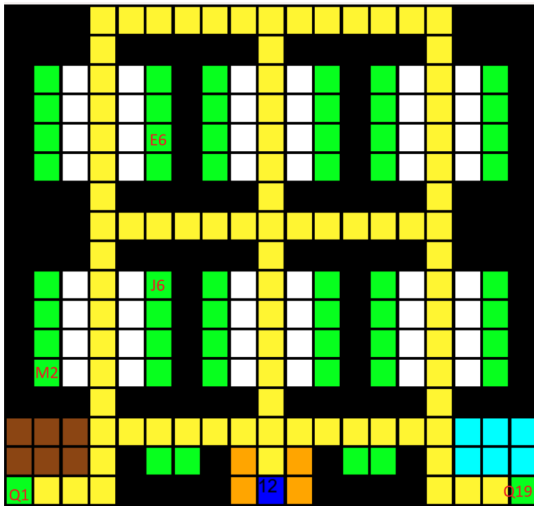


Figure 3.3: Visualization of the warehouse model in Pygame

3.5.1.3 A* Algorithm

To evaluate the A* algorithm, a grid representing the warehouse was constructed, with different numbers assigned to various elements, as previously described. The A* algorithm was then used to find the optimal path from the starting point to the goal, taking into account the costs associated with different types of cells, such as highways and obstacles.

Process of Evaluation The process of evaluating the A* algorithm involved the following steps:

- Defining the grid with various elements, as shown in Figure 3.3.
- Setting up the simulation environment using Pygame to visualize the grid and the robot's movement.
- Implementing the A* algorithm to navigate from the starting position (home) to the goal positions.
- Visualizing the pathfinding process and the final path in the Pygame simulation.

The goal sequence was as follows: home to goal 1 (M2), to goal 2 (Q19), to goal 3 (E6), to goal 4 (Q19), to goal 5 (J6), to goal 6 (Q1), and back to home.

3.5.1.4 Dijkstra's Algorithm

Dijkstra's algorithm was also tested to compare its performance with the A* algorithm. The steps involved in the evaluation were similar:

- Defining the grid with the same elements.
- Setting up the simulation environment in Pygame.
- Implementing Dijkstra's algorithm to navigate from the starting position to the goal positions.
- Visualizing the pathfinding process and the final path.

Comparison of Algorithms The performance of both algorithms was evaluated based on Path length (in nodes), Computation time (in seconds), and Number of nodes visited.

The results showed that while A* generally produced similar paths to Dijkstra's, its heuristic function made it much faster and more efficient at finding the best route. Dijkstra's algorithm was more reliable overall in finding the absolute shortest path, but it required significantly more time and computations. Both algorithms performed well in navigating the warehouse, but A* outperformed Dijkstra in terms of computational efficiency, as seen in Figure 3.4 which compares the number of nodes visited by each algorithm. Additionally, Figure 3.5 illustrates the time taken by both algorithms, further highlighting A*'s advantage in computational efficiency.

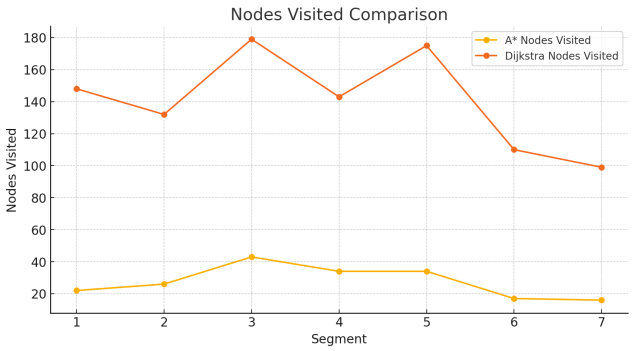


Figure 3.4: Nodes Visited Comparison between A* and Dijkstra

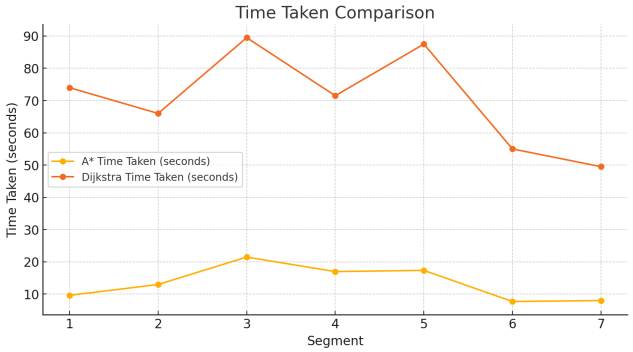


Figure 3.5: Time Taken Comparison between A* and Dijkstra

Comparison Summary The Dijkstra algorithm always found the best path, but the A* algorithm chose the best path 6 out of 7 times. Given the reduced number of computations needed for A*, a slight sacrifice can be made in terms of the best path. The comparison of path lengths and the number of turns between the two algorithms is shown in Table 3.12. Notably, the A* algorithm took a longer route in segment 5, which can be seen in Figure 3.6 and quantified in Table 3.12.

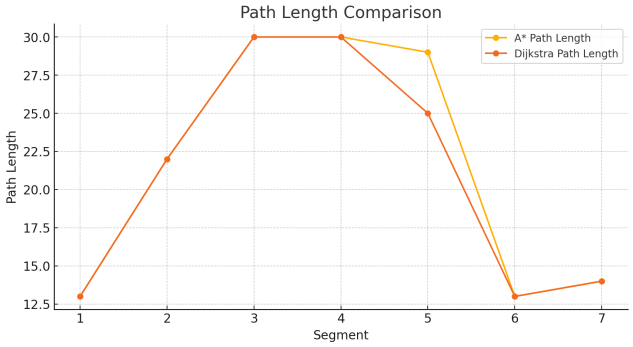


Figure 3.6: Comparison of Path Lengths and Number of Turns

Table 3.12: Comparison of A* and Dijkstra Algorithms

Segment	Start Time	End Time	Time Taken	Nodes Visited	Path length	Nr. of turns
A*						
1	00:00:00	00:09.60	9.62	22	13	3
2	00:09.60	00:22.60	12.98	26	22	4
3	00:22.60	00:44.10	21.48	43	30	4
4	00:44.10	01:01.10	16.98	34	30	4
5	01:01.10	01:18.40	17.36	34	29	4

Continued on next page

Segment	Start Time	End Time	Time Taken	Nodes Visited	Path length	Nr. of turns
6	01:18.40	01:26.10	7.7	17	13	2
7	01:26.10	01:34.10	8.01	16	14	3
Dijkstra						
1	00:00.00	01:14.00	74	148	13	3
2	01:14.00	02:20.00	66	132	22	4
3	02:20.00	03:49.50	89.5	179	30	4
4	03:49.50	05:01.00	71.5	143	30	4
5	05:01.00	06:28.50	87.5	175	25	4
6	06:28.50	07:23.50	55	110	13	2
7	07:23.50	08:13.00	49.5	99	14	3

This comprehensive evaluation using Python and Pygame provided valuable insights into the performance of different pathfinding algorithms in a simulated warehouse environment.

3.5.2 Concept Evaluation Matrix

To compare the two proposed concepts for the autonomous warehouse robot guidance system, a concept evaluation matrix was created. This matrix evaluates each concept against specific criteria, with each criterion weighted based on its importance to the success of the project. Both concepts are rated on a scale from 1 to 10 for each criterion, with higher scores reflecting better performance. The total score for each concept is calculated by multiplying the scores by their respective weights and then summing them. The concept with the highest total score is considered the most suitable for implementation. The justification for each score and weight can be found in Appendix A.

Table 3.13: Concept Evaluation Matrix

Criteria	**Weight**	**Concept 1**	**Concept 2**
Cost	20%	9	1
Implementation Complexity	15%	8	4
Scalability	10%	6	8
Maintenance	10%	9	5
Accuracy of Navigation	10%	6	9
Integration with WMS	10%	7	7
Power Requirements	5%	8	5
Battery Life	5%	8	5
Sensor Requirements	5%	9	6
Pathfinding Efficiency	10%	8	5
Total Score	100%	7.85	5.35

3.5.3 Final Concept Choice

Following a detailed evaluation using the Concept Evaluation Matrix, Concept 1: Basic Navigation and Cost-Effective System has been selected as the final choice for the autonomous warehouse robot guidance system. This decision is based on several key factors:

- **Cost:** Concept 1 is far more cost-effective than Concept 2, with a total estimated cost of R1622. It fits well within the budget while still delivering the essential functionalities required for the project.
- **Implementation Complexity:** The simpler implementation and development processes of Concept 1 reduce the time and resources needed for deployment, allowing for a quicker and smoother setup.
- **Maintenance:** With fewer components and a simpler design, Concept 1 requires less maintenance, ensuring reliability over time and easier upkeep, which is crucial for continuous warehouse operations.
- **Power Requirements:** Concept 1's use of fewer sensors and simpler processing demands results in lower power consumption. This leads to longer battery life and more efficient energy use, reducing downtime and operational interruptions.
- **Adequate Accuracy:** Although Concept 2 offers higher accuracy, Concept 1 provides sufficient accuracy for the project's needs. The line-following and QR-code-based localisation ensure dependable performance in a warehouse environment.
- **Integration with WMS:** Both concepts can integrate with the Warehouse Management System (WMS). Concept 1's simpler design does not limit its ability to communicate and synchronise with the WMS, ensuring smooth task execution.
- **Pathfinding Efficiency:** Concept 1's use of the A* algorithm makes it more computationally efficient and quicker in finding optimal paths, compared to Concept 2's Dijkstra algorithm. This is crucial for real-time navigation and task execution in a dynamic warehouse environment.

In conclusion, Concept 1 has been chosen for its cost-effectiveness, simpler implementation, reduced maintenance and power requirements, and adequate navigation accuracy. The added benefit of the A* algorithm for pathfinding makes it the optimal solution for the autonomous warehouse robot guidance system, providing a balance between performance and cost, and facilitating a more straightforward and efficient deployment process.

Chapter 4
System Design and Implementation

4.1 System Overview and Architecture

The system architecture for the warehouse robot is designed to ensure smooth communication between the web interface, the Personal Computer (PC) (which serves as the control hub), and the Raspberry Pi on the robot. This configuration allows the robot to navigate the warehouse autonomously while receiving real-time feedback from its various sensors. Figure 4.1 presents a high-level diagram illustrating how these components work together.

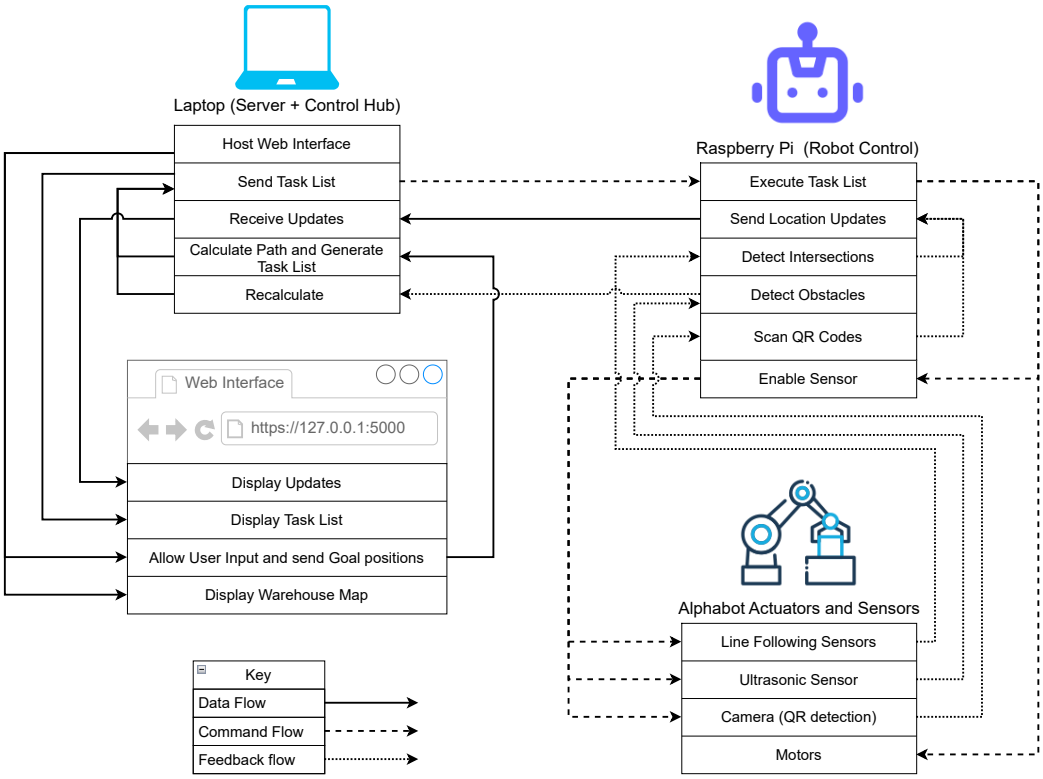


Figure 4.1: System Architecture Overview

4.1.1 Web Interface

The web interface serves as the main control point for the user. It waits for the user to select two goal positions for the robot to navigate to and monitor its progress in real-time. The web interface is run by a html file on the PC. Thus when the python code is running on the PC, the web-interface is accessible on any browser by entering the URL link "http://127.0.0.1:5000". The user first

has to select the two goal positions on the grid and then press the "Send list" button, this will trigger the python code on the PC, to create a task list which is then displayed on the interface along with any live updates from the robot.

4.1.2 Personal Computer (Control Hub)

The PC functions as the server for the entire system. It hosts the web interface and manages the pathfinding and obstacle avoidance for the robot. The PC generates task lists that break down the path the robot should take into actionable steps (such as turning or moving straight) the robot should make when it meets an intersection and sends these tasks to the Raspberry Pi. Additionally, the PC continuously receives updates from the robot regarding its position and obstacles it encounters.

4.1.3 Raspberry Pi 4B (Robot Controller)

The Raspberry Pi acts as the control centre for the robot itself, receiving task lists from the PC and executing them by controlling the robot's motors and sensors. The Pi is responsible for interpreting the input from the line-following sensors, ultrasonic sensors, and the camera for QR code scanning as well as sending updates to the PC, specifically information about its location at intersections, obstacles detected, and QR codes scanned.

4.1.4 Alphabot Sensors and Actuators

The robot's movement and ability to detect its environment are handled by a range of sensors and actuators. Infrared sensors allow the robot to follow lines on the floor, while the ultrasonic sensor detects obstacles in its path. The motors, which are controlled by the Raspberry Pi, manage the robot's movement, and the onboard camera scans QR codes to verify its precise location by confirming a loop closure. Together, these components enable the robot to navigate the warehouse autonomously and efficiently.

4.2 Hardware Design and Implementation

The hardware design for the warehouse robot is built on the Alphabot 2 platform, which provides the core components needed for movement, sensing, and control. The Raspberry Pi serves as the central controller, managing motor control, processing sensor input, and communicating with the PC.

4.2.1 Hardware Components

The Alphabot 2 comes equipped with several components that enable the robot's movement and sensing functions. Figure 4.2 illustrates the key hardware components of the Alphabot's base.

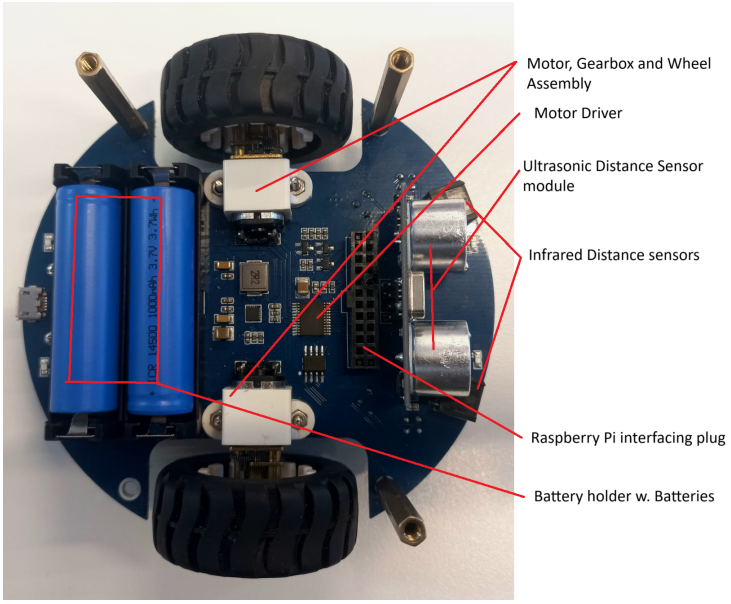


Figure 4.2: Alphabot 2 base hardware components.

Key hardware components include:

- **Motors and Wheels**: Two N20 micro gear motors (600 RPM, 6V) drive the wheels, enabling the robot to move and turn as needed.
- **Infrared Sensors**: These sensors handle both line tracking and short-range obstacle detection. ST188 sensors detect obstacles within 5 cm, while ITR20001/T sensors manage line following.
- **Ultrasonic Sensor**: An ultrasonic sensor provides longer-range obstacle detection (4m maximum), improving the robot's ability to avoid collisions beyond the infrared sensors' 5 cm range.
- **Camera Module**: Mounted on a servo, the camera scans QR codes for position verification, with the servo allowing dynamic adjustments for more accurate detection.
- **Omnidirectional Wheels**: These wheels enable smooth movement in multiple directions, allowing easier navigation at intersections and around obstacles.
- **TB6612FNG Motor Driver**: Controls motor speed and direction, allowing precise movement management via the Raspberry Pi.
- **Power Source**: Powered by two 14500 rechargeable batteries, providing stable power to the Raspberry Pi and motors.

4.2.2 Assembly and Custom Modifications

The robot was assembled according to the Alphabot 2 manufacturer's manual, but a modification was made to improve its functionality. The change was the addition of an ultrasonic sensor. The original obstacle avoidance system used two infrared sensors, but these had a limited range of only 5 cm. To overcome

this limitation, an ultrasonic distance sensor was added to allow for better obstacle detection at longer distances of up to 4 meters.

However, the ultrasonic sensor's interface plug was too large to fit in the robot's frame. To solve this, the interface plug was removed, and the sensor was directly soldered to the circuit board. This custom modification ensured it could fit within the available space without affecting its performance.

Additionally, the kit didn't include batteries, so two 14500 rechargeable batteries were sourced and installed in the battery holder. These batteries power both the Raspberry Pi and the motors, ensuring that the robot runs smoothly.

Figure 4.3 shows a bottom view of the Alphabot 2, highlighting the omnidirectional wheels and the photo-interrupt sensors responsible for line following.

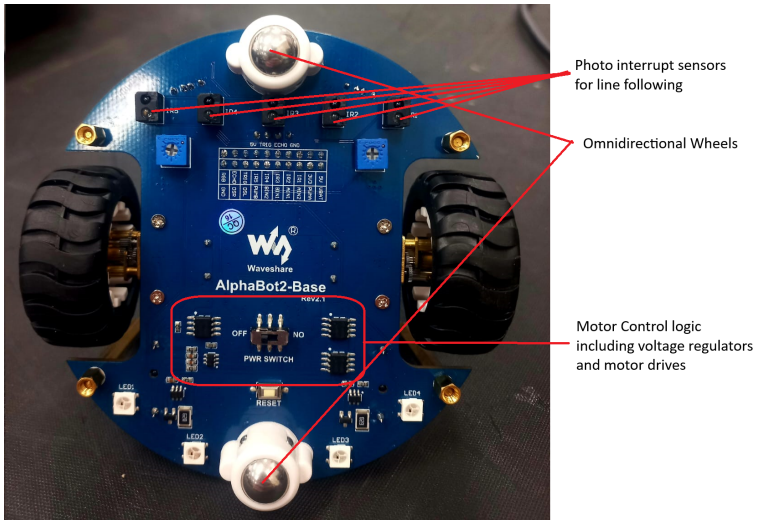


Figure 4.3: Bottom view of the Alphabot 2 base.

The Alphabot 2 comes with a camera module mounted on a servo, as shown in Figure 3.1. By default, the camera streams a video feed to a web interface using MJPG-streamer, which is the application recommended by the robot's manufacturer. While the camera doesn't natively detect QR codes, this functionality can be implemented, if needed, to provide precise localisation. The servo mechanism allows the camera to dynamically adjust its view, improving obstacle detection and scanning accuracy.

These hardware components, along with the custom modifications, are key to the robot's ability to navigate autonomously, detect obstacles, and verify its position when QR code functionality is added.

4.3 Software Design and Implementation

The software setup of the warehouse robot is built to enable communication between the different system components. This allows the robot to navigate

autonomously in the warehouse while getting real-time feedback from its sensors. The main software components include pre-installed scripts provided by the manufacturer for basic motor and sensor control, custom software for navigation and task execution, and communication layers that link the robot with the PC and web interface.

4.3.1 Overview of Software Components

At a high level, the software is divided into several components, each playing a distinct role in the system:

- **Pre-installed Software**: Basic motor control and sensor management, including motor speed control, sensor feedback, line following, and obstacle detection is handled by pre-installed scripts.
- **Custom Navigation Software**: Developed for warehouse navigation, this software calculates optimal paths using the A* algorithm, manages obstacle avoidance, and recalculates routes as needed. It also handles QR code-based position verification and runs on both the PC and Raspberry Pi.
- **Communication Layer**: Communication between the robot and PC is managed through Flask servers, allowing the PC to send task lists and receive real-time updates from the Raspberry Pi.
- **Web Interface**: Built using HTML and JavaScript, the web interface lets users input start and goal locations, track the robot's position, and monitor task progress in real-time.

4.3.2 Pre-installed Software

Several Python scripts were included in the Alphabot 2 package to manage the robot's basic functions. These scripts control motor movement, obstacle detection, and line following on the warehouse floor. The key pre-installed scripts are:

- **AlphaBot2.py**: Controls the motors and wheels, allowing the robot to move forward, backward, or turn left and right.
- **Infrared_Obstacle_Avoidance.py**: Manages the infrared sensors to detect obstacles. If an obstacle is detected within 5 cm, the robot stops or reroutes itself.
- **Ultrasonic_Obstacle_Avoidance.py**: Controls the ultrasonic sensor, which offers longer-range obstacle detection compared to the infrared sensors.
- **Line_Follow.py**: Enables the robot to follow lines on the warehouse floor, using reflective infrared sensors to keep it on track.
- **Joystick.py** and **IRemote.py**: Allow manual control of the robot using a joystick or infrared remote, though these were not used in this project.

4.3.3 Custom Software for Warehouse Navigation

The custom software for the robot manages pathfinding, task execution, and position verification. It takes user inputs from the web interface, calculates the path on the PC, and runs the tasks on the Raspberry Pi. The software libraries and modules used for these functions are listed in Appendix B.

4.3.3.1 User Input and Task List Generation

The navigation process starts when the user selects two goal positions (destinations) on the web interface. These positions mark key positions in the warehouse grid that the robot needs to navigate to. Once the positions are chosen, the user clicks "Send List" and the web interface sends the coordinates of goal 1 and goal 2 to the PC. The A* pathfinding algorithm then processes these locations and breaks the robot's journey into three tasks:

- From the home position to goal 1.
- From goal 1 to goal 2.
- From goal 2 back to the home position.

This ensures the robot follows an efficient path, minimizing unnecessary turns and avoiding obstacles. The web interface displays these tasks to the user and sends them to the robot for execution.

Figure 4.4 shows a screenshot of the web interface where the user selects the positions of goal 1 (green) and goal 2 (red) and views the generated task list along with some updates from the robot.

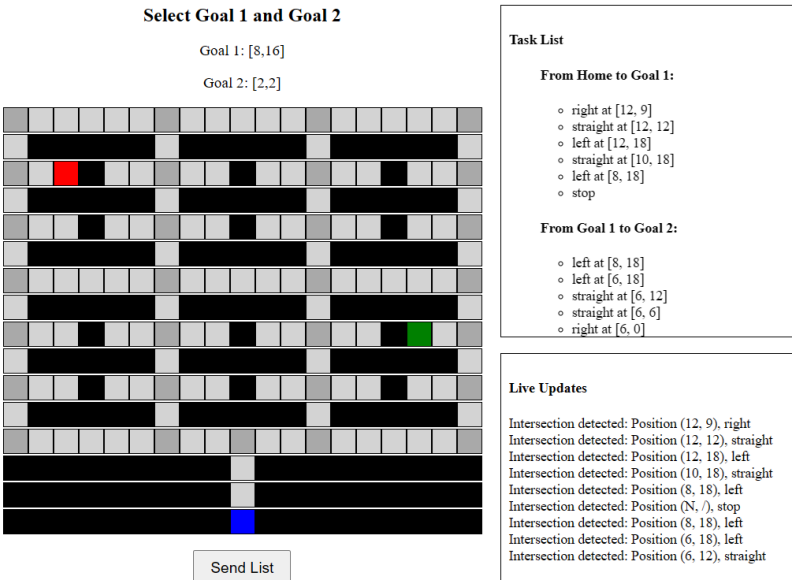


Figure 4.4: Web Interface for Goal Selection and Task List Display

Key JavaScript Code for User Input:

```
function sendSelection() {
    if (selectedGoals.length === 2) {
        const goal1 = selectedGoals[0];
        const goal2 = selectedGoals[1];
        fetch("/upload_goals", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ goal1, goal2 })
        }).then(response => response.json())
          .then(data => updateTaskList(data.tasks1, data.tasks2, data.tasks3));
    } else {
        alert("Please select two goals!");
    }
}

```

Listing 4.1: JavaScript for Grid Rendering and Goal Selection

The JavaScript code ensures that once the two goals are selected, their coordinates are sent to the backend running on the PC via a POST request. The PC's backend generates the task list, which is then sent to the robot for execution and also returned to the web interface, where the task list is displayed.

4.3.3.2 Task Execution and Intersection Detection

Once the robot receives the task list from the PC, it begins executing the tasks. These tasks include basic movements like moving forward, turning left, turning right or stopping at intersections. The robot uses its line-following sensors to stay on track and detect intersections, which are decision points for changing direction or continuing straight. The following Python code snippet shows how the tasks are received and combined into one list:

```
tasks = tasks1 + tasks2 + tasks3  # Combine all tasks into one list

```

Listing 4.2: Task List Reception

The function below is used to carry out each task in order. The robot moves forward until it detects an intersection, then moves on to the next task in the list. It constantly checks the line-following sensor values to stay on track.

```
def move_until_intersection(task):
    Ab.forward()  # Move forward
    # Continuously check the line sensors until an intersection is detected
    while True:
        position, sensor_values = tr_sensors.readLine()
        if line_follower.detect_intersection(sensor_values):
# Intersection detected
            Ab.stop()
            generate_logs(f"Intersection detected at {position}")
```

```
                send_intersection_update(position, task)  # Update PC with position
                break

```

Listing 4.3: Move Until Intersection

After detecting an intersection, the robot checks the task list to determine whether to turn left, right, or continue straight. The following Python code snippet shows how the robot processes these tasks:

```
if "left" in next_task:
    turn_left(position)
elif "right" in next_task:
    turn_right(position)
elif "straight" in next_task:
    move_straight(position)
elif "stop" in next_task:
    stop_robot(position, current_task_index + 1)

```

Listing 4.4: Task Execution at Intersection

When the robot reaches a "stop" task which the PC creates for when the robot reaches a goal, it halts, waits for a few seconds, performs a 180-degree turn, and then marks the task as complete. The robot then sends an update to the PC to notify it that the task is finished before continuing with the next task:

```
def stop_robot(position, task_number):
    Ab.stop()  # Stop robot
    time.sleep(5)  # Wait for 5 seconds
    Ab.left()  # Perform 180-degree turn
    time.sleep(1)  # Adjust for full turn
    Ab.stop()
    send_task_completion_update(task_number)  # Send task completion to web interface

```

Listing 4.5: Stop Robot

4.3.3.3 QR Code Scanning for Position Verification

Along with using line-following sensors, the robot uses its camera to scan QR codes placed at corner intersections in the warehouse to verify its position. These QR codes act as checkpoints, ensuring the robot stays on its intended path. When a QR code is scanned, the robot compares its actual location with the expected position from the task list. If they match, the robot sends a location update to the PC, and the web interface displays the updated position.

4.3.3.4 Obstacle Detection and Path Recalculation

When the robot detects an obstacle, it stops and sends an alert to the PC, including the grid location of the next intersection. The PC compares this with the last known intersection and assumes the obstacle is just before the next one.

Intersection Detection and Path Recalculation When an obstacle is detected, the robot sends an alert like, "Obstacle detected before (next intersection location)". The PC uses this to estimate the obstacle's position, assuming it's just before the next intersection. The system then figures out the robot's current direction and calculates a new path by turning the robot around and finding a route back to its home position. Figure 4.5 shows how the robot decides to send the alert to requests a new path.

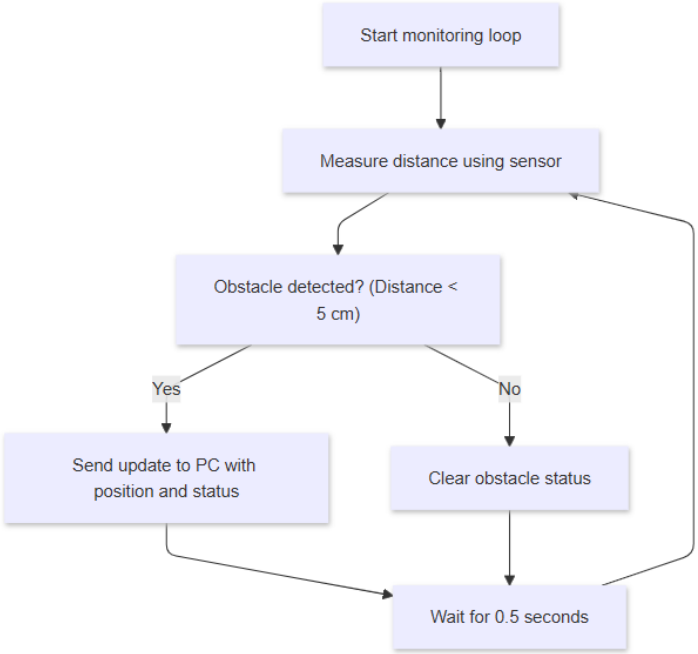


Figure 4.5: Flow diagram of the Obstacle detection logic

4.3.3.5 Communication Between Robot and PC

Communication between the robot and PC is handled via Flask servers running on both devices. The PC sends the task list to the Raspberry Pi over HTTP, and the Pi executes the tasks while continuously sending updates back. These updates include the robot's current location, any detected obstacles, and QR code scan results. Below in Figure 4.6 is a screenshot of the robot's terminal after receiving the task list, displaying the sequence of instructions it follows:

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.137.144:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 381-742-804
Task list received from laptop: ['left', 'right', 'straight', 'straight', 'strai
ght', 'straight', 'right', 'left', 'straight', 'left', 'right', 'right', 'right'
, 'straight', 'right', 'left']
Task list received: ['left', 'right', 'straight', 'straight', 'straight', 'strai
ght', 'right', 'left', 'straight', 'left', 'right', 'right', 'right', 'straight'
, 'right', 'left']
192.168.137.1 - - [03/Oct/2024 16:40:58] "POST /receive_tasks HTTP/1.1" 200 -
```

Figure 4.6: Robot Terminal After Receiving Task List

The robot communicates key updates to the PC, such as task completion and intersection detections, as shown in the Python code below:

```
1 def send_task_completion_update(task_number):
2     payload = {"task_number": task_number, "status": f"
    Completed Task {task_number}"}
3     requests.post(PC_url, json=payload)
4
```

Listing 4.6: Task Completion Update

Intersection Communication Each time the robot detects an intersection it sends an update to the PC which includes its current position, the fact that it detected an intersection and what its intended task is for that intersection, eg. left. This enables the system to track the robot's progress in real time.

Obstacle Communication When the robot detects an obstacle, it sends a notification to the PC and requests a new path. The PC then provides an updated task list, allowing the robot to avoid the obstacle and navigate back to its home position.

4.3.4 Task and Information Flow

The warehouse robot system follows a clear communication sequence between the user, web interface, PC, and robot. This ensures smooth task execution, navigation, and real-time updates during the robot's operation. Figure 4.7 illustrates the flow of information and tasks throughout the system.

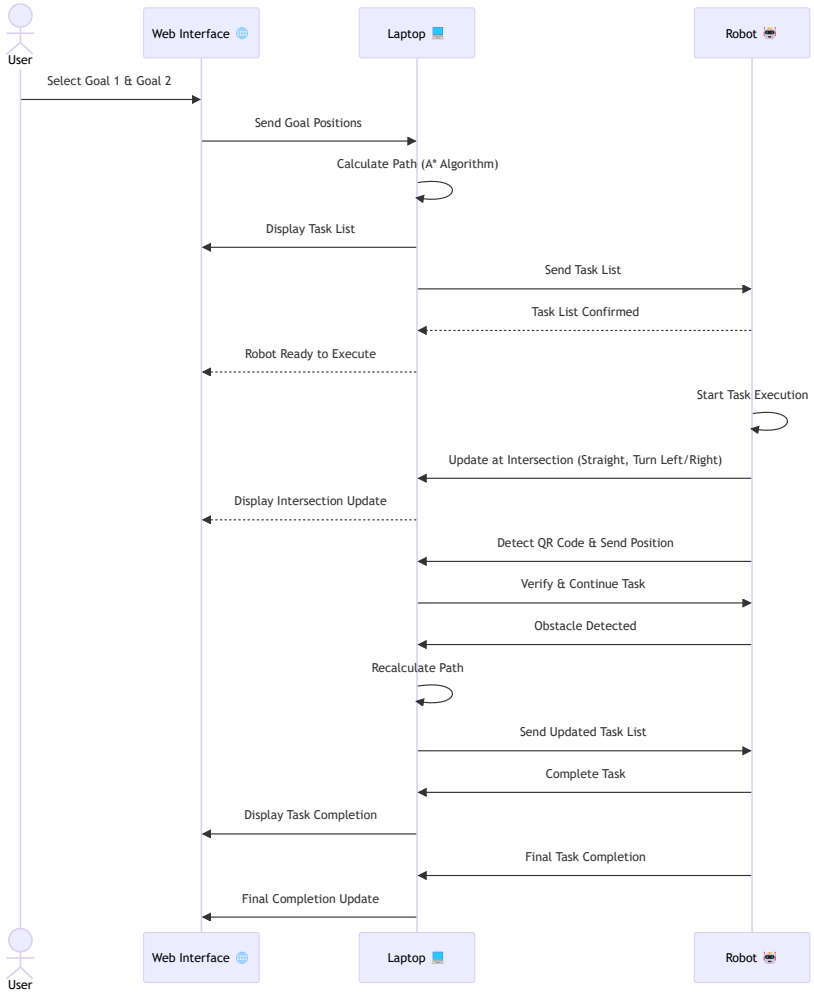


Figure 4.7: Task and Information Flow Diagram

1. User Input via Web Interface

The process starts with the user selecting two goal positions on the warehouse grid. Goal 1 is marked in green and Goal 2 in red. After selecting the goals, the user clicks the "Send List" button, which triggers a POST request to the PC, sending the goal 1 and goal 2 positions.

2. Path Calculation and Task List Generation

Once the PC receives the goal positions, it follows these steps:

- Runs the **A*** algorithm to calculate the shortest path.
- Divides the calculated path into three distinct tasks:
  - From the home position to **Goal 1**.
  - From **Goal 1** to **Goal 2**.
  - From **Goal 2** back to the home position.
- Generates a task list detailing the actions the robot should take at each intersection, such as:

  - *Turn left.*
  - *Turn right.*
  - *Go straight.*

- Displays the generated task list on the web interface, providing feedback to the user and sending the task list to the robot.

3. Task List Transmission to the Robot

The task list is sent from the PC to the robot through a POST request. The robot acknowledges receiving the list, confirming that it is ready to carry out the tasks.

4. Task Execution Begins

Once the task list is received, the robot begins navigating the warehouse using its line-following sensors. It carries out the assigned tasks at intersections and sends updates at each intersection.

5. Intersection Detection and Updates

At each intersection, the robot reviews the task list, performing actions like turning or going straight, and then sends position updates back to the PC. These updates are displayed on the web interface in real time, allowing the user to track the robot's progress. If the task is to stop, the robot halts, turns around and carries out the next task.

6. QR Code Scanning and Position Verification

As the robot moves through the warehouse, it scans QR codes with its onboard camera. The data is sent to the PC, which checks the robot's actual position against the expected one. If there's a mismatch, the PC sends instructions to guide the robot back to its home position.

7. Obstacle Detection and Path Recalculation

If the robot detects an obstacle using its ultrasonic sensors, it stops and requests a new task list. The PC records the obstacle's position, recalculates the path using the A* algorithm, and sends an updated task list that guides the robot around the obstacle and back to its home position.

4.4 Hardware Malfunction and Troubleshooting

This section focuses on the hardware problem that occurred during the early testing of the Alphabot 2, regarding the motor driver. The motor driver controls the robot's movement, and its failure made it difficult for the robot to perform tasks like navigation, line-following, and avoiding obstacles. The section goes through the steps taken to diagnose the issue, including voltage checks and motor testing. It also explains the attempts to fix the problem using voltage and torque adjustments, and how this affected the overall project timeline and testing.

4.4.1 Motor Driver Issue and Diagnosis

Early in the testing of the Alphabot 2's movement, an issue with the motor driver was discovered. The robot did not respond correctly to commands for basic functions like moving forward or turning. After investigating, it was found that the motor driver was sending irregular and inconsistent voltage to the motors, causing the robot's movement to be either erratic or non-functional.

4.4.1.1 Initial Observations

When testing the basic motor control script, `AlphaBot2.py`, which should make both motors move forward, only the right motor worked as expected. The left motor did not respond at all. Using the `joystick.py` file for manual control (forward, backward, left, right), it was noticed that the left motor could move in reverse but failed to go forward. This suggested a hardware issue with the motor driver.

4.4.1.2 Testing Setup and Voltage Measurements

To confirm the problem, a testing setup was created using an oscilloscope to check the PWM signals from the Raspberry Pi's GPIO pins (Figure 4.8). The signals from the Raspberry Pi were as expected, which ruled out any issues with the control signals from the Pi. The next step was to check the voltage output at the motor driver's pins.

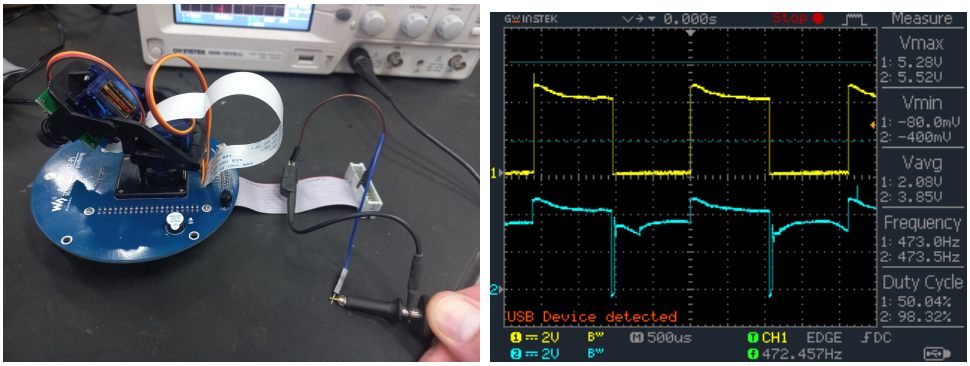


Figure 4.8: Testing Setup and Oscilloscope Readings at Motor Pins for 50% PWM

When the motors were set to run in reverse, both the left and right motors should have received similar signals. However, the left motor driver's output voltage was significantly higher, almost double that of the right motor. This confirmed that the motor driver was malfunctioning, failing to regulate the voltage correctly for the left motor. Figure 4.8 shows the oscilloscope results for both motors at a 50% PWM signal, with the right motor's signal in yellow and the left motor's signal in cyan (Channel 2). The left motor received an average voltage of 3.85V, while the right motor received 2.08V. The waveform also shows that the left motor's signal doesn't drop to 0V when the PWM signal is pulled down resulting in a duty cycle of nearly 100% when it should've been

50%. This indicates that the motor driver is leaking voltage, possibly from its Vcc pin into the left motor's reverse pin signal.

4.4.1.3 Troubleshooting and Testing Process

In an effort to resolve the issue, the following steps were taken:

- Verified the power supply and ensured the batteries were fully charged.
- Used a multimeter and oscilloscope to measure voltage at the motor driver pins, confirming irregular voltage delivery to the left motor.
- Ran diagnostic scripts to verify that control signals from the Raspberry Pi were correct, ruling out software issues.
- Test for continuity in the wiring leading to the motors to eliminate possible connection faults between the Raspberry Pi, motor driver, and motors.
- Compared motor winding resistance to ensure there was no motor damage.
- Disconnected the motors from the circuit and tested them directly with a power supply to ensure they operated correctly at varying voltages and also reversing the voltage to verify that the motor is capable of turning in both directions.
- Consult on online forums regarding similar motor driver issues.

Despite these efforts, the issue persisted. A similar problem was observed with a second Alphabot 2 unit, where the right motor showed the same behaviour—failing to move forward but functioning correctly in reverse. This suggested a recurring issue with the motor driver hardware.

4.4.2 Voltage Mapping and Compensation

Determined to get the robot working for testing, the decision was made to attempt a software fix. Since the right motor on the second robot did not run forward, the right motor was disconnected and reconnected with reversed polarity so that the reverse pin would drive the motor forward. Afterwards, voltage mapping of the motors at different PWM values was performed, which led to the development of voltage compensation code.

4.4.2.1 Voltage Mapping Tests

To better understand motor performance, voltage mapping tests were conducted to determine the relationship between PWM input values and the actual voltage delivered to the motors. As shown in Figure 4.9, the left motor exhibited a relatively linear and expected response between PWM and voltage, whereas the right motor displayed a non-linear relationship, with significantly higher voltages at the same PWM values.

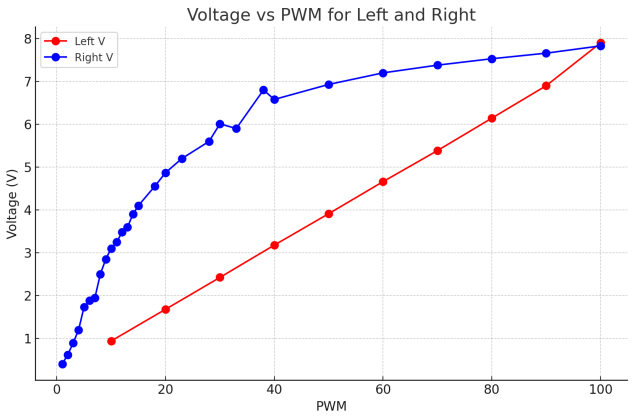


Figure 4.9: Unfitted Voltage vs. PWM for Left and Right Motors

4.4.2.2 Voltage Compensation and Fitting Curves

To address this voltage mismatch, a compensation function was created in the control script to adjust PWM values for the right motor, attempting to match the output voltages of the left motor. A fitted polynomial curve was applied for the right motor, while a linear fit sufficed for the left. The fitted curves for both motors are shown in Figure 4.10.

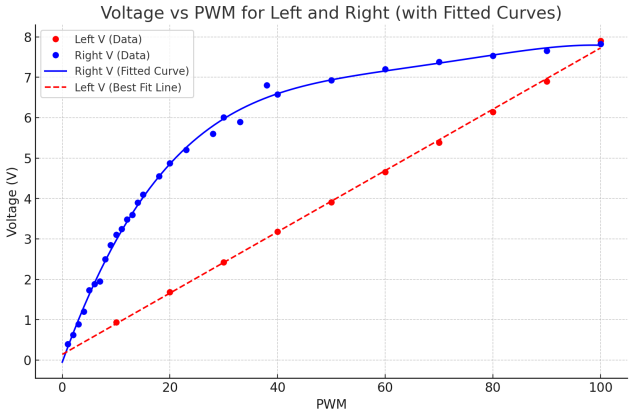


Figure 4.10: Fitted Voltage vs. PWM for Left and Right Motors

The equations used to map PWM values to voltages were as follows:

$$\begin{aligned} V_{\text{right}} = &-2.57 \times 10^{-7} \cdot \text{PWM}^4 + 7.23 \times 10^{-5} \cdot \text{PWM}^3 \\ &- 7.58 \times 10^{-3} \cdot \text{PWM}^2 + 0.37 \cdot \text{PWM} \\ &- 0.049 \end{aligned} \tag{4.1}$$

$$V_{\text{left}} = 0.078 \cdot \text{PWM} + 0.665 \tag{4.2}$$

Using these formulas, the PWM duty cycle of the right motor was adjusted to match the left motor's voltage, as shown in Figure 4.11. The plot compares

the compensated PWM signals of the right motor to those of the left motor. It clearly shows that to match the left motor's voltage, the PWM signals of the right motor need to be significantly reduced.

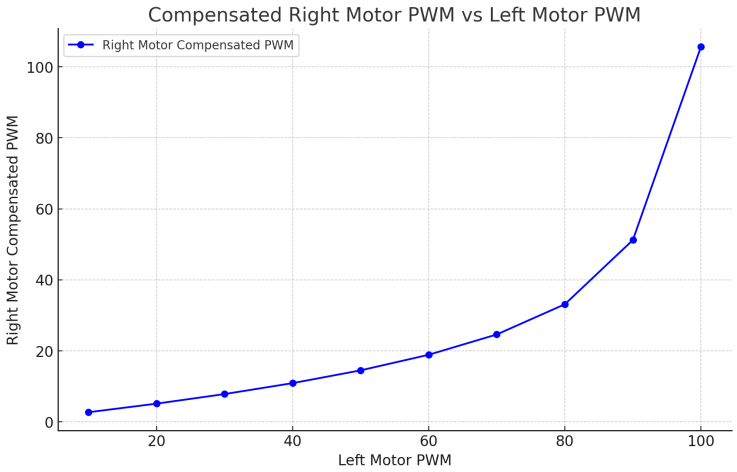


Figure 4.11: Compensated Right Motor PWM vs Left Motor PWM

4.4.2.3 Final Compensation Results and Torque Adjustment

With the voltage compensation in place, the motors were providing almost equal voltages across the same PWM range, as shown in Figure 4.12. However, the robot still veered to the right when moving forward, which suggests the right motor had lower torque even with the voltage adjustments.

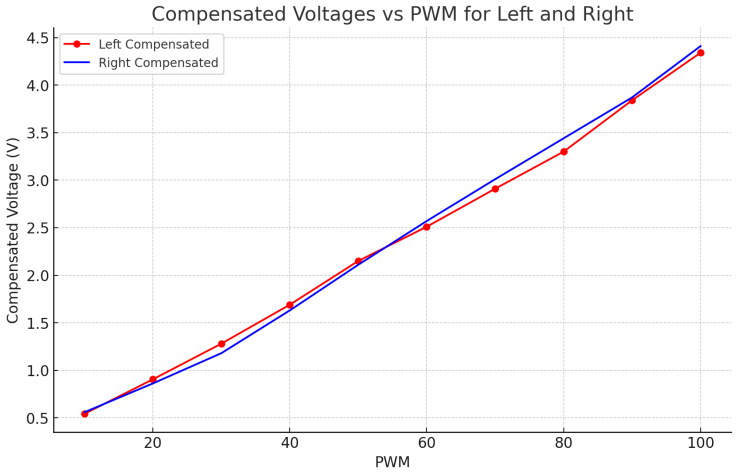


Figure 4.12: Compensated Voltages for Left and Right Motors

To fix the torque imbalance, a torque compensation system was added, boosting the right motor's voltage by 47% for the first 0.5 seconds of movement. This helped the robot straighten its path and move forward without veering. Figure 4.12 shows the final results of the torque compensation.

4.4.3 Motor Voltage Compensation Logic

In the robot's control system, voltage compensation is used to ensure both motors get equal voltage outputs, helping the robot move in a straight line. This compensation is necessary because the right motor showed non-linear voltage responses compared to the left motor, requiring adjustments.

4.4.3.1 Motor Compensation Overview

The compensation is handled by the function: compensate_right_pwm(). The goal is to calculate the correct PWM value for the right motor to match the voltage output of the left motor. Due to hardware differences, the motors don't behave the same when given the same PWM input, and this method corrects those differences.

4.4.3.2 Calculating Desired Left Motor Voltage

The desired voltage for the left motor is calculated using a linear model of the PWM-to-voltage relationship. The following function computes the voltage for any given left motor PWM:

```
def left_voltage(self, pwm):
    return 0.07584182 * pwm + 0.1408
```

Listing 4.7: Left Motor Voltage Calculation

This function models the left motor's behaviour as a linear equation, providing a predictable relationship between the PWM input and the motor's voltage.

4.4.3.3 Modeling Right Motor Voltage

In contrast, the right motor voltage requires a more complex model, expressed as a 4th-degree polynomial due to its non-linear behaviour:

```
def right_voltage_fourth(self, pwm):
    return (-2.56913920e-07 * pwm**4
            + 7.23355675e-05 * pwm**3
            - 7.58001776e-03 * pwm**2
            + 3.69975479e-01 * pwm
            - 4.92547993e-02)
```

Listing 4.8: Right Motor Voltage Calculation

This polynomial function accounts for the more erratic voltage response of the right motor as the PWM increases, ensuring a more accurate model of its behaviour.

4.4.3.4 Compensating Right Motor PWM

The core of the compensation logic occurs within the `compensate_right_pwm` function. This function calculates the required PWM value for the right motor to match the left motor's voltage:

```
def compensate_right_pwm(self, pwm_left):
    desired_voltage = self.left_voltage(pwm_left)
    def error_in_right_pwm(pwm_right):
```

```
4           return self.right_voltage_fourth(pwm_right) -
    desired_voltage
5       compensated_right_pwm = fsolve(error_in_right_pwm, pwm_left
    )[0]
6       compensated_right_pwm *= 0.97  # 3% reduction by tuning
7       return compensated_right_pwm
```

Listing 4.9: Right Motor PWM Compensation

This method works as follows:

- The left motor voltage is calculated using the linear model.
- An internal function `error_in_right_pwm` calculates the difference between the desired voltage and the actual right motor voltage.
- `fsolve` from the `scipy` library is used to solve for the right motor's PWM value that produces the desired voltage.
- A slight reduction (about 3%) was applied by tuning the compensation.

4.4.3.5 Applying the Compensated PWM

Once the compensation is calculated, it is applied to the right motor in the robot's movement methods. For example, in the `forward` method:

```
1  def forward(self):
2      GPIO.output(self.AIN1, GPIO.LOW)
3      GPIO.output(self.AIN2, GPIO.HIGH)
4      GPIO.output(self.BIN1, GPIO.LOW)
5      GPIO.output(self.BIN2, GPIO.HIGH)
6      self.PWMA.ChangeDutyCycle(self.PA)
7      self.PWMB.ChangeDutyCycle(self.PB * 1.47)
8      time.sleep(0.5)  # 50% boost for 0.5 seconds
9      compensated_pb = self.compensate_right_pwm(self.PB)
10     self.PWMB.ChangeDutyCycle(compensated_pb)
```

Listing 4.10: Forward Movement with Compensation

In this code:

- The left motor runs normally with its designated PWM.
- The right motor receives an initial 50% boost for 0.5 seconds to help it match the left motor's torque.
- After the boost, the right motor receives the calculated compensated PWM to ensure it runs in sync with the left motor.

4.4.4 Conclusion and Impact on Testing

The faulty motor driver affected key functions like line-following, task execution, and obstacle avoidance. Line-following needs constant adjustments to motor speed, which are usually handled by a PID control system. PID helps the robot adjust its motor speeds based on sensor feedback.

In a PID control system, the robot adjusts motor speeds based on sensor feedback to stay on course. These adjustments depend on three components:

proportional, integral, and derivative, which are all influenced by the system's response time (dt). The problem with the voltage and torque compensation is that it adds a 0.5-second boost to the right motor to fix its torque issue. This delay interferes with the precise timing needed for the PID loop, making the time derivative component unreliable. The system can't accurately measure how much the error changes over time, which prevents the PID from making quick corrections. Without this precise control, the line-following became impractical.

Since the project focused on the robot's navigation and guidance system, not its mechanical functioning, it was decided not to continue testing the line-following. Fixing the motor driver and making a working PID system would have taken too long and added unnecessary complexity.

The project timeline was heavily impacted by hardware delays. Although the robot was received on 14th August, work couldn't start until 9th September because important items like a display, keyboard, mouse, and a micro HDMI to HDMI cable were missing. These items were necessary to set up the Raspberry Pi and begin motor testing, which delayed early tests.

Once work started on 9th September, the motor driver problem was observed immediately. This diagnosis wasn't clear at first, and after discussing with the supervisor on 20th September, the decision was made to buy another robot. While the motors on the new robot worked at first, the same problem reappeared by 22nd September. Software fixes were tried, as hardware repairs weren't practical. Voltage and torque compensation were used to help the robot drive straight, which took until 2nd October to resolve. However, adding these fixes to the line-following code was too difficult, and on 30th September, just one week before the report draft was due on 7th October, a decision was made with the supervisor to stop working on motor control and line-following. With the final report due on 25th October, this left limited time to complete the remaining testing and finish the report.

The motor driver issue and the delays it caused left much less time for implementing and testing other important parts like task execution, intersection detection, communication, obstacle avoidance, and QR code scanning. Although progress was made up to obstacle avoidance, the QR code position verification wasn't fully completed due to the time constraints. Some of the work was started, but more time would be needed to finish and test it.

Chapter 5
Testing and Results

5.1 Testing Methodology

This chapter outlines the approach used to test the robot's subsystems. Due to hardware issues, namely the faulty motor driver, full system testing wasn't possible. However, key subsystems were still tested by manually pushing the robot over a simulated intersection and observing its behaviour during obstacle avoidance testing. Although QR code scanning wasn't implemented due to time constraints, as mentioned in Section 4.4.4, it could be added in the future.

5.1.1 Initial Testing and Identified Issues

During initial testing the robot faced a serious hardware issue with the motor driver, which made it impossible to reliably execute movement commands or follow the planned navigation method, specifically line-following. Efforts to solve the issue using voltage and torque compensation were unsuccessful due to the motor driver's faulty condition. This prevented full movement-based testing, such as dynamic line-following and obstacle avoidance, from being carried out.

However, important subsystems like communication between system components, intersection detection, line-following sensors, and obstacle avoidance were still tested by manually pushing the robot over a simulated intersection.

5.1.2 Test Setup

The testing environment was a controlled space where the robot's subsystems were evaluated. The Raspberry Pi on the robot was connected to the PC via a wireless hotspot, allowing task list communication and real-time updates. Tasks were sent through the web interface, and the PC handled path generation using the A* algorithm. To simulate movement, the robot was manually pushed over an intersection marked on A4 paper, allowing tests to be conducted on communication, intersection detection, and obstacle avoidance.

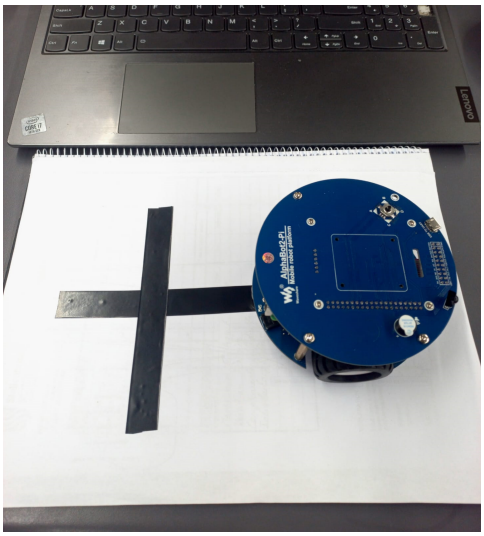


Figure 5.1: Simulated Intersection Setup for Manual Testing

The image in Figure 5.1 shows the setup used for manual testing, where the robot was physically pushed across a marked intersection to simulate movement.

5.2 Subsystem Testing and Performance Evaluation

Despite the motor driver issue, several subsystems were successfully tested to validate that the robot's core functionalities performed as expected.

5.2.1 Communication and Updates Testing

This test assessed the data flow between the robot, PC, and web interface, focusing on communication through HTTP requests.

- **Test Scenario:** The web interface was used to enter goal positions, which were sent to the PC via a POST request. After processing the goals, the PC sent the task list back to both the web interface and the robot, also using a POST request. The robot reviewed the list, executed the tasks, and sent updates back to the PC, which displayed them on the web interface at each intersection.
- **Results:** Updates were received consistently, with position data accurately displayed on the web interface in real-time. The PC received the goal positions immediately after the "Send List" button was pressed, and both the robot and web interface received the task list within 50 to 60ms.
- **Conclusion:** Communication between the PC and the robot, handled via HTTP POST requests, functioned as expected. This allowed real-time tracking of the robot's progress and ensured smooth data flow across the system.

5.2.2 Line Following and Intersection Detection Testing

Line following and intersection detection were tested manually by pushing the robot over a simulated intersection. The sensors detected lines and intersections accurately, though certain sensor limitations were observed.

- **Test Scenario:** The robot was manually pushed, and intersection detection was evaluated based on turning commands from the task list.
- **Results:** Intersections were detected correctly in most cases. However, the left two sensors exhibited lower resolution compared to the middle and right sensors, affecting accuracy in corner intersections.

When placed on a white surface, the sensor values (from left to right) were approximately [980, 969, 954, 954, 882]. On a black surface (insulation tape), the values dropped to around [949, 939, 222, 286, 149]. This showed a significant resolution difference, with about 30 on the left two sensors and an average of 711 on the other three when detecting the reflectivity between black and white surfaces. These differences in resolution called for adjustments to the intersection detection code. The new approach now detects an intersection when three sensors register values below 700, rather than the previous threshold of 300 for all sensors. This adjustment improved detection accuracy, but issues remained at corner intersections, particularly when the robot approached an edge intersection with only a line on the left side.

To account for the robot's inability to sense intersections where there is no tape on the right side of the robot, a simple solution was to extend the tape at edge intersections. This way, at least three sensors will always be able to detect an intersection. The robot manufacturer provided calibration logic for the sensors to address resolution differences, but this logic relied on motor control that slowly turns the robot left and right while placed over a black line. Since the robot's motor control doesn't work, manually calibrating the sensors wasn't feasible, so extending the tape at intersections was a simpler alternative.

5.2.3 Task List Execution and Command Processing Testing

This test was designed to verify the robot's ability to receive the task list from the PC, triggered by the user pressing the "Send list" button on the Web interface, and then being able to follow the task list's instructions when it detects an intersection. Due to the robot's motor control malfunction, the robot was manually pushed over a simulated intersection which triggered the intersection detection logic. The robot's ability to follow the task list's instructions at each intersection was recorded.

- **Test Scenario:** The task list generated by the PC based on user-input goal positions includes a mix of straight, left, and right commands. The

robot received the task list and waited to be pushed over intersections to make decisions according to the task list's instructions.

- **Results:** The task list was processed correctly, with correct decisions made at each intersection based on the instructions. Task execution was handled flawlessly by the robot, and updates were sent to the web interface at every intersection.
- **Conclusion:** Task list execution was verified in the software. The pathfinding algorithm provided optimal routes for each task, and real-time updates were transmitted to the web interface after pushing the robot over each consecutive intersection.

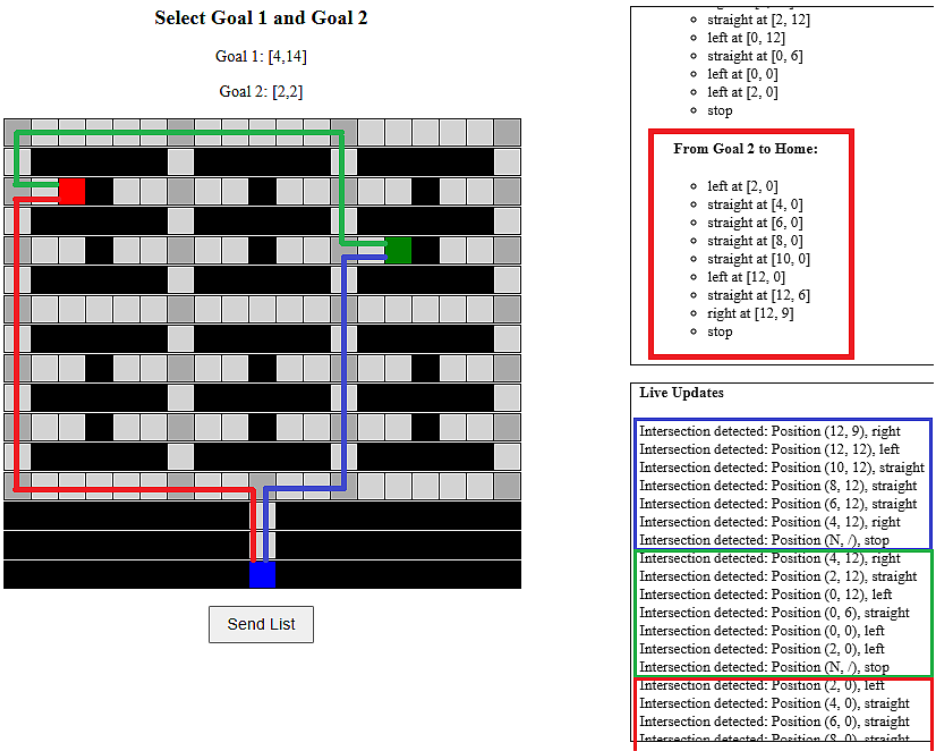


Figure 5.2: Task execution test illustrating robot's path and live updates at each intersection. Task 1 in blue, Task 2 in green, and Task 3 in red.

In Figure 5.2, the task flow is shown, with the robot executing three different tasks. Task 1 took the robot from the home position (blue) to Goal 1 (green). Task 2 was from Goal 1 to Goal 2 (red), and Task 3 took it from Goal 2 back to the home position. The blue, green, and red lines in the figure represent the paths the robot took for each task, showing that the pathfinding algorithm found the shortest routes.

The figure also shows the live updates sent by the robot during execution. The updates for Task 1 are encircled in blue, those for Task 2 in green, and part of Task 3's updates are shown in red. However, only part of Task 3's updates

are visible because the rest were cut off by the web interface's margins. The task instructions for Task 3 are also highlighted in red.

5.2.4 Obstacle Detection and Avoidance Testing

The main goal of this test was to check the robot's ability to detect and avoid obstacles. While full distance measurement accuracy tests were not conducted, a quick assessment using the setup in Figure 5.3 showed that the ultrasonic sensors had a minimal error margin, which was acceptable for the system's requirements.

The test focused on verifying the system's ability to detect obstacles, log their position, recalculate the path, and avoid the obstacle. An object was placed 10 cm in front of the robot during its task execution. The robot is programmed to detect obstacles at 12 cm, and it successfully detected the flask as expected.

After detecting the obstacle, the system logged its position, recalculated a new path, and sent an updated task list to the robot. The robot then followed the new task list, avoiding the obstacle and continuing towards its final goal.

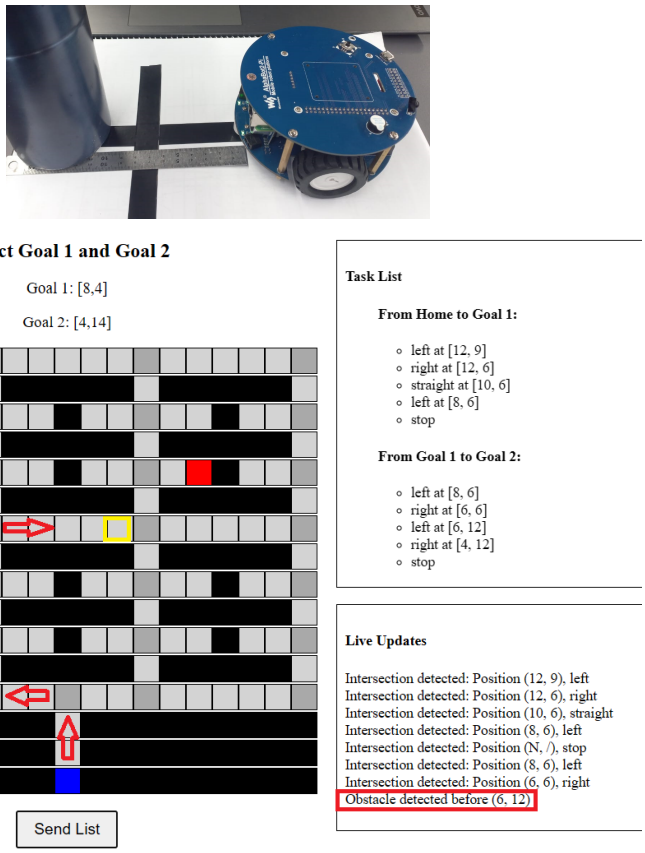


Figure 5.3: Obstacle detection testing setup and live updates on the web interface.

The following logs were recorded during the test:

PC Logs:

```
Received log message: Obstacle detected before (6, 12)
Last intersection position: [6, 6]
Next intersection position: [6, 12]
Handling obstacle: Detected before [6, 12] and after [6, 6]
(Direction: east)
Grid updated with obstacle at (6, 11)
Obstacle placed at (6, 11). Updating grid and recalculating path...
Robot's current position is assumed to be: (6, 10)
New path calculated.
Sending new task list to robot.
```

Robot Logs:

```
Task list received!
Task list: ['left at [6, 6]', 'straight at [8, 6]', 'straight
at [10, 6]', 'left at [12, 6]', 'right at [12, 9]', 'stop']
at 18:16:20.
```

In conclusion, the obstacle detection and avoidance system performed successfully. The robot detected the obstacle, logged its position, and recalculated a new path to navigate around it, confirming that the system can handle dynamic navigation tasks.

5.3 Proposed Full-System Testing Setup

If the motors had functioned as expected, the full-system testing would have been conducted in a simulated warehouse environment. The robot would navigate predefined paths, utilizing its sensors for line-following, intersection detection, and obstacle avoidance.

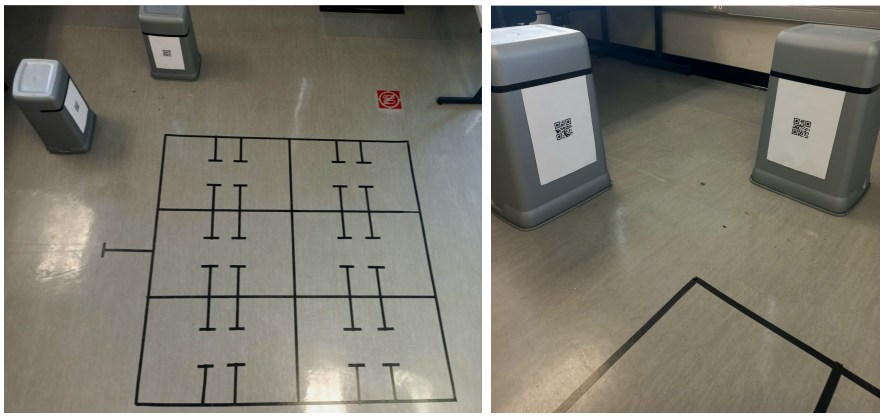


Figure 5.4: Proposed Testing Setup in a simulated warehouse environment

The images in Figure 5.4 represent the proposed test setup. The first image shows the layout of the testing environment, while the second illustrates

how and where QR codes would be placed at the grid corners for position verification.

5.4 Results Analysis

Despite the hardware challenges, key subsystems such as communication, task execution, and intersection detection were tested and validated. The manual testing process showed that these components functioned as expected, providing a strong basis for future testing once the hardware issues are resolved.

Subsystems like task processing and real-time communication between the robot, PC, and web interface performed well during testing. The consistent updates at intersections and successful obstacle detection and avoidance demonstrated the system's capability to manage warehouse navigation tasks when fully functional.

5.4.1 Comparison with Expected Outcomes

The tested subsystems met their expected outcomes. Both communication and intersection detection were reliable, and task list execution worked smoothly in the software. Although hardware issues limited full dynamic testing, the pathfinding algorithm and task execution logic showed that the system can navigate the warehouse grid accurately once the motors are functioning correctly.

Chapter 6
Discussion

6.1 Evaluation of System Performance

The system performed well in task execution, providing real-time updates, and detecting obstacles. Intersection detection was reliable, and the robot successfully followed pre-calculated paths. However, motor driver issues affected its ability to follow lines, requiring voltage compensation to maintain straight movement. While this solution worked for simple paths, it posed challenges in more complex scenarios. Overall, the system handled planned paths well but struggled with dynamic adjustments.

6.2 Strengths and Limitations

The system had several strengths, including flexible task management, real-time updates, and an easy-to-use web interface, making it efficient for basic tasks. However, the faulty motor driver limited movement precision, which reduced performance in more complex environments. Although the A* pathfinding algorithm worked effectively, hardware constraints prevented the system from achieving its full potential.

6.3 Comparison with Existing Systems

Compared to more advanced systems, this solution is simpler and more cost-effective. While other systems use LiDAR and SLAM for navigation, this system relies on A* pathfinding, making it suitable for basic tasks but lacking features like advanced obstacle avoidance and multi-robot coordination. Despite these limitations, the system's integration with a Warehouse Management System (WMS) and real-time updates aligns well with warehouse operations.

6.4 Lessons Learned

A key lesson from the project was the importance of hardware reliability. The motor driver issue emphasized the need for careful hardware selection and thorough testing. While the software performed well, the hardware limitations restricted the system's full capabilities. Future projects will require more rigorous hardware testing and closer integration with software, especially for handling dynamic tasks.

Chapter 7
Conclusion

The development and testing of the autonomous warehouse robot system showed that real-time task execution and pathfinding can work well in warehouse environments. The project successfully combined the web interface, A* pathfinding algorithm, and real-time communication between the robot and the laptop, ensuring tasks were completed as expected in basic tests. Key subsystems like task processing, obstacle detection, and intersection detection worked as intended.

However, there were some problems due to hardware issues, especially with the motor driver, which affected the robot's ability to make precise movements like line-following and small adjustments. While voltage and torque compensation helped with basic tasks, the inability to use PID control for fine-tuning the motors was a major challenge. This made it hard for the robot to perform more advanced tasks that required precise movement, limiting its usefulness in a dynamic warehouse.

If a fully functional and tested line-following robot had been available, the system would have performed better, especially with dynamic navigation. Another improvement would be centralizing the whole system so the user could run everything (laptop code, robot code, and the web interface) from one place, such as a Warehouse Management System (WMS). This would make the system easier to use and more efficient.

Additionally, the current obstacle avoidance system makes the robot return to its home position after detecting an obstacle. In the future, the system should be able to avoid the obstacle and continue with its current task, using flags in the code to track which tasks have been completed. The use of QR code position verification would also improve navigation accuracy. A future version of the system could also allow the robot to carry packages, scan their information with the camera, and communicate with the WMS to track the location of items in the warehouse.

In conclusion, while this project created a good starting point for an autonomous warehouse robot, more reliable hardware—especially a better motor driver—and further software improvements are needed to handle more complex and dynamic tasks. With these upgrades, the system could become a fully functional robot for precise navigation, task execution, and package handling in a real warehouse environment.

Chapter 8
List of references

(2023). Coppeliasim overview. Accessed: 23 October 2023.
Available at: `https://www.coppeliarobotics.com`

(2023). Gazebo overview. Accessed: 23 October 2023.
Available at: `https://gazebosim.org`

(2023). Matlab and simulink for robotics. Accessed: 23 October 2023.
Available at: `https://www.mathworks.com`

(2023). Pygame documentation. Accessed: 23 October 2023.
Available at: `https://www.pygame.org`

(2023). Webots overview. Accessed: 23 October 2023.
Available at: `https://cyberbotics.com`

Alatise, M.B. and Hancke, G.P. (2020). A review on challenges of autonomous mobile robot and sensor fusion methods. *IEEE Access*, vol. 8, pp. 39830–39846.

Ayala, R. and Mohd, T.K. (2021). Sensors in autonomous vehicles: A survey. *Journal of Autonomous Vehicles and Systems*, vol. 1, no. 3, p. 031003.

Bogue, R. (2016). Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot: An International Journal*, vol. 43, no. 6, pp. 583–587.

Cho, B.-S., Seo, W.-J., Moon, W.-s. and Baek, K.-R. (2013). Positioning of a mobile robot based on odometry and a new ultrasonic lps. *International Journal of Control, Automation and Systems*, vol. 11, pp. 333–345.

Farkh, R. and Aljaloud, K. (2023). Vision navigation based pid control for line tracking robot. *Intelligent Automation & Soft Computing*, vol. 35, no. 1, pp. 901–911.

Grover, A.K. and Ashraf, M.H. (2024). Leveraging autonomous mobile robots for industry 4.0 warehouses: a multiple case study analysis. *The International Journal of Logistics Management*, vol. 35, no. 4, pp. 1168–1199.

Hercik, R., Byrtus, R., Jaros, R. and Koziorek, J. (2022). Implementation of autonomous mobile robot in smartfactory. *Applied Sciences*, vol. 12, no. 17, p. 8912.

Huang, X., Zhang, N., Fan, K., Zhong, X. and Wu, Q. (2022). Improved a* method for high efficiency forklift path planning. In: *2022 IEEE International Conference on e-Business Engineering (ICEBE)*, pp. 136–140. IEEE.

Keith, R. and La, H.M. (2024). Review of autonomous mobile robots for the warehouse environment. *arXiv preprint arXiv:2406.08333*.

Lee, S.-J., Tewolde, G., Lim, J. and Kwon, J. (2015). Qr-code based localization for indoor mobile robot with validation using a 3d optical tracking instrument. In: *2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 965–970. IEEE.

Liu, Y., Wang, S., Xie, Y., Xiong, T. and Wu, M. (2024). A review of sensing technologies for indoor autonomous mobile robots. *Sensors*, vol. 24, no. 4, p. 1222.

Lu, Y., Ma, H., Smart, E. and Yu, H. (2021). Real-time performance-focused localization techniques for autonomous vehicle: A review. *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6082–6100.

Mecalux (2023). Autonomous mobile robots: warehouse applications and uses. *Mecalux Insights*.
Available at: `https://www.mecalux.com`

Permana, S.H., Bintoro, K.Y., Arifitama, B., Syahputra, A. *et al.* (2018). Comparative analysis of pathfinding algorithms a*, dijkstra, and bfs on maze runner game. *IJISTECH (International J. Inf. Syst. Technol*, vol. 1, no. 2, p. 1.

Qu, Y., Yang, M., Zhang, J., Xie, W., Qiang, B. and Chen, J. (2021). An outline of multi-sensor fusion methods for mobile agents indoor navigation. *Sensors*, vol. 21, no. 5, p. 1605.

TechBriefs (2023). Optimizing warehouse logistics with autonomous mobile robots. *TechBriefs Robotics Supplement*.
Available at: `https://www.techbriefs.com/component/content/article/tb/supplements/md/briefs/46685`

Yang, B., Li, W., Wang, J., Yang, J., Wang, T. and Liu, X. (2020). A novel path planning algorithm for warehouse robots based on a two-dimensional grid model. *IEEE Access*, vol. 8, pp. 80347–80357.

Zhao, J., Liu, S. and Li, J. (2022). Research and implementation of autonomous navigation for mobile robots based on slam algorithm under ros. *Sensors*, vol. 22, no. 11, p. 4172.

Appendix A
Concept Evaluation Justification

Weight Justification

- **Cost (20%):** High importance due to budget constraints.
- **Implementation Complexity (15%):** Significant as it impacts development time and resource allocation.
- **Scalability (10%):** Important but less critical than cost and complexity for the project's scope.
- **Maintenance (10%):** Essential to ensure long-term reliability and ease of upkeep.
- **Accuracy of Navigation (10%):** Necessary for operational efficiency but balanced with other criteria.
- **Integration with WMS (10%):** Crucial for real-time operations and task management.
- **Power Requirements (5%):** Important to ensure efficient energy use and longer operational hours.
- **Battery Life (5%):** Moderate importance to minimize downtime.
- **Sensor Requirements (5%):** Important for initial setup and ongoing management.
- **Pathfinding Efficiency (10%):** Essential for ensuring the robot can navigate the warehouse quickly and efficiently.

Explanation and Justification of Scores

- **Cost:** Concept 1 is significantly more cost-effective than Concept 2 due to fewer additional components.
- **Implementation Complexity:** Concept 1 has a simpler implementation, making it easier to develop and deploy.
- **Scalability:** While Concept 2 offers higher scalability, Concept 1 is sufficiently scalable for the intended application.
- **Maintenance:** Concept 1 requires less maintenance due to its simpler design and fewer components.
- **Accuracy of Navigation:** Concept 2 provides higher accuracy, but Concept 1 is adequate for the project requirements.
- **Integration with WMS:** Both concepts can be integrated with the WMS.
- **Power Requirements:** Concept 1 has lower power requirements due to fewer sensors and simpler processing needs.
- **Battery Life:** Concept 1 has better battery life because of lower power consumption.
- **Sensor Requirements:** Concept 1 requires fewer and simpler sensors, making it more straightforward to manage.

- **Pathfinding Efficiency:** Concept 1's use of the A* algorithm makes it more computationally efficient and quicker in finding optimal paths compared to Concept 2's Dijkstra algorithm.

Appendix B
Software Libraries and Modules

- **Flask**: Flask is used to create a lightweight web server, enabling communication between the robot and the server. It handles HTTP requests, such as sending task updates from the robot or receiving sensor data, and processes routes like '/receive_tasks' and '/update_log'.
- **Requests**: This library is responsible for sending HTTP requests from the robot to the server. It's used for operations like posting task completions or obstacle alerts back to the web interface.
- **Threading**: Threading is used to run multiple tasks concurrently. This allows the robot to detect obstacles, execute tasks, and communicate with the server without blocking other operations.
- **OpenCV (cv2)**: This library processes images and video streams, such as capturing camera feed. It's used for detecting QR codes and converting images to grayscale for easier processing.
- **Pyzbar**: Pyzbar decodes QR codes captured by the robot's camera. This allows the robot to interpret data from QR codes it encounters, helping it navigate or update its task list.
- **Time**: Time provides delay functions and timestamps, used for sensor initialisation, adding delays between operations, or handling camera warm-up times.
- **RPi.GPIO**: This library is used to control the Raspberry Pi's GPIO pins. It manages hardware components like motors and sensors, ensuring the robot can interact with its surroundings.
- **TRSensor**: Manages the line-following sensor, reading data from the sensor array to help the robot follow lines or paths in its environment.
- **PCA9685**: This module controls PWM signals for motors and servos, enabling precise control of motor speeds and directions via I2C communication.
- **Ultrasonic_Ranging**: Manages the ultrasonic sensor for obstacle detection, helping the robot calculate distances to nearby objects and avoid collisions.
- **Picamera**: Provides access to the Raspberry Pi camera, enabling the robot to capture images or video streams for tasks like QR code detection or environmental analysis.
- **smbus**: This library manages I2C communication, allowing the Raspberry Pi to interact with devices like motor drivers or sensors connected via I2C.
- **Adafruit_PCA9685**: Specifically used to interface with the PCA9685 PWM driver, this library controls the servos and motors, ensuring smooth and accurate movement.

- **json**: Manages the encoding and decoding of JSON data, which is the format used for communication between the robot and the server, enabling structured data exchange.

Appendix C
Gannt Chart

Stage/Task Name	Expected Start Date	Expected End Date	Duration (Days)	Deadline
Stage 1: Planning and Analysis	2024/03/16	2024/06/11	87	
Literature Review	2024/03/16	2024/05/31	14	
Customer Requirements Analysis	2024/05/15	2024/06/10	14	
Engineering Characteristics Definition	2024/05/16	2024/06/11	14	
Stage 2: Design	2024/06/10	2024/08/09	60	
Concept Generation	2024/06/10	2024/07/07	14	2024/05/10
Concept Selection	2024/07/07	2024/07/15	14	
Design Refinement	2024/07/15	2024/08/09	14	
Stage 3: Development	2024/08/09	2024/09/27	49	
Prototype Development	2024/08/09	2024/09/27	49	2024/06/18
Stage 4: Testing and Evaluation	2024/06/18	2024/07/15	27	
Testing and Data Collection	2024/06/18	2024/07/01	14	
Result Analysis	2024/07/02	2024/07/15	14	
Stage 5: Reporting and Finalization	2024/06/08	2024/11/10	155	
Progress Report Preparation	2024/06/08	2024/07/25	48	2024/07/26
Final Report Drafting	2024/07/16	2024/10/06	83	2024/10/10
Final Reporting	2024/07/16	2024/10/24	100	2024/10/24
Project Progress Presentations Prep	2024/08/02	2024/08/04	3	2024/08/04
Final Report Revision	2024/10/07	2024/10/17	11	2024/10/17
Presentation and Oral Exam Preparation	2024/10/25	2024/11/10	17	2024/11/10

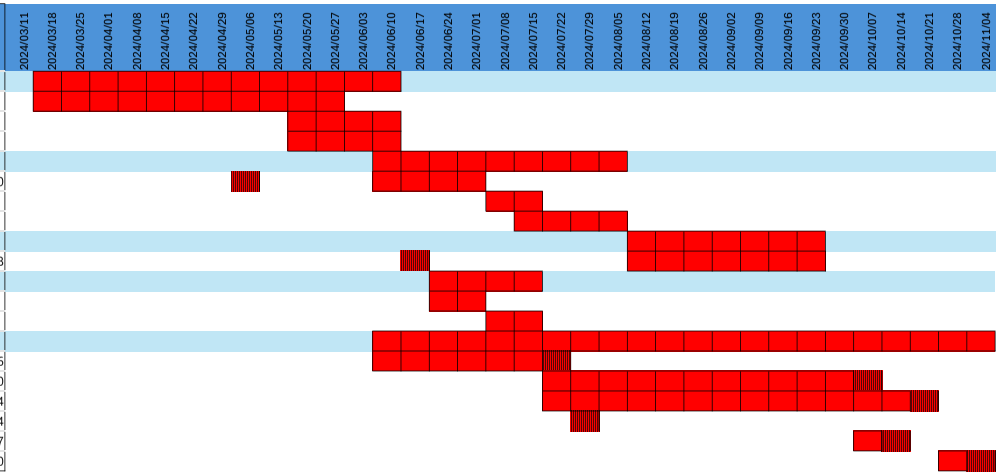
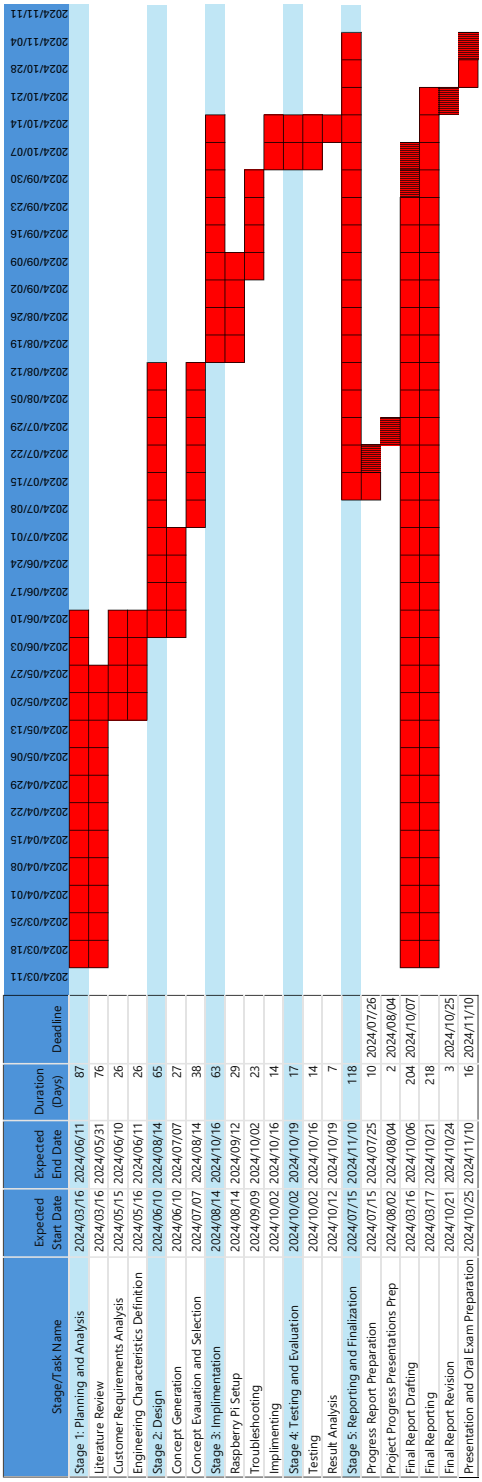


Figure C.1: Proposed Project Timeline



Stage/Task Name	Expected Start Date	Expected End Date	Duration (Days)	Deadline
Stage 1: Planning and Analysis	2024/03/16	2024/06/11	87	
Literature Review	2024/03/16	2024/05/31	76	
Customer Requirements Analysis	2024/05/15	2024/06/10	26	
Engineering Characteristics Definition	2024/05/16	2024/06/11	26	
Stage 2: Design	2024/06/10	2024/08/14	65	
Concept Generation	2024/06/10	2024/07/07	27	
Concept Evaluation and Selection	2024/07/07	2024/08/14	38	
Stage 3: Implimentation	2024/08/14	2024/10/16	63	
Raspberry Pi Setup	2024/08/14	2024/09/12	29	
Troubleshooting	2024/09/09	2024/10/02	23	
Implmenting	2024/10/02	2024/10/16	14	
Stage 4: Testing and Evaluation	2024/10/02	2024/10/19	17	
Testing	2024/10/02	2024/10/16	14	
Result Analysis	2024/10/12	2024/10/19	7	
Stage 5: Reporting and Finalization	2024/07/15	2024/11/10	118	
Progress Report Preparation	2024/07/15	2024/07/25	10	2024/07/26
Project Progress Presentations Prep	2024/08/02	2024/08/04	2	2024/08/04
Final Report Drafting	2024/03/16	2024/10/06	204	2024/10/07
Final Reporting	2024/03/17	2024/10/21	218	
Final Report Revision	2024/10/21	2024/10/24	3	2024/10/25
Presentation and Oral Exam Preparation	2024/10/25	2024/11/10	16	2024/11/10

Figure C.2: Actual Project Timeline

Appendix D
Techno-Economic Analysis

D.1 Budget

In this section, a comparison between the proposed budget and the actual budget is presented. Any variances between the two will be discussed, with explanations for additional expenses or savings that occurred during the project. This includes details on resource allocation and cost-saving strategies.

Table D.1: Proposed Budget Table

Activity	**Engineering Time (hr)**	**Engineering Cost (R)**	**Running Costs (R)**	**Facility Use (R)**	**Material Cost (R)**	**Total Cost (R)**
Stage 1: Planning and Analysis	60	24000	0	0	0	**24000**
Stage 2: Design	65	26000	0	0	0	**26000**
Concept Generation	25	10000	0	0	0	10000
Concept Selection	10	4000	0	0	0	4000
Design Refinement	20	8000	0	0	0	8000
Stage 3: Development	125	50000	200	300	5200	**55700**
Prototype Development	125	50000	200	300	5200	55700
Stage 4: Testing and Evaluation	30	12000	200	200	200	**12600**
Testing and Data Collection	15	6000	200	200	200	6600
Result Analysis	15	6000	0	0	0	6000
Stage 5: Reporting and Finalization	170	68000	0	0	0	**68000**
Progress Report Preparation	50	20000	0	0	0	20000
Final Report Drafting	90	36000	0	0	0	36000
Final Reporting	110	44000	0	0	0	44000
Project Progress Presentations Prep	3	1200	0	0	0	1200
Final Report Revision	12	4800	0	0	0	4800
Presentation and Oral Exam Preparation	18	7200	0	0	0	7200
Total						**186300**

Table D.2: Actual Budget Table

Activity	**Engineering Time (hr)**	**Engineering Cost (R)**	**Running Costs (R)**	**Facility Use (R)**	**Material Cost (R)**	**Total Cost (R)**
Stage 1: Planning and Analysis	**60**	**24000**	**0**	**0**	**0**	**24000**
Literature Review	30	12000	0	0	0	12000
Customer Requirements Analysis	15	6000	0	0	0	6000
Engineering Characteristics Definition	15	6000	0	0	0	6000
Stage 2: Design	**85**	**34000**	**200**	**0**	**0**	**34200**
Concept Generation	35	14000	200	0	0	15000
Concept Evaluation and Selection	50	20000	0	0	0	20000
Stage 3: Implementation	**125**	**50000**	**200**	**0**	**4081.35**	**54281.35**
Raspberry Pi Setup	25	10000	0	0	0	10000
Troubleshooting	100	40000	0	0	0	40000
Stage 4: Testing and Evaluation	**35**	**14000**	**400**	**0**	**0**	**14400**
Testing	25	10000	0	0	0	10000
Result Analysis	10	4000	0	0	0	4000
Stage 5: Reporting and Finalization	**145**	**58000**	**0**	**0**	**0**	**58000**
Progress Report Preparation	10	4000	0	0	0	4000
Final Report Drafting	60	24000	0	0	0	24000
Final Reporting	25	10000	0	0	0	10000
Project Progress Presentations Prep	10	4000	0	0	0	4000
Final Report Revision	10	4000	0	0	0	4000
Presentation and Oral Exam Preparation	30	12000	0	0	0	12000
Total						**184881.35**

D.1.1 Discussion

The budget analysis highlights several differences between the initial and actual cost projections for the project. These variations are primarily attributed to changes in material costs and time spent on specific tasks.

D.1.1.1 Material Costs

Interestingly, the actual material costs were lower than the proposed budget, even though two robots and two pairs of batteries had to be purchased due to hardware failures. The proposed budget overestimated the material costs, particularly for components like sensors and motor drivers, which were anticipated to be more expensive. Despite needing to replace hardware, the overall costs remained below the initial projections. This demonstrates that the planning phase overestimated the required materials.

D.1.1.2 Time Allocation

The overall time spent on the project closely matches the initial projections, but the distribution of time across the stages varied. Stage 3 (Development and Implementation) required more time due to hardware problems, specifically related to the motor drivers. This extended the time required for system integration and troubleshooting. However, final stages like Stage 5 (Reporting and Finalization) required less time than initially planned, as much of the report structure had already been developed earlier in the project.

D.2 Planning (Time Management)

The project faced some delays during the development and implementation stages due to unexpected technical issues. While the planning and design phases stayed mostly on track, the literature review and customer requirements analysis took longer than planned. This extra time allowed for a deeper understanding of the project and helped improve the design. Although this caused some initial delays, it led to better decisions in the concept generation and design refinement stages.

The biggest delays happened during the development phase, especially with setting up the Raspberry Pi and solving hardware problems. These issues took more time to fix than expected, but they were important for making sure the system worked properly. Once these problems were solved, the implementation moved forward, though it still took longer than planned. However, testing and data collection were completed faster than expected because the system worked well during testing, which helped make up for some lost time.

The reporting and finalization stages also took longer than planned. Writing the final report needed more time as adjustments were made to reflect the issues faced during testing and implementation. Despite these delays, the extra time helped ensure that everything was well-documented. Overall, while the project took longer than expected, the extra time was needed to deal with technical

challenges and complete the project successfully.

Discussion:

[Provide a brief analysis of any discrepancies between the planned and actual timelines. Include reasons for delays or acceleration and their impact on the project.]

D.3 Technical Impact

This project made several technical contributions significant for the field of autonomous robotics in warehouse environments. The integration of pathfinding algorithms, sensor systems for obstacle avoidance, and real-time communication demonstrates an innovative approach to warehouse automation. The key technical impacts include:

Technical Contributions:

- Development of an efficient navigation system based on the A* algorithm, optimized for dynamic warehouse environments.
- Implementation of real-time obstacle detection and avoidance using ultrasonic sensors.
- Creation of a web-based interface for task selection and monitoring, allowing real-time feedback and updates.
- Custom-designed motor compensation techniques to mitigate hardware issues and ensure smoother robot operation.

Discussion: The project's technical contributions are substantial, especially in terms of enhancing the efficiency and accuracy of autonomous navigation. By implementing an optimized pathfinding algorithm and a versatile task management system, the project demonstrated a high level of technical innovation. These advancements have the potential to improve not only individual robot operations but also multi-robot coordination in large-scale warehouses.

D.4 Return on Investment

The project offers both short-term and long-term value in terms of technical innovations and future development potential. While it involved significant initial investment in time and resources, the long-term benefits are promising in terms of improved efficiency and scalability.

Short-term Value:

- Immediate improvements in navigation and task execution for small-scale warehouses.

- Deployment of a cost-effective guidance system that can integrate with existing Warehouse Management Systems (WMS).
- Serving as a test platform for further research and development in autonomous systems.

Long-term Value:

- The system's scalability suggests it can be adapted for larger warehouses, offering long-term operational cost savings.
- It can serve as a foundation for more complex multi-robot systems as sensor technology and algorithms advance.
- Potential to become a commercial product for warehouse automation, offering ongoing revenue from licensing or direct sales.

Discussion: While the short-term benefits focus on operational improvements, the long-term value lies in the potential to evolve into a scalable commercial solution. The system's modular design and adaptability to various warehouse layouts suggest that further development could lead to significant productivity gains.

D.5 Potential for Commercialization

The project shows strong potential for commercialization, particularly in the growing warehouse automation market. With further development, it could be marketed as a cost-effective solution for small- to medium-sized warehouses.

Commercialization Potential:

- The system can be marketed as a scalable solution for warehouse navigation, appealing to businesses with limited budgets for automation.
- Integration with WMS platforms enhances marketability by reducing the need for companies to invest in entirely new systems.
- The system's ability to operate without pre-installed infrastructure (such as magnetic strips) offers a unique selling point.
- Expanding the system to include features like load carrying and item scanning would further increase its appeal in logistics.

Discussion: With the right investment, this system could become commercially viable. Its flexibility, scalability, and potential for integration make it attractive for businesses seeking warehouse efficiency improvements. Its modular design also allows for continuous improvements, making it a sustainable product for future market demand.

Appendix E
Safety

E.1 Safety Report 1

This section includes the Safety Report for the warehouse robot programming and implementation work in the Mechatronics Lab.

Safety Report: Mechatronics Lab

September 16, 2024

Programming and Implementation of a Warehouse Robot

Date:	September 16, 2024
Student & SU nr:	H. Aakmann-Visher (24660051)
Student contact nr:	0827596625
Supervisor:	Mr. P.E.J. Trinchero
Head of safety:	Cobus Zietsman

Emergency Contacts

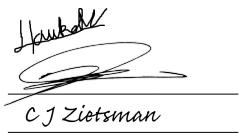
Contact:	**Room nr.**	**Work nr.**	**Cell nr.**
C Zietsman	M212	021 808 4275	-
Campus Security	-	021 808 2333	WhatsApp: 082 808 233
Fire Brigade	-	021 808 8888	-
Ambulance	-	021 883 3444	-

Signatures

Student: (H. Aakmann-Visher)

Supervisor: (Mr. P.E.J. Trinchero)

Head of Safety: (Cobus Zietsman)



Pressure Vessels or Pipes (check relevant box):

- ❑No pressure vessels or pipes with pressure in excess of 50kPa are involved in this project.
- ❑Pressure vessels or pipes in excess of 50kPa are involved – additional signature and report required.

Hot work / working at heights / confined entry / excavation (check relevant box):

- ❑No hot work / working heights / confined entry / excavation work involved in this project.
- ❑Hot work / working heights / confined entry / excavation work (underline relevant work type(s)) involved in this project – additional signature and report required.

Overview of Programming and Implementation

- **Type of Work:** Programming and implementation of the warehouse robot's systems in the Mechatronics Lab.
- **Location:** Mechatronics Lab, M&M Building.
- **Equipment to be Used:** Multimeters, oscilloscopes, power supplies, robotic components (motors, sensors, controllers), and programming interfaces (monitor, keyboard & mouse).
- **Sample Geometry:** Not applicable as this involves a complex robotic system rather than standard samples.
- **Robot to be Tested:** Alphabot (as shown in Figure 1).

Figure 1: The Alphabot robot to be used for programming and implementation.

General Lab Safety Instructions

- Wear closed shoes and appropriate PPE (e.g., safety glasses, gloves) at all times.
- No afterhours work alone.
- Follow proper procedures for turning equipment on and off.
- Procedures for power outages: Switch all equipment off at the wall socket.
- Emergency equipment must be located and easily accessible.
- Emergency exits must be known.
- No food or drink in the laboratory.
- Safety report must be visible and accessible during programming and implementation.
- No equipment or workstations may be left unattended.

Fire Safety

- No earphones allowed while working in lab or workshop areas.
- Identify and list all fire risks.
- Evacuation routes and plans relevant to the work area must be included.
- For any hot work or special permits, ensure relevant forms are completed and approved.

- Ensure that the Mechatronics Lab is equipped with accessible emergency exits and fire safety equipment.

In the event of a fire, follow the evacuation plan as shown in Appendix A. The nearest exits are highlighted, and the plan includes important safety information and emergency contacts.

Activity-Based Risk Assessment

Activity	**Risk**	**Risk Type (P/E)**	**Mitigating Steps**	**Classification of Risk Severity**
Programming the robot	Eye strain, repetitive strain injury	P	Take regular breaks, ensure ergonomic setup	Acceptable
Soldering electronic components	Burns, inhalation of fumes	P	Use fume extractor, wear protective gloves and goggles	High
Transporting components between labs	Tripping, equipment damage	P/E	Use carts for transport, ensure clear paths, seek assistance	Possible

Design and Housekeeping Measures

- Use personal protective equipment (PPE).
- Maintain good housekeeping at all times.
- Return all tools and equipment to their designated places.
- Dispose of waste properly.
- Report any damaged or broken components.
- Ensure the Mechatronics Lab is kept organized, with all setups dismantled and stored properly after use.

General Housekeeping

- Ensure the workspace is clear and clean.
- No personal belongings left behind.
- Follow specific steps for your laboratory setup.

Disciplinary Actions

Non-compliance with safety regulations or procedures will result in disciplinary action. Three warnings will result in a month's revocation of lab access.

Appendix A: Emergency Evacuation Plans

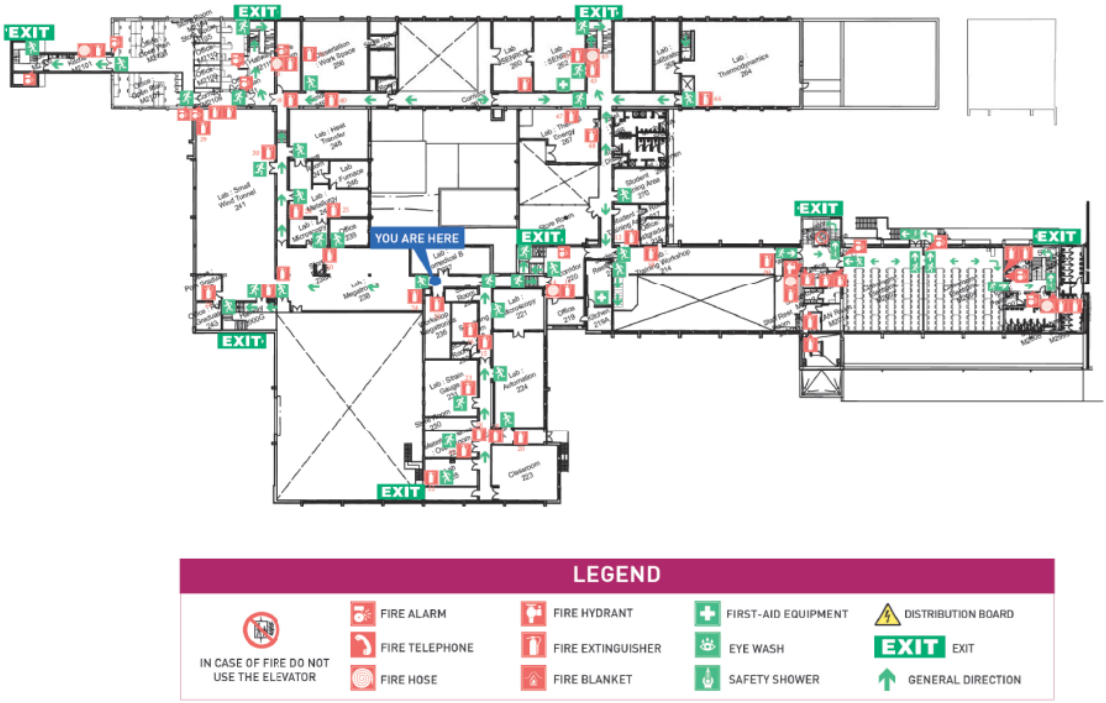


Figure 2: Emergency Evacuation Plan for Mechanical & Mechatronic Engineering, Level 1

E.2 Safety Report 2

This section includes the Safety Report for the additional safety considerations during the testing phase.

Safety Report: Classroom 223

September 16, 2024

Testing of a Warehouse Robot

Date:	September 16, 2024
Student & SU nr:	H. Aakmann-Visher (24660051)
Student contact nr:	0827596625
Supervisor:	Mr. P.E.J. Trinchero
Head of safety:	Cobus Zietsman

Emergency Contacts

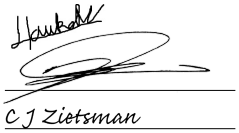
Contact:	**Room nr.**	**Work nr.**	**Cell nr.**
C Zietsman	M212	021 808 4275	-
Campus Security	-	021 808 2333	WhatsApp: 082 808 233
Fire Brigade	-	021 808 8888	-
Ambulance	-	021 883 3444	-

Signatures

Student: (H. Aakmann-Visher)

Supervisor: (Mr. P.E.J. Trinchero)

Head of Safety: (Cobus Zietsman)



Pressure Vessels or Pipes (check relevant box):

- ❑No pressure vessels or pipes with pressure in excess of 50kPa are involved in this project.
- ❑Pressure vessels or pipes in excess of 50kPa are involved – additional signature and report required.

Hot work / working at heights / confined entry / excavation (check relevant box):

- ❑No hot work / working heights / confined entry / excavation work involved in this project.
- ❑Hot work / working heights / confined entry / excavation work (underline relevant work type(s)) involved in this project – additional signature and report required.

Overview of Testing

- **Type of Work:** Testing of the warehouse robot in Classroom 223, simulating a real-world environment.
- **Location:** Room 223, M&M Building.
- **Equipment to be Used:** Robotic components, testing barriers.
- **Sample Geometry:** Not applicable as this involves a complex robotic system rather than standard samples.
- **Robot to be Tested:** Alphabot (as shown in Figure 1).

Figure 1: The Alphabot robot to be used for testing.

General Lab Safety Instructions

- Wear closed shoes and appropriate PPE (e.g., safety glasses, gloves) at all times.
- No afterhours work alone.
- Follow proper procedures for turning equipment on and off.
- Procedures for power outages: Switch all equipment off at the wall socket.
- Emergency equipment must be located and easily accessible.
- Emergency exits must be known.
- No food or drink in the classroom during testing.
- Safety report must be visible and accessible during testing.
- No equipment or test may be left unattended.

Fire Safety

- No earphones allowed while working in lab or workshop areas.
- Identify and list all fire risks.
- Evacuation routes and plans relevant to the work area must be included.
- For any hot work or special permits, ensure relevant forms are completed and approved.
- Ensure that Room 223 is equipped with accessible emergency exits and fire safety equipment.

In the event of a fire, follow the evacuation plan as shown in Appendix A. The nearest exits are highlighted, and the plan includes important safety information and emergency contacts.

Activity-Based Risk Assessment

Activity	**Risk**	**Risk Type (P/E)**	**Mitigating Steps**	**Classification of Risk Severity**
Testing robot movement	Collision, tripping	P/E	Clear the testing area, use barriers, monitor closely	Substantial
Assembling robotic components	Injuries from tools, electrical shock	P/E	Wear PPE, follow tool safety protocols, ensure proper insulation	Possible
Transporting components between labs	Tripping, equipment damage	P/E	Use carts for transport, ensure clear paths, seek assistance	Possible

Design and Housekeeping Measures

- Use personal protective equipment (PPE).
- Maintain good housekeeping at all times.
- Return all tools and equipment to their designated places.
- Dispose of waste properly.
- Report any damaged or broken components.
- Ensure Room 223 is kept organized, with all simulation setups dismantled and stored properly after use.

General Housekeeping

- Ensure the workspace is clear and clean.
- No personal belongings left behind.
- Follow specific steps for your testing setup.

Disciplinary Actions

Non-compliance with safety regulations or procedures will result in disciplinary action. Three warnings will result in a month's revocation of lab access.

Appendix A: Emergency Evacuation Plans

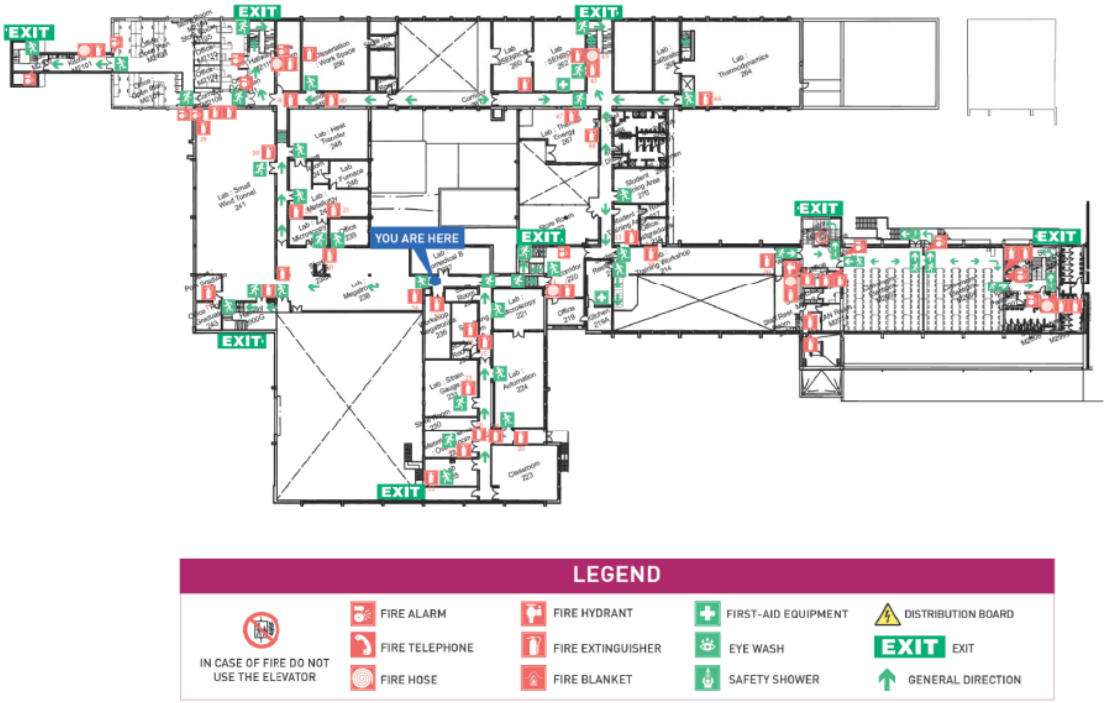


Figure 2: Emergency Evacuation Plan for Mechanical & Mechatronic Engineering, Level 1

Appendix F
End of Life

F.1 Project and Research

1. **Research Documentation and Dissemination**

   All findings, data, and analyses will be compiled into a comprehensive final report. Additionally, results will be shared through a poster or video presentation. This ensures thorough documentation for future reference, modification, and dissemination to a wider audience.

2. **Prototype and Hardware Management**

   The prototype robot and hardware will be evaluated to determine potential future uses. Since this project is a prototype, many components can be reused or modified for future research or educational projects. The robot may also be integrated into related research or kept securely for future upgrades or reference.

3. **Software and Intellectual Property**

   All software and algorithms developed during the project will be archived with proper documentation. Given this is a prototype, the software can be reused or modified in future iterations. The potential for open-sourcing or protecting it as intellectual property will be evaluated.

4. **Future Research and Development**

   The findings from this project will identify areas for further investigation. As a prototype, the insights gained will guide future research and system upgrades. A proposal for future work may be developed based on the outcomes.

5. **Stakeholder Engagement**

   A detailed report, including project outcomes and future plans for the prototype, will be shared with stakeholders and sponsors. The report will highlight the project's impact and suggest how the results may be leveraged in future initiatives.

6. **Community and Outreach Initiatives**

   The team will engage with the broader community to share knowledge from this prototype development. This may include public lectures, educational outreach, or social media engagement to highlight the project's contributions to robotics.

F.2 Hardware

1. **Recycling of Components**

   The potential for recycling various components will be explored, including electronic parts, metals, and plastics. Partnerships with local recycling facilities will be established to ensure responsible disposal of materials.

2. **Refurbishment and Reuse**

   As this project is a prototype, many hardware components can be reused in future projects. These parts will be assessed for refurbishment or repurposing in future research to extend their lifecycle and reduce waste.

3. **Safe Disposal**

   For components that cannot be reused or recycled, safe disposal methods will be employed, especially for hazardous materials like batteries. This ensures minimal environmental impact and safety.

4. **Donation for Educational or Research Purposes**

   If the prototype or parts of it are not reused, they may be donated to educational institutions or research organizations for learning or further study, extending their utility.