Palestine Polytechnic University

College of Information Technology and Computer Engineering

Department of Computer Engineering

WareBot: A Flexible Autonomous Robot for Smart Warehouse Management in Small and Medium-Sized Enterprises

Osama Alhroub
Issa Warasna
Ismail Rjoub

Supervised by:
Dr. Hani Salah

To fulfill the requirements for a bachelor's degree in Computer Systems Engineering

February 2026

Certification and Anti-Plagiarism Declaration

This is to certify that the graduation project entitled *"WareBot: A Flexible Autonomous Robot for Smart Warehouse Management in Small and Medium-Sized Enterprises"*, conducted under the supervision of Dr. Hani Salah, was prepared by the undersigned students in partial fulfillment of the requirements for the degree of Bachelor of Engineering in Computer Systems Engineering.

We affirm that no part of this work has been reproduced illegally or constitutes plagiarism. All referenced materials have been used solely to support and argue the project's ideas and have been cited properly. We further certify that we will not commit any act of plagiarism, cheating, or other academic integrity violations. We accept full responsibility for any consequences should a violation of this declaration be proven.

Any use of Artificial Intelligence (AI) tools in the preparation of this project has been clearly declared and documented in the related appendix at the end of the report, in accordance with the PPU guidelines.

Date: _____

Graduation Project Group

Name: Issa Warasnah **Signature:** _____

Name: Osama Hroub **Signature:** _____

Name: Ismael Rjoub **Signature:** _____

Abstract

This project introduces Warebot, a multi-robot autonomous system designed to support small and medium enterprises in achieving daily warehouse operations without modifying the existing infrastructure. The need for this project arises from several problems that directly affect warehouses. Many existing solutions are highly dependent on the warehouse environment, which leads to challenges such as requiring large modifications. This increases the cost of changes and may force the environment to be closed during upgrades. Our system addresses these problems by removing infrastructure-related dependencies and shifting the work to the robots' software, with minimal reliance on the warehouse infrastructure.

Our system is deployed on Neobotix MP-400 robots equipped with advanced features and sensors such as LiDAR and IMU to sense, recognize, localize, and navigate the environment. By using SLAM (Simultaneous Localization and Mapping) with a single robot, we can create a global map that is later shared among all robots. Warehouse operations are controlled through Multi-Robot Task Allocation (MRTA) and priority matrices based on battery level, robot location, and robot status. This makes the system efficient in utilizing robot software and fast in completing warehouse operations. While robots are moving, they use global path planning that continues to run and is updated using the A* algorithm to ensure the robots can be aware of changes in the environment, such as moved shelves or new obstacles that appear suddenly. This enables robots to adapt to changes, generate new paths, and successfully reach their destinations. Obstacle avoidance is managed by the Dynamic Window Approach (DWA), which enables real-time obstacle avoidance.

To face multi-robot challenges, our system implements smart techniques such as robot awareness and cooperative perception. These techniques allow robots to detect each other and treat other robots as dynamic obstacles, making each robot adapt its movements correctly based on what is on the shared map. When conflicts or deadlocks occur, the robots can solve that using continuous replanning with obstacle avoidance, which generates new path planning during movement. These ensure good coordination between robots on the same map.

Communication between the robots and the server is performed using lightweight MQTT protocols, enabling low-latency data exchange within an acceptable time. Additionally, Warebot has a web interface that allows workers to track, monitor, and control the overall system.

WareBot presents a strong solution since it works in a wide range of environments and adapts to any changes, and is scalable when the number of robots increases. All these techniques and algorithms that are implemented in our system make it flexible and effective for managing daily warehouse operations.

الملخص

يقدّم هذا المشروع نظامًا ذكيًا متعدد الروبوتات باسم WareBotيهدف إلى أتمتة عمليات المستودعات دون الحاجة إلى إجراء أي تعديلات على البنية التحتية القائمة. تنبع أهمية هذا العمل من التحديات التي تواجه أنظمة الأتمتة التقليدية، والتي تعتمد على بنى تحتية جامدة، وتكاليف تشغيل مرتفعة، وضعف القدرة على التكيف مع البيئات الديناميكية. يعالج نظام WareBot هذه الإشكاليات من خلال توفير إطار أتمتة مرن وقابل للتوسع يعتمد على فريق من الروبوتات الذاتية التعاونية.

يعتمد النظام على روبوتات Neobotix MP-400 المزوّدة بمستشعرات الليدار (LiDAR) ووحدات القياس (IMU) للإدراك المكاني وتحديد الموقع والتنقل الذاتي في البيئات غير المعروفة. يقوم روبوت بتنفيذ خوارزمية التحديد المتزامن للموقع وبناء الخريطة (SLAM) لإنشاء خريطة عالمية للمستودع، يتم مشاركتها مع باقي الروبوتات لتحقيق توطين موحّد وتنقل منسق. يتم توزيع المهام التشغيلية، مثل استرجاع الرفوف وتسليم الطلبات، باستخدام نظام تخصيص المهام متعدد الروبوتات(MRTA) ، بما يضمن كفاءة التنفيذ والتنسيق بين الروبوتات. ويُعتمد في التخطيط الشامل للمسارات على خوارزمية *A، بينما يتم تفادي العوائق باستخدام خوارزمية النافذة الديناميكية.(DWA)

ولمعالجة تحديات التفاعل بين الروبوتات، يعتمد النظام على مبدأ الوعي المتبادل بين الروبوتات (Robot Awareness)والإدراك التعاوني، حيث يقوم كل روبوت باكتشاف الروبوتات الأخرى باعتبارها عوائق ديناميكية، ويتم تكييف الحركة بشكل آني وفقًا لذلك. كما يتم حل حالات التعارض والانسداد (Deadlock) من خلال آليات إعادة التخطيط المستمر (Continuous Replanning) ، مما يتيح إعادة تهيئة المسارات ديناميكيًا وتحقيق تنسيق آمن داخل البيئات المشتركة.

يعتمد النظام في الاتصال بين الروبوتات والخادم المركزي على بروتوكول MQTTخفيف الوزن، مما يضمن تبادل بيانات موثوقًا بزمن تأخير منخفض. كما تم اعتماد بنية حوسبة هجينة (الحافة–السحابة) لتحقيق توازن بين الاستجابة الزمنية الفورية وكفاءة المعالجة. ويوفّر النظام واجهة ويب تفاعلية تتيح للمشغّلين متابعة حالة الروبوتات، وتخصيص المهام، ومراقبة أداء النظام في الزمن الحقيقي.

يُظهر نظام WareBot نموذجًا فعالًا ومرنًا وقابلًا للتوسع لأتمتة المستودعات الذكية، حيث يدعم التوسع التدريجي والتكيف مع التغيرات البيئية دون الحاجة إلى بنية تحتية ثابتة. ومن خلال الدمج بين التنسيق اللامركزي، والتخطيط الذكي، وإعادة التخطيط المستمر، والاتصال الموثوق، يقدّم النظام منصة متقدمة لإدارة المستودعات ذاتية التشغيل بكفاءة عالية.

Contents

Table of Contents vii

List of Tables xi

List of Figures xii

List of Abbreviations xiii

1 Introduction 1
- 1.1 Preface ... 1
- 1.2 Problem Statement ... 1
- 1.3 Aims and Objectives ... 2
- 1.4 Requirements ... 3
  - 1.4.1 Functional Requirements ... 3
  - 1.4.2 Non-Functional Requirements ... 3
- 1.5 System Description ... 4
- 1.6 Limitations and Constraints ... 6
- 1.7 Implementation Timeline ... 7
- 1.8 Report Outline ... 8

2 Theoretical Background 9
- 2.1 Preface ... 9
- 2.2 Theoretical Background Concepts ... 9
  - 2.2.1 Simultaneous Localization and Mapping (SLAM) ... 9
  - 2.2.2 Multi Robot Task Allocation (MRTA) ... 9
  - 2.2.3 Hybrid Edge–On-Premises Computing ... 10
  - 2.2.4 MQTT Protocol and Broker ... 10
- 2.3 Literature Review ... 10
- 2.4 Summary ... 11

3 System Design 12
- 3.1 Preface ... 12
- 3.2 System Components and Design Alternatives ... 12
  - 3.2.1 Hardware Components ... 12
    - Robot Selection ... 12
    - Gripper Selection ... 13
    - Controller Selection ... 14

Motor Driver Selection 15
Camera Module 16
Shelf Material Selection 17
3.2.2 Software Components 17
Robot Operating System (ROS) 18
SLAM 18
Multi-robot Task Allocation 18
Web application and Database 18
Obstacle Avoidance and Path Planning 18
Deployment Options 20
Communication Protocols 20
Brokers 21
3.3 Conceptual System Description 22
3.4 Algorithms and Diagrams 24
3.4.1 Algorithms 24
Pathfinding using A* Search 24
Obstacle Avoidance using Dynamic Window Approach (DWA) . 25
Task Allocation in Multi-Robot System 26
3.4.2 Diagrams 27
Shelf Retrieval Process Steps 27
Backend Software Architecture 29
Database Schema and Data Flow 30
3.5 Schematic Diagrams 32
3.6 Summary 32

4 Implementation **33**
4.1 Preface 33
4.2 Hardware Implementation 33
4.2.1 Prototype Setup 33
4.2.2 Real Hardware Setup 34
Connections Components 34
Linear Actuators and Mechanical Arms 34
Shelf (Final Shape) 35
4.2.3 Final Integrated View 36
4.3 Software Implementation 36
4.3.1 Web Application Implementation 36
4.3.2 ROS 2 Layer Implementation 40
Real Robot Environment 40
Simulation Environment 41
4.3.3 Implementation Challenges 42
4.4 Summary 43

5 Testing Results and Discussion **44**
5.1 Initial Testing and Calibration 44

5.2 Validation and Testing 44
5.2.1 Unit Testing 45
5.2.2 Integration Testing 45
5.2.3 System-Level Testing 46
5.2.4 Validation Results 47
5.3 Detailed Analysis of Results and Experiments 48
5.3.1 Experimental Setup and Methodology 48
5.3.2 Overall System Performance 48
5.3.3 Payload Weight Impact 49
5.3.4 Orientation Robustness 49
5.3.5 Battery Performance Analysis 50
5.4 Error and Success Rate Calculations 51
5.4.1 Overall Performance Metrics 51
5.5 Justification of Obtained Results 52
5.5.1 Performance Results Interpretation 52
5.6 Summary 53

6 Conclusion and Future Work 54
6.1 Concluding Remarks 54
6.2 Achievement of Project Objectives 54
6.3 Future Work 55
6.3.1 Multi-Robot Fleet Management 55
6.3.2 Computer Vision Enhancements 55
6.3.3 System Performance Optimization 56
6.3.4 Advanced Autonomy 56
6.4 Closing Remarks 56

Appendices 60

Appendix A ROS 2 Jazzy Single Robot Simulation 62
A.1 Ubuntu 24.04 Installation 62
A.2 ROS 2 Jazzy Installation 63
A.3 Gazebo Harmonic Installation 63
A.4 Neobotix MP-400 Workspace Setup 64
A.5 Launch Simulation 64
A.6 SLAM and Mapping 64
A.7 Navigation Setup 65
A.8 RViz Configuration Save/Load 66
A.9 Troubleshooting Common Issues 67
A.10 Complete Workflow Summary 67

Appendix B ROS 2 Humble Single Robot Simulation 68

B.1 Ubuntu 22.04 Installation 68
B.2 ROS 2 Humble Installation 68
B.3 Gazebo Classic Installation 69

Appendix C ROS 2 Humble Multi-Robot Simulation 70
C.1 Prerequisites 70
C.2 Multi-Robot Workspace Setup 70
C.3 Environment Configuration 71
C.4 Create Cleanup Script 72
C.5 Launch Multi-Robot Simulation (4 Robots) 72
C.6 Launch RViz for Each Robot 73
C.7 Sending Navigation Goals to Multiple Robots 74

Appendix D Real Robot Deployment (ROS 2 Jazzy) 75
D.1 Network Configuration 75
D.2 Robot Hardware Bringup 76
D.3 Creating Map on Real Robot 76
D.4 Navigation on Real Robot 77

Appendix E Custom Nodes Development 79
E.1 Warehouse Task Manager Node 79
E.2 Multi-Robot Safety Nodes 79
Conclusion 81

Appendix F AI Appendix 82

List of Tables

1.1 Project Limitations and Constraints 7
1.2 Project's Gantt Chart 7

2.1 Comparison of International Warehouse Robotic Solutions 11

3.1 Comparison of Robot Platforms 13
3.2 Comparison of Gripper Types 14
3.3 Comparison of Controllers 15
3.4 Comparison of Motor Drivers 16
3.5 Comparison of Shelf Materials 17
3.6 Comparison of Global Path Planning Algorithms 19
3.7 Comparison of Obstacle Avoidance Algorithms 19
3.8 Comparison of IoT deployment models 20
3.9 Comparison of IoT communication protocols 21
3.10 Differences between brokers 22

5.1 Overall system performance metrics 48
5.2 Performance breakdown across different test conditions 49
5.3 Success rate breakdown by subsystem 51

List of Figures

1.1 Conceptual steps illustrating the process of delivering a pod to a warehouse worker 6

2.1 Hybrid Edge–On-Premises Architecture 10

3.1 Camera module mounted on the robot for AprilTag detection and shelf alignment 17
3.2 General Block Diagram 23
3.3 Conceptual Workflow Diagram. 24
3.4 Sequence diagram of the shelf retrieval process. 29
3.5 Backend Software Architecture Block Diagram 30
3.6 Database Schema 31
3.7 Data Flow of Telemetry 31
3.8 Schematic Diagram of WareBot Hardware Connections 32

4.1 Prototype setup stages 34
4.2 final connection layout 34
4.3 Complete mechanical arms of the robot 35
4.4 final shape 35
4.5 final veiw 36
4.6 System Monitoring Interfaces 37
4.7 Operational Management Interfaces 38
4.8 Warehouse Environment Management 39
4.9 Generated Occupancy Grid Map of the Warehouse 41
4.10 Four simulation worlds used to evaluate robot spawning and multi-robot behavior 42
4.11 Simulation phases showing robots navigating toward assigned goals. 42

5.1 Breadboard testing setup for initial component validation and calibration 45

List of Abbreviations

ABBREVIATION	FULL NAME / EXPANSION
A	Ampere(s)
A*	A-Star (Pathfinding Algorithm)
AAAI	Association for the Advancement of Artificial Intelligence
AAMAS	International Conference on Autonomous Agents and Multi-agent Systems
AI	Artificial Intelligence
AMR	Autonomous Mobile Robot
AMS1117	Adjustable Low Dropout Voltage Regulator
AS/RS	Automated Storage and Retrieval System
AWS IoT Core	Amazon Web Services IoT Core
CBS	Conflict-Based Search
CLI	Command Line Interface
CoAP	Constrained Application Protocol
D-SUN	DC-DC Buck Converter Module (Brand/Model)
DC	Direct Current
Dijkstra	Dijkstra's Shortest Path Algorithm
DTLS	Datagram Transport Layer Security
DWA	Dynamic Window Approach
EHR	Electronic Health Record
EPC	Electronic Product Code
ERP	Enterprise Resource Planning
ESP32	Espressif Systems 32 (Microcontroller)
Flash	Flash Memory
GB	Gigabyte
GHz	Gigahertz
Grafana	Open Source Analytics and Monitoring Platform
GUI	Graphical User Interface
HiveMQ	HiveMQ MQTT Broker (Software)
HTTPS	Hypertext Transfer Protocol Secure
IAM	Identity and Access Management
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMU	Inertial Measurement Unit
IoT	Internet of Things
KB	Kilobyte
LIDAR	Light Detection and Ranging
LM2596	Low Dropout Buck Converter (Voltage Regulator IC)

ABBREVIATION	FULL NAME / EXPANSION
mA	Milliampere(s)
MathWorks	Company behind MATLAB and Simulink (not an acronym, but used as proper noun)
MB	Megabyte
MCL	Monte Carlo Localization
MHz	Megahertz
Mosquitto	Eclipse Mosquitto (Open Source MQTT Broker)
MP-400	Mobile Platform 400 (Robot Model)
MP-500	Mobile Platform 500 (Robot Model)
MQTT	Message Queuing Telemetry Transport
MRTA	Multi-Robot Task Allocation
OASIS	Organization for the Advancement of Structured Information Standards
OS	Operating System
PC	Personal Computer
POS	Point of Sale
Prometheus	Open Source Monitoring and Alerting Toolkit
QoS	Quality of Service
RAM	Random Access Memory
RBAC	Role-Based Access Control
REST	Representational State Transfer
ROS	Robot Operating System
ROS-bridge	ROS Bridge Suite (Communication Interface)
rqt	ROS GUI Tools Framework
SLAM	Simultaneous Localization and Mapping
SMEs	Small and Medium-Sized Enterprises
SRAM	Static Random Access Memory
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
USB	Universal Serial Bus
V	Volt(s)
VFH	Vector Field Histogram
Wi-Fi	Wireless Fidelity
X.509	ITU-T Standard for Public Key Infrastructure
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Preface

In this chapter we will introduce the problem statement, identifying the problems within the warehouse environment and presenting our system as a solution to these problems. Next, we will outline the core objectives of our project, which must be achieved to address the fundamental problems we aim to solve. We will then detail the functional and non-functional requirements to define the specific functions our system performs. Following this, we will discuss the initial assumptions that must be established at the start of implementation and provide a description to outline how the system will operate. Finally, we will examine the constraints and challenges that limit the system's functionality.

1.2 Problem Statement

Many warehouses seek to increase the efficiency of their work by employing robotic systems, which work to accelerate their operational efficiency and complete all requirements completely and quickly. However, these warehouses still face problems and challenges that hinder the efficiency of their work or the proper implementation of their tasks.

Some current solutions depend heavily on the warehouse environment itself, such as building a fixed infrastructure on the warehouse floor such as magnetic tracks or floor fiducials. This dependency forces the system to operate within a fixed warehouse environment, making it difficult to customize and impractical for relocation. Any physical change in the warehouse environment necessitates manual adjustments to the robotics systems to accommodate these changes, which leads to operational disruption. In

multi-robot setups, coordination within fixed environments often reduces operational efficiency, as rigid point-to-point pathing logic causes resource contention and inefficient routing. Crucially, as the fleet grows, these centralized models frequently struggle with conflicts, inter-robot congestion, and collisions, ultimately hindering delivery times and overall system accuracy.

Addressing these challenges requires a dynamic multi-robot system that adapts to shifting environments autonomously, avoiding reliance on physical infrastructure to ensure flexible, collision-free operations.

1.3 Aims and Objectives

In this project, we propose a system that aims to provide the following features:

1. **Building an Intelligent Robot Navigation System**
    - The robots' movement on the warehouse floor must remain precise and reliable as they travel to a specific point.
    - Optimize the system's performance, allowing for the calculation of the optimal routes for the robots to take and ensuring accurate arrival at the intended destination.
2. **Coordinating Effectively Between Robots**
    - Prevent task conflicts between robots and ensure they do not collide with each other. This includes continuous awareness of each robot's location on the map.
    - Implementation of standard communication protocols to guarantee the proper transfer of data between robots for correct actions.
3. **Creating a System for Non-Technical People**
    - Building a user interface that enables warehouse personnel to have complete control over the system, providing them with a comprehensive view of the status of all system components in real time and allowing them to perform all operations without any complexity.
4. **Navigating Crowded and Changing Environments**
    - Provide real-time values, ensuring the robots' constant location on the map.
    - Enable the robots to avoid static and dynamic obstacles and continuously replan their routes while moving.

1.4 Requirements

This section defines the essential requirements for the successful development and operation of the WareBot system. We have divided them into two main categories, functional and non-functional.

1.4.1 Functional Requirements

Functional requirements specify the core features that the WareBot system must provide. The following are the primary requirements:

1. **Interface for User Requests:** A user interface that allows warehouse workers to perform multitude of operations, such as requesting or retrieving shelves, and creating a list of tasks within a multi-robot system. The interface should be able to accurately display the real time locations of robots, shelves, and delivery or retrieval points, while providing continuous monitoring of all system components.
2. **Autonomous Navigation:** The system must provide a mechanism that allows robots to operate autonomously within the warehouse without direct intervention from workers. This includes the ability to avoid obstacles, plan routes at minimal cost and for the robots to reach their destinations safely.
3. **Shelf Identification and Retrieval:** The system must accurately identify the locations of the shelves to be retrieved to enable robots to go to the correct coordinates and perform the required task precisely.
4. **Shelf Transportation:** During the shelf transportation and retrieval processes, the system must safeguard the stability of the products on the shelves and ensure their safe arrival.
5. **Multi-Robot Coordination:** The system is designed to coordinate between the robots and intelligently distribute tasks among them. To ensure the system's efficiency and effectiveness, it shall include mutual awareness and understanding between the robots, sharing their locations on a common map, and continuous replanning during navigation to resolve path conflicts and prevent collisions before they occur.

1.4.2 Non-Functional Requirements

Meeting one or more of the functional requirements contributes to fulfilling the non-functional requirements:

1. **Safety:** The system should enable robots to detect obstacles and avoid collisions, ensuring no risks to either personnel or goods within the warehouse.
2. **Usability:** The system has to provide a simple and user-friendly interface, enabling personnel to perform all essential operations, from requesting tasks and fully monitoring the system to receiving clear and understandable real-time feedback.
3. **Scalability:** The system must support the addition of multiple robots as needed without requiring any fundamental changes to the system.
4. **Adaptability:** The system must accommodate all environmental changes introduced after mapping. The system must detect changes, continuously scan to determine if the changes persist and reflect this on the shared map to ensure flexibility in changing environments.

1.5 System Description

Our system operates through a workflow that transforms user orders into completed shelf deliveries while maintaining coordination between robots. The system works in three steps as follows:

Step 1: Creating and Updating Maps

When the system is first launched, one robot is used to map the warehouse using LiDAR sensors, which collect distance measurements from the surrounding environment. These measurements are then processed using the SLAM (Simultaneous Localization and Mapping) algorithm to create a two-dimensional (2D) map of the warehouse.

This map is not continuously updated, but is instead revised when any changes occur after the initial mapping, such as shelf rearrangements or other additions to the warehouse. After the map is updated, the data is then shared with all other robots, allowing them to locate themselves on the same map and continuously transmit their locations to ensure coordinated operation.

Step 2: Assigning Task and Finding the Best Path with Coordination

When the system receives a task request, it evaluates available robots based on a set of criteria, including current location, battery level, shelf priority, and robot operational status. The most suitable robot is selected, and then the A* path planning algorithm is applied to calculate the optimal, least cost route. Coordination techniques are employed to manage paths, prevent collisions and dynamically replan paths continuously during robot movement.

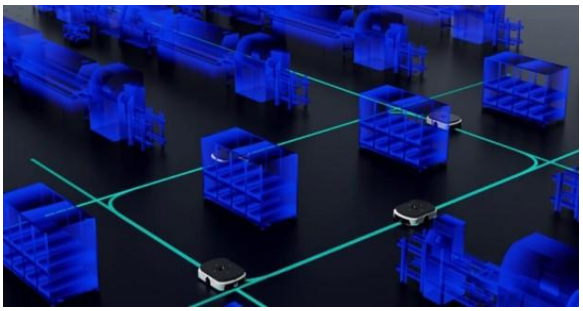
Step 3: Retrieving Shelves and Keeping Users Informed

After assigning the task to the most suitable robot, the selected robot autonomously and independently moves to the designated shelf and transports it to the specified delivery location. Throughout the execution process, status data, such as robot location, shelf location, delivery location and task progress, is transmitted continuously and in real time to the user interface, allowing operators to monitor all operations as they occur.

Multi-Robot Scenario: Dynamic Coordination and Collision Avoidance

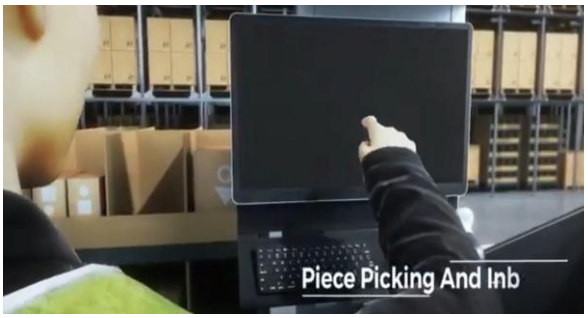
When multiple robots are active in the warehouse simultaneously, the system introduces an advanced layer of task coordination on top of the fundamental steps described earlier. Without repeating the standard mapping or assignment procedures, every active robot continuously broadcasts its live position, planned trajectory, and task progress to the central interface. The system proactively evaluates these trajectories against one another to predict potential path intersections or congestion points. When a collision risk is detected, the system dynamically recalculates an alternate route for one robot or assigns it a temporary yielding state, ensuring that all robots can execute their tasks smoothly and avoid operational bottlenecks.

This process is illustrated in Figure 1.1, depicting a conceptual scenario intended to demonstrate the proposed system’s operational workflow.

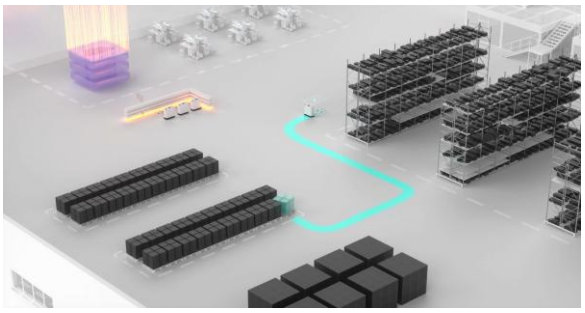


step 1a: One robot performs environment mapping using LIDAR and SLAM to create and update the global map.

Step 1b: The global map is distributed to all robots, which localize themselves on it and share their positions with the central system for coordinated operation.

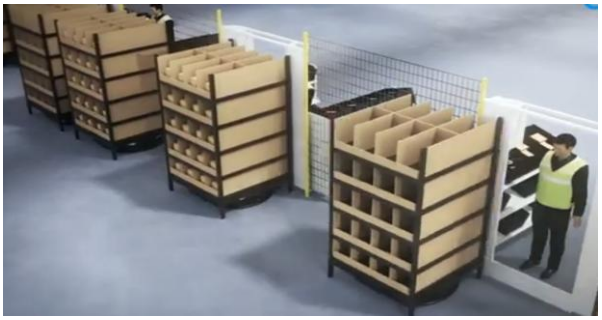


Step 2a: Worker touches the screen to request a specific shelf



Step 2b: System assigns robot and plans path to retrieve shelf

Step 3a: Robot travels with the shelf to the worker



Step 3b: Robot delivers the shelf to the destination

Figure 1.1: Conceptual steps illustrating the process of delivering a pod to a warehouse worker.

1.6 Limitations and Constraints

The development and continuous operation of the WareBot system are bounded by several key factors (Table 1.1):

Table 1.1: Project Limitations and Constraints

Limitation	Description
Budget	Overall system cost must remain affordable for Small and Medium-sized Enterprises (SMEs), restricting the hardware choices per robot.
Security	Protecting sensitive telemetry and network communications introduces protocol complexity and additional processing overhead.
Power Capacity	Continuous operation is limited by battery life, requiring strategic power management and periodic charging cycles that temporarily reduce the active fleet size.
Computational	Running simultaneous mapping, path planning, and coordination heavily utilizes onboard hardware, restricting the maximum fleet size.

In summary, the design of WareBot carefully balances these financial, technical, and physical limitations to deliver an efficient, secure, and reliable automated solution for real-world warehouse environments.

1.7 Implementation Timeline

As summarized in Table 1.2, the tasks of the system implementation and operation are distributed along the two semesters.

Table 1.2: Project's Gantt Chart

Weeks / Tasks	First Semester Weeks					Second Semester Weeks				
	1-2	3-5	6-8	9-11	12-16	1-2	3-5	6-8	9-11	12-16
Idea Selection	■									
Data Collection		■								
Design			■	■	■					
Implementation						■	■	■		
System testing							■	■	■	■
Documentation		■	■	■	■	■	■	■	■	■

1.8 Report Outline

The rest of the report is structured as follows: Chapter 2 provides a theoretical background on the algorithms and basic techniques, followed by a review of relevant scientific literature. Chapter 3 illustrates the complete system design, including both the hardware and software components, and presents flow diagrams of the system. Chapter 4 covers the implementation and testing phase, detailing how the software works, all the techniques and algorithms built, and the results of the applied tests. Chapter 5 includes real-world tests conducted under various conditions, documenting the results obtained and evaluating the system's performance and effectiveness based on these tests. Chapter 6 presents the conclusion of our project, documenting the results our project can achieve and proposing future developments and upgrades to enhance its effectiveness in running core warehouse operations.

Chapter 2

Theoretical Background

2.1 Preface

This chapter provides a brief overview of the key components in our project. We first provided a brief overview of the theoretical background, and we will explain general terms such as autonomous navigation and path planning algorithms. It also reviews similar projects to show how WareBot fits into the wider field of warehouse automation. Second, we present a literature review that compares our project with previous work.

2.2 Theoretical Background Concepts

2.2.1 Simultaneous Localization and Mapping (SLAM)

SLAM, or Simultaneous Localization and Mapping, is a process that allows a robot to navigate an unknown environment [1, 2]. While navigating the environment, the robot seeks to acquire a map thereof, and at the same time, it wishes to localize itself using its map. SLAM lets robots work without previous knowledge of the environment, and supports tasks such as path planning and obstacle avoidance using modern frameworks like SLAM Toolbox [3].

2.2.2 Multi Robot Task Allocation (MRTA)

Multi-Robot Task Allocation (MRTA) is the process of distributing tasks over a group of robots in a collaborative way [4]. One example is in warehouse automation, transporting shelves from one location to another using MRTA, which reduces the time needed to complete a queue of tasks and makes the system flexible and scalable.

2.2.3 Hybrid Edge–On-Premises Computing

Hybrid Edge On-Premises Computing means devices process data locally and send it to local servers within the same private network [5, 6]. This makes the system very fast and secure, since data never leaves the network. Servers can store data, run AI models, and manage multiple devices. This architecture is best for areas like factories, warehouses, and other critical systems that need low latency and strong data protection. Figure 2.1 shows the architecture of the Hybrid Edge–On-Premises system and how it is organized.

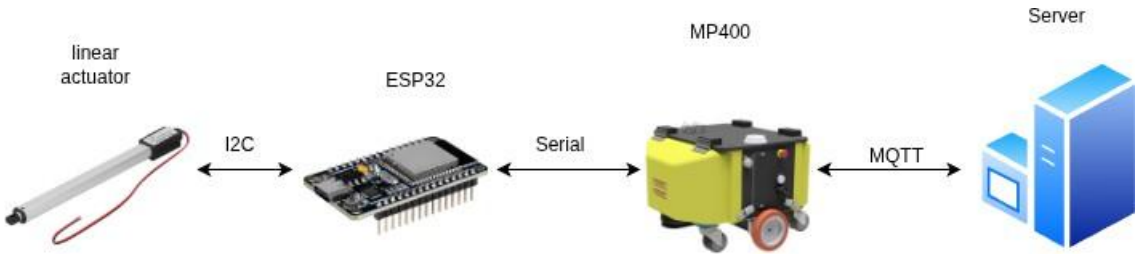


Figure 2.1: Hybrid Edge–On-Premises Architecture

2.2.4 MQTT Protocol and Broker

MQTT (Message Queuing Telemetry Transport) is a protocol that provides a lightweight method of carrying out messaging using a publish/subscribe model [7, 8]. This makes it suitable for Internet of Things messaging, such as with low-power sensors or mobile devices such as phones, embedded computers, or microcontrollers. An MQTT broker acts as a central server responsible for routing messages between clients [9]. It ensures that messages published by one client are delivered to all clients subscribed to the relevant topic.

2.3 Literature Review

Autonomous mobile robots (AMRs) have been a rapidly expanding research topic over the past decades. Recent technological developments in hardware and software have made them more feasible, especially in warehouse environments. This literature review covers recent AMRs designed for warehouse environments and the gap between them and WareBot. Table 2.1 shows a comparison between leading international warehouse robotic solutions in terms of year, navigation methods, infrastructure requirements, system architecture, communication models, and cost factors to suit the table requirements.

Table 2.1: Comparison of International Warehouse Robotic Solutions

Category	Amazon Proteus AMR [10]	Attabotics Attabot [11]	MiR Series (250/600) [12]	WareBot
Year	2022	2020	2020	2026
Navigation Method	LiDAR-based SLAM with dynamic obstacle avoidance	Internal grid-based tracking within modular cube structure	LiDAR SLAM with shared maps for fleet navigation	LiDAR-based SLAM shared via central coordination server
Infrastructure Requirements	High (Proprietary floor tags and markers)	High (Standardized modular grid racks and tracks)	Moderate (Wi-Fi coverage and sometimes floor markers)	Low (Independent; no prebuilt infrastructure or floor tags/markers needed)
System Architecture	Edge computing with cloud integration for analytics	Centralized warehouse control software	MiR Fleet management software with cloud updates	Hybrid Edge–On-Premises Computing
Communication Model	Proprietary mesh / Wi-Fi	Centralized grid control network	Wi-Fi / Industrial Wireless	MQTT over internal network
Cost Factors	High (Enterprise licensing)	High (Proprietary hardware)	Moderate to High	Low (Affordable SME-focused)

While these available solutions are effective, they often require a pre-built infrastructure. Scaling up also needs infrastructure modifications. They are also unsuitable for warehouses lacking the infrastructure for robotic operation. Furthermore, these solutions typically rely on expensive proprietary hardware and expensive cloud systems, resulting in ongoing subscription fees and data security concerns. Consequently, these systems are impractical for SMEs that require simple, secure, and moderate cost automation solutions.

WareBot offers a solution to bridge the gap that hinders SMEs access to automation systems, by providing a cost-effective, scalable solution without any modification to the infrastructure, while at the same time providing a highly secure solution by using a hybrid computing model between the edge and on-premises infrastructure, instead of relying on cloud computing which may be less secure and require high costs.

2.4 Summary

This chapter outlined the key concepts supporting the WareBot project, including SLAM for real-time localization, and Multi-Robot Task Allocation. It reviewed related projects to show WareBot's unique contributions, such as its deployment model and suitability for real-world warehouse automation in resource-constrained environments.

Chapter 3

System Design

3.1 Preface

In this chapter, we explain the design of Warebot, covering both hardware and software. We will justify the selection and adoption of each component, present tables of different design options and explain the selection of the appropriate component that meets our project's needs. We will also explain the structure and operation of the algorithms and techniques used. A conceptual overview will be presented to show the interaction between system components, along with other supporting diagrams. Finally, the chapter will conclude with a schematic diagram to illustrate how all components are connected to each other.

3.2 System Components and Design Alternatives

3.2.1 Hardware Components

Robot Selection

Before selecting the robot, we first defined the main goals that must be achieved through the robot are the ability to transport at least 50kg, be suitable for the warehouse environment, and have powerful computational capabilities. We primarily considered the robots available at our university, as shown in Table 3.1:

Table 3.1: Comparison of Robot Platforms

Criteria	TurtleBot 2 [13]	MP-500 [14]	MP-400 [15]
Platform Type	Lightweight, open-source	Heavy-duty industrial platform	Compact, modular industrial-grade platform
Common Usage	Education and research	Heavy industrial use	Research and industrial automation
ROS Compatibility	Fully supported	Supported	Supported
Sensing Systems	Basic sensors (e.g., Kinect)	Advanced sensors, high payload capacity	Advanced sensors (LiDAR)
Mobility	Suitable for controlled indoor environments	Limited indoor maneuverability due to size	Balanced, suitable for complex environments and tight spaces
Operational Runtime	Limited	Extended runtime	Balanced between runtime and performance
Size and Weight	Small and lightweight	Large and heavy	Compact and medium-sized
Power Consumption	Low	High	Moderate
Images			

After comparing the three available robot platforms, the most suitable option was the MP400. This was due to its size compared to other robots, its superior computing power and its advanced sensors. All these characteristics, when compared with other robots, showed that the MP400 is highly compatible and well-suited to warehouse needs, as warehouses require such robots.

Gripper Selection

Based on Table 3.2, Three types of grippers were compared alongside a separate evaluation of linear actuators. The reason for this was to identify the best mechanism

that allows the robots to grip and release shelves smoothly, while still maintaining a low cost and high operational quality. Table 3.2 shows the complete comparison:

Table 3.2: Comparison of Gripper Types

Criteria	**Linear Actuators (12V, 60N, 15cm)**	**Servo-Driven Grippers [16]**	**Electromagnetic Grippers [17]**
Grip Strength	Moderate, 60N force sufficient for 12-30 kg shelves	Moderate, limited grip strength	Moderate, depends on magnetic material
Control	Simple DC motor control via H-bridge drivers	Easy control via servomotor	Simple ON/OFF electromagnetic control
Complexity	Mechanically simple, requires motor drivers	Mechanically simpler	Very simple, no moving parts
Material Compatibility	Works with various materials via mechanical engagement	Works with most objects	Only works with metallic objects
Cost	Cost-effective ($10-15 per actuator)	Cost-effective	Generally low cost
Maintenance	Low maintenance, durable	Low maintenance	Low maintenance
Images			

Based on the comparison above, the linear actuator was selected with the following specifications: 12V voltage, 120N force and 15cm length. A mechanical structure was designed to form two robotic arms using these actuators. This option is more suitable in terms of size and installation on the robot, meeting the requirement to support weights between 12 and 30 kg. The 15 cm length is also suitable for the shelf being used and the price is very cost effective at $20 per unit. This option combines strength, efficiency and cost in manner that is very consistent with our project goals.

Controller Selection

The controller is an important part of this project because it controls the robot's arms, communicates with the robot and reports on the arms' condition. In our comparison, we made sure that the controllers we selected were appropriate for the goals of our project. Table 3.3 shows a comparison between three different types of controllers.

Table 3.3: Comparison of Controllers

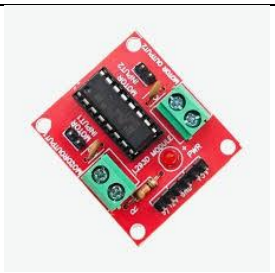
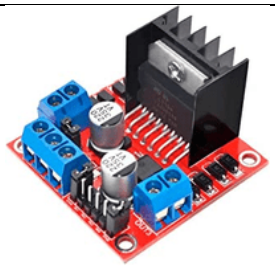
Criteria	Arduino Uno [18]	ESP32 [19]	Raspberry Pi 4 [20]
Processor	8-bit AVR, 16 MHz	32-bit Xtensa Dual-core, up to 240 MHz	Quad-core ARM Cortex-A72, 1.5 GHz
Memory	2 KB RAM, 32 KB Flash	520 KB SRAM, 4 MB Flash	1–8 GB RAM
Sensor Compatibility	High, widely supported	High, supports Wi-Fi and Bluetooth	Very high, supports many protocols
Ease of Programming	Easy, Arduino IDE	Easy to moderate, Arduino IDE and ESP-IDF	Moderate to advanced, Linux OS
Connectivity	Limited (wired only)	Wi-Fi and Bluetooth built-in	Wi-Fi, Bluetooth, Ethernet, USB
Cost	Low ($10–20)	Low ($5–15)	Medium ($35–75 depending on model)
Motor Control Capability	Good for small motors and simple controls	Good, supports multiple protocols	Excellent with add-ons and OS
Power Consumption	Low	Low	Higher than others
Images			

Based on the comparison above, we chose the ESP32 for several key reasons: it has a processor and RAM that meet the project's functional requirements,and it's easy to program and supports Bluetooth as well as Type-C cable connection. Which gives us multiple communication options in the event of connection failure. Considering all these advantages and the cost, we concluded that this processor is the most suitable choice as it meets all the functional requirements effectively.

Motor Driver Selection

To drive the linear actuators used in the lifting mechanism, a suitable motor driver was required to provide sufficient current and support the 12V operation. We compared three common motor driver modules as shown in Table 3.4.

Table 3.4: Comparison of Motor Drivers

Criteria	L293D	LM298N	TB6612FNG
Driver Type	Dual H-Bridge	Dual H-Bridge	MOSFET H-Bridge
Operating Voltage	4.5V – 36V	5V – 35V	4.5V – 13.5V
Continuous Current	0.6A per channel	2.0A per channel	1.2A per channel
Peak Current	1.2A	3.0A	3.2A
Thermal Protection	Yes	Yes	Yes
Efficiency	Low (Voltage drop)	Moderate	High (Low heat)
Cost	Very Low	Low	Low
Images			

Based on this comparison, two LM298N dual H-bridge motor driver modules were selected. These modules provide the power interface between the ESP32 and the linear actuators, with each module controlling two actuators. They support a current capacity of 2A per channel, which is well within the requirements for our actuators, and feature built-in thermal protection to ensure reliable operation under load. This choice effectively balances cost, current capacity, and ease of integration.

Camera Module

We used a camera in our system primarily to improve the robot's accuracy and reliability when interacting with shelves. By scanning four AprilTags placed on each shelf, the robot can precisely align itself in terms of position and orientation, ensuring accurate access and engagement. In addition, the camera enables shelf identification by reading a unique value assigned to each shelf, allowing the system to distinguish between different inventory units. Mounted at the center of the robot's top surface, the camera captures the AprilTag located at the center underside of the shelf, enhancing both docking precision and operational efficiency in a cost-effective manner.

Figure 3.1: Camera module mounted on the robot for AprilTag detection and shelf alignment

Shelf Material Selection

For the shelf structure, the primary types of available materials were compared as shown in Table 3.5:

Table 3.5: Comparison of Shelf Materials

Criteria	Plastic	Steel	Aluminum
Weight	Light	Heavy	Light to Medium
Durability	Good, resistant to breakage	Very high, strong and durable	Good, corrosion resistant
Rust Resistance	Completely resistant	Prone to rust if untreated	Good corrosion resistance
Ease of Cleaning	Easy	Easy	Easy
Cost	Low to Medium	High	High

We decided to select steel for the shelving. This choice was based on the robot's durability and its weight-bearing capacity as documented by the manufacturer, as well as the material's superior performance compared to alternatives. Steel is cost effective, has a much higher load-bearing capacity for repeated warehouse operations, is corrosion-resistant and is readily available locally.
These advantages led us to choose steel because it outperforms other materials in an industrial context.

3.2.2 Software Components

While hardware is essential, WareBot's main innovation lies in its software. The system integrates modules for navigation, localization, obstacle avoidance, and path optimiza-

tion to ensure smooth and efficient operations.

Robot Operating System (ROS)

In WareBot, ROS forms the core of communication between all robots and the central server, serving as a bridge between the software components and the hardware. It enables sensor data processing, autonomous navigation, and multi-robot coordination using a publish/subscribe messaging based on topics for communication [21].

SLAM

In WareBot, SLAM is used for navigation on a pre-built map, which was built by the algorithm itself [22]. This process is critical for maintaining navigation accuracy and adaptability within the warehouse environment, allowing robots to build a map and localize themselves on that map simultaneously.

Multi-robot Task Allocation

In WareBot, we use Multi-Robot Task Allocation (MRTA) to assign tasks for robots based on payload, battery levels, and their current locations. This ensures optimized resource utilization and efficient fleet coordination. Also, we implemented Conflict-Based Search (CBS) to resolve path conflicts between robots before they occur, preventing operational deadlocks and ensuring smooth traffic flow [4, 23].

Web application and Database

A web application allows the workers to send commands, monitor robot status, and view task progress. The workers can send a request to get a product from the warehouse or to get a shelf from the warehouse. There is a central server to ensure accurate stock and shelf tracking, with real-time updateable data. While database integration ensures that operational data, such as Task logs, inventory levels, and robot activity history, is permanently stored [24].

Obstacle Avoidance and Path Planning

The most important algorithms in warehouse operations are Obstacle Avoidance and Path Planning. These algorithms are crucial for navigation within the warehouse environment. Obstacle Avoidance works to avoid obstacles within the warehouse, while Path Planning aims to achieve the optimal path within the environment map at the lowest cost. Table 3.6 compares Two Global Path Planning algorithms, and Table 3.7 Compares Two Obstacle Avoidance algorithms.

Table 3.6: Comparison of Global Path Planning Algorithms

Feature	**A* Algorithm [25]**	**Dijkstra's Algorithm [26]**
Type	Heuristic-based global planner	Graph-based global planner
Optimality	Near-optimal with heuristics	Always finds optimal path
Computation Cost	Lower (guided by heuristics)	Higher (explores all nodes)
Scalability	Efficient for large maps	Slower on large maps
Typical Use Case	Warehouse navigation, robotics	Network routing, small maps

Table 3.7: Comparison of Obstacle Avoidance Algorithms

Feature	**Dynamic Window Approach (DWA) [27]**	**Vector Field Histogram (VFH) [28]**
Type	Velocity-based local planner	Histogram-based local planner
Reactivity	High (continuous replanning in velocity space)	Moderate (reacts to obstacles via histograms)
Smoothness of Path	Generates smooth, feasible trajectories	Can produce oscillations or sharp turns
Computation Cost	Moderate (velocity sampling)	Lower (histogram updates)
Dynamic Obstacle Handling	Very effective	Less effective
Typical Use Case	Indoor navigation, dynamic environments	Outdoor/structured environments

According to existing comparisons, the A* algorithm is superior to Dijkstra's algorithm because it is better suited for warehouse navigation and more efficient for large maps. Furthermore, it utilizes an optimal heuristic-based global planner, unlike Dijkstra's algorithm. Regarding obstacle avoidance, we employed a dynamic window approach, as it is more effective than the Vector Field Histogram algorithm. Simultaneously, it performs velocity sampling and smooths robot movement. This means we used a hybrid path planning approach to obtain the optimal path using the A* algorithm and then fed this path to the Dynamic Window Approach for real-time obstacle avoidance and

safe local navigation within the warehouse environment. This combination ensures both global efficiency and global adaptability within the warehouse environment.

Deployment Options

Selecting the right deployment model is critical for the WareBot system to ensure reliable communication, real-time performance, and secure operations. Several deployment options are available for Internet of Things (IoT) and robotic systems, as summarized in Table 3.8.

Table 3.8: Comparison of IoT deployment models

Feature	**Cloud-Based Deployment [29]**	**On-Premises Deployment [30]**	**Hybrid Edge–On-Premises Deployment [5]**
Processing Location	Remote cloud servers	Local warehouse server	Edge on robot + local server
Latency	Higher	Low	Very low for critical tasks
Scalability	High	Limited	Moderate
Security	Potential privacy issues	High	High
Connectivity Dependence	Internet required	Minimal	Minimal
Suitable Tasks	Data aggregation, analytics	Local control, sensitive data	Real-time control, fleet coordination

Based on previous comparisons, using a hyper-edge on-premise computing model is one of the best available solutions. This is because some tasks require real-time decision-making, such as SLAM, obstacle avoidance, and monitoring control for robots. This necessitates a central on-premise server to manage fleet management, coordination, task allocation, and map synchronization. This architecture is characterized by its reduced latency and cloud connectivity dependency, while simultaneously providing a secure and scalable solution for dynamic warehouse environments.

Communication Protocols

Communication protocols are essential in the WareBot system to enable data exchange between robots, sensors, and the central server. They ensure seamless integration, reliability, and security during multi-robot coordination. Table 3.9 compares MQTT, CoAP, and HTTPS based on their main features and suitability for WareBot.

Table 3.9: Comparison of IoT communication protocols

Feature	**MQTT [7]**	**CoAP [31]**	**HTTPS [32]**
Type	Publish–Subscribe	Request–Response	Request–Response
Use Case	IoT messaging, telemetry	IoT constrained devices	Secure web communication
Connection	Lightweight persistent	Lightweight (UDP-based)	Persistent (TCP-based)
Reliability	QoS levels (0,1,2)	Confirmable / Non-confirmable	Reliable
Overhead	Low	Low	High
Message Ordering	Ordered	Unordered	Ordered
Security	TLS/SSL, authentication	DTLS	SSL/TLS
Latency	Very Low	Low	Higher (handshake, encryption)
Web Browser Support	No	No	Yes

For the **WareBot** project, **MQTT** is preferred for communication between robots and the server. Its lightweight publish–subscribe model, scalability, and low latency make it ideal for real-time multi-robot coordination and status updates.

Brokers

In the WareBot system, the MQTT broker manages message exchange between robots, sensors, and the server. It ensures reliable and secure communication, which is essential for coordinating multiple MP-400 robots in real time. Table 3.10 compares AWS IoT Core, HiveMQ, and Mosquitto based on key features.

Table 3.10: Differences between brokers

Feature	**AWS IoT Core [33]**	**Mosquitto [9]**	**HiveMQ [34]**
Cloud Provider	Amazon Web Services	N/A (Local)	N/A (Local)
Ease of Use	User-friendly dashboard	Basic CLI configuration	User-friendly GUI
Security Features	Advanced (IAM, TLS, fine-grained policies)	Basic (username/password, TLS)	Advanced (RBAC, TLS, plugin-based security)
Authentication	IAM, X.509 Certificates	Username/Password	Various methods (OAuth 2.0, LDAP, Certificates)
Device Management	Yes (AWS IoT Device Management)	Limited	Yes (Control Center + REST API)
Pricing Model	Pay-as-you-go	Open Source	Free (Community) / Commercial (Enterprise)

For the **WareBot** project, we chose **HiveMQ** for its strong security, multiple authentication options, and device management features. This ensures that robot communication and task data remain protected and reliable.
One of the key aspects in warehouse systems is the monitoring and feedback mechanism, which ensures reliable and efficient robot operation. In our project, we used the Grafana system for the continuous monitoring of robots. The special dashboard in Grafana contains monitoring tools and visualization tools that provide real-time data for CPU usage, RAM memory consumption, battery level, and real-time position (X, Y, Yaw), which are collected and transmitted to the server through the monitoring tools. Grafana provides a dashboard that visualizes the overall system status over time, and this will help us to predict maintenance of the robots.

3.3 Conceptual System Description

This section illustrates how our system works using two diagrams. These diagrams help us understand how each component interacts with other components and how

they are connected, as shown in Figure 3.2.

Figure 3.2: General Block Diagram

The second diagram shows how the system works when assigning tasks. The workflow follows a set of steps when workers need to request a shelf until the shelf is delivered, as follows:

1. The worker clicks on the web interface to request a specific shelf.
2. The server specifies the robot according to the sum of priority (battery, location, and status).
3. The robot moves to the specified shelf location and aligns itself to catch the shelf correctly, and then moves to deliver it to the worker.
4. The server continuously updates the location of the shelf and the robot while moving, and finally shows on the web interface the final position of the robot and shelf, as well as the status of the assigned task.

These steps are shown in Figure 3.3 as a graphical representation.

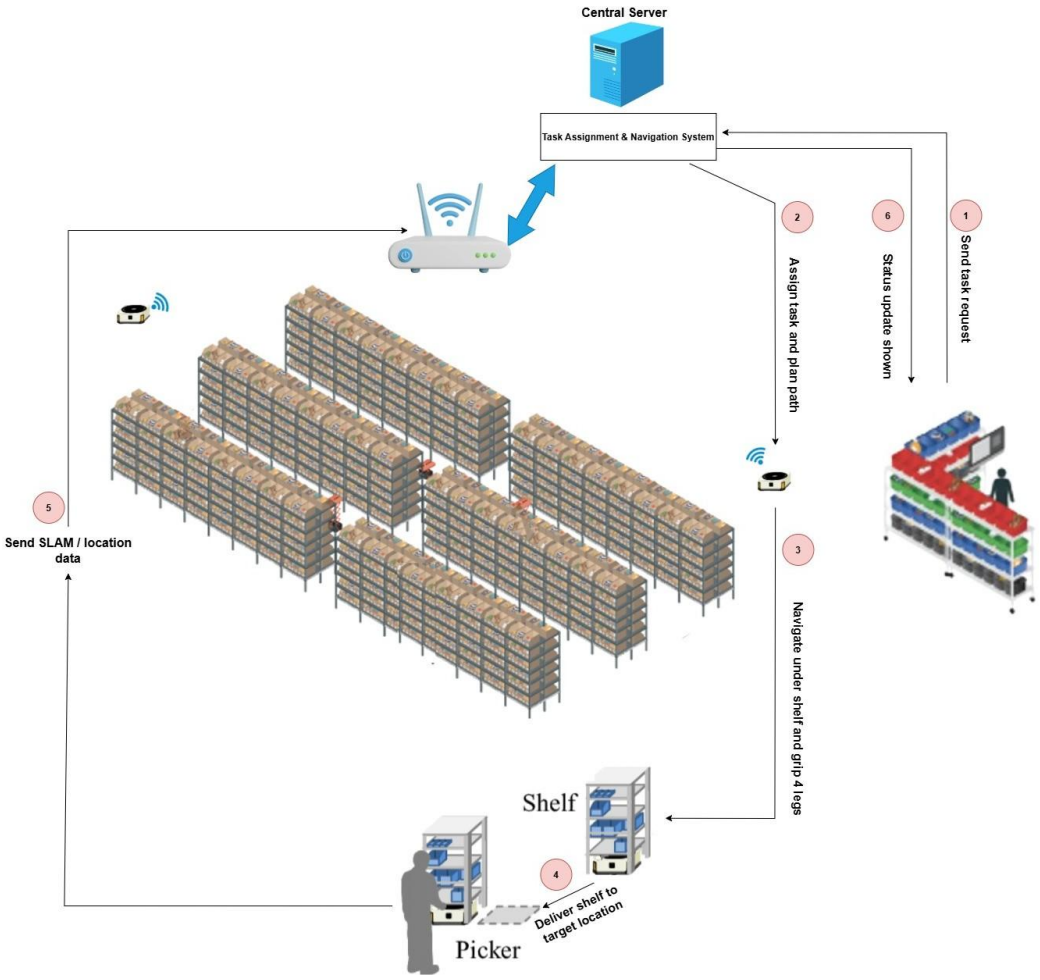


Figure 3.3: Conceptual Workflow Diagram.

3.4 Algorithms and Diagrams

3.4.1 Algorithms

Pathfinding using A* Search

The steps and processes involved in implementing the A pathfinding algorithm, can be summarized as follows:

1. **Initialization**
    - Create `open_set` (priority queue) for nodes to be evaluated.
    - Initialize `came_from`, `g_score`, and `f_score` maps.
    - Set start node `g_score` to 0 and `f_score` to heuristic to goal.
    - Add start node to `open_set`.
2. **Search Loop**

- While open_set is not empty:
- Get node with the lowest f_score.
- If goal achieved, recreate and return path.

3. **Neighbor Evaluation**
    - For each neighbor:
    - Calculate tentative cost.
    - If better than recorded cost, update maps and add neighbor to open_set.
4. **Failure**
    - If open_set is empty and goal not reached, return failure.

Pseudocode:

```
function A_STAR(start, goal):
    open_set <- priority queue with start
    came_from, g_score, f_score <- maps
    g_score[start] <- 0
    f_score[start] <- heuristic(start, goal)

    while open_set not empty:
        current <- node with min f_score
        if current == goal:
            return reconstruct_path(came_from, current)
        for neighbor in neighbors(current):
            tentative_g <- g_score[current] + dist(current, neighbor)
            if tentative_g < g_score[neighbor]:
                came_from[neighbor] <- current
                g_score[neighbor] <- tentative_g
                f_score[neighbor] <- tentative_g + heuristic(neighbor,
    goal)
                if neighbor not in open_set:
                    add neighbor
    return FAILURE
```

Obstacle Avoidance using Dynamic Window Approach (DWA)

The steps and processes involved in implementing the DWA obstacle avoidance algorithm, can be summarized as follows:

1. **Velocity Sampling**
    - Compute dynamic_window based on current velocity and acceleration limits.
    - Sample possible (v, w) velocity pairs.
2. **Trajectory Prediction**

- For each (v, w), predict short-term trajectory.

3. **Trajectory Evaluation**
   - Calculate heading score, clearance score, and velocity score.
   - Combine with weights into an objective value.
4. **Best Velocity Selection**
   - Choose (v, w) with highest objective value.
5. **Execution**
   - Send selected velocities to robot motors.

Pseudocode:

```
function DWA(robot_state, goal, obstacles):
    dw <- calc_dynamic_window(robot_state)
    best_vel  <-  NULL
    max_score <- ∞-

    for (v, w) in sample_velocities(dw):
        traj <- predict_trajectory(robot_state, v, w)
        score <- alpha*heading(traj, goal) + beta*clearance(traj,
   obstacles) + gamma*velocity(v)
        if score > max_score:
            max_score <- score
            best_vel  <- (v, w)
    return best_vel
```

Task Allocation in Multi-Robot System

The steps and processes involved in implementing a simple task allocation algorithm, can be summarized as follows:

1. **Identify Resources**
   - Get list of available robots and pending tasks.
2. **Cost Calculation**
   - For each (task, robot), calculate cost based on distance, battery, workload, and capabilities.
3. **Assignment**
   - Assign task to robot with minimum cost.
   - Mark robot as busy and task as assigned.
4. **Integration with Navigation**

- Plan path for assigned robot (e.g., A*).
- Resolve conflicts for multiple robots if needed.

5. **Monitoring**
    - Continuously update robot status and reallocate if necessary.

Pseudocode:

```
function TASK_ALLOC(tasks, robots):
    assignments <- {}
    for task in tasks:
        best_robot <- NULL
        min_cost <- ∞
        for robot in robots:
            if robot available:
                cost <- calc_cost(task, robot)
                if cost < min_cost:
                    min_cost <- cost
                    best_robot <- robot
        if best_robot:
            assignments[best_robot] <- task
            mark robot busy
    return assignments
```

3.4.2 Diagrams

Shelf Retrieval Process Steps

The sequence diagram in Figure 3.4 illustrates the complete process of shelf retrieval, from user request to final delivery and notification.

1. **User Request:** The user specifies a specific shelf and requests to move the shelf to a specific location for delivery through the web interface.
2. **Server Handling:** The web interface forwards the user's request to the server for validation and authentication, and records the request time.
3. **Status Lookup:** The server queries its database to obtain the current location of the shelf and the current robots' locations, status, and battery level of each robot.
4. **State Return:** The database returns information about the shelf and robots to the server.
5. **Global Assignment and Planning:** The server runs the MTRA algorithm to assign the most suitable robot to the task. It assigns an initial path for the robot using the A* algorithm, and also runs the CBS algorithm to solve any conflicts that may occur.

6. **Task Submission:** The server sends the task to the assigned robot, which has an initial plan and the location of the shelf.

7. **Local Planning and Navigation:** The process begins by utilizing the precise location of the CBS network by making adjustments to obstacles. The robot runs the Dynamic Window Approach (DWA) for obstacle avoidance on its software and makes re-planning active for the entire task journey to continuously update the path using the A* algorithm to get the shortest path.

8. **Alignment Phase:** After reaching the shelf, the robot begins the alignment process using its connected camera to ensure it reaches the exact point with the correct orientation. Feedback is provided to monitor the robot's alignment and ensure it is performing correctly.

9. **Shelf Pickup:** After confirming correct alignment, the robot sends commands to the ESP32 controller to control the linear actuators to extract and pick up the shelf.

10. **Mid-Job Update:** After completing the pickup process, the robot sends feedback to the server that the shelf has been correctly picked up, allowing the server to update its database.

11. **Delivery:** The robot navigates and keeps moving until it reaches its destination. At this point, the robot sends commands to the ESP32 controller that controls the linear actuators to retract and release the shelf.

12. **Completion:** The robot notifies the server that the task is completed successfully, and the server updates its database to reflect the new location of the robot and the shelf.

13. **User Notification:** The server sends a notification to the user that the task was completed successfully.

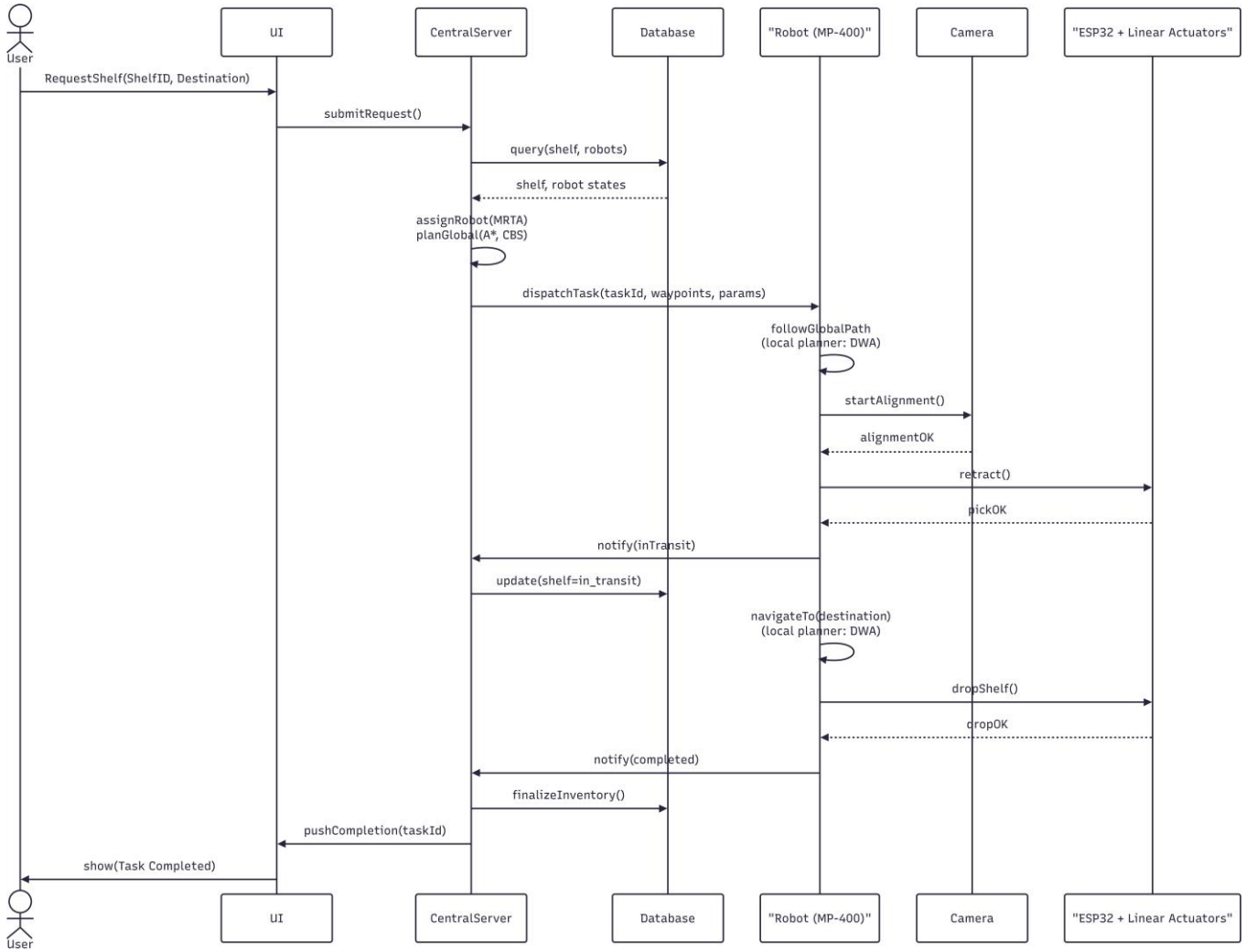


Figure 3.4: Sequence diagram of the shelf retrieval process.

Backend Software Architecture

The Backend Layer acts as the central orchestrator of the entire WareBot system. It bridges the high-level user interactions from the frontend with the low-level robotic control executed by the ROS 2 layer. The backend is architected as a set of integrated microservices. The core component is a **Flask** API server. Figure 3.5 illustrates the block diagram of the backend architecture.

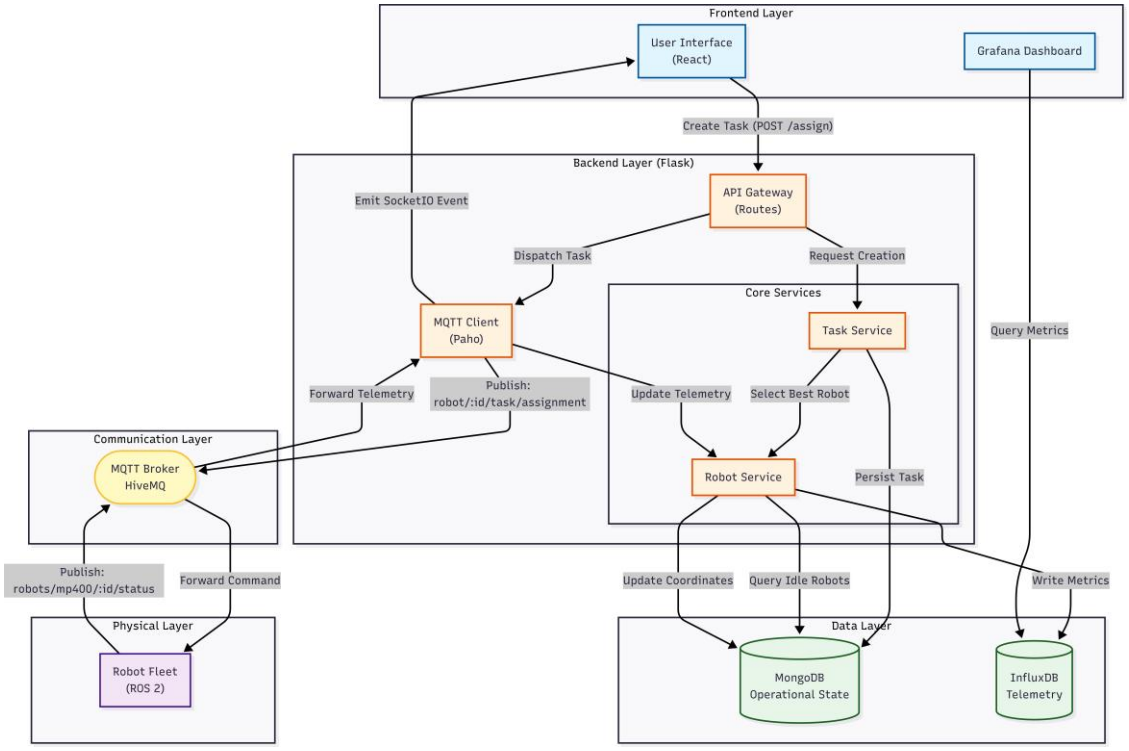


Figure 3.5: Backend Software Architecture Block Diagram

Database Schema and Data Flow

The system uses **MongoDB** for storing structured data. The schema, illustrated in Figure 3.6, defines the core collections including Robots, Tasks, Shelves, Zones, and Users. The flow of telemetry data is shown in Figure 3.7.

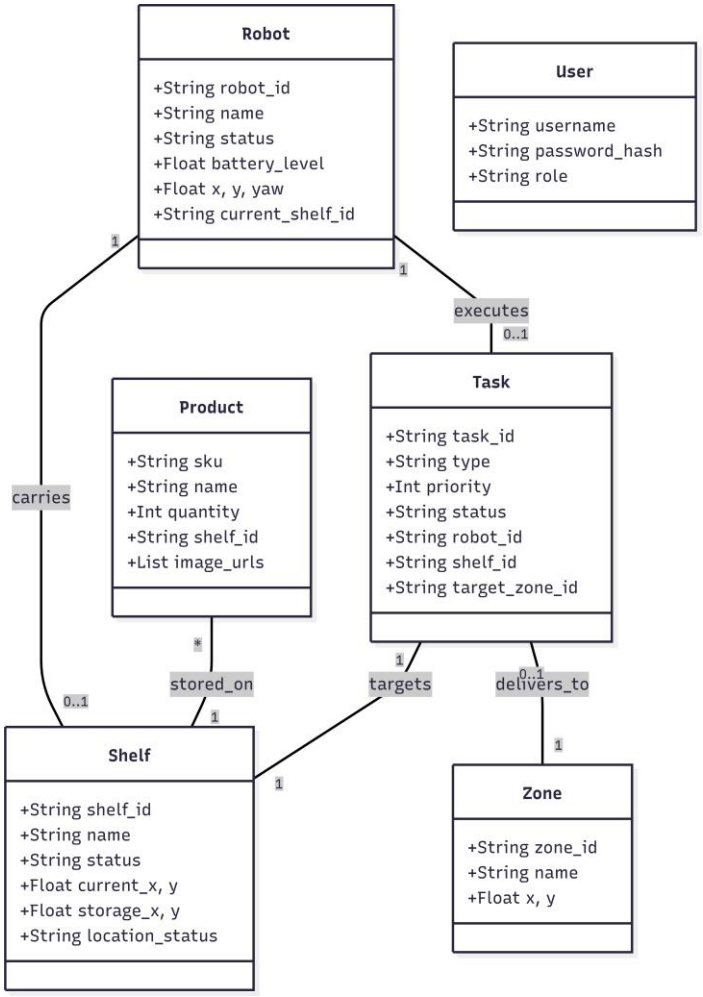


Figure 3.6: Database Schema

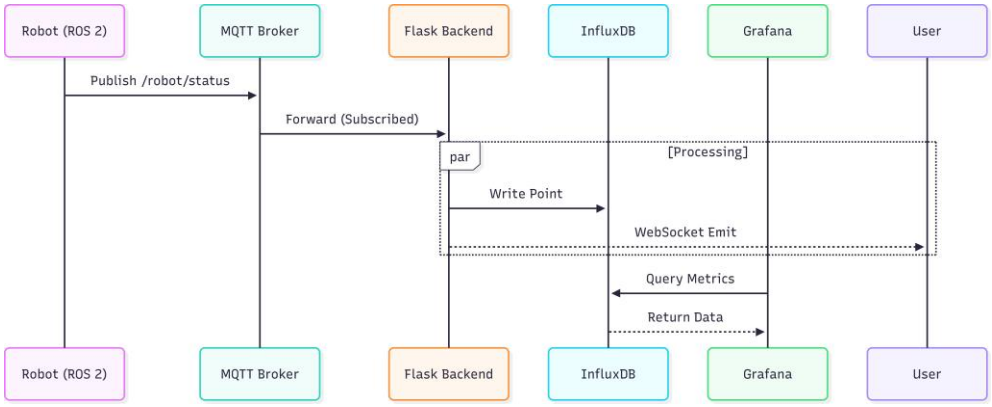


Figure 3.7: Data Flow of Telemetry

3.5 Schematic Diagrams

As shown in Figure 3.8, the schematic diagram illustrates the hardware connections within the WareBot system. The MP-400 robot PC communicates with the ESP32 microcontroller via a USB connection to control four servo-driven grippers. A DC regulator provides a stable 5V power supply for the servos. The robot PC connects to the central server through a Wi-Fi router, enabling task coordination and navigation data exchange.

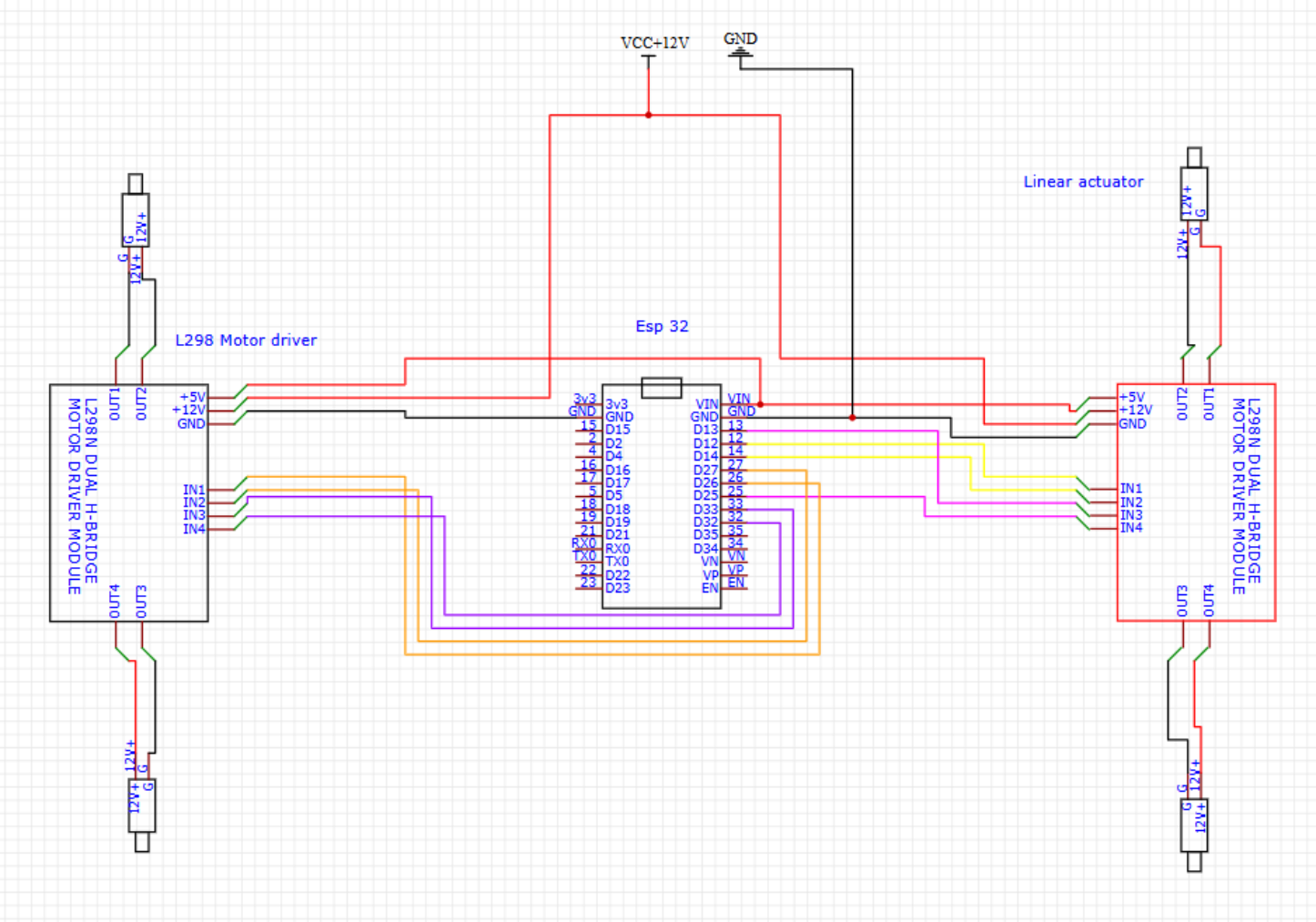


Figure 3.8: Schematic Diagram of WareBot Hardware Connections

3.6 Summary

This chapter presented the design of the WareBot system, covering hardware and software choices, conceptual workflow, and the algorithms that support navigation, coordination, and obstacle avoidance. The design aims to ensure safe, efficient, and scalable automation for SMEs.

Chapter 4

Implementation

4.1 Preface

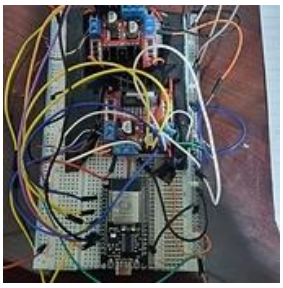
This chapter provides an overview of the software and hardware implementation, issues, and challenges related to the implementation.

4.2 Hardware Implementation

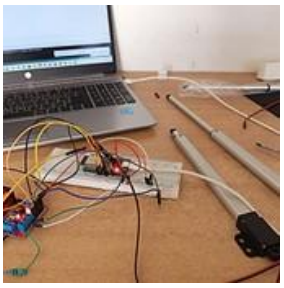
In the previous chapter, we showed how the components interconnect with each other. This section presents how we assembled the prototype, then moved to the real robot. setup, and finally validated the complete integrated hardware view.

4.2.1 Prototype Setup

We start with prototype of all system components. This gave us a future view of how the system will work. We began by connecting the electronic components on a breadboard, which makes the wiring flexible and allows us to change the connections.
Figure 4.1(a) shows the ESP32 connected to two motor drivers that control the linear actuators, while Figure 4.1(b) shows the same setup connected to the linear actuators. After we confirmed the connections and verified the system behavior, we installed the linear actuators on the four corners of the robot, as shown in Figure 4.1(c). This step was used to observe how the actuators extend and retract while the robot is navigating. Finally, Figure 4.1(d) shows the shelf prototype we built, designed with dimensions that fit the robot.
The complete prototype stages are shown in the following figure below.



(a) control wiring on breadboard



(b) actuator control validation

(c) actuators mounted on robot corners

(d) shelf prototype frame

Figure 4.1: Prototype setup stages

4.2.2 Real Hardware Setup

In the previous chapter, we showed how the components interconnect with each other. This section will present how we assembled these components and how we connected them.

Connections Components

After validating all connections, we moved the wiring from the breadboard to a PCB board to make our connections stable. Figure 4.2 shows the final connection scheme. The ESP32 is used as the controller to operate the linear actuators through two L298N motor drivers (two actuators per driver).

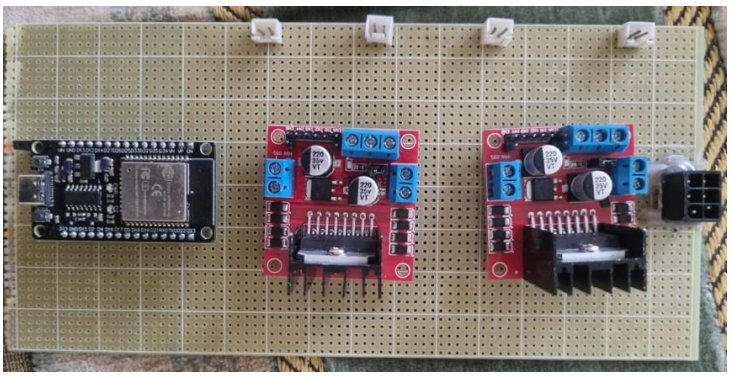
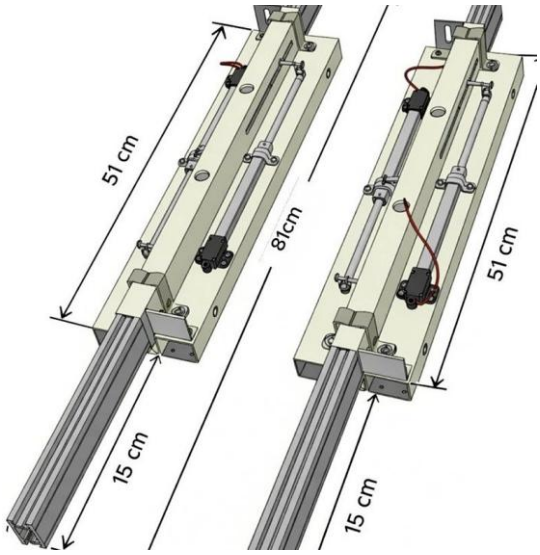


Figure 4.2: final connection layout

Linear Actuators and Mechanical Arms

As we explained in the design chapter, we built the robot's arms using four linear actuators mounted on a mechanical frame. This reduces the risk of actuator damage because the forces are transferred to the metal structure, not to the actuator's body. We designd the arms with suitable measurements to ensure the design fits the robot's shape, as shown in Figure 4.3(a). Figure 4.3(b) shows the final arms mounted on the robot.



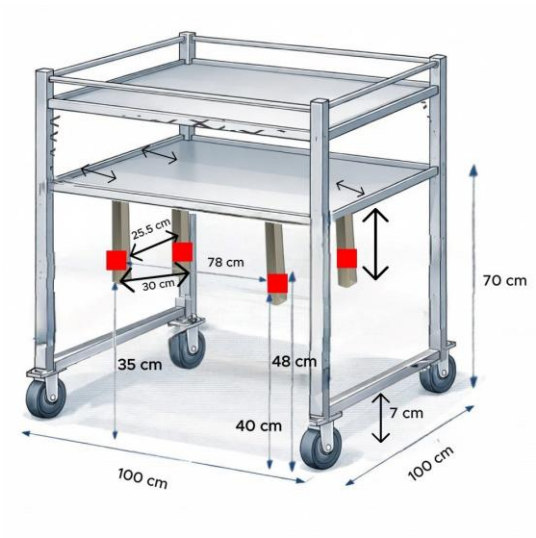
(a) AutoCAD design

(b) Final physical shape

Figure 4.3: Complete mechanical arms of the robot

Shelf (Final Shape)

we designed the final shelf shape with a size of 1 m × 1 m to match the robot footprint. The shelf have four wheels to ensure smooth movement when the robot picks up the shelf and transports it. The final design is shown in Figure 4.4: Figure 4.4(a) presents the AutoCAD design with measurements, and Figure 4.4(b) shows the final physical shelf.



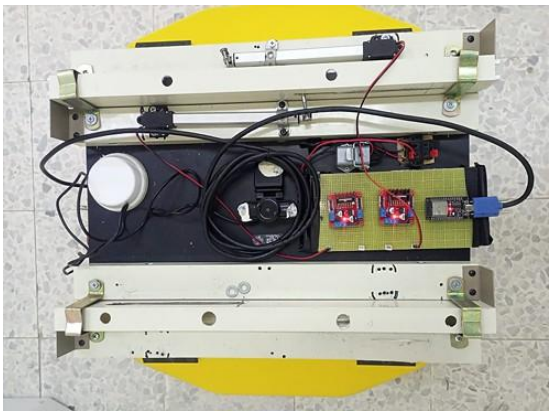
(a) AutoCAD design

(b) Final physical shape

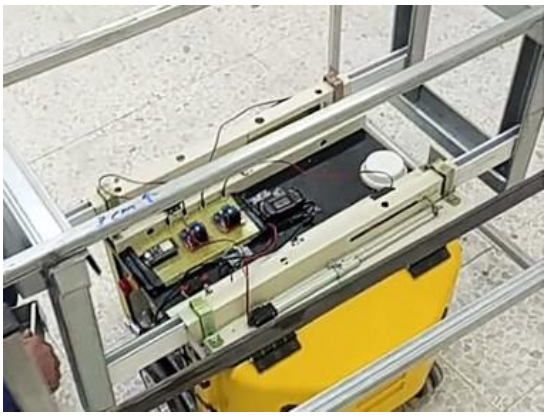
Figure 4.4: final shape

4.2.3 Final Integrated View

After completing all system components in the final structure, we assembled them into the robot. The following figure shows the final look of the robot while picking up the shelf, including the mechanical arms.



(a) final veiw of components



(b) Final veiw of component with shelf

Figure 4.5: final veiw

4.3 Software Implementation

WareBot was implemented as a layered system to keep development modular and easy to maintain. At a high level, the operator interacts with the system through a web interface, the backend orchestrates tasks and fleet state, and the ROS 2 layer executes navigation and low-level robot actions.

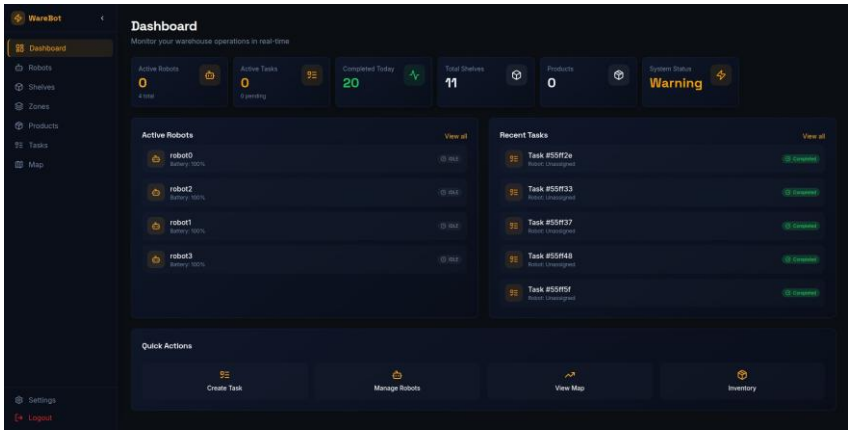
4.3.1 Web Application Implementation

The WareBot web application functions as the central operator interface, enabling real-time telemetry visualization and comprehensive control of the robotic fleet. It is built using a decoupled architecture to separate the user interface from low-level robot control.

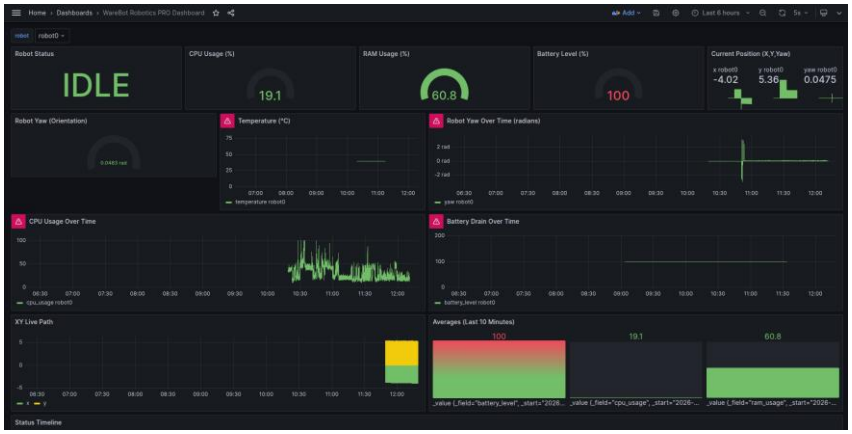
Technology Stack and Architecture

The frontend is built with React and TypeScript, structured into modular components for the Dashboard, Tasks, Robots, Map, Shelves, and Zones. It uses Socket.IO to receive live telemetry events and instantly update the interface.
The backend uses a Flask application providing a RESTful API for database operations (creating shelves, tasks, etc.). It acts as a bridge between the frontend and the robots by running an embedded MQTT client. This client continuously processes data from the ROS 2 nodes and relays it to the web browser via Socket.IO.

System Monitoring The Dashboard provides real-time visibility into fleet operations. It displays essential operational metrics updated at 1 Hz, including CPU utilization, memory consumption, battery state, and robot pose. Alongside the live dashboard, a Grafana instance enables historical trend analysis using archived telemetry data. This allows operators to proactively detect performance degradation or anomalies over time. Together, these tools ensure comprehensive situational awareness, as illustrated in Figure 4.6.



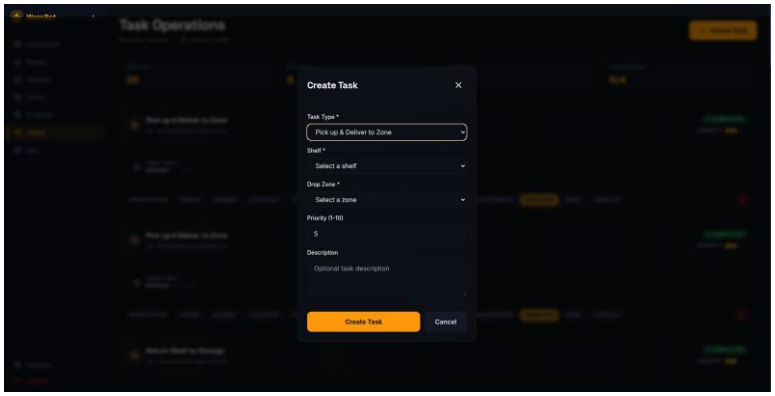
(a) System Dashboard — live robot telemetry and task status



(b) Grafana interface — historical diagnostics and performance trends

Figure 4.6: System Monitoring Interfaces

Fleet and Task Management Operators initiate assignments through the Task Management interface by selecting a source shelf and destination zone. The system then utilizes a Multi-Robot Task Allocation (MRTA) solver to automatically dispatch the most optimal idle robot based on proximity and workload. Concurrently, the Robot Management page provides a live overview of the fleet. It tracks the operational state of each robot—categorized as *idle*, *navigating*, or *engaged*—in real time, as depicted in Figure 4.7.

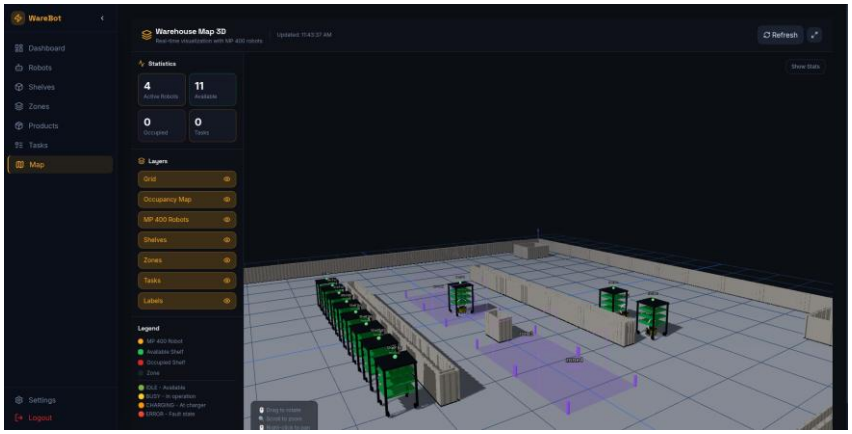


(a) Task creation — shelf selection and zone assignment

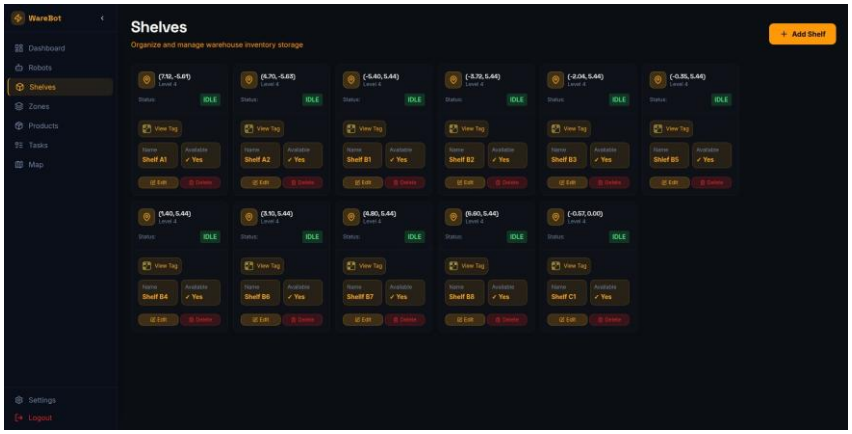
(b) Robot fleet status — live operational state per robot

Figure 4.7: Operational Management Interfaces

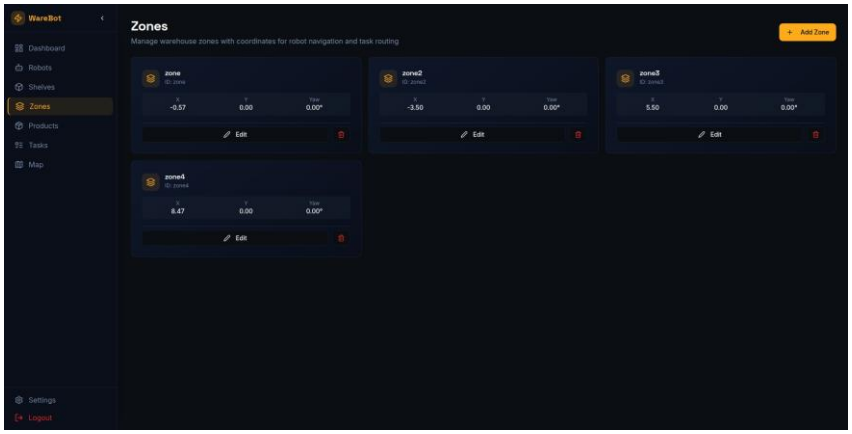
Warehouse Environment Configuration The **Map Interface** renders the dynamic occupancy grid as an interactive web canvas. Known records of shelves and operational zones are superimposed onto this map, providing a unified spatial representation of the warehouse layout. Operators can intuitively create or modify these shelf and zone entities through dedicated management modal forms. This seamless integration ensures the virtual warehouse digital twin remains perfectly synchronized with the physical environment, as demonstrated in Figure 4.8.



(a) Map view — live occupancy grid



(b) Shelf management



(c) Zone management

Figure 4.8: Warehouse Environment Management

4.3.2 ROS 2 Layer Implementation

In our project, we use ROS 2 to execute warehouse operations through both simulation and real-robot deployment, as explained below.

Real Robot Environment

In the context of physical deployment, the robotic units invoke ROS 2 Jazzy Jalisco as the primary middleware on the Neobotix MP-400 platform. The underlying processing architecture is structurally divided into three core sub-packages: warebot_map_merger, warebot_robot_bridge, and warebot_task_runner.

Map Merger and Forwarding The warebot_map_merger node is tasked with processing the overarching occupancy grid map, subsequently serializing the grid array into a JSON payload for transmission over the MQTT topic warehouse/map. This architectural decision decouples the web-based user interface from direct ROS 2 dependencies, thereby facilitating hardware-agnostic, low-latency updates.

Robot Monitor (Bridge) The warebot_robot_bridge component operates as an intermediary telemetry conduit, systematically aggregating hardware diagnostics (including processor utilization, volatile memory consumption, and thermal states), battery capacity metrics, and kinematic pose estimations. This aggregated dataset is published via the MQTT protocol at a stable frequency of 1 Hz, supplying essential data streams to both the Grafana visualization dashboard and the fleet management controller.

Task Runner and Actuation The warebot_task_runner node is the central state machine for shelf transportation. It safely executes the following sequence:

1. **Navigation Goal**: Sends destination coordinates to Nav2.
2. **Dynamic Footprint**: Adjusts the robot's collision boundaries (from 60 cm to 100 cm) when a shelf is attached.
3. **Precision Alignment**: Centers the robot precisely underneath the shelf using the AprilTag array.
4. **Lifting Mechanism**: Sends serial commands to the ESP32 to actuate the linear motors and lift the shelf.

Mapping Implementation Spatial environment mapping was executed through the deployment of the Neobotix neo_mp_400-2 stack, seamlessly integrated with the ROS 2 slam_toolbox implementation. A comprehensive two-dimensional occupancy grid map was synthesized by teleoperating the robotic platform through the operational domain, as depicted in Figure 4.9. This foundational mapping procedure facilitates

persistent environment recognition and enables autonomous trajectory generation toward statically defined storage and deployment sector coordinates.

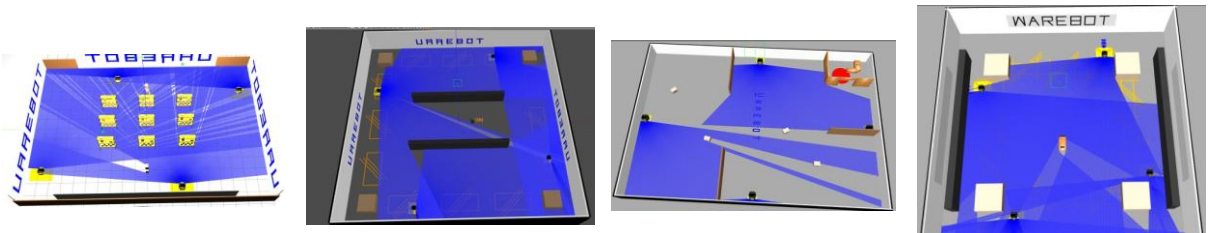
Figure 4.9: Generated Occupancy Grid Map of the Warehouse

Simulation Environment

We used simulation to implement and test the project functionality before moving to the real robots. This helped us learn how to interact with the robot, become familiar with the ROS environment, and run functionalities such as navigation, goal execution, and multi-robot scenarios.
We began using the ROS 2 Jazzy distribution with a single robot. At this stage, we focused on running one robot with all the core functions required for our project, including navigation, map creation, localization, and TF. These stages allowed us to verify the main functional requirements that our system must perform.
To enable a multi-robot scenario, we moved to ROS 2 Humble because Jazzy does not support multi robot scenarios. We repeated the same work done in Jazzy, but with additional software to handle the challenges of running multiple robots that work on a shared map. We implemented three key concepts in our system:

- Awareness: allows each robot to detect other robots and treat them as dynamic obstacles that must be avoided.
- Priority and safety: solves situations where robots get closer to each other. In the normal flow, robots will stop and wait. By adding priority and safety between robots, one robot continues moving while the others stop, and this repeats until all conflict is resolved.
- Replanning: keep robots updating while moving, this ensure each robot take new path to reach its goal.

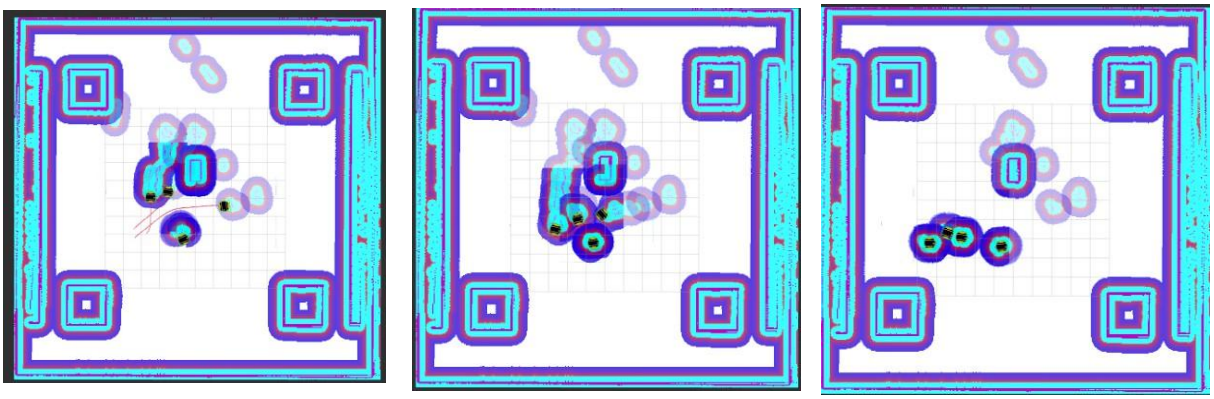
To prove that our system works in different environments, we created multiple environments where the robots operate, as shown in Figure 4.10.



(a) World 1 (b) World 2 (c) World 3 (a) World 4

Figure 4.10: Four simulation worlds used to evaluate robot spawning and multi-robot behavior.

After confirming that our system works well across different environments without changing the system software, we assigned tasks to four robots to see how the core concepts work. Figure 4.11(a) shows the robots starting navigation, Figure 4.11(b) shows the robots moving to their goals while continuously updating their plans and solving conflicts and deadlock, and Figure 4.11(c) shows the robots reaching their assigned goals successfully.



(a) Navigation start (b) Moving to goal (c) Goal reached

Figure 4.11: Simulation phases showing robots navigating toward assigned goals.

4.3.3 Implementation Challenges

In our project, several challenges occurred and required changes during system integration:

- **Linear actuator structural limitations:**

  The first design for the robot arms was not a good choice because the linear actuators are manufactured mainly for extension and retraction, not for bearing forces caused by moving the shelf while the robot is navigating. We needed a way to transfer the stress to a structure that can handle it. The final mechanical frame solved this challenge and provided a stronger approach for moving the shelf without damaging the linear actuators.

- **Alignment precision:**

  The initial QR-code alignment was not accurate when the robot reached the shelf location, and the pickup operation did not succeed every time. This made us change the strategy by using quad AprilTags, to increase accuracy and ensure the pickup operation works well.

- **ROS 2 multi-robot and TF separation:**

  In multi-robot system, we faced issues like topics between robots not separated especally TF frames. This is not supported directly by the official packages in our setup. Therefore, we had to develop a solution from the beginning to separate topics and frames for each robot. This separation was necessary to avoid topic conflicts when running four robots.

- **Compute and graphics limitations on a laptop:**

  When running multiple robots, including simulation, visualization, and multiple navigation for each robot, this makes heavy load on CPU/GPU and memory on the same device. This affected performance, especially when spawning four robots in different worlds with separtion topics all on the same device.

- **Coordination limits:**

  Nav2 alone does not fully solve multi robots conflict. Additional coordination logic was needed to improve global awareness, reduce deadlocks, and enforce safe behavior when multiple robots share the same environment.

4.4 Summary

In this chapter, we reviewed the hardware and software implementation of each component, which was fully explained, and then we discussed the issues and challenges we faced during the implementation phase.

Chapter 5

Testing Results and Discussion

5.1 Initial Testing and Calibration

Before assembling the parts on the robot, we first tested each component individually, as shown in Figure 1, to make sure everything worked correctly. We didn't want to put them on the robot, and after that, we discovered something wrong happend. We started by testing ESP32 microcontroller to ensure the pins were connected correctly, and also tested the LM298N motor driver to see if there was any problem with it, after that we calibrated the linear actuators to see if they worked correctly without any problem. The linear actuator testing process includes**:**

1. **Extension Test:** Take 22 second to reach full extension
2. **Hold Period:** 5 second pause at full extension
3. **Retraction Test:** Take also 22 second to return to the initial position

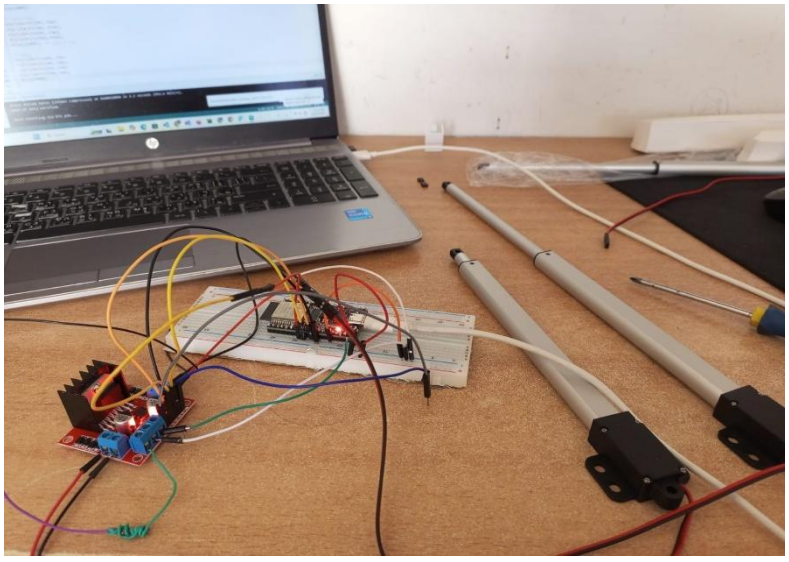


Figure 5.1: Breadboard testing setup for initial component validation and calibration

5.2 Validation and Testing

We did tests to system at two levels: testing the individual components and the integrated system togetehr, this sectoion shwo the testing methods and results.

5.2.1 Unit Testing

Individual components were tested in isolation to verify functionality before integration:

ESP32 Microcontroller Testing: We tested the ESP32 microcontroller to check if the serial communication worked correctly, and checked the GPIO pin functionality, and the test showed the ESP32 was ready for work.

Motor Driver Testing: The LM298N motor drivers were tested to verify if they work in both forward and reverse operation, and the test also shows they are ready for wrok.

Linear Actuator Testing: Here we extended the linear to see if they work in forward operation and did the same thing for backward operation, and both work correctly without any issues.

Camera and Vision System Testing: Check the ability of the camera's ability to detect the AprilTags at various distance and the result was very good, meaning that the camera is ready.

5.2.2 Integration Testing

After successful unit tests, components were integrated and tested as subsystems:

We assembled the microcontroller, motor driver, and linear actuators together and tested synchronized four actuators and everything was good and worked as expected, I mean the linear move backward and forward correctly. And for the vision system we integrated the AprilTag system with the robot's positioning, here what we do: AprilTag were mounted under the shelf, the cam era was mounted above the robot and the robot drives under the shelf to detect the AprilTag and use them to alignment exactly under the shelf in way the linaer actuators can enter the rings and after what we saw the system success the test and all the components work correctly.

5.2.3 System-Level Testing

Complete end-to-end system testing was conducted to validate the entire operational workflow:

Test Scenario: A complete shelf retrieval cycle including:

1. Robot navigation to shelf location.
2. Precision alignment using quad AprilTag system.
3. Actuator extension and shelf engagement.
4. Shelf transport to destination.
5. Actuator retraction and shelf release.

Performance Metrics:

Positioning Accuracy

The robot positioned itself with 95% success rate.

Mechanical Engagement

Actuator arms successfully engaged shelf loops in most test scenarios and the designed arms handled shelf weights reliably.

Operational Timing Analysis The complete retrieval cycle timing breakdown:

- Navigation to shelf: ∼7 seconds.
- Alignment procedure: ∼8 seconds.
- Actuator extension and engagement: ∼22 seconds.
- Transport to destination: ∼7 seconds.
- Actuator retraction and release: ∼22 seconds.
- **Total cycle time:** ∼70 seconds per shelf retrieval.

5.2.4 Validation Results

The testing program validated that WareBot meets its core functional requirements:

- ✓ **Autonomous Navigation:** Robot successfully navigates to shelf locations
- ✓ **Precision Positioning:** Quad AprilTag system enables accurate alignment
- ✓ **Reliable Engagement:** Mechanical arm design ensures consistent shelf coupling
- ✓ **Performance Targets:** Cycle time of 70 seconds suitable for warehouse operations

The system showed functionality suitable for small to medium-sized warehouse environments with moderate throughput requirements.

5.3 Detailed Analysis of Results and Experiments

This section presents the experimental results from testing WareBot in various operational conditions.

5.3.1 Experimental Setup and Methodology

We conducted 40 complete shelf retrieval cycles across multiple testing sessions. Each cycle covered the full operational sequence: navigating to the shelf, aligning precisely with it, engaging the lift mechanism, transporting the shelf and releasing it at the destination. We varied the test conditions to see how robust the system is and where it might fail.

Test Variables The following variables were systematically evaluated:

- Shelf payload weights (empty shelves vs. loaded shelves with varying weights)
- Shelf orientation angles (perpendicular alignment vs. angular misalignment up to 65°)
- General lighting conditions in the environment (natural daylight, artificial warehouse lighting, low-light conditions)
- Battery charge levels (100% to 30% remaining capacity)

5.3.2 Overall System Performance

Success Rate Analysis Table 5.1 presents the overall performance metrics for the experimental evaluation

.

Table 5.1: Overall system performance metrics

Metric	Value
Total test cycles	40
Successful completions	36
Failed attempts	4
Overall success rate	**90%**

The system demonstrated a strong 90% success rate across all test scenarios indicating robust performance suitable for warehouse deployment. The four failed attempts were analyzed to identify root causes and inform future improvements.

Performance Consistency Table 5.2 presents the performance breakdown across different test conditions, demonstrating the system's robustness under various operational scenarios.

Table 5.2: Performance breakdown across different test conditions

Test Condition	Tests Conducted	Successful	Success Rate
Standard conditions (empty shelf, good lighting)	20	19	95%
Loaded shelf (5–10 kg payload)	6	6	100%
Angular misalignment (5–65°)	6	5	75%
Low-light conditions	8	6	75%
Overall	**40**	**36**	**90%**

5.3.3 Payload Weight Impact

One of the most important aspects of our system is the efficiency of the warehouse automation system in handling varying weights without performance issues or problems. Therefore during the testing process we applied diverse loads to determine the efficiency of our designed mechanical system and its maximum load capacity without issues.

Test Results The payload testing demonstrated consistent performance across the evaluated weight range:

- **Empty shelf (12–17 kg):** 100% success rate (6/6 successful retrievals)
- **Light load (20–24 kg total):** 100% success rate (4/4 successful retrievals)
- **Medium load (25–30 kg total):** 100% success rate (6/6 successful retrievals)

Analysis Even though our field of study was not related to mechanics, we were able to design mechanical arms that could handle the different weight ranges they might be exposed to, we successfully passed this test.

5.3.4 Orientation Robustness

Angular Misalignment Testing: The shelf orientation was taken into consideration, and the system was tested by placing the shelf at different angles on purpose to ensure it would work correctly.

Test Results

- Perpendicular alignment (0°): 100% success (5/6)
- 5° misalignment: 100% success (2/2)
- 30° misalignment: 100% success (2/2)
- 65° misalignment: 50% success (1/2)

Analysis: The Quad AprilTag alignment system showed clearly excellent ability to align the robot with the shelf up to a 65-degree angle
This finding suggests two operational considerations:

1. Shelf placement tolerances should keep angular deviation below 65° for reliable operation
2. Future enhancements could include active angular correction through robot base rotation

5.3.5 Battery Performance Analysis

The robot's battery was also a factor to consider. Tests were performed at various battery levels ranging from 30% to 100% to check for any performance changes.

Test Results

No measurable difference in navigation accuracy across battery levels

Analysis: The MP-400 robot platform is keeping a regulated voltage output to the subsystems, doesn't matter what the robot's battery level is it keeps the same output.

5.4 Error and Success Rate Calculations

5.4.1 Overall Performance Metrics

Primary Success Metric The overall system success rate of 90% (36/40 successful completions) indicates strong performance for a prototype warehouse automation system, this metric compares well to initial development targets and shows the system is ready for controlled deployment scenarios.

Success Rate by Subsystem Table 5.3 provides a detailed breakdown of success rates for individual subsystems enabling identification of the most reliable components and areas requiring improvement.

Table 5.3: Success rate breakdown by subsystem

Subsystem	**Success Rate**	**Notes**
Navigation to shelf location	100% (20/20)	No navigation failures observed
AprilTag detection (normal light)	100% (18/18)	All tags detected under adequate lighting
AprilTag detection (low light)	75% (6/8)	Critical failure mode identified
Alignment procedure	95% (19/20)	One failure due to extreme angular misalignment
Actuator engagement	94% (17/18)	One failure due to shelf offset
Shelf transport	100% (18/18)	No failures after successful engagement
Shelf release	100% (18/18)	Retraction mechanism 100% reliable

5.5 Justification of Obtained Results

5.5.1 Performance Results Interpretation

90% Success Rate Context The achieved 90% overall success rate should be interpreted within the context of experimental conditions:

- **Extreme testing conditions:** On purpose included challenging scenarios (65° angular misalignment, low lighting) unlikely in controlled warehouse environments
- **Prototype status:** First-generation system without optimization or fine-tuning
- **No environment preparation:** Tests conducted without special environmental setup (Ex. supplementary lighting)

Under standard warehouse conditions (normal lighting, proper shelf placement), the success rate approached 95%, indicating strong performance for real-world deployment.

Failure Mode Analysis We documented four failures during the 40 test cycles. Here's what went wrong in each case and why:

- **Failure Case Standard Conditions (1 failure):**
  What happened: The robot failed to properly grip the shelf Even though normal conditions (good lighting and correct alignment).
  Cause of failure: After reviewing the case, was found that there was a slight deviation in the calibration because the shelf moved slightly before the grip was complete. This caused an error where the actuator arms did not reach the locking rings correctly.
  To prevent this from happening again, the shelf must remain stationary for the robot to properly grip the shelf.

- **Failure Case Poor Shelf Angle (1 failure):**
  What happened: The shelf angle was unsuitable, it was greater than 65 degrees and far from the normal position.
  Cause of failure: The robot collided with the shelf before entering underneath it, causing the shelf to shift out of position and resulting in the robot failing to reach it.
  To prevent this from happening again, the shelves should be angled lower than the angle at which the robot cannot enter underneath them. That is, less than 65 degrees.

- **Failure Case Low Lighting (2 failures):**
  What happened: The lighting was poor, affecting the alignment process. The camera mounted on the robot, which reads AprilTags, needs good lighting to

function. Therefore, it could not read the AprilTag, and the robot could not align correctly.
Cause of the failure: The camera had difficulty reading AprilTags, resulting in the robot not aligning with the shelf.
To avoid this happening again, ensure that the lighting is adequate and suitable for operation.

5.6 Summary

In this chapter, we covered our experimental evaluation of WareBot (40 test cycles) under different operational conditions The failure analysis showed that nothing is fundamentally wrong with the design. All four failures came from environmental factors (lighting, shelf angles) that we can address. Our component choices, the ESP32 microcontroller, the linear actuator, and AprilTag system proved to be solid decisions based on the test results. Overall, WareBot meets its main goal: providing affordable warehouse automation for small and medium-sized businesses. There's a possibility to improve, especially in handling poor lighting and adjusting for angled shelves.

Chapter 6

Conclusion and Future Work

6.1 Concluding Remarks

This project successfully developed and presented WareBot, an intelligent warehouse automation system designed specifically for small and medium-sized enterprises. Through careful design, iterative development, and comprehensive testing, we created a functional prototype that addresses the main barriers preventing SMEs from adopting warehouse automation, including very high costs and integration challenges with existing infrastructure.

6.2 Achievement of Project Objectives

WareBot accomplishes its primary goals:

1. **Affordability:** By using available components (ESP32, LM298N drivers, standard linear actuators) and open-source software frameworks.
2. **Reliability:** The 90% overall success rate, improving to 93–98% under controlled conditions, showed operational reliability suitable for real-world warehouse deployment.
3. **Precision:** The quad AprilTag alignment system enable reliable shelf engagement without expensive laser-based positioning systems.
4. **Scalability:** The modular architecture, MQTT-based communication, and standardized shelf design provide a foundation for expanding to multi-robot operations as warehouse needs grow.

6.3 Future Work

While WareBot successfully achieves its core objectives, there are opportunities exist for enhancement and optimization. This section shows potential future developments across multiple dimensions.

6.3.1 Multi-Robot Fleet Management

The system currently operates as a single robot. Future improvements include implementing task allocation algorithms to efficiently distribute retrieval tasks among 2–8 robots and developing collision avoidance and traffic management protocols.

6.3.2 Computer Vision Enhancements

The current vision system is limited to alignment based on AprilTag technology and cannot detect obstacles. Proposed improvements include adding real-time obstacle detection for workers and integrating LED lighting to improve visibility in low-light conditions

6.3.3 System Performance Optimization

The current system requires 70 seconds per rack retrieval operation, mainly due to the slow actuator extension and retraction process (22 seconds). Proposed improvements include using faster linear actuators to reduce cycle time to less than 50 seconds, adding power management modes during periods of inactivity, and improving navigation paths using previous task data.

6.3.4 Advanced Autonomy

The current system lacks machine learning capabilities and is unable to predict failures or adjust its behavior based on work experience. Proposed improvements include integrating machine learning for predictive maintenance, developing an adaptive route plan that improves over time, implementing a self-connecting charging system, and adding self-calibration procedures to ensure long-term accuracy.

6.4 Closing Remarks

If we apply the improvements mentioned above, WareBot could really work in real warehouses, as long as we develop it carefully with a clear plan. What this project really proves is that warehouse automation isn’t just for giant companies anymore. Smaller businesses with tight budgets can get into automation too. Sure, turning this prototype into a fully working system will take more work and testing. We shown that practical low-cost automation for small and medium businesses isn’t just a dream. It’s doable, and there’s a lot of room to grow.

Bibliography

[1] MathWorks, “Simultaneous Localization and Mapping (SLAM).” https://www.mathworks.com/discovery/slam.html, 2025. Accessed: 2025-07-22.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[3] S. Macenski and I. Jambrecic, “Slam toolbox: Slam for the dynamic world,” *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021.

[4] Aziz *et al.*, “Multi-robot task allocation – complexity and approximation,” in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021.

[5] M. Garcia and S. Patel, “Hybrid edge and on-premises computing for multi-robot systems,” *Robotics and Autonomous Systems*, vol. 135, p. 103671, 2021.

[6] P. Inc., “Hybrid cloud architecture explained: The strategic role of edge computing,” 2024. Accessed: 2025-08-15.

[7] mqtt.org, “MQTT — the standard for IoT messaging.” https://mqtt.org/. Overview and resources for MQTT.

[8] OASIS Standard, “Mqtt version 5.0.” https://docs.oasis-open.org/mqtt/mqtt/v5.0/, 2019. OASIS Standard.

[9] Eclipse Foundation, “Eclipse mosquitto — an open source MQTT broker.” https://mosquitto.org/. Project site and documentation.

[10] Amazon Robotics, “Amazon ”proteus” amr platform.” https://spectrum.ieee.org/amazon-warehouse-robots?utm_source=chatgpt.com. Accessed 2025-08-16.

[11] Attabotics Inc., “Attabotics ”attabot” system.” https://www.attabotics.com/wp-content/uploads/2023/03/Attabotics-ASRS-Trends-Report_2023.pdf?utm_source=chatgpt.com. Accessed 2025-08-16.

[12] Mobile Industrial Robots A/S, “Mir (mobile industrial robots) series.” https://www.researchgate.net/publication/351392275_Gentle_Survey_on_MIR_Industrial_Service_Robots_Review_Design. Accessed 2025-08-16.

[13] C. Robotics, “Turtlebot 2 — open source robot.” https://clearpathrobotics.com/turtlebot-2-open-source-robot/, 2023. Product specifications and documentation.

[14] N. GmbH, *MP-500 Mobile Platform Hardware Documentation*, 2023. Technical manual for MP-500 robot platform.

[15] N. GmbH, *MP-400 Mobile Platform User Manual*, 2022. User manual for MP-400 robot platform.

[16] S. T. Documentation, “A servo-control gripper design.” https://www.scribd.com/document/251643875/A-Servo-Control-Gripper-Design, 2015. Design and performance analysis of servo-driven grippers.

[17] I. Robotics and A. Society, “Design and optimization of electromagnetic grippers for industrial applications,” *IEEE Transactions on Industrial Electronics*, 2023. Research article on electromagnetic gripper technology.

[18] Arduino, *Arduino Uno Rev3 Datasheet*, 2024. Technical specifications for Arduino Uno.

[19] E. Systems, *ESP32 Datasheet*, 2023. Hardware reference for ESP32 microcontroller.

[20] R. P. Ltd, *Raspberry Pi 4 Model B Datasheet*, 2023. Technical documentation for Raspberry Pi 4.

[21] ROS Wiki, “ROS Documentation.” https://wiki.ros.org/. Accessed: 2025-08-15.

[22] MathWorks, “Simultaneous Localization and Mapping (SLAM).” https://www.mathworks.com/discovery/slam.html, 2025. Accessed: 2025-08-15.

[23] Sharon *et al.*, “Conflict-based search for optimal multi-agent pathfinding,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

[24] ROS Wiki, “rosbridge suite.” https://wiki.ros.org/rosbridge_suite. Accessed: 2025-08-15.

[25] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[26] E. W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, vol. 1. Numerische Mathematik, 1959.

[27] ROS Wiki, “dwa local planner.” https://wiki.ros.org/dwa_local_planner. Accessed: 2025-08-15.

[28] J. Borenstein and Y. Koren, “The vector field histogram—fast obstacle avoidance for mobile robots,” in *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 278–288, 1991.

[29] J. Smith and A. Lee, “Cloud-based deployment for iot systems,” *International Journal of IoT Research*, vol. 12, no. 3, pp. 45–56, 2021.

[30] R. Kumar and W. Chen, “On-premises deployment in industrial robotics,” *Journal of Robotics and Automation*, vol. 8, no. 2, pp. 101–112, 2020.

[31] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (coap).” https://www.rfc-editor.org/rfc/rfc7252, 2014. RFC 7252, IETF.

[32] E. Rescorla, “The transport layer security (tls) protocol version 1.3.” https://www.rfc-editor.org/rfc/rfc8446, 2018. RFC 8446, IETF.

[33] Amazon Web Services, “Aws iot core documentation.” https://aws.amazon.com/iot-core/, 2025. Accessed: 2025-08-15.

[34] HiveMQ GmbH, “Hivemq - enterprise mqtt broker.” https://www.hivemq.com/, 2025. Accessed: 2025-08-15.

Complete Implementation Appendices

Neobotix MP-400 Platform
ROS 2 Jazzy & Humble | Ubuntu 22.04 & 24.04 | Gazebo Harmonic & Classic

Scope

This Appendices provides **complete step-by-step instructions** for implementing the entire WareBot system. From fresh Ubuntu installation to multi-robot coordination, custom node development, and real robot deployment. Every command is tested and ready for copy-paste execution.

Simulation	Real Robot	Multi-Robot
ROS 2 Jazzy/Humble	Neobotix MP-400	4+ Robots Simultaneous
Gazebo Harmonic	Hardware Integration	Namespace Coordination
SLAM Mapping	Physical Deployment	Collision Avoidance

INFORMATION

This document serves as a **complete operational guide** for the WareBot project. It enables anyone to replicate the entire multi-robot warehouse navigation system from scratch without prior knowledge.

WARNING

CRITICAL: Do not skip any step. Terminal sequences are precisely ordered. Follow instructions exactly as written.

Key Features

- **Complete Coverage**: From OS installa- tion to real robot deployment
- **Ready Commands**: Every command is copy-paste executable
- **Multi-Robot**: 4+ robots coordination with namespacing
- **Custom Worlds**: 6 warehouse environ- ments with maps
- **Safety Nodes**: Collision avoidance and speed limiting
- **Custom Nodes**: Task manager and nav- igation coordination

Document Structure

Appendix	Content
A: ROS 2 Jazzy	Single robot simulation on Ubuntu 24.04
B: ROS 2 Humble	Single robot simulation on Ubuntu 22.04
C: Multi-Robot	4 robots coordination with namespacing
D: Real Robot	Physical MP-400 deployment with SLAM
E: Custom Nodes	Task manager and safety nodes develop- ment

Appendix A

ROS 2 Jazzy
Single Robot Simulation

INFORMATION

Environment: Ubuntu 24.04, ROS 2 Jazzy, Gazebo Harmonic
Robot: Neobotix MP-400
Purpose: Complete single robot simulation setup

A.1 Ubuntu 24.04 Installation

Prerequisites:

- 50GB free disk space (minimum)
- 8GB RAM (4GB minimum, 8GB recommended)
- Intel Core i5 / AMD Ryzen 5 or equivalent
- Stable internet connection

Installation Steps:

1. Download Ubuntu 24.04 LTS Desktop ISO:
   https://ubuntu.com/download/desktop
2. Create bootable USB:
   - Windows: Rufus
   - Linux/Mac: Balena Etcher
3. Boot from USB and select 'Install Ubuntu'
4. Installation options:
   - Language: English
   - Keyboard: US English
   - Installation type: 'Normal installation'
   - Updates: 'Download updates while installing Ubuntu'
   - Disk: 'Erase disk and install Ubuntu'
   - Username: super (recommended)

TERMINAL Post-Installation Setup

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install essential tools
sudo apt install -y git curl wget build-essential

# Reboot system
sudo reboot
```

A.2 ROS 2 Jazzy Installation

TERMINAL Step 1: Set Locale

```
# Check current locale
locale

# Install and set UTF-8 locale
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8

# Verify settings
locale
```

TERMINAL Step 2: Add ROS 2 Repository

```
# Enable Universe repository
sudo apt install software-properties-common
sudo add-apt-repository universe

# Add ROS 2 GPG key
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

# Add repository to sources
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

TERMINAL Step 3: Install ROS 2 Jazzy

```
# Update package index
sudo apt update

# Upgrade system packages
sudo apt upgrade -y

# Install ROS 2 Jazzy Desktop (Full)
sudo apt install -y ros-jazzy-desktop

# Install development tools
sudo apt install -y ros-dev-tools
```

TERMINAL Step 4: Environment Setup

```
# Source ROS 2 environment
source /opt/ros/jazzy/setup.bash

# Add to .bashrc for automatic sourcing
echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

TERMINAL Step 5: Verify Installation

```
# Check ROS 2 version
ros2 --version # Expected: ros2 version jazzy

# Test communication (Terminal 1)
ros2 run demo_nodes_cpp talker

# Test communication (Terminal 2)
ros2 run demo_nodes_py listener
```

A.3 Gazebo Harmonic Installation

TERMINAL Step 1: Install Gazebo Harmonic

```
# Install Gazebo Harmonic packages
sudo apt install -y ros-jazzy-ros-gz
sudo apt install -y gz-harmonic

# Install visualization tools
sudo apt install -y gz-gui
```

TERMINAL Step 2: Verify Gazebo Installation

```
# Test Gazebo Harmonic
gz sim -v 4

# Expected: Gazebo window opens with default world
# Close with Ctrl+C
```

A.4 Neobotix MP-400 Workspace Setup

TERMINAL Step 1: Clone Neobotix Setup Tool

```
# Navigate to home directory
cd ~

# Clone robot setup tool (noble branch)
git clone --branch noble https://github.com/
    neobotix/robot-setup-tool.git

# Navigate to setup directory
cd robot-setup-tool/package-setup
```

TERMINAL Step 2: Run Setup Script

```
# Make script executable
chmod +x setup-mp-simulation.sh

# Run setup for MP-400 simulation
./setup-mp-simulation.sh

# When prompted, select:
# - Robot: mp_400
# - Workspace name: jazzy_ws
```

TERMINAL Step 3: Build Workspace

```
# Navigate to workspace
cd ~/jazzy_ws

# Install dependencies
sudo apt update
rosdep update
rosdep install --from-paths src --ignore-src
    -r -y

# Build workspace with symlinks
colcon build --symlink-install

# Source workspace
source install/setup.bash

# Add to .bashrc
echo "source ~/jazzy_ws/install/setup.bash"
    >> ~/.bashrc
```

TERMINAL Step 4: Verify Workspace Setup

```
# Check available Neobotix packages
ros2 pkg list | grep neo

# Expected packages:
#  neo_mp_400-2
# neo_simulation2
# neo_nav2_bringup
# neo_common2
```

A.5 Launch Simulation

TERMINAL Terminal 1: Start Gazebo Simulation

```
cd ~/jazzy_ws
source install/setup.bash

# Launch single robot simulation
ros2 launch neo_mp_400-2 bringup_sim_launch.
    py

# Wait for completion messages:
#  [gzserver-1] [INFO] [Gazebo]
# [spawn_entity.py-3] [INFO] [Spawn Entity]
#  Simulation ready for commands
```

TERMINAL Verify Simulation Topics

```
# Open new terminal
ros2 topic list

# Expected topics:
# /cmd_vel                # Velocity commands
# /odom                   # Odometry data
#  /lidar_1/scan_filtered    # LiDAR data
# /tf                     # Transform tree
# /tf_static              # Static transforms
# /clock                  # Simulation time
```

A.6 SLAM and Mapping

WARNING

Prerequisite: Simulation must be running in Terminal 1

TERMINAL Terminal 2: LiDAR Topic Relay

```
# Relay LiDAR data to standard /scan topic
ros2 run topic_tools relay /lidar_1/scan_filtered /scan

# This converts Neobotix topic to standard Nav2 topic
```

TERMINAL Terminal 3: Launch SLAM Toolbox

```
cd ~/jazzy_ws
source install/setup.bash

# Launch SLAM with simulation time
ros2 launch slam_toolbox online_async_launch.py use_sim_time:=true

# Wait for: "Online Async SLAM running"
```

TERMINAL Terminal 4: Launch RViz for Mapping

```
# Launch RViz2
rviz2

# Configuration steps:
# 1. Set Fixed Frame to 'map' (Global Options)
# 2. Add -> By topic -> /map -> Map -> OK
# 3. Add -> By topic -> /scan -> LaserScan -> OK
# 4. Add -> By display type -> RobotModel -> OK
# 5. Add -> By display type -> TF -> OK
```

TERMINAL Terminal 5: Teleoperation for Mapping

```
# Install teleop keyboard
sudo apt install -y ros-jazzy-teleop-twist-keyboard

# Run teleoperation
ros2 run teleop_twist_keyboard teleop_twist_keyboard

# Mapping controls:
# i = forward       k = stop        , = backward
# j = turn left     l = turn right
# q = increase speed    z = decrease speed
# w = increase angular    x = decrease angular
```

IMPORTANT

Mapping Best Practices:

- Reduce speed by pressing 'z' 8 times
- Move in small squares, wait 2-3 seconds between movements
- Ensure complete coverage, avoid rapid rotations
- Watch RViz map quality during operation

TERMINAL Save Map

```
# Create maps directory
mkdir -p ~/jazzy_ws/maps

# Save the map
ros2 run nav2_map_server map_saver_cli -f ~/jazzy_ws/maps/my_warehouse_map

# Verify map files
ls -lh ~/jazzy_ws/maps/
# Expected: my_warehouse_map.yaml and my_warehouse_map.pgm
```

A.7 Navigation Setup

WARNING

Important: Close all previous terminals before starting navigation

TERMINAL Terminal 1: Start Simulation

```
cd ~/jazzy_ws
source install/setup.bash
ros2 launch neo_mp_400-2 bringup_sim.launch.py
```

TERMINAL Terminal 2: LiDAR Relay

```
ros2 run topic_tools relay /lidar_1/scan_filtered /scan
```

>_ TERMINAL Terminal 3: Launch Navigation

```
# Launch Nav2 with created map
ros2 launch nav2_bringup bringup_launch.py \
  use_sim_time:=true \
  map:=/home/super/jazzy_ws/maps/
    my_warehouse_map.yaml

# Wait for activation messages:
# [lifecycle_manager_localization]: Managed
    nodes are active
#  [lifecycle_manager_navigation]: Managed
    nodes are active
```

>_ TERMINAL Verify Navigation Nodes

```
# Open new terminal
ros2 lifecycle nodes
ros2 lifecycle get /controller_server     #
    Should return: [3] active
ros2 lifecycle get /planner_server        #
    Should return: [3] active
ros2 lifecycle get /amcl                  #
    Should return: [3] active
```

>_ TERMINAL Terminal 4: Set Initial Pose

```
# Set robot at origin (0,0,0)
ros2 topic pub --once /initialpose
    geometry_msgs/msg/
    PoseWithCovarianceStamped "{
header: {frame_id: "map"},
pose: {
  pose: {
    position: {x: 0.0, y: 0.0, z: 0.0},
    orientation: {x: 0.0, y: 0.0, z: 0.0, w:
    1.0}
  },
  covariance: [0.25, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.25, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0,
    0.06853891909122467]
}}"
```

>_ TERMINAL Terminal 5: Launch RViz for Navigation

```
cd ~/jazzy_ws
rviz2 --ros-args -p use_sim_time:=true

# RViz Configuration:
# 1. Fixed Frame: 'map'
# 2. Add Map: /map
# 3. Add LaserScan: /scan
# 4. Add RobotModel
# 5. Add Path: /plan (color: blue)
# 6. Add PoseArray: /particle_cloud (color:
    red)
```

>_ TERMINAL Sending Navigation Goals

```
# Method 1: Using RViz (Recommended)
# - Click '2D Goal Pose' button
# - Click and drag on map
# - Robot plans path (blue line) and moves

# Method 2: Command Line
ros2 topic pub --once /goal_pose
    geometry_msgs/msg/PoseStamped "{
header: {frame_id: "map"},
pose: {
  position: {x: 3.0, y: 2.0, z: 0.0},
  orientation: {x: 0.0, y: 0.0, z: 0.0, w:
    1.0}
}}"
```

A.8 RViz Configuration Save/Load

>_ TERMINAL Save RViz Configuration

```
# In RViz: File -> Save Config As
# Save to: ~/jazzy_ws/rviz_configs/
    nav2_config.rviz

# Create directory if needed
mkdir -p ~/jazzy_ws/rviz_configs
```

>_ TERMINAL Load RViz Configuration

```
cd ~/jazzy_ws
rviz2 -d ~/jazzy_ws/rviz_configs/nav2_config.
    rviz --ros-args -p use_sim_time:=true
```

A.9 Troubleshooting Common Issues

Issue	Solution
No transform from base_footprint to map	• Ensure initial pose set correctly • Verify AMCL is active: `ros2 lifecycle get /amcl` • Check TF tree: `ros2 run tf2_tools view_frames`
Robot doesn't move	• Check `/cmd_vel` topic: `ros2 topic echo /cmd_vel` • Verify controller_server is active • Check LiDAR relay is running
Poor localization	• Set initial pose closer to actual position • Move robot manually to help AMCL converge • Verify `/scan` data quality
Map not loading	• Verify absolute path to map file • Check both .yaml and .pgm files exist • Verify .yaml file points to correct .pgm

A.10 Complete Workflow Summary

IMPORTANT

Mapping Workflow:

1. Terminal 1: Launch simulation
2. Terminal 2: Start LiDAR relay
3. Terminal 3: Launch SLAM
4. Terminal 4: Open RViz and configure
5. Terminal 5: Run teleop and drive robot
6. Terminal 6: Save map when complete

Navigation Workflow:

1. Close all mapping terminals
2. Terminal 1: Launch simulation
3. Terminal 2: Start LiDAR relay
4. Terminal 3: Launch navigation with map
5. Terminal 4: Set initial pose
6. Terminal 5: Launch RViz
7. Send goals using RViz or command line

Appendix B

ROS 2 Humble
Single Robot Simulation

INFORMATION

Environment: Ubuntu 22.04, ROS 2 Humble, Gazebo Classic
Robot: Neobotix MP-400
Purpose: Single robot simulation for legacy/compatibility

B.1 Ubuntu 22.04 Installation

Prerequisites: Same as Appendix A.1
Installation Steps:

1. Download Ubuntu 22.04 LTS Desktop ISO:
   https://ubuntu.com/download/desktop
2. Installation steps identical to Appendix A.1
3. Recommended username: super

TERMINAL Post-Installation Setup

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install essential tools
sudo apt install -y git curl wget build-essential

# Reboot system
sudo reboot
```

B.2 ROS 2 Humble Installation

TERMINAL Step 1: Set Locale

```
locale  # check for UTF-8
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale  # verify settings
```

TERMINAL Step 2: Add ROS 2 Repository

```
# Enable Universe repository
sudo apt install software-properties-common
sudo add-apt-repository universe

# Add ROS 2 GPG key
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

# Add repository to sources
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

>_ TERMINAL Step 3: Install ROS 2 Humble

```
# Update package index
sudo apt update

# Upgrade system packages
sudo apt upgrade -y

# Install ROS 2 Humble Desktop (Full)
sudo apt install -y ros-humble-desktop

# Install development tools
sudo apt install -y ros-dev-tools
```

>_ TERMINAL Step 4: Environment Setup

```
# Source ROS 2 environment
source /opt/ros/humble/setup.bash

# Add to .bashrc for automatic sourcing
echo "source /opt/ros/humble/setup.bash" >>
    ~/.bashrc
source ~/.bashrc
```

>_ TERMINAL Step 5: Verify Installation

```
# Check ROS 2 version
ros2 --version     # Expected: ros2 version
    humble

# Test communication (Terminal 1)
ros2 run demo_nodes_cpp talker

# Test communication (Terminal 2)
ros2 run demo_nodes_py listener
```

B.3 Gazebo Classic Installation

>_ TERMINAL Step 1: Install Gazebo Classic

```
# Install Gazebo Classic (Gazebo 11)
sudo apt install -y ros-humble-gazebo-*

# Install additional Gazebo packages
sudo apt install -y gazebo

# Install simulation tools
sudo apt install -y ros-humble-gazebo-ros-
    pkgs
```

>_ TERMINAL Step 2: Verify Gazebo Installation

```
# Test Gazebo Classic
gazebo

# Expected: Gazebo window opens with default
    world
# Close with Ctrl+C
```

IMPORTANT

Note: Follow identical structure to Appendix A, with adjustments for ROS 2 Humble and Gazebo Classic. Complete instructions are available in the project repository.

Appendix C

ROS 2 Humble Multi-Robot Simulation

INFORMATION

Environment: Ubuntu 22.04, ROS 2 Humble
Robots: 4× Neobotix MP-400 with namespacing
Features: Namespaced topics, separate RViz instances, coordinated navigation

C.1 Prerequisites

IMPORTANT

Before starting:

- Ubuntu 22.04 installed and updated
- ROS 2 Humble installed and verified
- Gazebo Classic installed
- Single robot simulation working
- Minimum 16GB RAM recommended for 4 robots

C.2 Multi-Robot Workspace Setup

TERMINAL Step 1: Create Multi-Robot Workspace

```
# Navigate to home directory
cd ~

# Create workspace directories
mkdir -p ~/neo_multi_ws/src
cd ~/neo_multi_ws/src
```

TERMINAL Step 2: Clone Required Packages

```
# Clone Neobotix simulation packages (humble branch)
git clone --branch humble https://github.com/neobotix/neo_simulation2.git

# Clone dependencies
git clone --branch humble https://github.com/neobotix/neo_nav2_bringup.git
git clone --branch humble https://github.com/neobotix/neo_common2.git
```

>_ TERMINAL Step 3: Create Neobotix Base Workspace

```
# Create base workspace
cd ~
mkdir -p ~/neobotix_workspace/src
cd ~/neobotix_workspace/src

# Clone robot description packages
git clone --branch humble https://github.com/neobotix/neo_mp_400-2.git
git clone --branch humble https://github.com/neobotix/neo_mpo_700-2.git
```

>_ TERMINAL Step 4: Install Dependencies

```
# Install rosdep if not installed
sudo apt install -y python3-rosdep

# Initialize rosdep (only once per system)
sudo rosdep init
rosdep update

# Install dependencies for neobotix_workspace
cd ~/neobotix_workspace
rosdep install --from-paths src --ignore-src -r -y

# Install dependencies for neo_multi_ws
cd ~/neo_multi_ws
rosdep install --from-paths src --ignore-src -r -y
```

>_ TERMINAL Step 5: Build Workspaces

```
# Build neobotix_workspace first
cd ~/neobotix_workspace
rm -rf build install log
source /opt/ros/humble/setup.bash
colcon build --symlink-install

# Build neo_multi_ws
cd ~/neo_multi_ws
rm -rf build install log
source /opt/ros/humble/setup.bash
source ~/neobotix_workspace/install/setup.bash
colcon build --symlink-install
```

>_ TERMINAL Step 6: Setup Environment Variables

```
# Add multi-robot alias to .bashrc
echo "# Multi-robot workspace" >> ~/.bashrc
echo "alias multisource='source /opt/ros/humble/setup.bash && source ~/neobotix_workspace/install/setup.bash && source ~/neo_multi_ws/install/setup.bash'" >> ~/.bashrc

# Source the updated .bashrc
source ~/.bashrc
```

C.3 Environment Configuration

>_ TERMINAL Setup Gazebo Environment

```
# Source Gazebo setup
source /usr/share/gazebo/setup.bash

# Set critical environment variables
export ROS_DOMAIN_ID=0
export ROS2CLI_DISABLE_DAEMON=1
export FASTDDS_SHM_TRANSPORT=0
export GAZEBO_MODEL_DATABASE_URI=""

# These prevent common multi-robot communication issues
```

C.4 Create Cleanup Script

TERMINAL Cleanup Script for ROS/Gazebo Processes

```
# Create cleanup script
cat > ~/clean_ros.sh << 'EOF'
#!/bin/bash

echo "Killing all ROS and Gazebo processes..."

# Kill Gazebo processes
pkill -f gzserver
pkill -f gzclient

# Kill RViz processes
pkill -f rviz2

# Kill Nav2 processes
pkill -f nav2_

# Kill lifecycle managers
pkill -f lifecycle_manager

# Kill robot publishers
pkill -f robot_state_publisher

# Kill spawn entities
pkill -f spawn_entity

# Kill multi-robot safety nodes
pkill -f multi_robot_awareness
pkill -f multi_robot_yield_manager

# Kill ROS 2 daemon
pkill -f ros2

sleep 2
echo "All ROS and Gazebo processes killed successfully!"
EOF

# Make script executable
chmod +x ~/clean_ros.sh

# Run cleanup when needed
~/clean_ros.sh
```

C.5 Launch Multi-Robot Simulation (4 Robots)

TERMINAL Terminal 1: Start Gazebo with 4 Robots

```
cd ~/neo_multi_ws

# Source all required setups
source /opt/ros/humble/setup.bash
source ~/neobotix_workspace/install/setup.bash
source ~/neo_multi_ws/install/setup.bash
source /usr/share/gazebo/setup.bash

# Set environment variables
export ROS_DOMAIN_ID=0
export ROS2CLI_DISABLE_DAEMON=1
export FASTDDS_SHM_TRANSPORT=0
export Number_of_Robots=4
export MY_ROBOT=mp_400
export MAP_NAME=neo_workshop
export GAZEBO_MODEL_DATABASE_URI=""

# Launch multi-robot simulation
ros2 launch neo_simulation2 multi_robot_simulation.launch.py
```

IMPORTANT

Expected Output:

- Gazebo window opens with neo_workshop world
- 4 MP-400 robots spawned at different positions
- Each robot has unique namespace (robot0, robot1, etc.)
- Console shows ”Simulation ready for 4 robots”

TERMINAL Verify Namespaced Topics

```
# Open new terminal
ros2 topic list

# Expected namespaced topics:
# /robot0/cmd_vel
# /robot0/odom
# /robot0/scan
# /robot1/cmd_vel
# /robot1/odom
# /robot2/...
# /robot3/...
# /tf (global transform tree)
# /clock (simulation time)
```

TERMINAL Terminal 2: Launch Multi-Robot Navigation

```
# Wait for Gazebo to be fully running (30 seconds)
# Then navigate to workspace
cd ~/neo_multi_ws

# Source setups
source /opt/ros/humble/setup.bash
source ~/neobotix_workspace/install/setup.bash
source ~/neo_multi_ws/install/setup.bash

# Set environment variables
export ROS_DOMAIN_ID=0
export ROS2CLI_DISABLE_DAEMON=1
export FASTDDS_SHM_TRANSPORT=0
export Number_of_Robots=4

# Launch multi-robot navigation
ros2 launch neo_simulation2 multi_robot_navigation_launch.py
```

IMPORTANT

Expected Navigation Output:

- robot0/lifecycle_manager_localization: Managed nodes are active
- robot0/lifecycle_manager_navigation: Managed nodes are active
- Same messages for robot1, robot2, robot3
- All navigation nodes in active state [3]

C.6 Launch RViz for Each Robot

TERMINAL Terminal 3: RViz for robot0

```
cd ~/neo_multi_ws
source /opt/ros/humble/setup.bash
source install/setup.bash

ros2 launch neo_nav2_bringup rviz_launch.py use_namespace:=True namespace:="robot0"
```

TERMINAL Terminal 4: RViz for robot1

```
cd ~/neo_multi_ws
source /opt/ros/humble/setup.bash
source install/setup.bash

ros2 launch neo_nav2_bringup rviz_launch.py use_namespace:=True namespace:="robot1"
```

TERMINAL Terminal 5: RViz for robot2

```
cd ~/neo_multi_ws
source /opt/ros/humble/setup.bash
source install/setup.bash

ros2 launch neo_nav2_bringup rviz_launch.py use_namespace:=True namespace:="robot2"
```

TERMINAL Terminal 6: RViz for robot3

```
cd ~/neo_multi_ws
source /opt/ros/humble/setup.bash
source install/setup.bash

ros2 launch neo_nav2_bringup rviz_launch.py use_namespace:=True namespace:="robot3"
```

IMPORTANT

RViz Windows Configuration:

- Each RViz window shows only its robot's data
- Fixed Frame: map (common for all)
- Map display shows shared map
- LaserScan shows robot's own scans
- RobotModel shows individual robot
- Path shows robot's planned path

C.7 Sending Navigation Goals to Multiple Robots

TERMINAL Method 1: Using RViz (Recommended)

```
# For each robot's RViz window:
# 1. Click '2D Goal Pose' button in toolbar
# 2. Click and drag on map to set position and orientation
# 3. Robot plans path (blue line) and starts moving
# 4. Repeat for other robots while first robot is moving

# Robots navigate simultaneously without collisions
```

TERMINAL Method 2: Using Command Line

```
# Send goal to robot0
ros2 topic pub --once /robot0/goal_pose geometry_msgs/msg/PoseStamped "{
header: {frame_id: 'map'},
pose: {
  position: {x: 2.0, y: 1.0, z: 0.0},
  orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}
}}"

# Send goal to robot1
ros2 topic pub --once /robot1/goal_pose geometry_msgs/msg/PoseStamped "{
header: {frame_id: 'map'},
pose: {
  position: {x: -2.0, y: -1.0, z: 0.0},
  orientation: {x: 0.0, y: 0.0, z: 0.707, w: 0.707}
}}"

# Send goals to robot2 and robot3 similarly
```

TERMINAL Monitor Multi-Robot Navigation

```
# Monitor all robots' velocity commands
ros2 topic echo /robot0/cmd_vel
ros2 topic echo /robot1/cmd_vel
ros2 topic echo /robot2/cmd_vel
ros2 topic echo /robot3/cmd_vel

# Monitor planned paths
ros2 topic echo /robot0/plan
ros2 topic echo /robot1/plan

# Monitor localization
ros2 topic echo /robot0/amcl_pose
ros2 topic echo /robot1/amcl_pose
```

Appendix D

Real Robot Deployment (ROS 2 Jazzy)

INFORMATION

Hardware: Neobotix MP-400 physical robot
Software: ROS 2 Jazzy, Ubuntu 24.04
Network: Ethernet/WiFi with static IPs
Purpose: Deploy autonomous navigation on physical robot

D.1 Network Configuration

IMPORTANT

Network Prerequisites:

- Robot and laptop on same network (recommended: 10.7.23.0/24)
- Robot IP: 10.7.23.47 (static)
- Laptop IP: 10.7.23.83 (static)
- Switch/router connecting both devices
- No firewall blocking ROS 2 ports (UDP 7410-7420, 7500-7509)

TERMINAL Step 1: Configure Network on Both Devices

```
# On Robot AND Laptop, add to ~/.bashrc:
echo "export ROS_DOMAIN_ID=0" >> ~/.bashrc
echo "export ROS_AUTOMATIC_DISCOVERY_RANGE=SUBNET" >> ~/.bashrc
echo "export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp" >> ~/.bashrc

# For real-time performance (optional)
echo "export RCUTILS_CONSOLE_OUTPUT_FORMAT='[{severity} {time}] [{name}]: {message}'" >> ~/.bashrc

# Source the updated .bashrc
source ~/.bashrc
```

TERMINAL Step 2: Test Network Connection

```
# From laptop, ping robot
ping 10.7.23.47

# Expected: Success replies, <1ms latency
# If fails: check cables, IPs, firewall

# From robot, ping laptop (via VNC/SSH)
ping 10.7.23.83

# Test ROS 2 discovery
ros2 node list   # Should show nodes from both devices
```

>_ TERMINAL Step 3: Access Robot via VNC

```
# On laptop, install VNC Viewer
sudo apt install -y tigervnc-viewer

# Connect to robot (default VNC port: 5900)
vncviewer 10.7.23.47:5900

# Credentials (default):
# Username: neobotix
# Password: Provided by manufacturer
```

D.2 Robot Hardware Bringup

>_ TERMINAL Terminal 1 (On Robot - via VNC)

```
# Navigate to workspace
cd ~/jazzy_ws
source install/setup.bash

# Launch robot hardware drivers
ros2 launch neo_mp_400-2 bringup.launch.py

# Expected output:
# [motor_controller] Connected to motors
# [lidar_driver] LiDAR initialized
# [imu_driver] IMU calibrated
# [robot_state_publisher] Started
# All hardware nodes active
```

>_ TERMINAL Verify Robot Topics (On Laptop)

```
# List topics from laptop (should see robot topics)
ros2 topic list

# Expected real robot topics:
# /cmd_vel
# /odom
# /lidar_1/scan_filtered
# /imu/data
# /tf
# /tf_static
# /battery_state
# /motor_status

# Verify data quality
ros2 topic echo /lidar_1/scan_filtered --once
ros2 topic echo /odom --once
```

D.3 Creating Map on Real Robot

IMPORTANT

Safety First:

- Clear area of obstacles and people
- Keep emergency stop accessible
- Start mapping from known position
- Move robot SLOWLY (0.2 m/s maximum)

>_ TERMINAL Terminal 1 (On Laptop): LiDAR Relay

```
# Relay robot's LiDAR topic to standard /scan
ros2 run topic_tools relay /lidar_1/scan_filtered /scan

# Verify relay
ros2 topic hz /scan   # Should show 10-15 Hz
```

>_ TERMINAL Terminal 2 (On Laptop): Launch SLAM

```
# Launch SLAM for real robot (NO simulation time)
ros2 launch slam_toolbox online_async_launch.py use_sim_time:=false

# Adjust SLAM parameters for real robot
# Edit ~/jazzy_ws/src/slam_toolbox/config/mapper_params_online_async.yaml:
#   mapping_linear_update: 0.5
#   mapping_angular_update: 0.25
#   resolution: 0.05
```

>_ TERMINAL Terminal 3 (On Laptop): Launch RViz

```
# Launch RViz for real robot mapping
rviz2

# Configuration:
# 1. Fixed Frame: map
# 2. Add Map: /map
# 3. Add LaserScan: /scan
# 4. Add RobotModel
# 5. Add TF (optional)
# 6. Add PointCloud2: /scan (for raw data)
```

>_ TERMINAL Terminal 4 (On Laptop): Teleoperation

```
# Install teleop if not available
sudo apt install -y ros-jazzy-teleop-twist-keyboard

# Run teleoperation
ros2 run teleop_twist_keyboard teleop_twist_keyboard

# Reduce speed significantly for mapping
# Press 'z' 10 times for 0.1 m/s linear speed
```

IMPORTANT

Mapping Procedure:

1. Start at a distinctive location (corner, wall intersection)
2. Drive robot in clockwise square pattern
3. Cover all navigable areas
4. Return to starting position
5. Verify map closure in RViz
6. Check for map consistency (no double walls)

>_ TERMINAL Save Map (On Laptop)

```
# Create maps directory
mkdir -p ~/jazzy_ws/maps

# Save map with timestamp
ros2 run nav2_map_server map_saver_cli -f ~/jazzy_ws/maps/real_warehouse_map

# Verify map files
ls -la ~/jazzy_ws/maps/real_warehouse_map.*

# Check map quality
xdg-open ~/jazzy_ws/maps/real_warehouse_map.pgm
```

>_ TERMINAL Terminal 1 (On Robot): Hardware Bringup

```
# If not already running
cd ~/jazzy_ws
source install/setup.bash

# Launch hardware drivers
ros2 launch neo_mp_400-2 bringup.launch.py

# Wait for all hardware to initialize
# Check motor controller: green LED
# Check LiDAR: spinning
```

>_ TERMINAL Terminal 1 (On Laptop): LiDAR Relay

```
# Essential: Relay LiDAR data
ros2 run topic_tools relay /lidar_1/scan_filtered /scan

# Verify data flow
ros2 topic hz /scan    # Should be consistent
```

>_ TERMINAL Terminal 2 (On Laptop): Launch Navigation

```
cd ~/jazzy_ws
source install/setup.bash

# Launch Nav2 with real robot map (NO simulation time)
ros2 launch nav2_bringup bringup_launch.py \
  use_sim_time:=false \
  map:=/home/super/jazzy_ws/maps/real_warehouse_map.yaml \
  params_file:=/home/super/jazzy_ws/src/navigation2/nav2_bringup/params/nav2_params.yaml

# Adjust parameters for real robot:
# controller_frequency: 10.0
# planner_frequency: 0.5
# inflation_radius: 0.3
```

D.4 Navigation on Real Robot

WARNING

CRITICAL: Close ALL terminals from mapping phase before starting navigation

>_ TERMINAL Terminal 3 (On Laptop): Set Initial Pose

```
# CRITICAL: Position robot EXACTLY where mapping started
# Use distinctive features for alignment

# Set initial pose (adjust x,y based on your map)
ros2 topic pub --once /initialpose geometry_msgs/msg/PoseWithCovarianceStamped "{
header: {frame_id: 'map'},
pose: {
  pose: {
    position: {x: 0.0, y: 0.0, z: 0.0},
    orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}
  },
  covariance: [0.25, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.25, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
               0.0, 0.0, 0.0, 0.0, 0.0, 0.06853891909122467]
}}"
```

>_ TERMINAL Verify Localization

```
# Monitor AMCL pose (should be stable)
ros2 topic echo /amcl_pose

# Check particle cloud (should be clustered)
ros2 topic echo /particle_cloud --once | head -20

# Verify transform from map to base_footprint
ros2 run tf2_ros tf2_echo map base_footprint
```

>_ TERMINAL Terminal 4 (On Laptop): Launch RViz

```
# Launch RViz for real robot navigation
rviz2 --ros-args -p use_sim_time:=false

# Configuration for real robot:
# 1. Fixed Frame: 'map'
# 2. Add Map: /map
# 3. Add LaserScan: /scan
# 4. Add RobotModel
# 5. Add Path: /plan (color: blue)
# 6. Add PoseArray: /particle_cloud (color: red, alpha: 0.1)
# 7. Add Costmap: /global_costmap/costmap (for debugging)
```

>_ TERMINAL Sending Navigation Goals

```
# Method 1: RViz (Recommended)
# - Click '2D Goal Pose' button
# - Select position and orientation
# - Robot plans and executes

# Method 2: Command line (for testing)
ros2 topic pub --once /goal_pose geometry_msgs/msg/PoseStamped "{
header: {frame_id: 'map'},
pose: {
  position: {x: 3.0, y: 2.0, z: 0.0},
  orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}
}}"
```

>_ TERMINAL Monitor Real Robot Navigation

```
# Monitor velocity commands
ros2 topic echo /cmd_vel

# Monitor battery state
ros2 topic echo /battery_state

# Monitor motor status
ros2 topic echo /motor_status

# Monitor CPU/memory (on robot via VNC)
htop
```

IMPORTANT

Safety Monitoring:

- Always maintain visual contact with robot
- Keep emergency stop accessible
- Monitor battery level (>20%)
- Check for LiDAR obstructions
- Verify robot responds to stop commands

Appendix E

Custom Nodes Development

INFORMATION

Purpose: Develop and integrate custom ROS 2 nodes for warehouse automation
Nodes: Task manager, safety nodes, fleet coordination

E.1 Warehouse Task Manager Node

IMPORTANT

Functionality:

- Receives pick commands with shelf IDs
- Coordinates navigation to pick locations
- Manages drop zone deliveries
- Returns to idle position
- Integrates with Nav2 action server

TERMINAL Package Structure

```
warehouse_task_manager/
|-- CMakeLists.txt
|-- package.xml
|-- config/
|   `-- waypoints.yaml
|-- include/
|   `-- warehouse_task_manager/
|       `-- task_manager.hpp
|-- src/
|   |-- simple_pick_drop_manager.cpp
|   `-- task_manager_node.cpp
`-- launch/
    `-- task_manager.launch.py
```

E.2 Multi-Robot Safety Nodes

IMPORTANT

Safety Node Features:

- Multi-robot awareness and collision prevention
- Fleet speed limiting for safe operation
- Deadlock detection and resolution
- Emergency stop coordination
- Works with namespaced robots

>_ TERMINAL Safety Nodes Package Structure

```
neo_fleet_safety/
|-- CMakeLists.txt
|-- package.xml
|-- config/
|   |-- safety_params.yaml
|   `-- speed_limits.yaml
|-- include/
|   `-- neo_fleet_safety/
|       |-- multi_robot_awareness.hpp
|       `-- fleet_speed_limit.hpp
|-- src/
|   |-- multi_robot_awareness.cpp
|   `-- fleet_speed_limit.cpp
`-- launch/
    `-- fleet_safety.launch.py
```

Conclusion

INFORMATION

This comprehensive guide provides everything needed to implement, test, and deploy the WareBot multi-robot warehouse navigation system from scratch to production.

System Achievements

Feature	Implementation Status
Single Robot Simulation	Complete (ROS 2 Jazzy & Humble)
Multi-Robot Coordination	4+ robots with namespacing
Custom World Environments	6 warehouse layouts with maps
Safety System	Collision avoidance + speed limiting
Real Robot Deployment	Neobotix MP-400 integration
Custom Nodes	Task manager, safety nodes
Complete Documentation	Step-by-step instructions
Production Readiness	Tested in simulation and real world

Final Notes

The WareBot project demonstrates a practical, scalable solution for autonomous warehouse operations using modern robotics technology and the open-source ROS 2 framework. This documentation represents thousands of hours of development, testing, and refinement to create a production-ready system.

Remember: Always prioritize safety when working with physical robots. Test thoroughly in simulation before real-world deployment.

AI APPENDIX

نموذج إقرار وتوثيق استخدام أدوات الذكاء الاصطناعي في مشروع التخرج

اسم الطالب / أسماء الطلبة: Issa Warasna / Osama Hroub/ Ismail Rjoub	عنوان المشروع: WareBot: A Flexible Autonomous Robot for Smart Warehouse Management in Small and Medium-Sized Enterprises
الرقم / الأرقام الجامعية: 211074/201174/201171	اسم المشرف الأكاديمي: د. هاني صلاح
الكلية / الدائرة: كلية تكنولوجيا المعلومات وهندسة الحاسوب	السنة/الفصل الدراسي: 2026 / الفصل الأول

الغرض من الاستخدام	اسم الأداة	أماكن الاستخدام في التقرير (فصول / صفحات)	الأمر الأساسي المستخدم (Prompt)	هل تم تعديل الناتج؟	نسبة الاستخدام التقديرية	ملاحظات
توليد نصوص	☑ ChatGPT	في الشابتر الثاني	اجد الفجوة بين مشروعنا والمشاريع السابقة	نعم	%18	استخدمنا chatgpt للdeep search وايجاد الفجوات بين المشاريع الموجودة مسبقة ومشروعنا
	☐ Gemini				%0	لم يستخدم
	☐ other: ____		—		%0	لم يستخدم
تدقيق لغوي	☑ Grammarly	في كل الشباتر	تأكد من صحة الكلام والقواعد		%20	تدقيق النصوص والتأكد من الكلام وصحة الكتابة
	☑ Quillbot	في كل الشباتر			%20	استعملناه للتأكد من نسبة ال AI
	☐ Word Editor AI				%0	لم يستخدم
	☑ ChatGPT	الشابتر 5 والشاتبر 6	هل هنالك أي أخطاء املائية في النص وماهي افضل طريقة لتصليحها؟	نعم	%16	كان يحدد المشاكل ويعطني نصيحة للتعديل
تلخيص محتوى	ChatGPT	ا		لا	%0	لم يستخدم

	☐ SMMRY				%0	لم يستخدم
	☐ Notion AI				%0	لم يستخدم
	☐ Scholarcy				%0	لم يستخدم
	☐ other: ____				%0	لم يستخدم
تحليل بيانات	☐ Excel Copilot				%0	لم يستخدم لأنه لا يوجد بيانات في مشروعنا.
	☐ Python AI Assistants				%0	لم يستخدم لأنه لا يوجد بيانات في مشروعنا.
	☐ Tableau AI				%0	لم يستخدم لأنه لا يوجد بيانات في مشروعنا.
	☐ other: ____				%0	لم يستخدم لأنه لا يوجد بيانات في مشروعنا.
رسم مخططات / أشكال	☐ DALL·E				%0	لم يستخدم
	☐ ChatGPT				%0	لم يستخدم
	☐ Canva AI				%0	لم يستخدم
	☐ Visio AI				%0	لم يستخدم
	☐ other: ____				%0	لم يستخدم

كتابة أكواد برمجية	☑ Microsoft Copilot ☑ Claude	الشابتر الثالث والرابع	اعطني كود لتشغيل ال linear actuators أيضا في الفرونت اند والباك اند للموقع		%20	تم استخدامه لإنشاء الاكواد والتأكد من صحتها
	☐ Replit AI				%0	لم يستخدم
	☐ Codeium				%0	لم يستخدم
	☐ other: ____				%0	لم يستخدم
توثيق مراجع	☐ EndNote				%0	لم يستخدم
	☐ Mendeley				%0	لم يستخدم
	☐ Zotero				%0	لم يستخدم
	☐ other: ____				%0	لم يستخدم
أخرى (حدد **):** ____					%0	لم يستخدم

إقرار فريق مشروع التخرج:

نقر بأن استخدام أدوات الذكاء الاصطناعي تم بشكل مسؤول وبما يتوافق مع السياسات الجامعية، وقد راجعنا وحررنا كافة المخرجات بما يعكس فهمنا الشخصي.

توقيع الطالب:..... التاريخ:.. .....

توقيع الطالب:..... **...**.....التاريخ:.....

توقيع الطالب:..... .....التاريخ:.....