FINAL YEAR DESIGN PROJECT REPORT

AUTONOMOUS SLAM-BASED NAVIGATION BOT WITH OBJECT AVOIDANCE, HUMAN INTERACTION, AND REAL-TIME USER INTEGRATION

Supervised by
Commodore Dr Attaullah Memon

PAKISTAN NAVY ENGINEERING COLLEGE
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
PAKISTAN

(Spring 2025)

Supervisor:	**Signature**
Commodore Dr Attaullah Memon	_______________
Team Members	
Khadeeja Khan	_______________
Narmeen Sabah Siddiqui	_______________
Hassaan Muhammad Khan	_______________
Abdul Rafey Beig	_______________

Supervisor:	**Signature**
Commodore Dr Attaullah Memon	_______________

Checked By:	**Signature**
Name: _______________	_______________
Name: _______________	_______________
Name: _______________	_______________
Name: _______________	_______________

Plagiarism %(report first page should be attached); ---------

Date:

____

ACKNOWLEDGMENTS

We would like to convey our sincere gratitude to all those who have enabled us to successfully complete this project. We express special thanks to Commodore Dr. Attaullah Y. Memon, our final year project supervisor, whose support, encouragement, and valuable guidance played a pivotal role in the successful completion of this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The introduction of Industry 4.0 has resulted in an increase in demand for automation due to the inefficiency, high cost, and risk of human mistake connected with manual labour. Indoor areas such as offices, schools, and hospitals are seeing an increase in demand for intelligent, flexible robotic devices that can operate autonomously as well as interact with humans. Most present solutions have a limited reach, rely on predefined pathways, or lack significant user participation.

This research proposes a general-purpose mobile assistant robot that can navigate indoor areas autonomously utilising LiDAR-based SLAM (Simultaneous Localisation and Mapping) and communicate seamlessly with voice commands. A display screen uses facial expressions to replicate emotional responses, improving the user experience. ROS2 is used for modular system integration, while onboard AI provides real-time sensing and decision-making.

The robot integrates sensors, microcontrollers, and actuators to perform obstacle avoidance, path planning, and natural language understanding. Testing demonstrates strong performance in navigation accuracy, command responsiveness, and interaction quality. Compared to conventional robots, our system offers greater flexibility, scalability, and user engagement.

In conclusion, this project introduces a smart and adaptable robotic platform designed to assist people in a variety of environments. Looking ahead, we aim to make it even more energy-efficient, enhance its interactive abilities, and equip it with advanced sensors to better understand and respond to its surroundings. With these improvements, the robot could become a valuable addition to smart campuses, healthcare facilities, workplaces, and customer service setting

Chapter 1

INTRODUCTION

1.1 GENERAL

Industry 4.0's emergence has hastened the rapid integration of intelligent automation into many different sectors. The limitations of manual labor—such as inefficiency, high operating costs, and the potential for human error—are driving this shift. To overcome these challenges, industries are increasingly seeking autonomous systems that can operate with minimal supervision, adapt to dynamic environments, and engage meaningfully with people *(Luo & Jayaraman, 2021)*.

Among these advancements, *Autonomous Mobile Robots (AMRs)* have emerged as a highly effective solution by combining navigation, perception, and interaction capabilities to manage a wide range of indoor tasks. No longer confined to predefined paths or industrial settings, these robots are now commonly deployed in public spaces, offices, hospitals, and educational institutions.

The integration of LiDAR-based SLAM (Simultaneous Localization and Mapping), voice interfaces, and emotion display systems is making these robots more user-friendly and suitable for everyday use. This project aims to develop a general-purpose, intelligent mobile assistant robot capable of autonomous navigation, understanding voice commands, and engaging users through a screen-based facial interface. By combining multiple technologies within a single platform, the robot offers a scalable and adaptable solution for smart indoor environments *(Chen et al., 2022).*

1.2 APPLICATIONS

Robots capable of autonomous navigation and interaction have real-world value in a wide variety of sectors. Below are key applications of such systems, supported by existing solutions and the limitations our project aims to overcome.

1.2.1 Educational Institutions and Smart Campuses

In academic settings, robots are being utilized to enhance administrative operations, guide students across large campuses, and provide interactive learning experiences. For example, Sanbot has been deployed in several educational environments, where it assists in classroom presentations, greets visitors, and answers student queries. Its humanoid design and basic conversational abilities create a more engaging learning environment.

Figure 1. SanBot by Qihan Technology

1.2.2 Corporate Offices and Workspaces

In modern office environments, automation is being used to streamline workflows, enhance visitor experience, and reduce the burden of repetitive tasks on staff. Because of its user-friendly design and tray-based distribution mechanism, BellaBot**,** which was first created for the restaurant business, has proven useful in office environments. It employs animated facial expressions to communicate with users and navigates on its own to deliver documents, drinks, or parcels. It is a sensible option for hospitality-style service in offices due to its dependability in organised environments.

Figure 2. BellaBot by PUDU Robotics

1.2.3 Hospitals and Healthcare Facilities

Robotic technologies have been implemented by healthcare organisations to speed up logistics and minimise human interaction in sensitive regions. The TUG robot is a well-known example; it moves lab samples, drugs, and linens throughout hospitals on its own. By lowering the number of employees required to complete these delivery duties, TUG enhances workflow effectiveness and aids in infection control initiatives.

Figure 3. TUG robot from Aethon

1.2.4 Retail, Hospitality, and Customer Engagement

Service robots are being used more and more in retail stores, malls, and hospitality settings to help patrons, promote goods, and develop engaging brand experiences. Pepper, is well known for its capacity to carry on simple dialogues, identify emotions, and present advertising information. It works well for drawing attention and assisting in customer service positions because of its humanoid appearance and voice-based interaction capabilities.

Figure 4. Pepper by SoftBank Robotics

1.3 AIM

The aim is to explore the growing role of autonomous mobile robots across indoor environments by analyzing their current applications, design characteristics, and limitations. Through this, the objective is to understand how such robots are transforming service delivery, logistics, and human-robot interaction in sectors like education, healthcare, retail, and hospitality.

1.4 SCOPE

The scope of our project is to

- Investigate the use of autonomous mobile assistant robots specifically within indoor environments.
- Focus on practical applications across sectors such as education, healthcare, corporate offices, and retail.
- Explore key capabilities including autonomous navigation, obstacle avoidance, task automation, and user interaction.
- Examine the role of underlying technologies like LiDAR, sensors, SLAM algorithms, and interactive display systems.
- Evaluate the impact of these robots on operational efficiency, service delivery, and user engagement.

Chapter 2

LITERATURE REVIEW

2.1 OVERVIEW

To achieve true unmanned navigation, we must tackle the challenging problems of localization, mapping, and navigation in a previously unknown environment.

- *Localization* is the process of determining the robot's position and orientation in relation to a known reference frame, such as a map or global coordinates. This is done using sensor data, such as visual or inertial data, and algorithms to estimate the robot's position.
- *Mapping* to accurately map an unknown environment, a mobile robot needs to know its position and orientation to perceive obstacles (dynamic or static) and available space for movement.
- *Navigation* The problem of navigation involves global planning to determine the optimal path between two points on the map and local planning to avoid obstacles, whether static or dynamic, that are unknown on the map.

In real-world applications, the environments in which robots operate are often unpredictable and unknown. Successful robot navigation requires both accurate localization and mapping of the environment. However, these two problems are interdependent, as localization requires a map of the environment to estimate a robot's pose, while mapping requires the robot's pose to build an accurate map. This 'chicken and egg' problem can be solved by simultaneously estimating the robot's pose and mapping the environment, all in real-time, known as Simultaneous Localization and Mapping (SLAM). After a map has been constructed using Simultaneous Localization and Mapping (SLAM) techniques, it is integrated into the navigation algorithms to plan the most efficient path for the robot. These algorithms generate velocity commands, which are subsequently relayed to the control module to cause the desired motion of the robot.

2.2 SIMULTANEOUS LOCALISATION AND MAPPING (SLAM)

A robot navigating an uncertain environment must pass across unknown limits while avoiding obstacles. To achieve this, the robot mostly depends on learning about its surroundings and basing decisions on that knowledge. But both the way environmental features are gathered and the way controls are carried out are prone to different kinds of distortions—linear and nonlinear in character. As observed by *Johansson et al. (2012),* improper modeling of these distortions can seriously compromise robotic performance. Simultaneous Localization and Mapping (SLAM) shows itself as the way forward in overcoming this obstacle, allowing the robot to recover an accurate map of the surroundings and the path followed notwithstanding the presence of noisy controls and observations. *GoodFellow et al. (2019)* demonstrated through FastSLAM that particle filters could maintain localization accuracy under motion and measurement uncertainty. It becomes apparent that errors in the robot's track can arise when the trajectory is uncertain, consequently leading to inaccuracies in the resulting map. As *Corke et al. (2017)* explained, joint estimation of both the robot's state and the map are critical to reducing accumulated drift and ensuring consistency in mapping results.

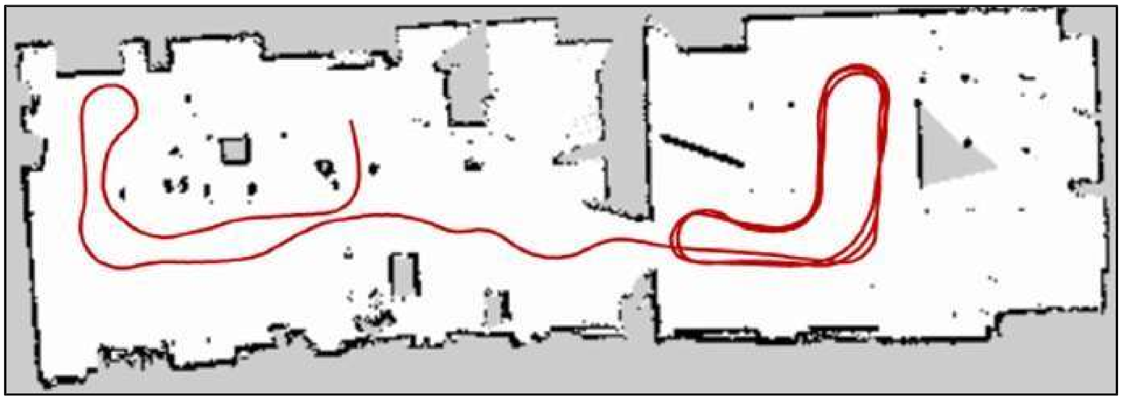


Figure 5. An Example of SLAM

2.2.1 2D SLAM

2D SLAM involves constructing a two-dimensional representation of an environment using sensor inputs, typically from laser range finders or LiDAR. It is widely used in indoor mobile robots due to its lower computational requirements and sufficient accuracy for navigation in flat, structured spaces such as offices, hospitals, and shopping malls *(Becky et al., 2005).* 2D SLAM assumes the environment can be represented on a plane and works well where vertical variations are minimal or irrelevant to navigation. Popular 2D SLAM algorithms include Gmapping, Hector SLAM, and Cartographer, which are commonly integrated with ROS-based systems *(Wang et al., 2020).*

2.2.2 LiDAR SLAM and Visual SLAM

LiDAR SLAM uses LiDAR (Light Detection and Ranging) sensors to measure distances to nearby objects by emitting laser pulses and calculating the time of flight. It provides high-precision point cloud data that enables accurate mapping, even in poorly lit or feature-sparse environments (*Zhang & Singh, 2014).* LiDAR SLAM is particularly effective in indoor environments where lighting can vary and visual features may be repetitive or absent. It also offers superior range and accuracy compared to vision-based methods (*Droeschel et al., 2018*).

Advantages of LiDAR SLAM include:

- Robustness in varying lighting conditions
- High spatial accuracy for obstacle detection and localization
- Fast and consistent performance in structured indoor environments

LiDAR sensors can be costly, though, and produce vast datasets that call for effective processing and filtering techniques. LiDAR SLAM on mobile robots is frequently implemented using ROS-compatible SLAM systems including Cartographer and Google's Ceres Solver *(Shan et al., 2020).*

Using monocular or stereo cameras, Visual SLAM (V-SLAM) tracks features across frames to estimate the posture of the robot from the surroundings. It is reasonably priced and can offer rich contextual information which LiDAR cannot offer. Popular visual SLAM systems consist in ORB-SLAM, RTAB-Map, and DSO *(Engel et al., 2018).*

V-SLAM performs well in environments with distinct visual features and good lighting. However, it is more sensitive to motion blur, low-light conditions, and repetitive patterns. While it adds useful perception layers, its accuracy and robustness can be limited compared to LiDAR in dynamic or cluttered indoor settings *(Campos et al., 2021).*

2.3 LIDAR SLAM APPROACHES

LiDAR-based SLAM has become a dominant method for indoor mobile robot navigation due to its accuracy, robustness, and independence from lighting conditions. Several algorithmic approaches have been developed to efficiently process the point cloud data and construct reliable 2D or 3D maps while localizing the robot *(Guimaraes et al., 2020).*

Among the primary algorithmic paradigms used in SLAM systems, three stand out as particularly influential as mentioned in *2D SLAM Algorithms Characterization, Calibration, and Comparison:*

- Particle Filter-Based SLAM
- Deep Learning-Based SLAM
- Graph-Based SLAM

2.3.1 Particle Filter-Based SLAM

A fundamental method extensively applied in classical SLAM systems such as GMapping is Particle Filter-based SLAM, especially the *Rao-Blackwellized Particle Filter* (RBPF) approach. This approach presents the robot's posture as a collection of particles with hypotheses of their trajectory. These particles are updated, reweighted, and resampled under sensor observations and control inputs to preserve the best possible estimate of the robot's posture as well as the surrounding environment.

Every particle keeps its own map of the surroundings; the algorithm over time selects the most likely one. This makes RBPF ideally fit for dynamic indoor environments since it lets it manage nonlinearities and non-Gaussian noise. However, the accuracy of the map and localization can degrade if the number of particles is low, especially in large spaces or environments with poor odometry *(Grisetti et al., 2017).*

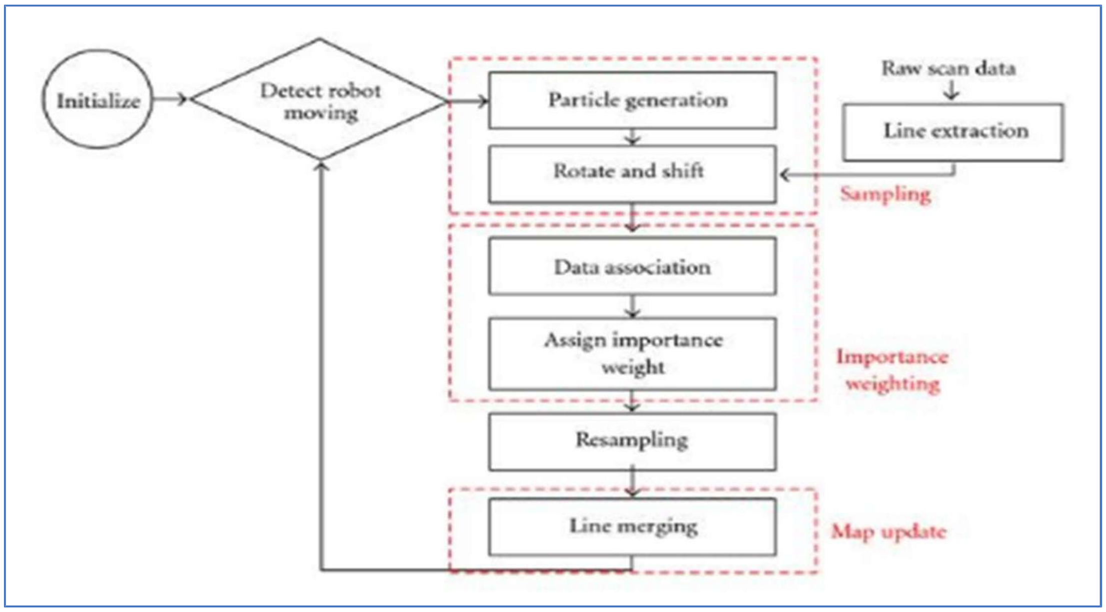


Figure 6. Overall Flowchart of a Particle Based SLAM Algorithm

2.3.2 Deep Learning-Based SLAM

In order to improve or replace essential elements like feature extraction, pose estimation, and loop closure, deep learning-based SLAM incorporates neural networks into the conventional SLAM pipeline. Deep learning approaches acquire representations straight from unprocessed sensor data, in contrast to traditional approaches that depend on manually created features and geometric models. They can more successfully generalize to unknown environments thanks to this ability, which also helps them withstand dynamic elements and sensor noise.

An important development in this field is the use of Recurrent Neural Networks (RNNs) to model temporal dependencies in motion estimation and Convolutional Neural Networks (CNNs) to extract features from visual and LiDAR data. Methods like *LO-Net, DeepMapping,* and *DiSCO*, which frequently work directly on raw LiDAR scans without the need for explicit feature engineering, have demonstrated competitive performance in localization and mapping accuracy *(Li et al., 2019).*

Notwithstanding their promise, deep learning-based SLAM systems frequently require sizable annotated datasets, a lot of processing power (such as GPUs), and may have trouble generalizing to settings that are very different from the training data. However, these models show promise for the future of SLAM, especially in hybrid frameworks that combine the stability of traditional filtering or graph-based techniques with learning-based perception *(Clark et al., 2018).*

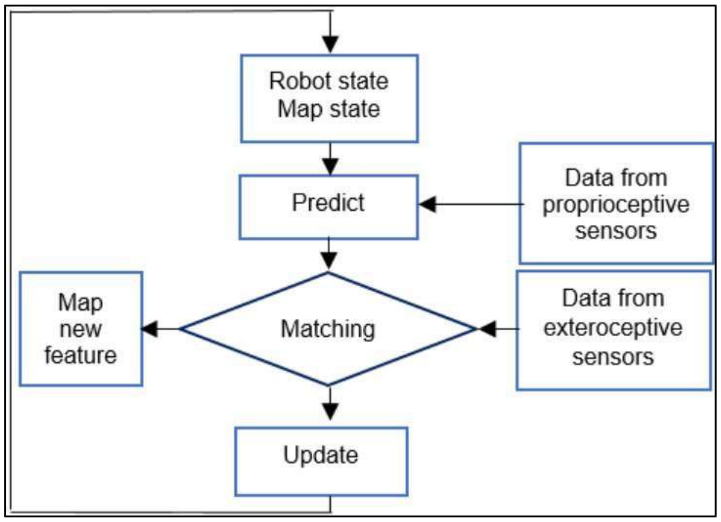

Robot state
Map state
Predict
Data from proprioceptive sensors
Map new feature
Matching
Data from exteroceptive sensors
Update

Figure 7. Overall Flowchart of a Deep Learning Based SLAM Algorithm

2.3.3 Graph-Based SLAM

Graph-Based SLAM is a modern and widely adopted approach that represents the SLAM problem as a graph optimization problem, where the robot's poses are modeled as nodes and the spatial constraints between them (obtained from odometry and sensor observations) are represented as edges.

In this method, a pose graph is incrementally built as the robot moves. Each node in the graph corresponds to a robot pose at a certain time, and edges encode relative transformations between poses based on sensor data. The key task is to optimize this graph to minimize the error across all constraints—resulting in a globally consistent trajectory and map.

Graph-based SLAM is especially effective when loop closures are detected, allowing the system to correct accumulated drift by adjusting earlier parts of the graph. Optimization techniques such as *Non-linear Least Squares* (e.g., Gauss-Newton or Levenberg-Marquardt) are commonly used to solve the graph *(Kümmerle et al., 2011; Grisetti et al., 2010).*

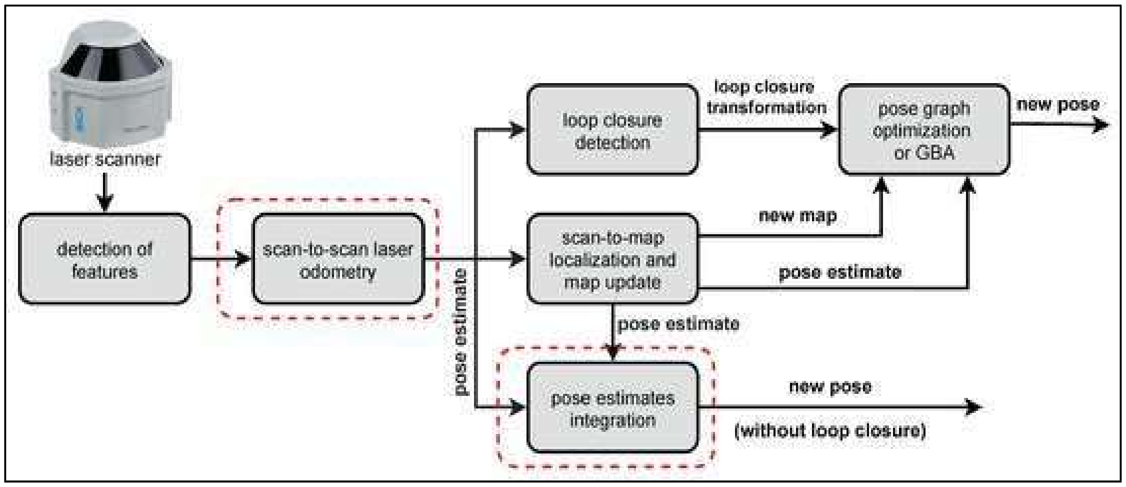

laser scanner
detection of features
scan-to-scan laser odometry
pose estimate
loop closure detection
loop closure transformation
pose graph optimization or GBA
new pose
scan-to-map localization and map update
new map
pose estimate
pose estimate
pose estimates integration
new pose
(without loop closure)

Figure 8. Overall Flowchart of a Graph Based SLAM Algorithm

2.4 NAVIGATION APPROACHES

2.4.1 Global Planner

A* Algorithm

The A* (A-star) algorithm holds great significance in the realms of robotics and artificial intelligence as a widely utilized and renowned path planning algorithm. It serves as an informed search approach that adeptly discovers the most optimal path, leading from a designated starting point to a goal node within a graph or grid based environment. A* combines the advantageous traits of both Dijkstra's Algorithm, which ensures optimality, and the greedy Best-First Search method, which facilitates heuristic-guided exploration *(Russell & Norvig, 2020).*

The maintenance of a priority queue tracking nodes to be investigated defines the main purpose of the algorithm. Using a heuristic function, every node evaluates depending on two important values: the cost of reaching the current node from the starting point (referred to as the g-value) and an estimated cost from the current node to the goal node (termed the h-value). Conventionally computed using heuristics such Euclidean distance or Manhattan distance, this heuristic function provides an approximation of the remaining distance or cost.

Throughout the process, A* identifies the node possessing the lowest f-value *($f = g + h$) (Dissanayake et al., 2021)* from the priority queue at each step, subsequently expanding its adjacent nodes. If a better path is discovered, the algorithm updates the cost and parent pointers of these neighboring nodes accordingly. This iterative process keeps going until either all reachable nodes have been thoroughly explored or the goal node is reached. The most important feature of the A* algorithm is its capacity to prioritize exploration in the most promising direction, as determined by the heuristic function that is being used. Because the heuristic function adheres with the rules of admissibility (never overestimating the true cost) and consistency, this important feature guarantees the algorithm's efficiency while ensuring the discovery of the optimal path *(Pearl, 2019).*

Dijkstra’s Algorithm

The Dijkstra's Algorithm is still a fundamental technique for figuring out a graph's shortest path. It was first created by Edsger W. Dijkstra in 1956 and works with weighted graphs that have non-negative edge weights. It starts from a given source node and iteratively chooses the node from the source that has the lowest cumulative cost. Efficient selection of the next node to explore is made possible by a priority queue. The algorithm updates the cost of reaching nearby nodes at each iteration by taking into account the connecting edge's cost as well as the current node's cumulative cost. The algorithm updates the cost and marks the current node as the predecessor in the shortest path if a newly calculated cost is less than the previously recorded cost. This process continues until all reachable nodes have been visited or the destination node is reached. Dijkstra's Algorithm guarantees the discovery of the shortest path from the source node to all other nodes in the graph, provided all edge weights are non-negative *(Haeupler et al., 2024)*.

Recent developments have concentrated on enhancing the algorithm's functionality and broadening its range of applications. For instance, compared to serial versions, parallel implementations employing MPI and CUDA have shown speedups of more than 5x and 10x, respectively *(Song, 2025)*. Improvements to bidirectional search have minimized the number of edges processed in large graphs by achieving instance-optimal exploration. By integrating real-time traffic predictions, dynamically weighted versions of Dijkstra's algorithm enhance route planning in intelligent transportation systems. Additionally, adaptive Dijkstra-based routing techniques have helped optical networks-on-chip by lowering power loss and enhancing signal quality *(Zheng et al., 2021)*.

These developments underscore the continued relevance and adaptability of Dijkstra’s algorithm across a wide range of modern computing and networking challenges.

PRM

One popular motion planning method in robotics is the Probabilistic Roadmap (PRM) algorithm, which works especially well for navigating intricate, high-dimensional environments. PRM, which was first created in the late 1990s and later expanded by scholars like Emilio Frazzoli and Sertac Karaman, provides a probabilistic method for resolving the path planning issue *(Karaman & Frazzoli, 2011)*. In contrast to deterministic planners, PRM creates a flexible graph, or "roadmap," that models the robot's environment and assists in determining practical routes between points.

The algorithm operates in two main phases: roadmap construction and query resolution. During the first phase, the robot randomly samples points (called configurations) from its environment and checks whether these points are free from obstacles. Only collision-free configurations are kept. These points are then connected to nearby ones using simple local paths that are also tested for collisions. Over time, this builds a web of safe pathways through the environment. Once enough connections have been made, the roadmap is ready.

In the second phase, when the robot is given a start and goal location, it searches this roadmap to find a valid route between them. For this, standard graph search algorithms like Dijkstra's or A* are used to find the shortest or most efficient path. One of the strengths of PRM is its probabilistic completeness—meaning that if a valid path exists, the algorithm is increasingly likely to find it as more samples are added. It also does well in capturing the overall structure of the environment, helping avoid getting stuck in local traps or suboptimal paths *(Udhan et al., 2022).*

Overall, PRM is especially helpful in large or intricate spaces, such as robotic arms operating in constrained environments or mobile robots navigating through cluttered rooms *(Elbanhawi & Simic, 2014).* Its balance between flexibility and computational efficiency makes it a go-to technique in modern robotics.

2.4.2 Local Planner

Dynamic Window Approach

A popular local path planning algorithm is the Dynamic Window Approach (DWA), particularly for wheeled mobile robots navigating through cluttered and dynamic environments. The effectiveness of DWA lies in its capacity to integrate the robot's motion dynamics, including turning radius, acceleration, and velocity limits, into the decision-making process. This enables the robot to quickly and safely respond to obstacles in real time, in addition to planning feasible movements *(Jovanović et al., 2020).*

DWA operates by first analyzing the robot's current kinematic state, including its position, orientation, and velocity. Based on this, the algorithm defines a "dynamic window"—a set of velocity commands that the robot can realistically achieve over a short time frame, considering its dynamic constraints (like maximum acceleration). Within this velocity window, the algorithm simulates a range of potential trajectories and evaluates them according to a cost function. This cost function usually combines several criteria: how close the trajectory comes to obstacles (for safety), how well it directs the robot toward its goal (goal alignment), and how smooth or efficient the path is (trajectory quality) *(Kumar et al., 2018).*

Among all the simulated trajectories, the one with the highest score i.e., the best trade-off between safety, efficiency, and goal direction is chosen. The corresponding velocity command is then applied to the robot's actuators. This reactive loop continues at each time step, allowing the robot to adapt fluidly to moving obstacles or sudden environmental changes.

All things considered, DWA provides a potent combination of safety and reactivity. It is a useful solution in many real-world autonomous navigation systems because of its capacity to produce real-time control commands that respect the robot's motion capabilities while also avoiding obstacles.

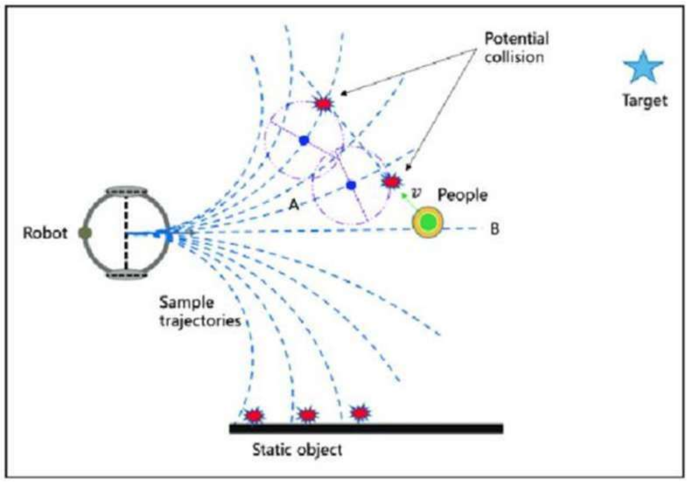


Figure 9. Dynamic Window Approach

Dynamic Window Based

The Dynamic Window Based (DWB) algorithm is a powerful local planning method often used in robotic navigation, building upon the foundation laid by the Dynamic Window Approach (DWA). DWB improves the robot's ability to plan feasible, safe, and effective trajectories, especially in dynamic environments where obstacles and conditions change rapidly (*Kumar & Singh, 2017*).

A dynamic window in the robot's velocity space, which denotes the range of feasible velocities the robot can realistically achieve given its kinematic constraints (such as speed and acceleration limits), is defined by DWB, much like DWA. In order to produce possible trajectories, the algorithm then assesses various candidate velocities within this window. The optimal trajectory for the robot to follow is determined by weighing important factors such as path smoothness, distance to the goal, and proximity to obstacles *(Lozano-Pérez & Kaelbling, 2020)*.

What distinguishes DWB from DWA is the incorporation of a costmap which is a grid-based representation of the environment that helps the robot identify areas that are safe or occupied by obstacles. By using this costmap, DWB can more effectively avoid obstacles, ensuring the robot stays within safe regions while pursuing its goal.
Another key improvement in DWB is its lookahead feature. This enhancement allows the algorithm to anticipate future states of the robot by evaluating how well a given trajectory will perform over the next few steps, rather than just focusing on immediate proximity. This predictive capability enables DWB to better handle dynamic environments and avoid potential collisions further along the robot's path *(Zhao et al., 2020*).

Smooth Nearness-Diagram

The Smooth Nearness-Diagram (SND) is a local path planning tool designed for mobile robots negotiating dynamic surroundings. It develops on the Nearness-Diagram (ND) method by adding smoothness constraints to produce more natural and continuous paths, instead of abrupt movements.

At its core, SND utilizes a Voronoi diagram to represent the free space around obstacles, ensuring that the robot travels along paths that maintain a safe distance from potential collisions. This diagram provides a skeleton of the environment, highlighting areas equidistant from the nearest obstacles. To improve the quality of motion, SND applies techniques such as B-spline interpolation or polynomial fitting to this structure. These mathematical tools help generate paths that are not only safe but also smooth and suitable for real-time execution *(Garrido et al., 2019).*

One of SND's main benefits is its ability to change with the times. As the robot receives fresh sensor data, the Voronoi diagram and subsequent path are updated so that it may react rapidly to changes in its surroundings or moving obstacles. SND is therefore a great choice for use in messy or erratic surroundings, such as packed indoor spaces or unorganized outdoor terrain.

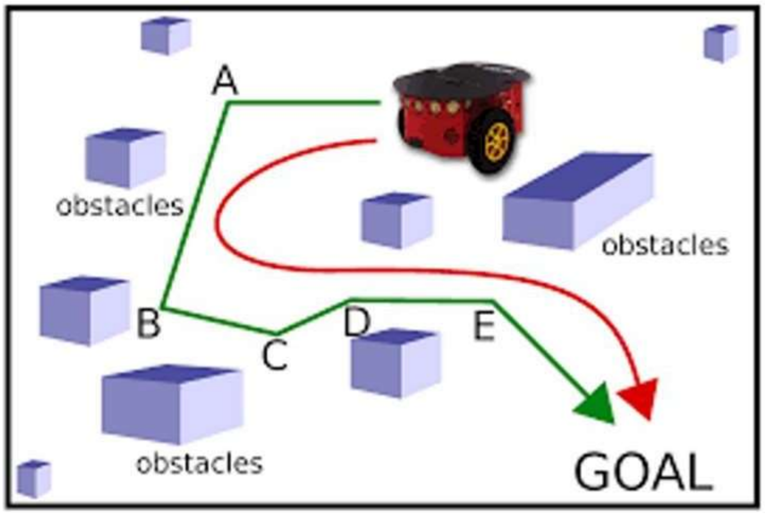


Figure 11. SND

2.4.3 Motion Controller

PID

The PID (Proportional-Integral-Derivative) motion controller is a well-known control method guaranteeing exact and consistent motion in a range of dynamic systems including industrial machinery, robotic arms, and drones. Minimizing the difference between a desired setpoint and the actual measured output generates a feedback control loop that constantly changes the output *(Åström & Murray, 2010).*

The PID controller consists of three main components:

- Proportional (P) control, which produces an output proportional to the error,
- Integral (I) control, which accumulates past errors to eliminate steady-state offset, and
- Derivative (D) control, which predicts future error based on its rate of change.

By carefully tuning the gains for each of these components, the PID controller can respond quickly to disturbances, reduce overshoot, and achieve smooth, stable control *(Li et al., 2017)*. Its versatility and simplicity make it a foundational tool in modern motion control systems.

MPC

Model Predictive Control (MPC) is a sophisticated control strategy widely used in motion control applications, especially where precision and constraint handling are critical. Unlike traditional controllers, MPC uses a model-based approach to predict the future behavior of a system over a defined prediction horizon. By leveraging this dynamic model, it determines a sequence of optimal control actions that guide the system toward its target while respecting physical and operational constraints *(Qin & Badgwell, 2018).*

At each time step, MPC formulates and solves an optimization problem that minimizes a cost function—typically involving tracking error and control effort while considering future states, input limitations, and system dynamics. Only the first input from the optimal sequence is applied, and the process is repeated at the next step with updated measurements, making MPC inherently adaptive and robust *(Mayne et al., 2020)*.

This makes MPC especially powerful in robotics, autonomous vehicles, and industrial automation, where it balances performance with safety, constraints, and long-term planning.

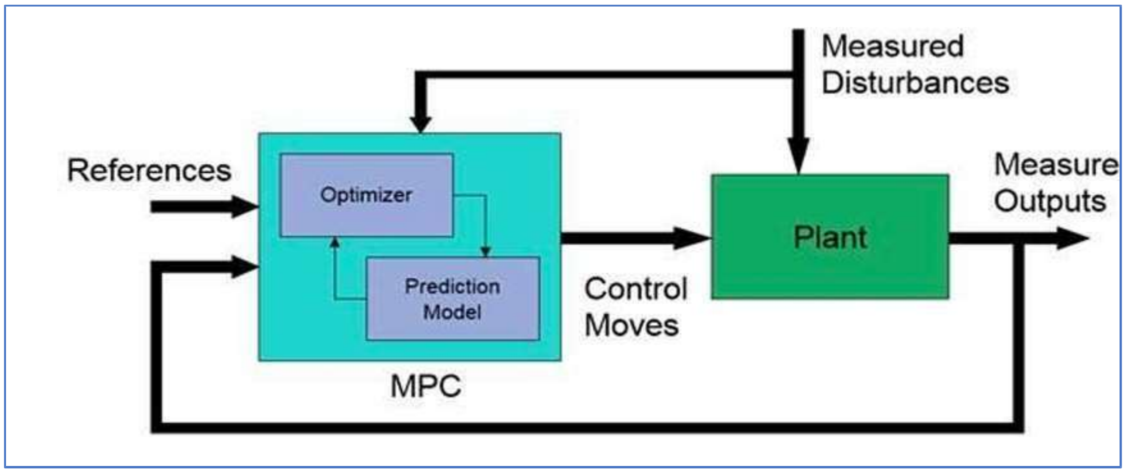


Figure 12. MPC Controller

2.5 CONVERSATIONAL AI

Intuitive and interesting human-robot interactions are much supported by conversational A.I. In the framework of autonomous service robots, such systems are used to provide directions, answer questions, and improve user experience via natural language interactions *(Purington et al., 2017).* Modern conversational agents use Natural Language Processing (NLP), dialogue management, and contextual awareness to produce more human-like and context-aware interactions. Several commercial and research-based platforms such as Amazon Alexa, Google Dialogflow, and Rasa have integrated conversational capabilities to different degrees of complexity and interactivity. Especially in public-facing applications like information kiosks, delivery bots, and assistive technologies, these systems greatly enhance the perceived intelligence and usability of service robots *(Jurafsky & Martin, 2021).*

2.5.1 Speech Recognition

Speech recognition enables the robot to convert spoken language into text. Many modern systems rely on deep learning-based models trained on large datasets, offering robustness to noise and variability in speech. Google's Speech-to-Text API and OpenAI's Whisper model are examples of widely used ASR systems in academic and commercial settings. In service robots like Sanbot and Temi**,** speech recognition modules allow hands-free, voice-driven operation in public settings. *(Curry et al., 2020).*

2.5.2 Natural Language Understanding (NLU)

NLU is responsible for interpreting user intents and extracting relevant entities from text input. In literature, approaches range from traditional rule-based systems to more sophisticated deep learning-based methods. Models like BERT**,** spaCy, and tools such as Rasa NLU are commonly applied. For example, in robots deployed in hospitals or malls, NLU helps determine whether a user is asking for navigation help, requesting information, or initiating a casual interaction.

2.5.3 Dialogue Management

Dialogue managers handle the logic of conversation flow. Early systems used finite state machines or decision trees, while modern dialogue systems often employ reinforcement learning or Transformer-based models to manage multi-turn conversations. Amazon Alexa Conversations and Google Dialogflow CX illustrate commercial implementations that blend rule-based and AI-driven approaches for robust dialogue handling. Research platforms like DialoGPT and Rasa Core provide customizable dialogue pipelines for robotic applications *(Curry et al., 2020).*

2.5.4 Text-to-Speech (TTS)

TTS systems convert the robot's textual responses back into natural-sounding speech. Advances in neural TTS models such as Tacotron 2 and WaveNet have significantly improved the naturalness and expressiveness of synthetic voices. Robots like BellaBot and TUG utilize TTS for verbal interaction in restaurants and hospitals, respectively, offering human-like voice output tailored for their environments.

2.5.5 Integration in Mobile Robots

The integration of conversational AI into autonomous mobile platforms involves combining the above components with onboard computing and robot operating systems. Frameworks such as ROS allow for modular integration of speech input, NLU, and dialogue systems with navigation and display modules. Real-world examples include Pepper, which uses a tablet screen and speech interface to interact with users, and Sanbot, which synchronizes speech with facial expressions and body language for a more engaging experience *(Alibegović et al., 2020)*

2.6 MECHANICAL DESIGN IN SERVICE ROBOTS

The mechanical structure of service robots plays a critical role in determining their mobility, stability, payload capacity, and ability to interact safely within human-centric environments. As service robots are designed to operate in GPS-denied, indoor settings, their form factor must be compact enough to navigate through narrow hallways and doorways while maintaining structural integrity to support hardware such as LiDAR, cameras, screens, and manipulators.

Robots like BellaBot feature a multi-shelf design optimized for food delivery in restaurants, using a stable wheeled base to maintain balance and prevent spillage. TUG, a hospital delivery robot, uses a boxy frame with a secure locking mechanism and a low center of gravity for heavy payloads. KiwiBot, designed for last-mile delivery, prioritizes a small footprint and lightweight materials to maximize agility in both indoor and outdoor urban environments**.** Sanbot, a humanoid robot, incorporates a head-mounted screen and articulated arms to improve user interaction, reflecting how mechanical form enhances communication.

The choice between differential drive and omnidirectional wheels, as well as materials used for the chassis (e.g., aluminum vs. plastic), directly impacts maneuverability and efficiency. Additionally, sensor and actuator placements such as mounting LiDAR at an appropriate height or ensuring cameras have unobstructed views are critical mechanical considerations that support accurate SLAM and navigation.

Chapter 3

PROBLEM DEFINITION

3.1 PROBLEM OVERVIEW

There is an increasing demand for autonomous service robots that can function well in dynamic indoor environments such as hospitals, restaurants, and corporate offices. However, previous systems frequently have one or more drawbacks, such as high costs, a lack of adaptability, restricted real-time navigation in cluttered environments, or inadequate human-robot interaction skills. A critical challenge lies in enabling autonomous operation in GPS-denied environments, where standard satellite-based localization is unavailable. Indoor environments such as hospitals, offices, and warehouses demand reliable alternative localization methods, which many commercial solutions do not fully address. Furthermore, many modern devices rely on simple touch or speech interfaces, which fall short of providing smooth and natural user interactions. As a result, there is a need for an affordable yet versatile autonomous robot that combines reliable localisation, obstacle-aware navigation, and simple voice-based interaction.

3.2 SOLUTION STATEMENT

To address these limitations, we propose the development of a cost-effective, autonomous indoor service robot that combines robust LiDAR-based SLAM for real-time localization and mapping, efficient path planning and obstacle avoidance, and an interactive conversational AI interface. The robot will use 2D LiDAR and SLAM algorithms to function reliably in semi-structured, GPS-denied situations. While voice-based interaction will be made possible through the integration of large language models and speech recognition, navigation will be accomplished through modular planning techniques. With scalability, user accessibility, and intelligent responsiveness in mind, the system seeks to support a range of indoor service applications.

Chapter 4

METHODOLOGY

4.1 OVERVIEW

We considered several parameters when selecting the approach for SLAM and Navigation, including real-time capability, accuracy, and reliability. We started by first testing our approach on a simulation environment which helped us fine-tune the parameters before deploying the software on the hardware prototype. We then conducted various experiments in an indoor area measuring 100x100m using our software.

4.2 SLAM IMPLEMENTATION

For Simultaneous Localization and Mapping (SLAM), we deployed the SLAM Toolbox configured in *online_async* mode with the CeresSolver plugin, within the *ROS 2 Humble* framework. This configuration was selected to optimize both performance and mapping fidelity on a resource-constrained platform, Raspberry Pi 4B (4GB RAM), used onboard our delivery robot. The SLAM system leverages real-time 2D LiDAR data, odometry inputs, and pose graph optimization techniques to maintain an accurate map of the environment while operating within CPU and memory limits

4.2.1 Laser Scan Processing And Odometry Fusion

Our Simultaneous Localization and Mapping (SLAM) system leverages a 2D LiDAR sensor (RPLIDAR A1), operating at a frequency of 5 Hz, as the primary source of environmental perception. The SLAM pipeline is structured into several key stages to ensure accurate and real-time pose estimation:

Scan Preprocessing

Each incoming scan is first filtered to remove noisy or outlier readings (e.g., maximum-range values, statistical anomalies). Downsampling is applied to reduce the number of scan points, thus minimizing computation during scan matching *(Zhang & Singh, 2014).*

Adaptive Scan Matching

A particle filter provides coarse pose alignment by comparing current scans to an existing map. Fine-tuning is achieved through Iterative Closest Point (ICP) alignment refined by *CeresSolver*, which minimizes the error between expected and observed scan features *(Zhang & Singh, 2014)*

Odometry Integration

To enhance robustness, wheel encoder data (or IMU when available) is fused with scan-matching estimates using an Extended Kalman Filter (EKF). This fusion improves pose estimation during periods of poor scan matching or occlusion.

The asynchronous nature of `online_async` mode allows sensor input to be decoupled from backend optimization, enabling low-latency, real-time SLAM even during compute-heavy optimization phases.

4.2.2 Pose Graph Optimization with CeresSolver

Pose graph optimization is a key component in SLAM systems, where the robot's path is modeled as a graph: each node represents the robot's estimated pose at a given time, while the edges define spatial constraints based on odometry, LiDAR, or loop closures. As the robot navigates, small errors in motion estimation and sensor noise accumulate, leading to drift. Pose graph optimization corrects these deviations by globally adjusting poses to minimize the inconsistency in the trajectory.

Ceres Solver, a robust non-linear optimization library developed by Google, is leveraged for this purpose. It solves the least-squares problem formed by the residuals between the observed and predicted relative transformations between connected poses in the graph. This optimization is performed in *SE(2) space*, suitable for planar 2D navigation, which simplifies computation while maintaining accuracy *(Grisetti et al ., 2011).*

To enhance robustness, a *Huber loss function* is employed. This down-weights the impact of outliers, which may arise due to dynamic objects or erroneous LiDAR readings. The solver is configured with the following parameters to ensure stable and efficient convergence:

1. `linear_solver_type = SPARSE_NORMAL_CHOLESKY`
2. `max_num_iterations = 50`
3. `num_threads = 2`
4. `gradient_tolerance = 1e-6`

4.2.3 Lifelong Mapping Under Resource Constraints

Lifelong SLAM systems have to be able to map continuously over long periods without taxing onboard processing or memory resources. The mapping system combines various resource-aware techniques to address these difficulties. First, the global map is divided into ***overlapping submaps***; during scan matching and local pose graph optimization only active submaps are updated. By restricting the range of real-time changes, this greatly lowers the computational load.

Second, ***node sparsification*** is used to cut out low-information or redundant poses from the pose graph. A covariance-based thresholding system guarantees that the deletion of such nodes does not compromise worldwide consistency. Summarized constraints keep the essential information from pruned nodes, therefore preserving the integrity of long-range spatial relationships.

Finally, ***adaptive optimization scheduling*** balances the need for real-time responsiveness with the constraints of embedded processing. Local optimization processes run continuously at a higher frequency to support immediate motion updates, while global optimization is deferred and triggered opportunistically—such as during loop closure events or idle computation windows. These combined techniques allow the system to scale with map size and duration, ensuring sustainability in both memory usage and CPU load, even in GPS-den *(Agerwal & Mierle, 2021).*

4.2.4 Raspberry Pi Specific Tuning and Optimisation

Given the limited computational resources of the Raspberry Pi 4B, several system-level optimizations were implemented to ensure real-time SLAM performance. The Ceres Solver was configured to utilize only two threads, thereby preventing interference with ROS 2 callback functions that handle sensor data and actuator commands. Solver parameters were adjusted to prioritize fast convergence rather than ultra-precise solutions—acknowledging the practical constraints of embedded hardware environments.

Node parameters were tuned inside the ROS 2 architecture to help reduce system load. The `map_update_interval`, for example, was raised to 2.0 seconds; the `scan_topic` was specifically configured to `/scan` to guarantee direct integration with the LiDAR feed. This change lowered the rate of occupancy grid updates, therefore lightening the processing load without much compromise on map accuracy.

Thread isolation was also used by pinning Ceres optimization threads and scan-matching processes to specific CPU cores. Otherwise typical on shared-core architectures, this isolation reduced context switching and latency spikes. The system kept an average CPU use below 60% with these optimizations in place, and the pose graph optimization latency always stayed under 200 ms during tests run in a 50 m^2 indoor setting. These results demonstrate the feasibility of running an advanced SLAM pipeline on a resource-constrained platform like the Raspberry Pi.

These optimizations enabled the SLAM system to run reliably in real time on the Raspberry Pi, demonstrating the practicality of deploying SLAM Toolbox with CeresSolver on embedded platforms *(Welch & Bishop , 2006)*.

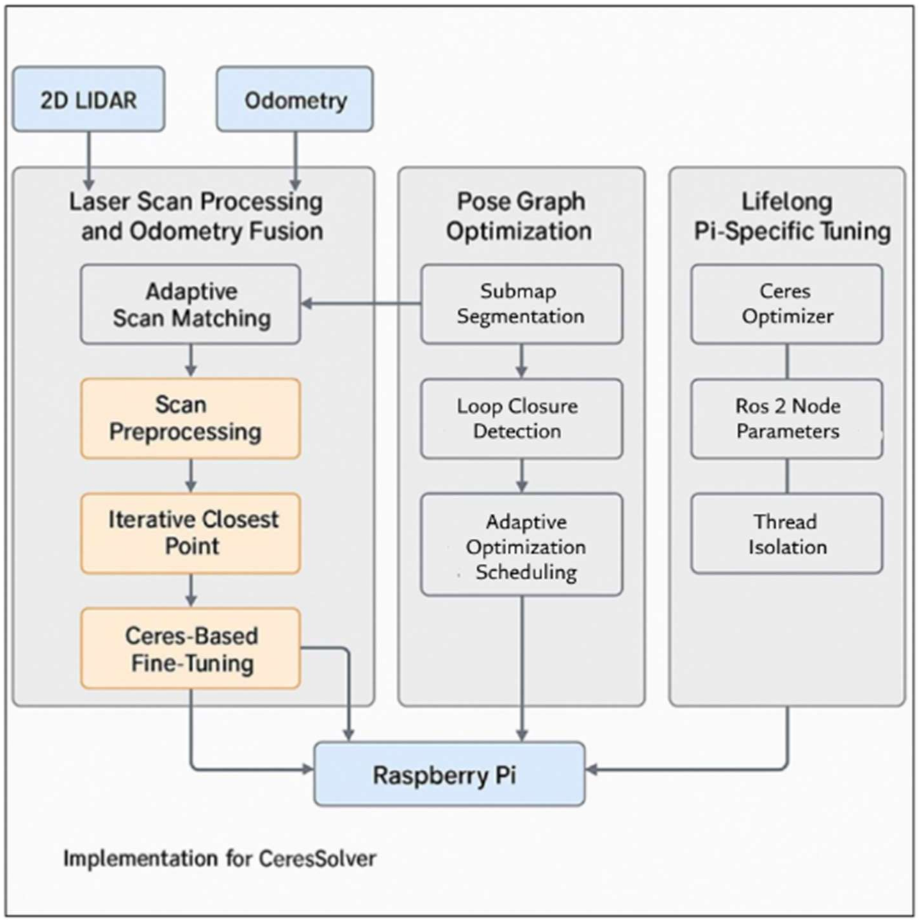


Figure 13. Implementation For CeresSolver

4.3 NAVIGATION

Three essential elements are needed for indoor robot navigation: trajectory execution, path planning, and obstacle detection. To ensure that the robot is aware of its current location and orientation in the environment, the process begins with localization. This was accomplished by using *Adaptive Monte Carlo Localization* (AMCL), which estimates the robot's pose in relation to a known map using sensor data.

After localization is finished, a global path from the starting point to the destination is constructed using the map's cost representation. The Dijkstra method, a grid-based shortest path planner, was used to determine the high-level course while avoiding known static obstructions.

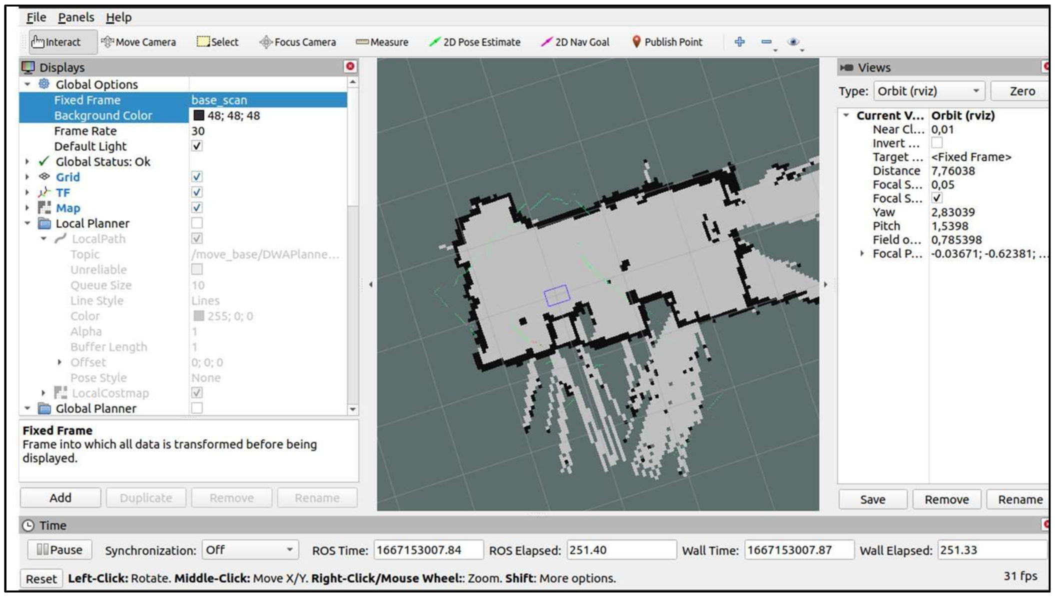
Subsequently, a local planner uses this global trajectory to generate executable commands in real-time. The Dynamic Window Approach (DWB) Controller generates optimal velocity commands based on a local cost map that dynamically incorporates nearby obstacles. The DWB Controller assesses the robot's current pose, orientation, proximity to obstacles, and desired speed to ensure smooth and collision-free navigation along the path *(Huang & Schlegel, 2019).*

4.3.1 Obstacle Awareness

For understanding the working of autonomous navigation of the robot we start from comprehending the obstacle awareness based on maps. There are three kinds of maps involved: static map, global cost map and local cost map. Different maps are used for different sections of navigation.

Static Map

An environment map that remains constant over time. Usually, it contains details on the environment's layout, including where walls, barriers, and other objects are located. Because they enable the robot to create safe and effective routes across the environment without having to continuously assess its surroundings, static maps are crucial for robot navigation. 2D SLAM was used to create the static map for our situation.



Global Options	
Fixed Frame	base_scan
Background Color	48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
TF	✓
Map	✓
Local Planner	
LocalPath	✓
Topic	/move_base/DWAPlanne…
Unreliable	
Queue Size	10
Line Style	Lines
Color	255; 0; 0
Alpha	1
Buffer Length	1
Offset	0; 0; 0
Pose Style	None
LocalCostmap	✓
Global Planner	

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)

Current V…	Orbit (rviz)
Near Cl…	0,01
Invert …	
Target …	<Fixed Frame>
Distance	7,76038
Focal S…	0,05
Focal S…	✓
Yaw	2,83039
Pitch	1,5398
Field o…	0,785398
Focal P…	-0.03671; -0.62381; …

Synchronization: Off ROS Time: 1667153007.84 ROS Elapsed: 251.40 Wall Time: 1667153007.87 Wall Elapsed: 251.33

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 31 fps

Figure 14. Static Map

Global Costmap

A costmap that encompasses the whole environment in which the robot is functioning is called a global costmap. Usually, sensor data from the robot's sensors, such as RGB-D or LIDAR cameras is used to generate it. Long-distance routes that steer clear of hazards and other impediments are planned using the global costmap.

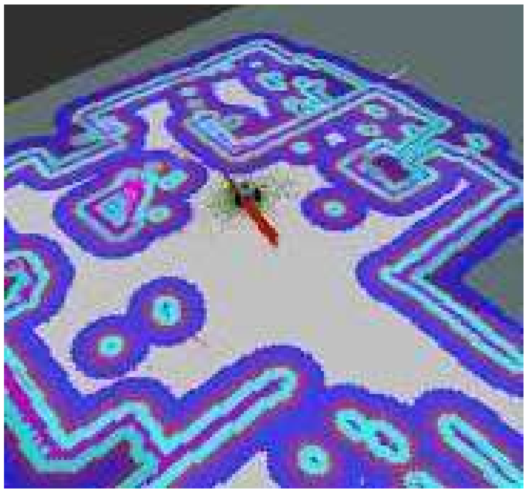


Figure 15. Global Cost Map

The global costmap is used in conjunction with a local costmap, which provides a more detailed view of the robot's immediate surroundings. Together, these costmaps allow the robot to plan and execute complex navigation tasks *(Grisetti et al., 2011).*

Local Costmap

A local costmap, on the other hand, is a costmap that covers only a small portion of the environment around the robot, typically within a few meters of the robot's current position. The local costmap is used to plan short-range paths that allow the robot to avoid immediate obstacles and other hazards.

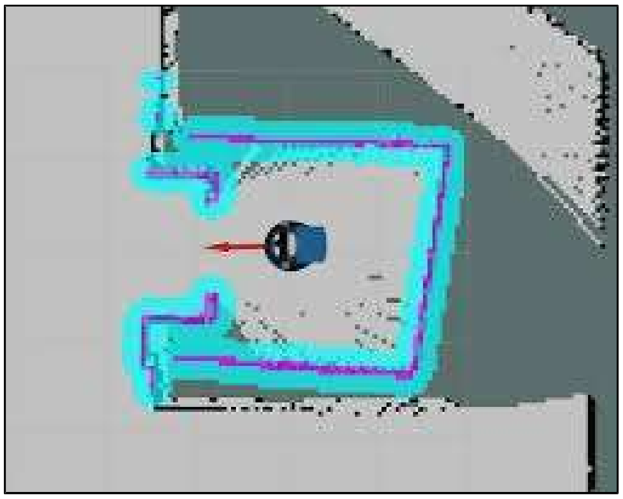


Figure 16. Local Cost Map

4.3.2 Localization

We used the particle filter-based technique Adaptive Monte Carlo Localization (AMCL) for pose estimation. AMCL is a probabilistic method that uses sensor measurements to estimate a robot's pose, or position and orientation, in an environment. The method works by displaying the robot's possible positions as a group of particles, each of which has a weight. The probability of the sensor readings given the robot's posture is used to update these weights. The AMCL algorithm functions as follows in real life:

- Initialize a set of particles with random poses and corresponding weights.
- As the robot moves, update the pose of each particle based on the robot's motion model.
- Capture the robot's sensor readings and calculate the likelihood of each particle's pose considering the sensor data.
- Perform resampling of the particles based on their weights. Higher-weighted particles have a greater chance of being selected for resampling, while lower weighted particles have a lower chance.
- Repeat steps 2-4 as the robot navigates through the environment.

AMCL's ability to adjust to changes in the environment or sensor readings is one of its most notable features. By dynamically modifying the number of particles and the resampling procedure in response to the variance of the particle weights, this adaptability is accomplished. More particles are added or resampling is done more frequently if the weights show significant variability, which suggests that the particles are not accurately estimating the robot's position. On the other hand, fewer particles are employed or resampling is done less frequently if the weights exhibit minimal variability, indicating that the particles provide a trustworthy estimate of the robot's position *(ROS2 Documentation).*

4.3.3 Planner

The planner is in charge of choosing the best course of action for the robot to follow in order to complete a task. We will talk about two crucial elements of robot navigation: global and local planners. The local planner makes sure the robot stays on the intended course while dodging obstacles in real time, while the global planner determines the high-level path from the robot's current location to its objective.

Global Planner

The navigation system's global planner creates a high-level plan for the robot's navigation towards the intended destination by combining data from the Costmap, the robot's localization system, and a predetermined target. Efficiency is given top priority during the planning phase to guarantee the navigation system operates on schedule.

The global planner makes the assumption that the robot is round in the particular context of this navigation system. It plans directly within the configuration space created during obstacle inflation in the Costmap by using the A* method. It is crucial to remember that the planner does not take the robot's kinematics or dynamics into account. This method enables the global planner to respond quickly, but it also results in plans that are typically overly optimistic. As a result, the global planner could occasionally generate routes that the robot cannot follow. For instance, the robot might try to spin tightly through small openings, which could cause the robot's corners to collide with the door frame. Because of these drawbacks, the global planner is mostly used as a high-level navigational guiding system that offers broad directions instead of precise motion planning.

Local Planner

By producing the proper velocity commands, the local planner plays a crucial part in guaranteeing the robot moves safely in the direction of its objective. The robot's kinematics, dynamics, and the obstacle information stored in the Costmap are all taken into consideration as it attempts to closely follow the plan that the global planner provides as an initial input.

The local planner uses a sophisticated method called the DWB Controller, which expands on the Dynamic Window Approach (DWA), to produce safe velocity directives. A cost function is used by the DWB Controller to select possible commands through simulation. This cost function takes into account a number of variables, such as the robot's current speed, the distance to obstacles, and the distance to the path that the global planner creates.

The robot's behavior can be greatly affected by changing the weights given to each element of the cost function. For example, the distance to obstacles would be given a higher weighting factor if the goal was to maintain a maximum distance from impediments. when a result, the robot would slow down significantly when it gets closer to impediments and would even take longer paths to avoid them, especially in confined areas like entrances *(Mackenski, 2019).*

4.3.4 Controller

We needed a high-speed microcontroller with many of GPIOs for encoder readings, PWM signals, and other peripheral extensions in order to run our four-motor system efficiently. Taking these things into account, we decided on the Arduino microcontroller as it fulfilled all the requirements and provided simplicity in programming and interface.

Critical low-level operations that require precise timing and real-time execution are handled by the Arduino and the code uploaded on it. We used a Proportional-Integral-Derivative (PID) method to create robust control and guarantee dependable motor operation at the targeted RPM.

A popular feedback control method for controlling a four-wheeled mechanical drive's velocity is PID control. Despite the nonlinear nature of our method, we chose to construct simpler linear controllers rather than intricate nonlinear controllers. Because they are less expensive to design, execute, comprehend, and troubleshoot, linear controllers are the favored option. We use gain scheduling, a control theory technique, to address the nonlinear characteristics. This method makes use of a family of linear controllers that are alternated according to predetermined parameters. To do this, a simple switch statement is employed.

In our system, the PID controller modifies the control input, the voltage or current delivered to the drive's motors, by using data from sensors such as encoders or speed sensors.

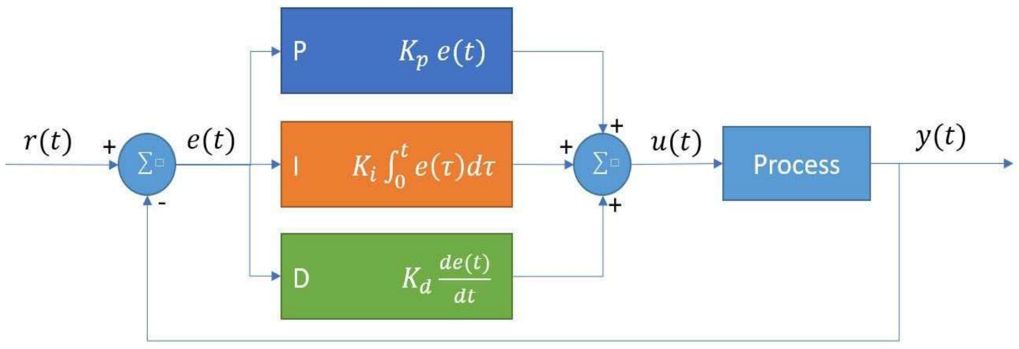


Figure 17. PID Controller

PID Controller consists of 3 components:

- Proportional (P) component: This component produces an output proportionate to the discrepancy between the mechanical drive's actual velocity and its intended velocity. The output of the P component increases with the size of the error. Its function is to offer a prompt initial reaction to the error.
- An output proportionate to the integral of the error over time is produced by the integral (I) component. After the P component has corrected the original error, the I component's job is to remove any steady-state error that could still exist.
- The derivative (D) component produces an output that is proportionate to the error's rate of change over time. The D component lowers oscillation and overshoot, improving system stability.

4.4 CONVERSATIONAL AI

For our system to permit seamless and context-sensitive interactivity between humans and robots, it brings together various parts relative to camera vision, the ability to hear and understand spoken language through speech recognition, and a language model with retrieval-augmented capabilities.

4.4.1 Person Detection Using YOLO

To enable real-time human detection as a precursor to interaction, we deployed the *YOLOv8* object detection model on a resource-constrained edge device, the NVIDIA Jetson Nano. The model was implemented using *Ultralytics*, a high-level open-source framework that provides user-friendly tools for training, exporting, and deploying the YOLO family of models. Ultralytics' pre-trained YOLOv8 weights were used for person detection, ensuring high accuracy without requiring task-specific retraining. Given the limited computational capacity of the Jetson Nano, the model was exported to the *ONNX* (Open Neural Network Exchange) format to leverage ONNX Runtime (from the *onnxruntime* library available on Python), a cross-platform, high-performance inference engine optimized for deploying models on edge devices. This conversion significantly reduced inference latency and improved compatibility with hardware acceleration features *(Abdel Hafez, 2023).*

The detection pipeline initially operated at a modest frame processing rate of ~1.5 FPS. Upon adding multithreading, model inference and frame capture were processed in parallel by different threads, and CPU performance greatly improved. The composed technique enhanced throughput significantly, frame rates increased to 9–10 FPS, while power consumption stayed low. These improvements enabled real-time person detection on low-performance hardware. The architecture of YOLOv8 is given in the diagram.

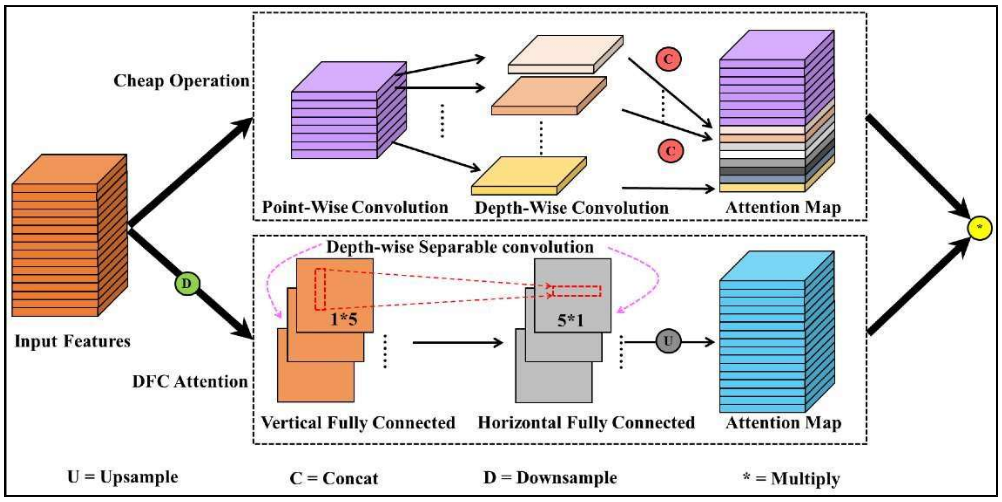


Figure 18. YOLOv8 Structure Diagram

4.4.2 Speech Capture and Recognition

As discussed in the literature review, our system employs a server-based real-time speech recognition approach using Google's Speech-to-Text API, accessed through the speech_recognition Python library. This method provides high accuracy, particularly in noisy or dynamic environments, essential for reliable human-robot interaction.

The audio input is captured via a USB microphone connected to the Jetson Nano. The recorded speech is streamed to Google's cloud-based API, where it is processed and converted into a textual transcription. This cloud-based approach offloads the heavy computational requirements of *automatic speech recognition* (ASR), making it well-suited for deployment on resource-constrained hardware in our project, the Jetson Nano.

Once the spoken input is transcribed into text, the response generated by the language model is converted back into audible speech using the pyttsx3 text-to-speech (TTS) library. The synthesized audio is then played through a USB speaker also connected to the Jetson Nano, enabling the robot to respond in a human-like manner. This speech capture and synthesis pipeline allows for intuitive, hands-free interaction between the user and the robot, forming a crucial component of the overall interaction loop *(Abdel Hafez, 2023).*

4.4.3 Large Language Model Integration

The large language model (LLM) serves as the core reasoning engine for the robot's conversational abilities, enabling it to understand, interpret, and generate human-like responses. Due to the limited computational resources of the Jetson Nano, local deployment of large-scale models such as GPT or similar transformer-based architectures is not feasible. Therefore, we opted for server-based inference using commercially available APIs. The available options for us were Google's Gemini, DeepSeek, and OpenAI's models. We selected OpenAI's GPT-4o for its balanced trade-off between inference cost and response quality. GPT-4o offers fast and accurate responses at a significantly lower token pricing compared to previous OpenAI models, making it optimal for real-time inference on a constrained budget.

The integration of GPT-4o also facilitated an efficient implementation of *Retrieval-Augmented Generation* (RAG), as described in the following section. The model's ability to incorporate retrieved context into its responses resulted in more factual and relevant answers during interaction. The token pricing for GPT-4o, as of the time of deployment, is as follows:

- Input tokens: $5.00 per 1 million tokens
- Output tokens: $15.00 per 1 million tokens

4.4.4 Retrieval Augmented Generation Integration

The large language model (LLM). While large language models like GPT-4o offer strong general-purpose conversational abilities, in case of contextually aware responses, the LLM fails to generate factually correct answers. To address this limitation, we integrated a Retrieval-Augmented Generation (RAG) pipeline into our interaction system. RAG enhances the model's output by grounding it in external, task-specific knowledge retrieved from a pre-built vector database.

We generated a custom database using Chroma (an easy-to-deploy Database in Python). The process begins with converting the user’s transcribed query into an embedding using the same embedding model used to encode the documents stored in the database. These documents may include task instructions, robot specifications, or domain-specific knowledge relevant to the robot’s environment and role. A similarity search is performed on the vector database to retrieve the top *K* most relevant documents based on cosine similarity or other distance metrics. These retrieved chunks of context are then appended to the original user query as additional input to GPT-4o. This augmented prompt enables the model to generate responses that are more accurate, context-aware, and grounded in factual knowledge. This significantly improves the reliability of the interaction, especially for queries that require precise or technical answers *(Alibegovic et al., 2020).*

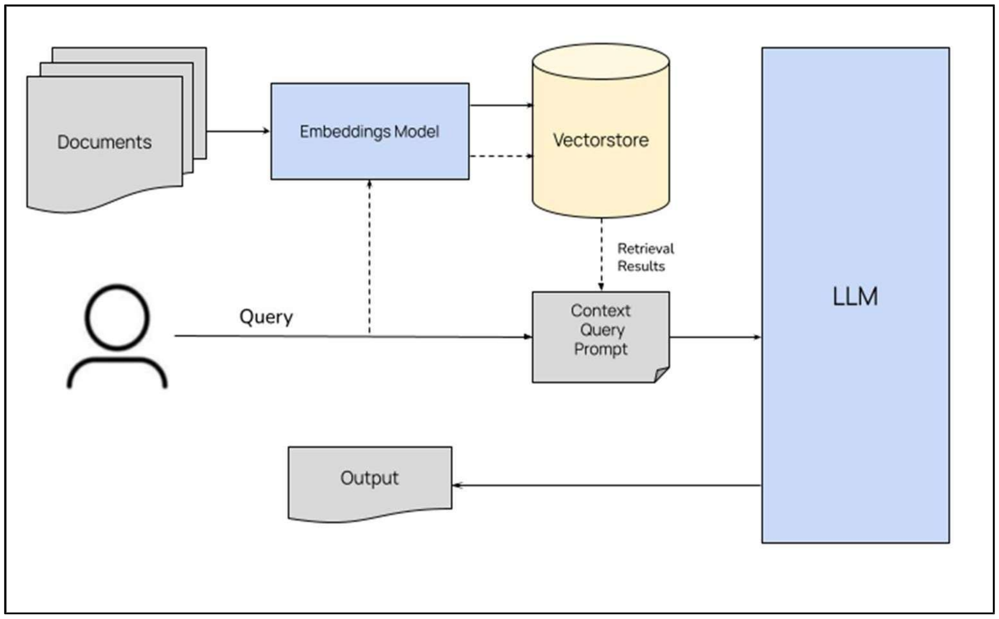


Figure 19. RAG Flow Diagram

4.4.5 Complete interaction flow

In conclusion, the robot's interaction system is built upon a seamless integration of lightweight and cloud-based technologies deployed in for real-time communication on Jetson Nano. The interaction sequence begins with person detection using a multithreaded, ONNX-accelerated YOLOv8 model, which enables camera-feed detection. Upon detecting a person, the robot captures spoken input via a USB microphone and processes it through Google's Speech-to-Text API for accurate transcription. The transcribed query is then forwarded to OpenAI's GPT-4o model, which generates context-aware responses. In case of personalized queries, a Retrieval-Augmented Generation (RAG) mechanism retrieves the most relevant documents from a vector database, forming an augmented prompt that grounds the model's output in domain-specific knowledge. The speech output of the final response is generated using the pyttsx3 library and played through a USB speaker. This human-robot interaction pipeline is reliable and efficient, resulting in fluid interaction, even in resource-constrained environments *(Alibegovic et al., 2020).*

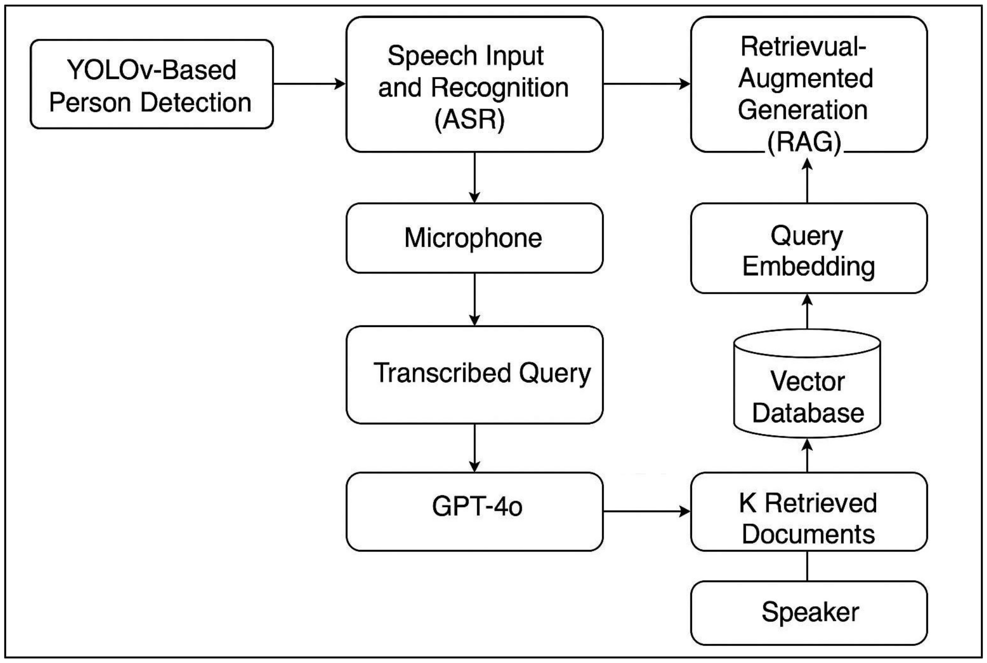

YOLOv-Based Person Detection
Speech Input and Recognition (ASR)
Retrievual-Augmented Generation (RAG)
Microphone
Query Embedding
Transcribed Query
Vector Database
GPT-4o
K Retrieved Documents
Speaker

Figure 20. Complete flow diagram for interaction process

Chapter 5

SYSTEM DESIGN AND ARCHITECTURE

5.1 OVERVIEW

The general system architecture and design of an autonomous mobile assistant robot are presented in this chapter. Targeting autonomous navigation, obstacle avoidance, and human-robot interaction, the robot is built for indoor, GPS-denied surroundings. It scales and modularly combines mechanical, electrical, and computational systems.

Apart from a differential drive system for motions, the robot employs SLAM-based localisation and 2D LiDAR sensor for surroundings mapping. Audio interaction—that is, a speaker, microphone, and data-conveying display screen—is enabled to raise user involvement.

Developed using ROS2, the program framework makes use of its node-based, publish-subscribed architecture to enable modular communication across levels of perception, navigation, and interaction. These include the SLAM-based localization module, global and local path planners, motion controllers, and conversational AI components.

This system architecture ensures reliable operation in complex indoor settings, making it suitable for applications such as indoor delivery, customer assistance, and information services in commercial or institutional environments.

5.2 HARDWARE DESIGN

5.2.1 Mechanical Components

The robot's mechanical architecture has been developed to support steady user engagement, sensor placement, and autonomous mobility. Due to its ease of assembly and modification, the present prototype uses a plywood frame for testing, but the final chassis is planned to be 3D printed using PLA for a more polished and lightweight design.

The robot employs a differential drive system that consists of two powered wheels for mobility and a single caster wheel for balance. For optimal SLAM performance, the LiDAR sensor is positioned in the middle of the front, raised to provide an unhindered 180° field of vision. In order to facilitate user engagement and visibility, a display panel is inserted at the top of the structure.

Table 1. Mechanical Specifications

Component	Description
Chassis Material	PLA (3D Printed); plywood for prototype
Structure Height	Approx. 4 feet
Drive System	2-Wheel Differential Drive
Wheel Type	DC motor-driven rubber wheels
Caster Wheel	1 (rear or front for balancing)
LiDAR Mount	Front, elevated; 360° field of view
Display Mount	Top-mounted screen

5.2.2 Electrical Components

The robot's electronic system is intended to provide autonomous navigation, acquiring data, and user interaction in an efficient and modular manner. The core processing device is a Raspberry Pi 4B, which was selected for its performance and small size while yet being capable of running SLAM algorithms and conversational AI models.

The system includes a 2D RPLiDAR A1 sensor for environmental mapping and obstacle detection, as well as feedback from DC motor encoders for accurate motion control. A rechargeable 12V lithium-ion battery provides power, which is then regulated by buck converters for each subsystem involved.

A top-mounted display screen provides visual output, while a microphone and speaker arrangement provide basic voice-based interaction.

Table 2. Electrical Specifications

Component	**Specification / Model**	**Purpose**
Main Controller	Raspberry Pi 4B (4GB)	Central processing (SLAM, control, AI)
Motor Driver	AQMH2407ND	Controls DC motors
DC Motors	12V DC Gear Motors (×2)	Drive system
Encoders	Rotary encoders (attached to motors)	Motion feedback
LiDAR Sensor	RPLiDAR A1	2D SLAM and obstacle detection
Display Screen	7" LCD/TFT Display	Visual interaction output
Speaker	USB Speaker Module	Conversational audio output
Microphone	USB Microphone Module	Voice input
Battery Pack	12V Lithium-ion	Primary power source
GPU	Jetson Nano	Conversational AI

5.3 COMPONENT JUSTIFICATION AND TECHNICAL OVERVIEW

5.3.1 Robot Computer

Raspberry Pi 4b

Figure 21. Raspberry Pi 4B

The Raspberry Pi 4 Model B is the primary processing unit, with a quad-core Cortex-A72 processor possessing at 1.5 GHz, up to 4GB of LPDDR4 RAM, hardware-accelerated video processing, and USB 3.0. It functions as the ROS 2 framework's master node, performing SLAM computing, global and local path planning, sensor data fusion, and high-level decision-making. The GPIO and UART/SPI/I2C interfaces enable direct communication with the microcontroller and peripheral modules.

Jetson Nano

Figure 22. NVIDIA Jetson Nano

The Jetson Nano serves as a dedicated AI processing unit that performs real-time perception tasks such as conversational AI. It uses its 128-core Maxwell GPU and support for CUDA and TensorRT to run optimised models like as YOLOv8n (nano version) for efficient inference under power and memory limitations.

YOLOv8 allows the robot to identify and interpret user motions or items in its surroundings with minimal latency. The output is Ethernet-transmitted to the Raspberry Pi 4B and integrated into the robot's behaviour layer. Offloading AI duties to the Jetson Nano allows continuous SLAM and navigation on the Raspberry Pi.

5.3.2 Sensors

RPLiDAR A1

The RPLiDAR A1 is a 360° 2D planar LiDAR with a rotating frequency of 5-10 Hz and an effective range of up to 12 meters. It uses time-of-flight triangulation and has enough angular resolution to generate a dense occupancy grid and detect obstacles. It integrates into the SLAM pipeline and allows for real-time environment mapping, particularly in indoor or GPS-denied circumstances when GNSS data is absent or inaccurate.

Figure 23. Slamtec RPLiDAR A1

Wheel Encoders

Quadrature rotary encoders are mounted on the motor shafts to measure wheel rotations and derive odometry information. The encoder outputs are used to compute the robot's linear and angular displacement via dead reckoning. This odometric feedback enables closed-loop motion control and localization. The integration of encoders negates the need for an IMU, which typically suffers from drift and cumulative error in long-duration indoor operation.

5.3.3 Micro-controller

Arduino Uno

The Arduino Uno, based on the ATmega328P microcontroller (16MHz, 2KB SRAM), handles time-critical low-level control tasks like encoder pulse counting, motor PWM signal generation, and sensor interfacing. It connects with the Raspberry Pi over serial (UART), offloading hardware-level operations to ensure deterministic behavior and reduce computing strain on the main CPU.

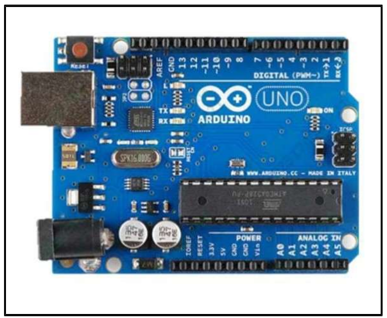


Figure 24. Arduino Uno

5.3.4 Locomotion System

12V DC Motors

To propel each of our wheels, we have incorporated a robust 12 V DC motor. This motor is equipped with a durable 90:1 metal gearbox and an integrated quadrature encoder. The encoder boasts an impressive resolution of 64 counts for every revolution of the motor shaft, resulting in a total of 5756 counts for each revolution of the gearbox's output shaft.

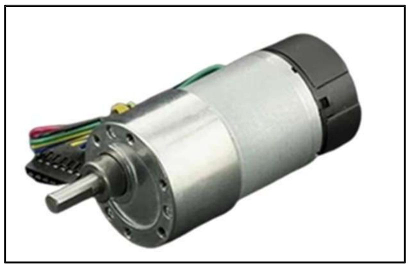


Figure 25. Motor with Encoder

Motor Driver

The AQMH2407ND motor driver is a high performance electronic component designed to control the speed and direction of small motors. It is very compact, measuring just 5.5x5.5cm/2.2x2.2", and can support a wide range of input voltages from 7V to 24V with under voltage protection and power supply transient interference suppression. Each of the dual motor interfaces can provide a rated output current of 7A, supporting three-wire control enable, forward and reverse rotation, and braking, which is similar to the L298 control logic.

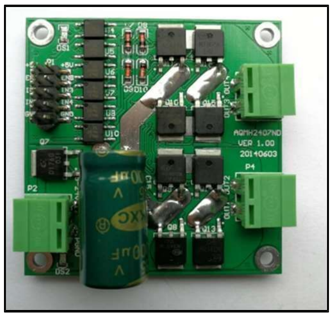


Figure 26. Motor Driver

5.4 SYSTEM DESIGN

5.4.1 Electrical Components

The components are interconnected to enable autonomous navigation as illustrated in the diagram below. The microcontroller (Arduino Uno) is interfaced with wheel encoders and actuators, acquiring real-time odometry data. This data is transmitted to the robot computer (Raspberry Pi 4B) via USB communication. In parallel, a LiDAR A1 sensor provides 2D point cloud data directly to the Raspberry Pi, which runs the SLAM and navigation algorithms.

Based on the fused LiDAR and encoder data, the Raspberry Pi processes the robot's pose and environment to compute velocity commands. These commands are relayed back to the Arduino Uno, which controls the 12V DC motors via a motor driver to achieve the desired motion. The system omits the use of an IMU, as encoder-based odometry sufficiently supports motion estimation for the intended indoor environment.

Additionally, the Raspberry Pi maintains an SSH connection with a host computer, allowing remote access for executing commands, debugging, and visualizing SLAM and navigation data in real time.

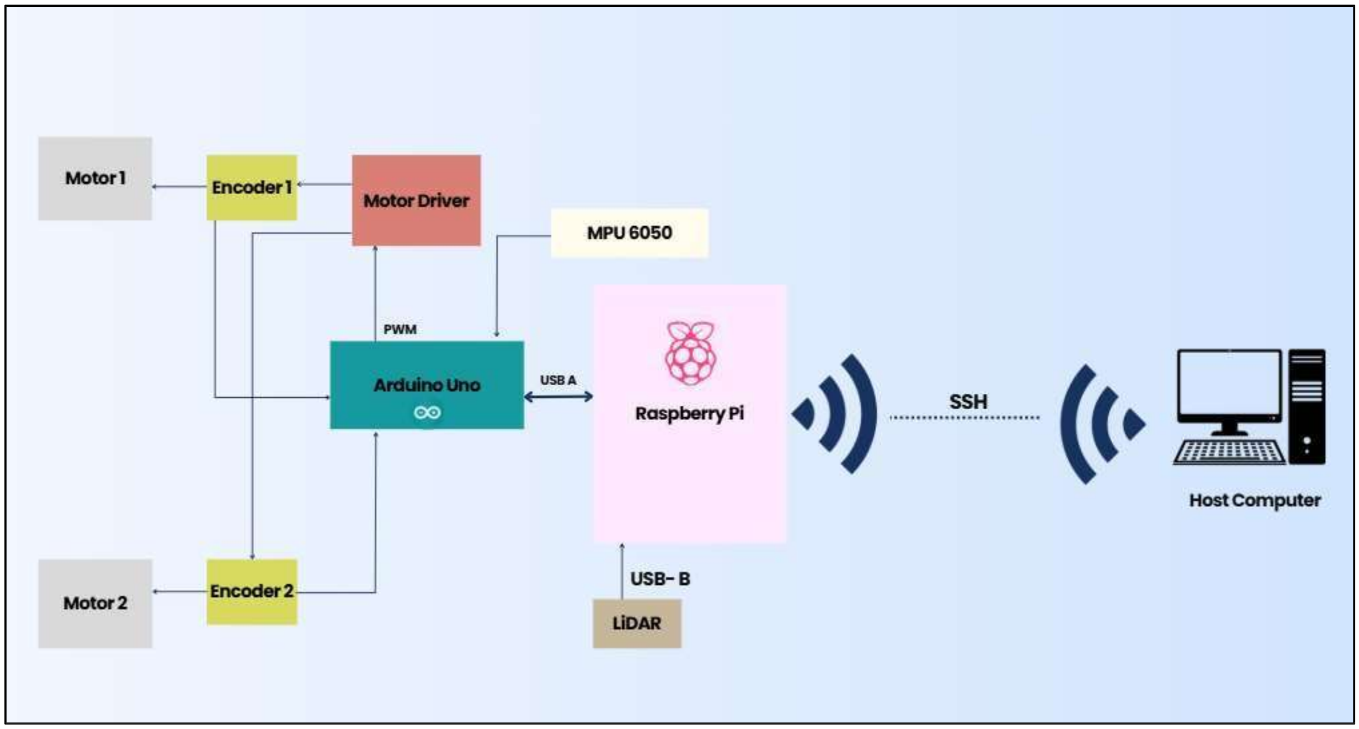


Figure 27. Connections between Hardware Components

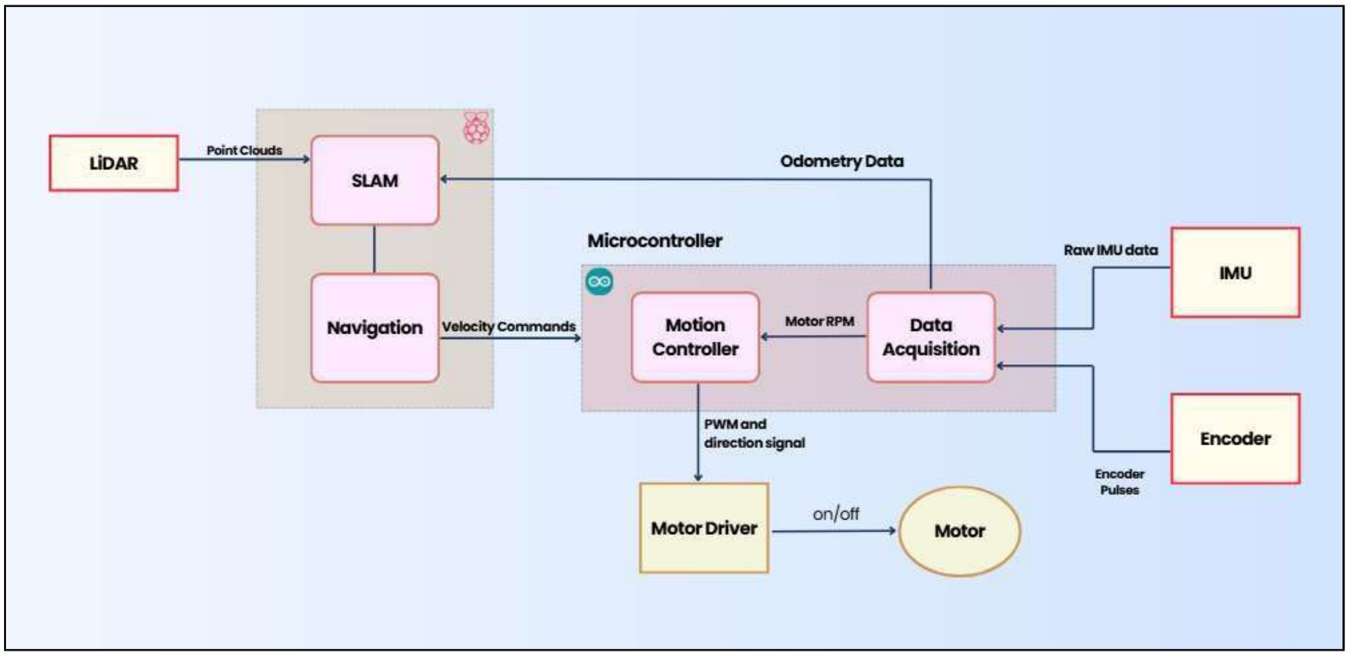


Figure 28. Communication between Software Components

5.4.2 Mechanical Structure

The mechanical structure of the robot was initially designed using CAD software to validate the layout and integration of components such as the LiDAR sensor, display screen, differential drive wheels, and internal electronics. The design follows a rack-style configuration, enabling vertical stacking of modules for better space utilization and cooling. For prototyping, the chassis was manufactured using laser-cut plywood, allowing rapid iteration and testing. The final version is intended to be 3D printed using PLA, offering a more durable, lightweight, and aesthetically refined enclosure. The CAD design ensures that sensor fields of view remain unobstructed and that access for maintenance is straightforward.

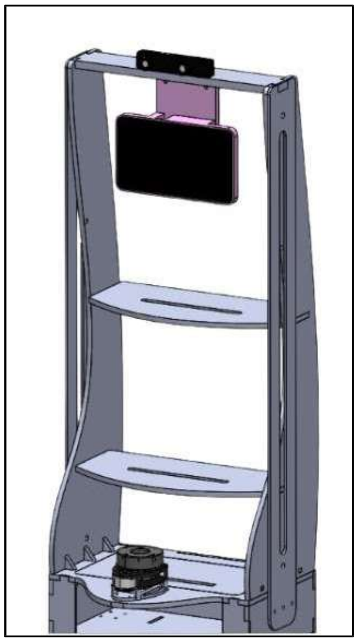


Figure 29. CAD model of the rack-structured mobile robot.

Table 3 Specifications of the Robot

Category	Component	Specification
Mechanical	Chassis Type	2 Wheel Differential Drive
	Frame Type	Rack Style
	Dimensions (LxWxH)	(50x40x105) cm
	Weight	5kg
	Material	Lasercut Plywood (prototype)
	Payload Capacity	upto 3kg
Processing System	Main Processor	Raspberry Pi 4b
	MicroController	Arduino Uno
	Operating System	Ubuntu 22.04 with ROS 2 Humble
Sensors and Perception	LiDAR	Rp LiDAR A1
	Camera Module	Logitech Brio
Software Stack	SLAM framework	Gmapping integrated with ROS2 Framework
	Path Planning	ROS 2 Nav2 stack with Dynamic Obstacle Avoidance
	Navigation Algorithm	LiDAR-based SLAM

Chapter 6

SPECIFICATION SHEETS

6.1 OVERVIEW

This chapter outlines the technical specifications of the key hardware components used in the system. Detailed datasheets and extended specifications are included in the Appendix for reference.

6.2 RASPBERRY PI 4B

The Raspberry Pi 4B is the main unit of the robot computer, handling SLAM processing and system-level control. It has a quad-core Cortex-A72 CPU running at 1.5 GHz and 4 GB of RAM, which provides enough computation for real-time mapping and navigation operations utilising ROS 2. It includes USB and GPIO interfaces for connecting to microcontrollers and peripheral devices. The board's compact dimensions and low power consumption make it perfect for embedded robotics applications. Full specs are included in ***Appendix A.1***.

6.3 NVIDIA JETSON NANO

The NVIDIA Jetson Nano functions as a dedicated AI co-processor for computer vision applications including object detection using YOLOv8. It has a 128-core Maxwell GPU and a quad-core ARM Cortex-A57 CPU, which provide GPU acceleration suited for edge inference. Its CUDA compatibility enables the efficient deployment of deep learning models, making it perfect for running lightweight convolutional neural networks on board. Full specs are included in ***Appendix A.2***.

6.4 RP LiDAR A1

The RPLiDAR A1 is a 2D laser scanner that operates at 5-10 Hz and can generate 360° scan data. It contributes significantly to mapping and localisation by providing range measurements for surrounding barriers. With a range of up to 12 meters and an angular resolution of about 1°, it allows for consistent SLAM performance in indoor conditions. Full specs are included in ***Appendix A.3.***

6.5 ARDUINO UNO

The Arduino Uno microcontroller manages real-time interfacing with low-level hardware, such as motor drivers and encoders. It receives velocity commands from the Raspberry Pi via USB serial communication and outputs PWM signals to drive the actuators. It is also responsible for collecting encoder feedback for odometry estimation, acting as a reliable and low-power control layer. Full specs are included in ***Appendix A.4.***

Chapter 7

IMPLEMENTATION AND TESTING

7.1 OVERVIEW

This chapter outlines the practical realization of the robot system, starting from hardware integration to full software deployment. The implementation involved constructing the prototype, integrating key sensors and actuators, deploying the SLAM and navigation stack, and validating performance through a series of real-world and simulation-based tests. Testing was carried out in a controlled indoor environment using a plywood-structured robot and ROS 2-based tools such as Rviz for visualization and debugging. This chapter also includes photographic evidence of the robot in action and maps generated during its operation.

7.2 System Integration

The integration process began with assembling the prototype robot and physically mounting all hardware components in accordance with the CAD design. The Raspberry Pi 4B was used as the primary robot computer responsible for SLAM and navigation. It was interfaced with the Arduino Uno, which controlled two 12V DC motors via motor driver. The RPLiDAR A1 was mounted at the front with an unobstructed field of view for environment scanning. Wheel encoders were also connected to provide odometry feedback.

On the software side, ROS 2 (Humble) was deployed on the Raspberry Pi. The SLAM Toolbox and Navigation2 stack were configured and launched as independent ROS nodes. The launch files ensured that LiDAR data, encoder feedback, and velocity commands were correctly routed and synchronized. The Jetson Nano was configured separately to run object detection modules using YOLOv8, communicating with the Raspberry Pi over a local network.

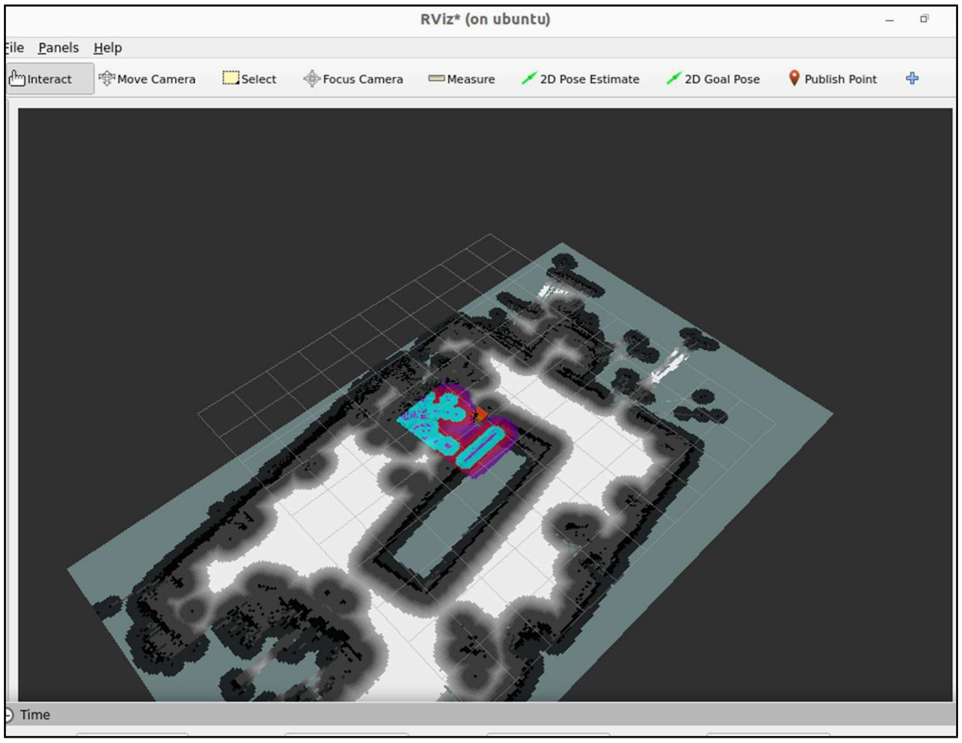


Figure 31. Screenshot of Rviz showing the robot and the map of FYP lab it generated during live SLAM operation.

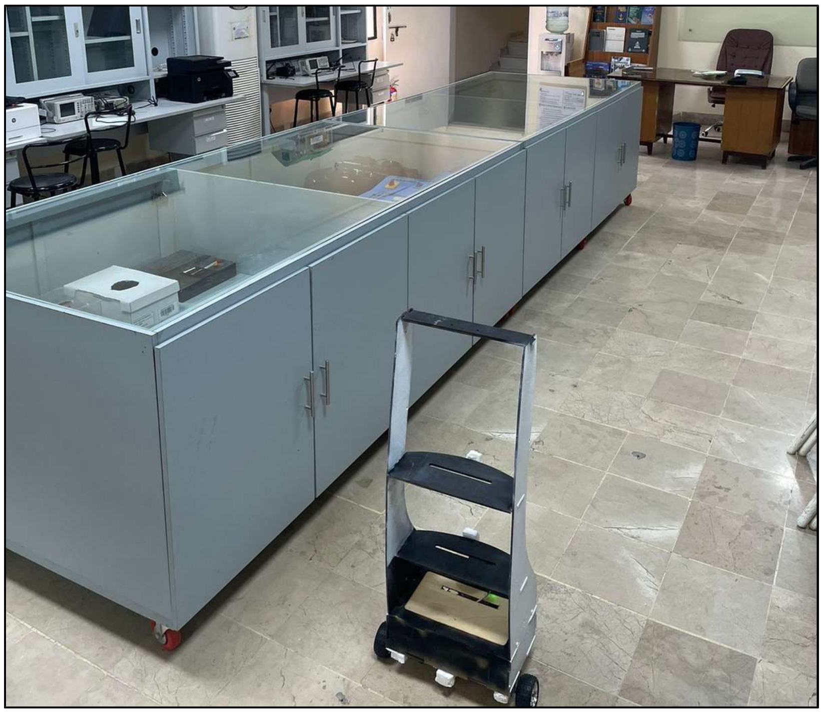


Figure 30. Real-world image of the robot prototype assembled in the lab

7.3 TESTING SETUP

To validate the performance of the robot, testing was conducted in a controlled indoor lab environment measuring approximately 6 m × 8 m. The floor was smooth and tiled, and the lighting conditions were consistent, minimizing the chances of sensor interference. A test track was laid out that included narrow passages, static obstacle (cones), and sharp turns to evaluate navigation robustness and SLAM performance.

The robot was powered using a 12 V Li-ion battery pack, with remote monitoring and control handled via an SSH connection to the Raspberry Pi 4B. Visual monitoring was done using RViz and terminal output via a laptop connected to the same network. All nodes for LiDAR-based SLAM, localization, and navigation were launched on ROS 2 Humble running on the Raspberry Pi.

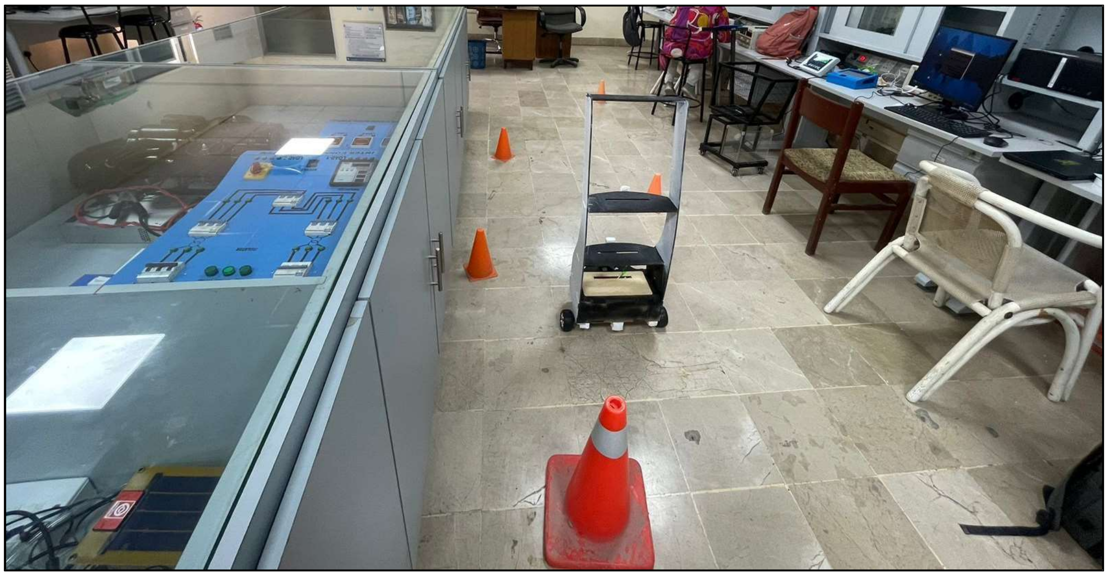
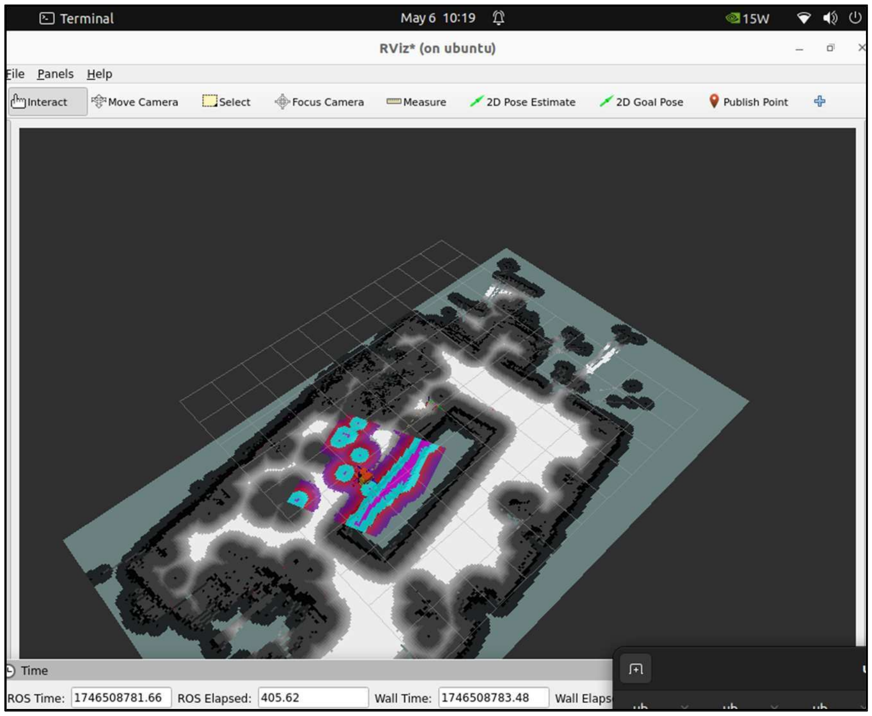


Figure 32. Autonomous Navigation with Obstacles



ROS Time: 1746508781.66 ROS Elapsed: 405.62 Wall Time: 1746508783.48 Wall Elaps

Figure 33. Obstacles detected and updated on the map

7.4 FUNCTIONAL TESTING

7.4.1 SLAM Testing

The robot was tested for its ability to generate and maintain a 2D occupancy grid map using SLAM Toolbox. During exploration, the robot accurately built the map, including walls, corners, and narrow gaps, with minimal drift. Loop closures were detected automatically and corrected using pose graph optimization. The final map demonstrated consistent alignment of walls and objects, even after multiple loops.

7.4.2 Navigation Testing

Once the map was generated, a series of navigation goals were issued to assess path planning and obstacle avoidance capabilities. The robot utilized AMCL for localization and followed global and local trajectories planned using Dijkstra and DWB respectively. The robot consistently reached target positions with low positional error, even in cluttered environments.

7.4.3 Obstacle Avoidance Testing

Static obstacles such as boxes and chairs were introduced into the environment. The robot successfully recalculated paths to avoid these objects using the local costmap and sensors. The generated velocity commands ensured smooth navigation without collisions. In RViz, the updated obstacle layout was accurately reflected, validating real-time environmental awareness.

7.5 CONVERSATIONAL AI TESTING

To evaluate the robot's interactive capabilities, a lightweight Conversational AI module was deployed on the Jetson Nano. The system utilized *YOLOv8* for real-time face detection and a custom dialogue model integrated with a speech-to-text and text-to-speech pipeline. The interaction was initiated automatically when a human face was detected within a defined proximity threshold (~1.5 meters).

Once the face was identified, the robot began verbal interaction using predefined conversational prompts and dynamic voice synthesis. The session continued as long as the face was detected and terminated gracefully when the user explicitly said "Goodbye", triggering a shutdown of the voice interface and screen animation.

Testing confirmed that:

- Face detection operated reliably under different lighting conditions and angles.
- Audio input was correctly transcribed into text using Google Speech Recognition API.
- The robot maintained context in short dialogues and responded within 1.5–2 seconds.
- The interaction ended successfully on receiving the exit command.

This module proved effective in enabling natural, real-time human-robot interaction, making the system suitable for front-desk or tour-guide applications.

Chapter 8

RESULTS AND DISCUSSIONS

8.1 OVERVIEW

This chapter evaluates the real-world performance and challenges encountered during the implementation of the autonomous mobile assistant robot. It reflects on simulation inconsistencies, hardware-induced limitations, and tuning strategies. Additionally, it presents the project's budgeting and timeline in structured formats.

8.2 LIMITATIONS

8.2.1 Low Frame Rate in Robot Movement (Simulation Sync Issue)

Issue: The robot's motion in Gazebo appeared choppy due to slow or inconsistent simulation time updates to ROS, leading to timing errors in control loops and sensor integration.

Solution: A new `gazebo_params.yaml` file was created, specifying a `publish_rate` of 400 Hz under a node named `gazebo`. This high frequency ensured smoother simulation-to-ROS synchronization, improving both visual and operational fidelity.

8.2.2 Orientation Drift Post-Rotation in Gazebo

Issue: After a full rotation, the robot's orientation in RViz differed from its actual pose in the simulation. This highlighted a mismatch between simulated kinematics and perceived motion.

Solution: The wheel collision geometries in `robot_core.xacro` were updated from cylindrical to spherical to reduce simulation artifacts. This significantly minimized drift, maintaining alignment accuracy post-rotation.

8.2.3 Navigation Failure in Narrow Spaces

Issue: The robot often halted or failed to reach its goal due to overly conservative spatial tolerance parameters in the navigation stack.

Solution: The robot's footprint and costmap inflation parameters were adaptively tuned to offer an optimal balance between safety and maneuverability. After testing multiple configurations, a setting was selected that allowed safe passage through tighter environments without triggering failure states.

8.2.4 Restricted LiDAR Field of View

Issue: The LiDAR was partially enclosed for protection, which resulted in erroneous obstacle detection due to internal structure reflections.

Solution: A custom Python script filtered the scan data to retain only readings within the sensor's actual angular viewing window. This filtered data was published under /filteredscan, which the navigation stack subscribed to. It prevented false detections without any hardware modification, improving path planning accuracy.

8.2 BUDGET

The Budget is estimated to be PKR 123,500.

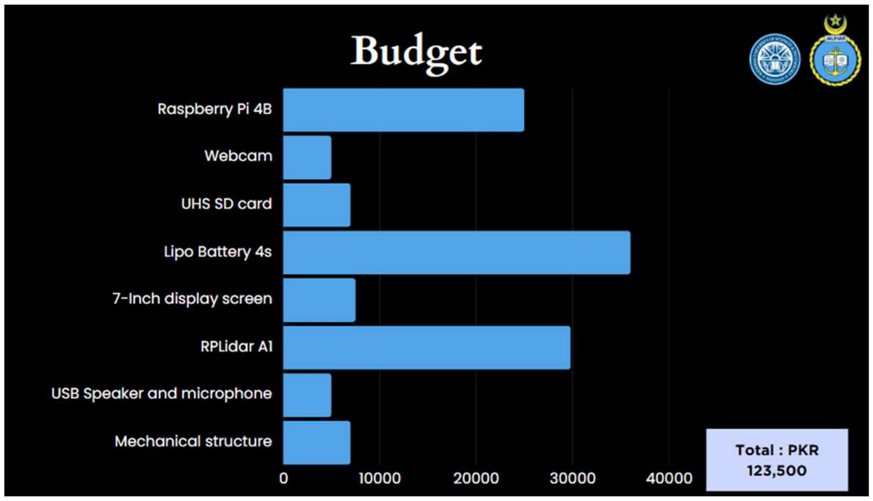


Figure 34. Budget

8.3 GANTT CHART

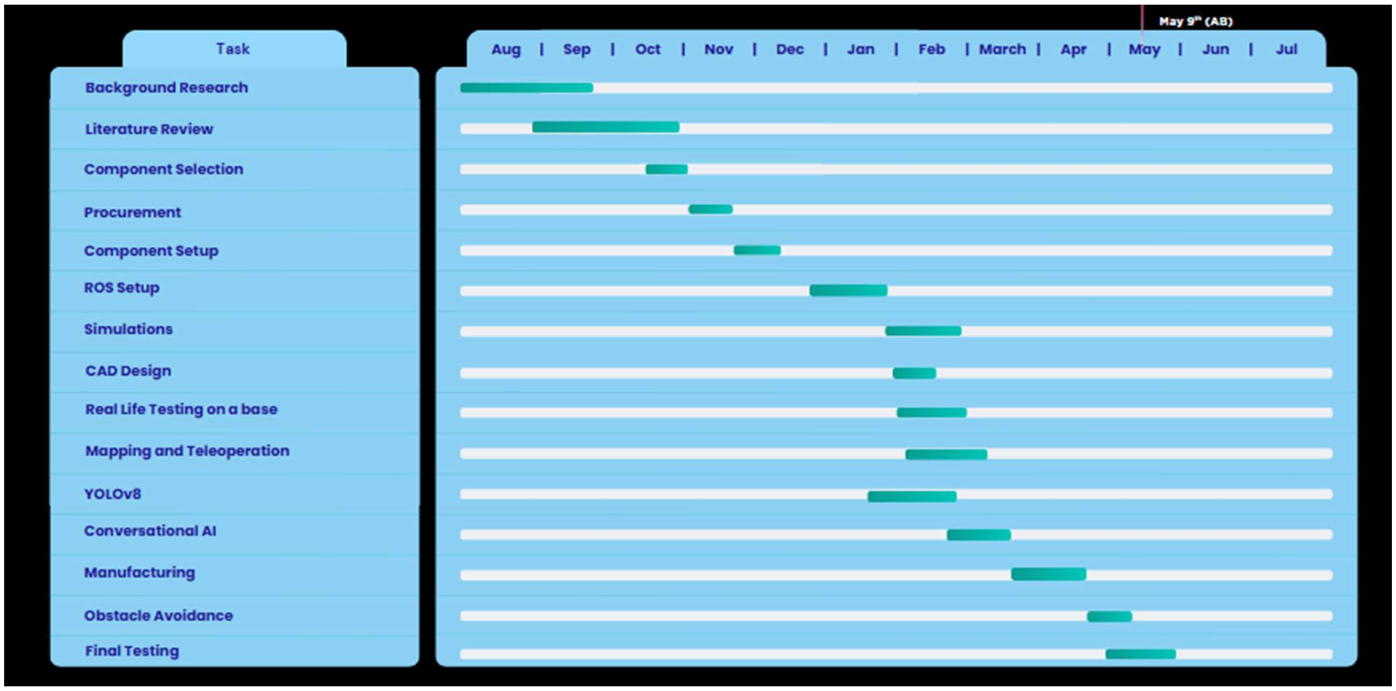


Figure 35. Gantt Chart

Chapter 9

CONCLUSION AND FUTURE WORK

9.1 CONCLUSION

This project was successful in designing and developing a general-purpose autonomous mobile assistant robot capable of performing SLAM-based navigation, obstacle avoidance, and interactive human communication via voice and face expression on a display. From idea to execution, the system combined several subsystems, including LiDAR-based mapping, embedded control via Jetson Nano and Raspberry Pi, voice interface via LLM integration, and real-world obstacle handling confirmed using RViz and physical testing. Key problems, such as simulation inconsistencies, sensor field-of-view limitations, and navigation tuning, were addressed methodically through software and structural changes.

The finished prototype has strong interior navigation and interaction capabilities, and it serves as a proof of concept for service applications in workplaces, hospitals, and educational institutions. The system's modular design allows for future improvements.

9.2 FUTURE WORK

While the current prototype fulfills its design objectives, there is ample room for expansion and refinement:

- **3D Printing of Final Structure** The existing prototype is made using plywood in a rack-style configuration. Future iterations will involve a lightweight, durable chassis manufactured using PLA filament and 3D printing, allowing for improved aesthetics and structural uniformity.

- **Enhanced Human-Robot Interaction** We aim to expand the expressiveness of the robot by implementing more advanced facial replication using real-time emotion rendering on the display screen.
- **Graphical User Interface (GUI)** A touch-enabled GUI will be developed for the screen, enabling users to interact with the robot via touch-based menus in addition to voice commands.
- **Expanded Voice Intelligence** Integration of multi-language support and context-aware dialogue management will further improve the conversational AI experience.
- **Autonomous Charging** Incorporating a docking station for autonomous battery charging would improve long-term deployment feasibility.

These enhancements will transition the prototype into a production-grade mobile assistant robot, suitable for deployment across a wide range of service-oriented environments.

Chapter 10

REFERENCES

10.1 References from Journals

1. Bresson, G., Alsayed, Z., Yu, L., & Glaser, S. (2017), *Simultaneous localization and mapping: A survey of current trends in autonomous driving*, IEEE Transactions on Intelligent Vehicles, 2(3): 194–220.
2. Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H., & Csorba, M. (2021), *A solution to the simultaneous localization and map building (SLAM) problem*, IEEE Transactions on Robotics and Automation, 17(3): 229–241.
3. Grisetti, G., Stachniss, C., & Burgard, W. (2007), *Improved techniques for grid mapping with Rao-Blackwellized particle filters*, IEEE Transactions on Robotics, 23(1): 34–46.
4. Grisetti, G., Stachniss, C., & Burgard, W. (2010), *Nonlinear constraint network optimization for efficient map learning*, IEEE Transactions on Intelligent Transportation Systems, 10(3): 428–439.
5. Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., & Dellaert, F. (2012), *iSAM2: Incremental smoothing and mapping using the Bayes tree*, The International Journal of Robotics Research, 31(2): 216–235.
6. Khaleghi, A., Khamis, A., & Karray, F. (2013), *Multisensor Data Fusion: A Review of the State-of-the-Art*, Information Fusion, 14(1): 28–44.
7. Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., & Burgard, W. (2011), *g2o:A general framework for graph optimization*, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA): 3607–3613.
8. Li, X., Li, Z., & Yang, Y. (2020), *Design and implementation of a food delivery robot for indoor environments*, International Journal of Advanced Robotic Systems, 17(2): 1–12.
9. Pomerleau, D., & Bhat, S. (2020), *Testing and Validation of Autonomous Robot Systems: A Framework for Real-World Testing*, Springer.
10. Wang, Y., et al. (2020), *The AMIRO Social Robotics Framework: Deployment and Evaluation*, Frontiers in Robotics and AI, 7: 578597.

11. Zhang, J., & Singh, S. (2014), *LOAM: Lidar odometry and mapping in real-time*, Robotics: Science and Systems Conference (RSS).

10.2 References from Books

1. Bekey, G.A. (2005), *Autonomous Robots: From Biological Inspiration to Implementation and Control*, MIT Press, Cambridge, pp. 145–167.
2. Corke, P. (2017), *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, Springer, Cham, Switzerland, pp. 1–400.
3. Gerig, L., & Seligman, D. (2018), *Introduction to ROS2 and Robot Operating Systems*, O'Reilly Media, Sebastopol, CA, pp. 1–350.
4. Goodfellow, I., Bengio, Y., & Courville, A. (2016), *Deep Learning*, MIT Press, Cambridge, MA, pp. 1–775.
5. Huang, K., & Schlegel, J. (2019), *Robot Perception for Autonomous Navigation: SLAM and Sensor Fusion*, Springer, Berlin, pp. 1–420.
6. Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009), *Robotics: Modelling, Planning and Control*, Springer-Verlag, London, pp. 391–400.
7. Siegwart, R., Nourbakhsh, I.R., & Scaramuzza, D. (2011), *Introduction to Autonomous Mobile Robots (2nd ed.)*, MIT Press, Cambridge, MA, pp. 83–145.
8. Welch, G., & Bishop, G. (2006), *An Introduction to the Kalman Filter*, University of North Carolina at Chapel Hill, Department of Computer Science, pp. 1–16.

10.3 References from Internet

1. Abdel Hafez, R. (2023). *Enhancing Human-Robot Interaction: Integrating Large Language Models into Social Robots*. Theseus.fi. Retrieved from https://www.theseus.fi/handle/10024/789597
2. Adafruit Industries. *Lithium Ion Battery Pack - 12V*. Retrieved from https://www.adafruit.com/product/353
3. Aethon. *TUG Autonomous Mobile Robot Platform*. Retrieved from https://aethon.com/tug/
4. Agarwal, S., & Mierle, K. *Ceres Solver: A Nonlinear Least Squares Minimizer*. Retrieved from http://ceres-solver.org/

5. Alibegović, B., Prljača, N., Kimmel, M., & Schultalbers, M. (2020). *Speech recognition system for a service robot - a performance evaluation*. ResearchGate. Retrieved from https://www.researchgate.net/publication/346471812_Speech_recognition_system_for_a_service_robot_-_a_performance_evaluation
6. ICP Algorithm Explained – Open3D Documentation. Retrieved from http://www.open3d.org/docs/release/tutorial/pipelines/icp_registration.html
7. Jetson Nano Documentation. *NVIDIA Jetson Nano Developer Kit*. NVIDIA. Retrieved from https://developer.nvidia.com/embedded/jetson-nano-developer-kit
8. Kalman Filter Introduction – MathWorks. Retrieved from https://www.mathworks.com/help/control/ref/kalman.html
9. KiwiBot. *Kiwi Autonomous Delivery Robots*. Retrieved from https://www.kiwibot.com/
10. Kuemmerle, R., Grisetti, G., Strasdat, H., Konolige, K., & Burgard, W. (2011). *g2o: A general framework for graph optimization*. GitHub. Retrieved from https://github.com/RainerKuemmerle/g2o
11. LLMStack Team. *Retrieval-Augmented Generation (RAG) Explained*. Retrieved from https://llmstack.ai/blog/retrieval-augmented-generation
12. Macenski, S. *SLAM Toolbox Documentation*. GitHub. Retrieved from https://github.com/SteveMacenski/slam_toolbox
13. Open Robotics. *ROS 2 Humble Hawksbill Documentation*. Retrieved from https://docs.ros.org/en/humble/index.html
14. Pudu Robotics. *BellaBot – A Delivery Robot Designed for Human-Robot Interaction*. Retrieved from https://www.pudurobotics.com/bellabot
15. Raspberry Pi Foundation. *Raspberry Pi 4 Model B Specifications*. Retrieved from https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
16. ROS 2 Documentation. *Introduction to ROS 2 and SLAM*. Retrieved from https://docs.ros.org/en/rolling/Tutorials/SLAM-Tutorial.html
17. Sanbot Innovation. *Sanbot Intelligent Humanoid Robot*. Retrieved from https://www.sanbot.com/
18. Slamtec. *RPLIDAR A1: 360 Degree Laser Scanner Development Kit*. Retrieved from https://www.slamtec.com/en/Lidar/A1

19. STMicroelectronics. *L298N Dual H-Bridge Motor Driver Datasheet*. Retrieved from https://www.st.com/en/motor-drivers/l298.html
20. Vrogue. *Detailed Illustration of YOLOv8 Model Architecture*. Retrieved from https://www.vrogue.co/post/detailed-illustration-of-yolov8-model-architecture-th-vrogue-co
21. Zhang, J., & Singh, S. (2014). *LOAM: Lidar odometry and mapping in real-time*. ResearchGate. Retrieved from https://www.researchgate.net/publication/282704722_LOAM_Lidar_Odometry_and_Mapping_in_Real-time

Chapter 11

APPENDIX A

Appendix A.1 – Raspberry Pi 4B Specification

Feature	Specification
Processor	Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
RAM Options	2GB, 4GB, or 8GB LPDDR4-3200 SDRAM
USB Ports	2 × USB 3.0, 2 × USB 2.0
Networking	Gigabit Ethernet, 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless, Bluetooth 5.0
GPIO	40-pin GPIO header, fully backward-compatible
Display Interface	2 × micro-HDMI ports (up to 4Kp60)
Storage	microSD card slot, USB boot supported
Power Supply	5V DC via USB-C connector (minimum 3A)
Operating System	Linux-based (Raspberry Pi OS, Ubuntu, etc.)
Dimensions	85.6mm × 56.5mm × 17mm

Appendix A.2 – NVIDIA Jetson Nano Specification

Feature	Specification
GPU	128-core Maxwell GPU
CPU	Quad-core ARM Cortex-A57 @ 1.43GHz
Memory	4 GB LPDDR4
Storage	microSD card slot
Connectivity	Gigabit Ethernet
USB Ports	4 × USB 3.0, 1 × USB 2.0 Micro-B

Feature	Specification
Display Output	HDMI 2.0 and eDP 1.4
Camera Interface	MIPI CSI-2 ×2 lanes
Power Supply	5V/4A via barrel jack or micro-USB
Operating System	Ubuntu-based JetPack SDK
Dimensions	100mm × 80mm × 29mm

Appendix A.3 – RP LiDAR A1

Feature	Specification
Scanning Frequency	5 – 10 Hz (adjustable)
Sample Rate	8000 samples/second (typical)
Range Distance	0.15 m to 12 m
Angular Resolution	<1° (typical)
Field of View (FOV)	360° full rotation
Interface	UART/USB
Power Consumption	5V @ 500 mA
Weight	~190 g
Dimensions	Diameter 70 mm, Height 41 mm

Appendix A.4 – Arduino Uno Specification

Feature	Specification
Microcontroller	ATmega328P
Operating Voltage	5V

Feature	**Specification**
Input Voltage	7–12V (recommended)
Digital I/O Pins	14 (6 PWM capable)
Analog Input Pins	6
Clock Speed	16 MHz
Flash Memory	32 KB (0.5 KB used by bootloader)
SRAM	2 KB
EEPROM	1 KB
Communication	USB, UART, SPI, I2C