Autonomous Navigation System For Indoor Delivery Robots

Software Requirements Specification

By

Syed Muhammad Hussain CS-24	(sh06892@st.habib.edu.pk)
Syed Muhammad Daniyal CS-24	(sz06880@st.habib.edu.pk)
Muhammad Ahmed Atif CS-24	(ma06413@st.habib.edu.pk)
Laiba Ahmed EE-24	(la06855@st.habib.edu.pk)

January 9, 2024

In partial fulfillment of the requirement for

Bachelor of Science

Computer Science

Dhanani School of Science and Engineering

Habib University

Fall 2023

1 Introduction

This interdisciplinary project aims to develop an autonomous delivery system inspired by the Industry 4.0 paradigm, with a focus on SLAM (Simultaneous Localization and Mapping), Navigation, and Voice Bot technology. The project envisions a future where society functions with minimal human intervention, enhancing operational efficiency and safety. The project seeks to create a versatile autonomous mobile robot platform adaptable to various sectors, including industrial logistics, healthcare, unmanned missions, and security operations. For its final year milestone, the project focuses on the development of a self-driving delivery robot for indoor environments, notably the Habib University cafeteria. The robot will integrate SLAM to create a map of its surroundings and localize itself within the map. This will allow the robot to navigate safely and efficiently, even in dynamic and cluttered environments. The robot will also use navigation algorithms to plan and execute paths to its destination, avoiding obstacles and other hazards. An interactive food ordering system with a voice bot interface will enhance the user experience and streamline the ordering process. The voice bot will allow users to place orders using natural language, making the system more accessible and user-friendly.

2 Purpose Of Document

To ensure the successful development and implementation of autonomous navigation capabilities for indoor delivery robots, a System Requirement Specification (SRS) for this project is being created. This specification aims to define and document the specific functional and non-functional requirements of the system. Elaborating between which use-cases are of functional type and which are non-functional type helps associate priority in Project Scope and Deliverable.

This document also ensures smooth execution of project deliverable by establishing a project plan. This project plan helps the stakeholders to identify the dependencies for fulfilling functional and non-functional requirements of this projects. These dependencies can include project objectives, tasks, timelines, resources, and risks.

3 Project Plan

3.1 Project Scope and Objectives

Robot Simulation on ROS (Gazebo, Rviz) Platform: For Kaavish 1

1. Development of a realistic simulation environment on the Robot Operating System (ROS) platform, incorporating Gazebo for physics-based simulation and Rviz for visualization.
2. Implementation of path planning algorithms to enable the robot to navigate dynamic indoor environments.
3. Integration of Simultaneous Localization and Mapping (SLAM) techniques to enable the robot to create maps of its surroundings.

Interactive Food Ordering Website: For Kaavish 2

1. Development of an interactive and user-friendly website for food ordering.
2. Implementation of a seamless ordering process, allowing customers to place orders conveniently.
3. Integration of communication and coordination features, enabling efficient interaction between customers and the autonomous delivery robot.

Delivery Robot Prototype: For Kaavish 1 and 2

1. Designing the robot structure
2. Interfacing and Integration of website and sensor module in the Robot prototype platform.

3.2 Resources

Hardware Resources

1. Raspberry Pi4 board
2. NVIDIA's Jestson Nano (2GB)
3. Arduino AtMega board
4. Slamtec RPLIDAR A3 2D
5. SR305 Intel Realsense Camera
6. Encoder DC motors
7. Robot Chasis

Software Resources

1. ROS2 Humble LTS
2. Gazebo11
3. Ubuntu 22 LTS
4. RViz
5. Nav2 Stack

3.3 Risk Analysis

The anticipated Risk for this project are as follow:

1. Lack of funds or availability of desired hardware can slow the pace of the project. In more extreme case it can also poses the risk of modifying Project Statement.
2. There is a risk of meeting planned deadline. Since there are a lot of moving part to this project and most of the things are new to learn hence it might take longer than anticipated to fully understand the elements involved in this project.
3. There is a slight risk of not being able to transition the project from simulation to practical working robot, as the simulations are too ideal and on contrary the real world setting comes with a lot of variable that are not accounted in the model exported from simulation to real setting.

3.4 Project Timeline

You can access the Project Timeline, Tasks and corresponding Gantt chart for more details and updates at the following link:
`https://docs.google.com/spreadsheets/d/1TRYLb7lYOgMUk7PVofgamzo7w5hh2eDoUBj9Yw-dJXA/edit?usp=sharing`

4 System Diagram and Description

4.1 User Journey High-Level Diagram and Description

This section of the document aims to elucidate the user experience within our system in a comprehensive manner. The high-level diagram presented in Figure 1 offers a visual representation of the step-by-step process that users go through when interacting with our system.

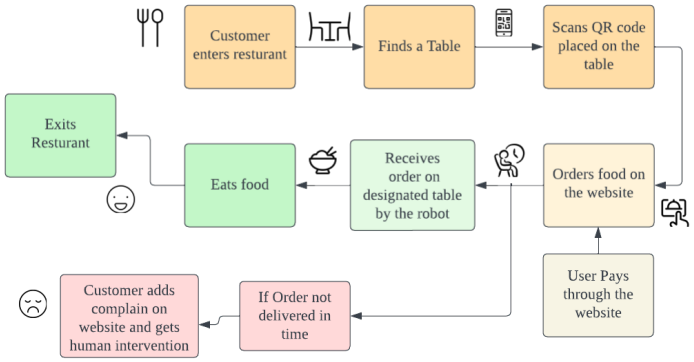

Figure 1: Customer/User Journey

In this diagram, we capture the entire journey, from the initial interaction with our system to the point where users successfully order food through the unique mechanism of QR code scanning. This ordering process is a crucial aspect of our system and is designed to seamlessly guide users from their point of entry to the user-friendly website interface.

Key Components:

1. **Initial Interaction:** The starting point of the user's journey, where they initiate contact with our system. It may involve activities such as accessing a menu or encountering a QR code.
2. **QR Code Scanning:** A crucial element in our system, allowing users to easily scan QR codes to access our platform, facilitating a smooth transition from the physical world to the digital realm.
3. **Website Interface:** Users transition to our website interface, where they can explore our offerings, browse menus, and conveniently place their food orders.
4. **Order Placement:** This step emphasizes the core functionality of our system—food ordering. Users can select their desired items, customize their orders, and finalize the purchase, all within the website interface.
5. **Order Confirmation:** After submitting their order, the system confirms the details, providing users with order confirmation and estimated delivery times.
6. **Payment Processing:** If applicable, this stage showcases the secure and efficient payment processing procedures that our system employs.
7. **Delivery or Pickup:** Depending on user preferences, our system handles the logistics of delivering the food to the specified location or notifying users when their order is ready for pickup.
8. **Feedback and Follow-up:** The user journey extends beyond the point of purchase, acknowledging the importance of post-order feedback, support, and any follow-up communication with users.

Purpose and Benefits:

By presenting this high-level user journey diagram and description, we aim to provide a comprehensive understanding of the user's experience within our system. This visual representation helps stakeholders, team members, and users grasp the system's functionality, anticipate user needs, and identify potential areas for improvement. Furthermore, it underscores the user-centric approach our system adopts, prioritizing a smooth, efficient, and satisfying interaction for every user.

4.2 Robot Journey High-Level Diagram and Description

In this section, we provide an in-depth exploration of the high-level diagram depicting the journey of our robot within the system. Figure 2 illustrates a detailed, step-by-step process showcasing how our robot interacts with the system. This diagram specifically highlights the process of collecting

food from the food counter and delivering it to the user's table while intelligently navigating to avoid both static and dynamic obstacles.

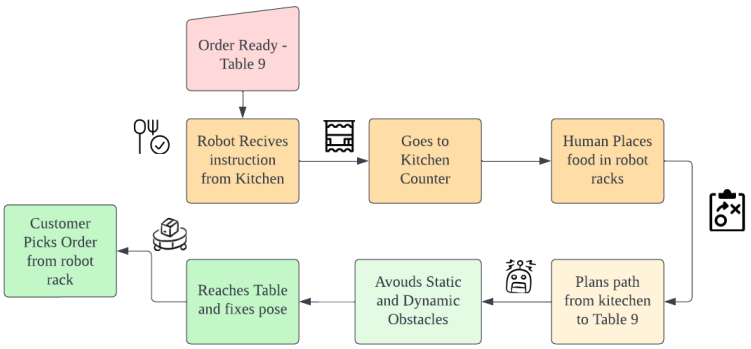

Figure 2: Robot Journey

Key Components:

1. **Food Collection:** The journey commences with the robot's task of collecting the user's food order from the designated food counter. This phase involves the robot's interaction with the food preparation area.

2. **Navigation System:** Our system employs an advanced navigation system that equips the robot with the ability to map the restaurant's layout, recognize static obstacles such as walls and furniture, and detect dynamic obstacles like moving patrons or objects in its path.

3. **Route Planning:** The robot's journey includes route planning, ensuring the most efficient path to reach the user's table, minimizing delivery time and avoiding any encountered obstacles.

4. **Obstacle Avoidance:** The robot's sensors and software enable it to dynamically adjust its path in real-time, avoiding obstacles, ensuring safety, and maintaining a seamless delivery process.

5. **User Interaction:** At the user's table, the robot interacts with the user, delivering the food and possibly offering a user-friendly interface for feedback or additional services.

6. **Return to Base:** After completing the delivery, the robot may return to a designated base or service area for maintenance, charging, or further instructions.

Purpose and Benefits:

The high-level robot journey diagram and description serve as a valuable tool for understanding the intricate process by which our robotic system opMuhammaderates. It provides a visual representation of the robotic interaction, emphasizing its ability to intelligently and safely navigate through the restaurant environment while ensuring timely and accurate food delivery to the user's table.

4.3 Module-Level System Diagram and Description

The system diagram provides an abstract overview of the primary components and their interactions within our project. It illustrates the two main modules that form the core of our system, showcasing their connections and dependencies.

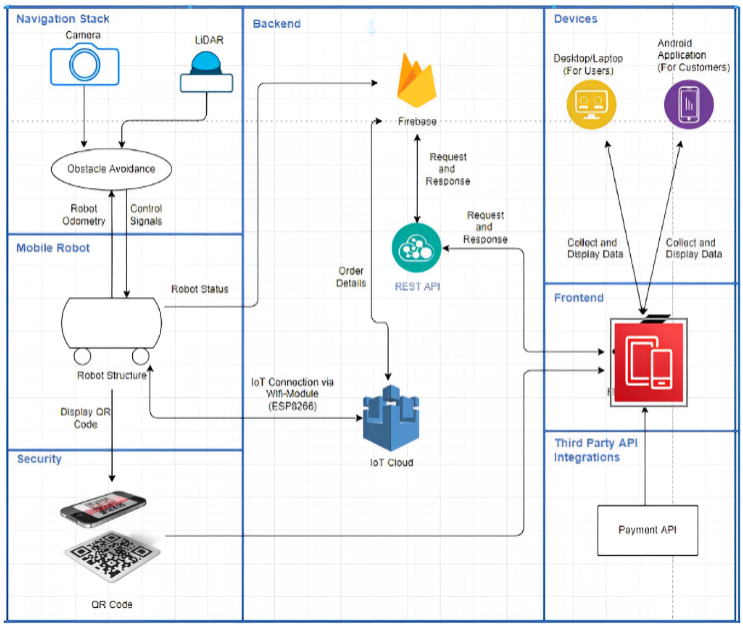

Figure 3: Module-Level Diagram

Module 1: Navigation Module

Description:

The Navigation Module is a critical component of our system responsible for guiding the robot through its environment while ensuring obstacle avoidance. It leverages two key technologies: LIDAR (Light Detection and Ranging) and camera systems. Here are the essential details:

- **LIDAR:** LIDAR technology is employed to create a 360-degree view of the robot's surroundings, detecting both static and dynamic obstacles in real-time. The data obtained from LIDAR assists in building a comprehensive map of the environment.
- **Camera:** A camera system complements LIDAR, providing visual information to enhance obstacle recognition and assist in the robot's navigation.
- **LIDAR-Camera Fusion:** The fusion of LIDAR and camera data is a crucial step in our obstacle avoidance strategy, combining the strengths of both technologies to improve accuracy and reliability.

- **Path Planning Algorithm:** Our Navigation Module incorporates a sophisticated path planning algorithm, which takes input from LIDAR and camera systems to determine the most efficient and obstacle-free route for the robot.

Module 2: Application Back-End

Description:

The Application Back-End serves as the central control hub of our system, facilitating seamless communication and coordination with the robot. It's crucial for enabling fast M2M (Machine-to-Machine) communication, ensuring prompt task assignment without human intervention. Key features include:

- **IoT Connectivity:** The back-end is equipped with IoT (Internet of Things) capabilities, establishing a secure and efficient connection with the robot. This connection enables real-time data exchange and remote control.
- **Wireless Connectivity:** The back-end utilizes wireless technology to maintain a constant link with the robot. This ensures rapid response times and instant task allocation upon order placement.
- **Automated Task Assignment:** As soon as a customer places an order, the Application Back-End seamlessly allocates the relevant tasks to the robot, automating the delivery process without requiring human intervention.

The combined operation of these two modules forms the backbone of our system, ensuring efficient, automated, and obstacle-avoiding navigation while delivering orders to customers.

4.4 Low-Level System Diagram

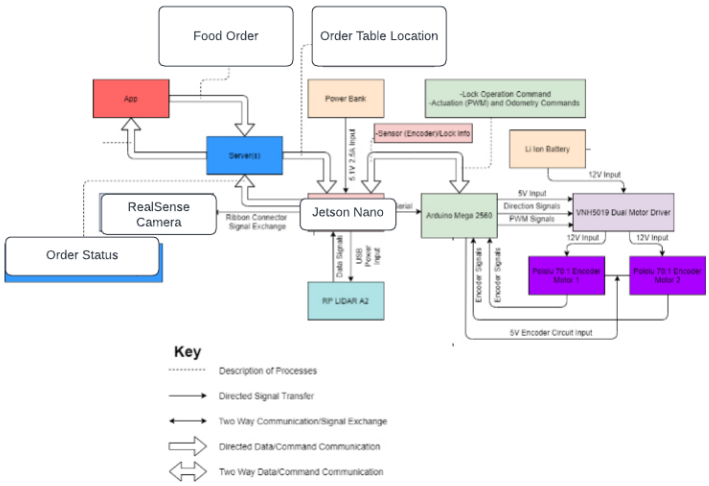

Figure 4: Low-Level Diagram

5 Functional Requirements

This section outlines the various functions and features offered by our system, categorizing them into modules based on their intended purpose, user requirements, or mode of operation, creating a functional hierarchy that might appear as follows:

Web App

- Scanning the QR code should open a designated web page on the user's device.
- The website should provide a user-friendly interface for selecting food items.
- The user should be able to browse a menu and select items they want to order.
- After selecting items, a form should be displayed on the website requesting order information.
- The submitted data, including the user's name, payment information, product quantity, payment option and table number.
- By selecting the table Number in the form, the coordinates of that table (which is hard coded on the map) should be sent to the server.
- The Payment information should be encrypted and securely stored.
- The user should receive confirmation of their order after successful submission.

- In case of any issues with the submission, clear error messages and guidance should be provided.

Web Server

- The web server should be capable of receiving and processing the data submitted from the form.
- It should validate the data and handle payment processing securely.
- The server should record the order and send a confirmation or acknowledgment to the user.
- The system should provide error messages and validation checks to ensure that the user inputs the required information correctly.
- In case of errors, the user should receive feedback and be able to correct their entries.
- Secure communication protocols (e.g., HTTPS) should be used to transmit sensitive data.

Admin Panel

- The Admin Panel should allow administrators to view all booking and user details.
- It should allow the admins to control the robot for indoor mapping.
- It should show the status of the delivery.
- The Admin Panel should require login with the correct credentials for security.

Robot

- By Navigating through the mapped environment, the robot should safely reach the table Number (where the person is sitting) while avoiding both static and dynamic obstacles through appropriate maneuvers.
- By Utilizing the A* algorithm, the robot should plan its path and adapt it as per the prevailing conditions.
- The robot's trajectory should be designed to be nearly seamless and straightforward, ensuring timely deliveries.
- The Environmental friendliness is a core principle, as the robot should be programmed to avoid harming any living beings or structures.
- Concealing its internal circuitry and components within compartments, the robot should maintain an appealing appearance.
- The robot should make a map of the surroundings by utilizing Lidar sensor data, intel Real Sense SR305 Camera input and Google Cartographer's mapping algorithm.
- Appropriate braking system should be established to ensure food or juice remains stable on the robot.

5.1 Use Case Diagram

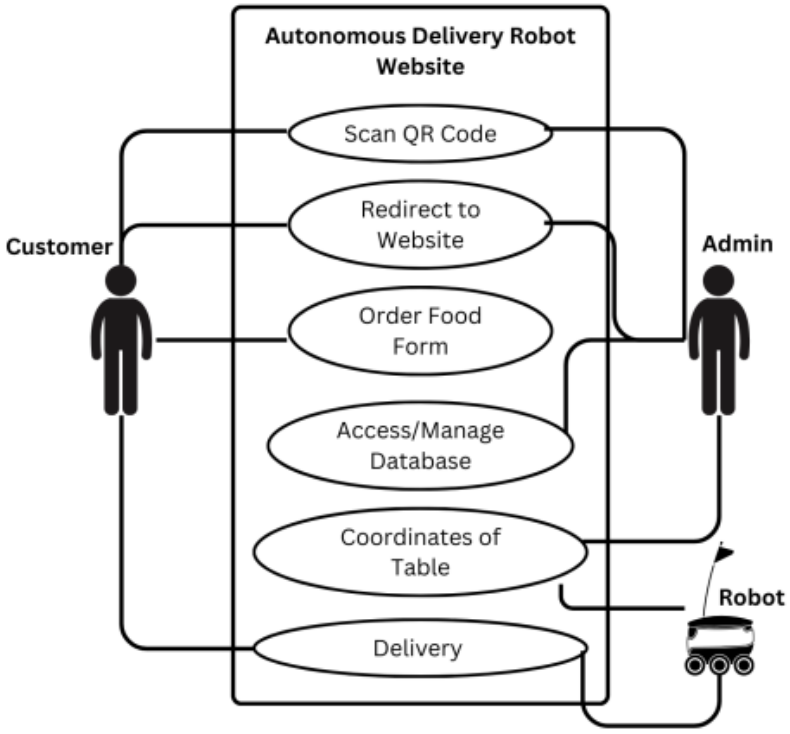

Figure 5: Low-Level Diagram

6 Non-Functional Requirements

Performance:

- The system should provide a fast and responsive user experience, with a web page load time of less than 5 seconds.
- The form submission process should be completed within 5 seconds.
- The robot should be time efficient in performing its task

Security:

- The system should implement strong data encryption (e.g., SSL) to protect user information during transmission.
- User authentication should be required to access and submit data.

- The system must have measures in place to prevent SQL injection and other common web application security vulnerabilities.

Reliability:

- The system should have an uptime of at least 99.9% to ensure it is available to users at all times.
- It should have regular backup and disaster recovery procedures in place to minimize data loss and system downtime.
- The robot should be easily deployable without interfering with any infrastructure of the organization's space.
- The structure of the robot should be strong.

Scalability:

- The web app should be capable enough to handle the load of high number of users without noticeable effects on its performance, navigation or glitches
- The codebase should be scale able so that more features can be added in the app.
- It should be able to accommodate a minimum of 10 concurrent users.
- The robot should scale the payload under its limit.

Usability:

- The user interface should be intuitive and user-friendly, ensuring that users can easily navigate the website and complete the order process.
- The website should be accessible on various devices and screen sizes.
- The robot should be human friendly

Compliance:

- The system should comply with relevant data protection regulations (e.g., GDPR, HIPAA, or other applicable laws).
- It should also adhere to payment processing and security standards (e.g., PCI DSS) if handling payment information.

Adaptibility:

- The robot should have an adaptive speed so that if any object comes within its boundary, it can take appropriate measures.

Data Integrity and Backup:

- The system should regularly back up user data and order information to prevent data loss.
- Data integrity checks should be in place to ensure that submitted data is accurate and complete.

7 Software Design Specification (SDS)

We have discussed the crux of our project in the previous part of our report (SRS document). In this section the important documents have been attached in relation to the design and implementation of the project.

7.1 Software Design

This section provides a concise overview of the server's connection with the robot over the internet, outlining the communication between the robot, the backing database, and the server. Attributes and methods of each class, along with relationships among classes, are presented. Note that these details are tentative, and additional information will be added as the project progresses.

7.1.1 Robot

This module is exclusively related to the robot's functionality. It is designed to send and receive information concerning the user's distance from the robot, estimated time of arrival, current location, and details about the next delivery. The robot will also validate its arrival at the destination and communicate this information with the server.

7.1.2 Central Server

The Central Server module is interconnected with all other modules in the system. It facilitates the exchange of information related to customers, orders, and locations that the robot needs to visit.

7.1.3 Customer

The User module manages customers' details, along with other order-related information.

7.1.4 Order

This module is responsible for sending delivery order details to the server. It also handles current orders and details received from the app.

7.2 UML Diagram

Figure 6 presents the UML diagram representing the relationships and structure of the Swiftbot software design. The diagram illustrates the connections and interactions among different modules.

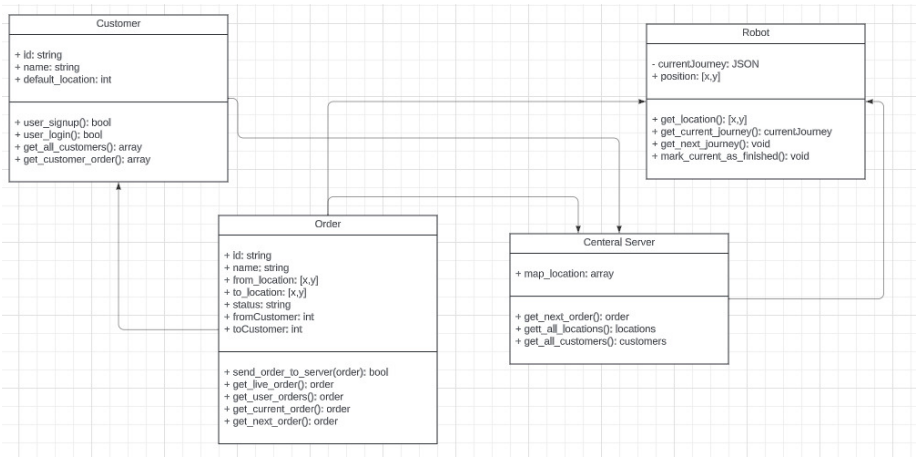

Figure 6: UML Diagram

7.3 Main Loop

Additionally, a Main Loop has been designed to link all the modules together, as explained in previous Section. The program flow diagram is shown below.

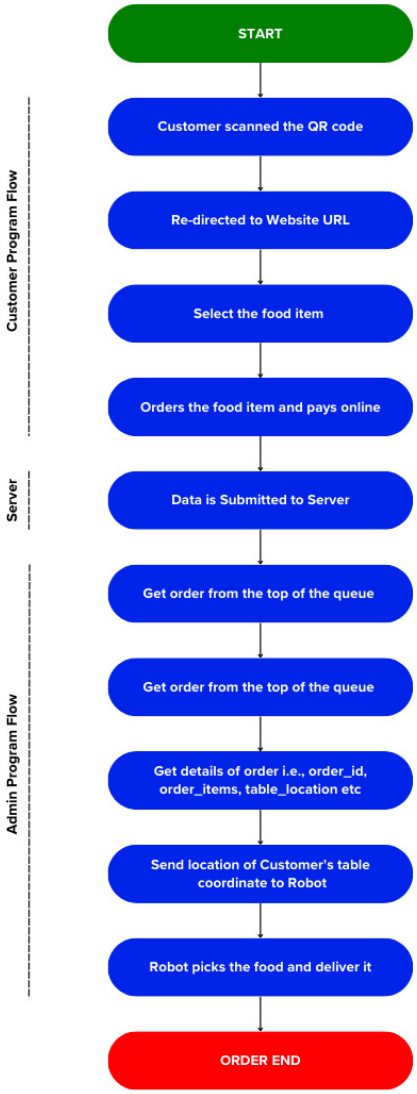

Figure 7: Flow Diagram

This Main Loop orchestrates the interaction and coordination among the various modules, ensuring seamless communication and functioning of the entire system.

7.4 Data Design

This section outlines the structure of the database implemented for persistent data storage in our project. The design is presented as a normalized data model for relational databases. The representation includes entities, attributes, relationships with their cardinalities, and identification of primary and foreign keys. To provide a clear overview of the data maintained, DB Designer was

employed.

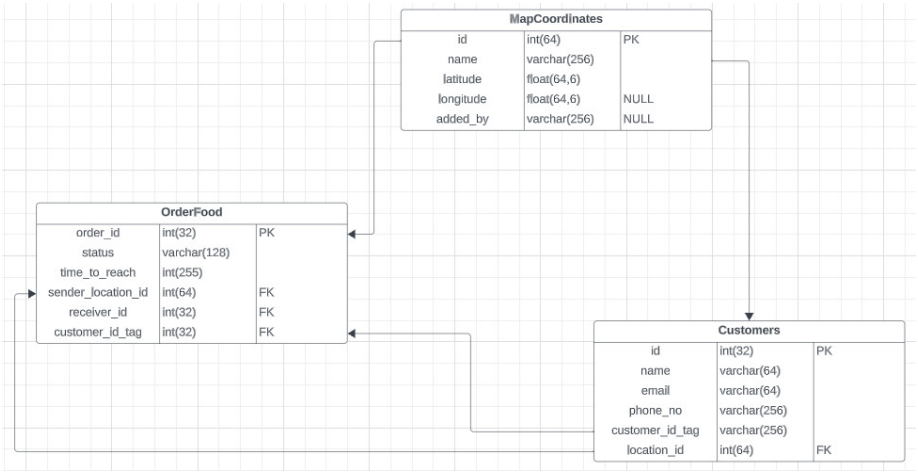

Figure 8: Entity Relation Diagram of the Backing Database

Figure 8 illustrates the Entity Relation Diagram (ERD) of the backing database, showcasing the relationships and structure of the entities. The diagram is instrumental in visualizing the organization of data within the relational database.

7.5 Technical Details

Our project consists of various integrated modules, each serving a specific purpose. In this section, we delve into some of these details to elucidate the objectives of our project.

7.5.1 ROS: Robot Operating System

The Robot Operating System (ROS) is an open-source framework designed for the robotics industry. Operating on the Linux architecture, ROS provides features such as low-level hardware device control, sensor fusion, and robot simulation through tools like Gazebo.

7.5.2 Why ROS?

ROS boasts a substantial community support, regularly updated libraries, and widespread availability, making it convenient for new users. Given the project's reliance on localization and mapping fundamentals, ROS serves as an ideal choice. Additionally, managing data routing between the server and low-level hardware (Arduino) necessitates the use of an embedded system, such as a Raspberry Pi or Jetson Nano.

The decision to load the Ubuntu OS on Raspberry Pi/Jetson Nano was made to enable multi-threading of processes, reducing memory usage. The Ubuntu OS packaged with the ROS framework was installed, and a Catkin workspace was set up. Essential libraries for the project's software base were downloaded following workspace creation.

7.5.3 Fundamental Reasons for Using ROS

1. **Wide Applicability:** ROS is a renowned open-source framework encompassing packages for diverse applications, with pre-built packages for a majority of actuators, sensors, and hardware components, as well as packages for building drivers for components lacking drivers.

2. **Simulation Capabilities:** Complex projects necessitate thorough simulations before execution. ROS includes Rviz and Gazebo modules for creating a customizable virtual environment, facilitating necessary testing of the robot's functions. These simulations also contribute to runtime benefits, with Rviz capable of displaying live maps for SLAM.

3. **Language Compatibility:** ROS offers extensive language compatibility, a valuable trait when diverse programming backgrounds collaborate on large teams. Systems based on ROS can incorporate packages and libraries from different programming languages, fostering collaboration across various platforms and accommodating a wide range of packages written in multiple languages.

7.6 ROS Concepts

ROS is distinguished by certain characteristics and paradigms, encompassing concepts like nodes, topics, publishers, subscribers, messages, and launch files.

7.6.1 Message

Messages are data structures with specific fields that define the type of data (e.g., String, Boolean, Integer) being transferred between nodes.

7.6.2 Node

A ROS node is a process performing a computation, essentially an executable program within the developer's application. Nodes form a graph and communicate through protocols such as topics, services, and actions. Encapsulating computations in nodes reduces code complexity, enhances scalability, and modularizes the application. Nodes improve fault tolerance by communicating through ROS without direct links, ensuring that if one node crashes, others remain unaffected.

7.6.3 Topic

ROS facilitates information exchange between nodes through buses known as topics. Topics determine the type of data transmitted between nodes, creating a decoupling between data production and usage. Nodes publish to a specific topic when sending data and subscribe to relevant topics when receiving data, unaware of the specific nodes involved in communication.

7.6.4 Publisher and Subscriber

The publisher/subscriber (pub/sub) mechanism is the core paradigm in ROS, enabling data exchange between nodes. This mechanism involves subscribing to topics to receive data and publishing to topics to send data. The pub/sub model supports many-to-many communication, allowing multiple publishers and subscribers (transmitting and receiving nodes) connected by topics.

7.6.5 Launch File

Launch files offer a convenient way to initiate multiple nodes and initialize parameters efficiently. These files typically contain code storing application parameters (e.g., PID settings for the robot) and code for initializing ROS packages.

7.7 ROS Setup

The project is developed on ROS Melodic, chosen for its stability and Long Term Stable (LTS) version availability. Despite newer ROS versions, the time taken for package migration makes the LTS version more reliable than the latest unstable release.

7.8 Desktop/Laptop Setup

ROS, originally designed for Linux, is still transitioning to full compatibility with Windows. For optimal functionalities and stability, Debian-based environments like Ubuntu are preferred. The team utilized Ubuntu 18.04 LTS, compatible with ROS Melodic. Installation instructions for desktop/laptop machines can be found on the ROS Wiki site.

7.9 Raspberry Pi Setup

The project involves multiple stationary and mobile nodes. Instead of installing Ubuntu directly on Raspberry Pi, the team opted for a pre-compiled Ubuntu and ROS image provided by Ubiquity Robotics. The image, along with installation instructions, can be obtained from the Ubiquity Robotics site. This image can be directly burned onto an SD card, bypassing the need for manual installation on desktop/laptop.

7.10 Catkin Workspace Setup

Catkin workspace serves as the build system for ROS. Targets in ROS can be libraries, packages, or dynamically created scripts. The build system handles tasks like generating targets from all program scripts, allowing end-users to utilize targets without concerns about underlying processes or libraries used. Further details about the Catkin workspace can be found on the ROS Wiki site.

Once ROS is installed, the Catkin workspace needs to be built and initialized for first use. Tutorials for setting up and initializing the Catkin workspace can be found on the ROS Wiki site.

7.11 ROS Packages Used

This section provides a brief overview of the ROS packages utilized in the project. All open-source third-party packages can be installed into ROS by cloning their GitHub repositories to the Catkin workspace's `./src` folder and rebuilding the workspace using the `catkin_make` command in the CLI.

7.12 Rosserial

The Rosserial package establishes communication between the Raspberry Pi and the Arduino. It transfers PWM values for motors and lock mechanism commands. The Arduino subscribes to ROS

topics and publishes encoder values back to the Pi. The package uses the Universal Asynchronous Receiver Transceiver (UART) connection with a baud rate of 115200. More details can be found on the ROS Wiki site.

7.13 rplidar ros

The `rplidar ros` package channels the RP LIDAR sensor data to ROS, publishing scan data on the `/scan` topic. Additional information about this package is available on the ROS Wiki site .

7.14 Navigation Stack

The Navigation Stack navigates robot control using odometry and a pre-generated map created with the Cartographer library. Rviz sets the current location, pose, and goal position of the robot. The Navigation Stack employs the `amcl` node for localization, the `differential_drive` node for speed and direction, and the `map_server` node to subscribe to the map. The commands are translated into a trajectory and transmitted to the robot using the `cmd_vel` topic. Further details can be found on the ROS Wiki site.

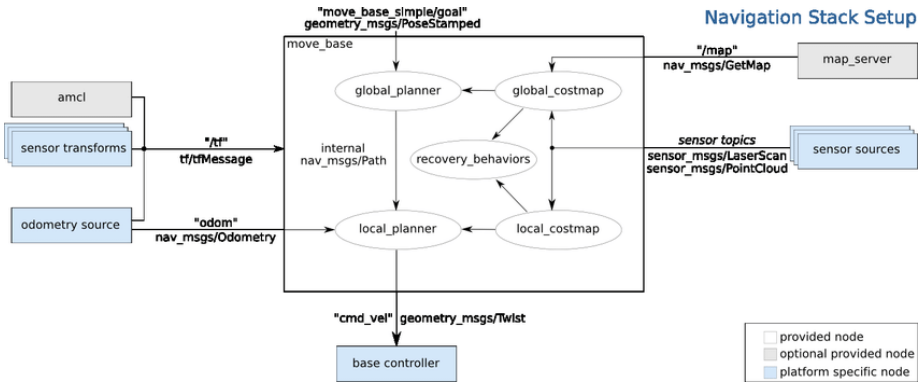

Figure 9: The ROS Navigation Stack

7.15 Gmapping

Gmapping is a ROS package providing the OpenSlam Gmapping algorithm, constructing a 2D occupancy grid map using laser data and odometry information. More information is available on the ROS Wiki site and the OpenSlam site .

7.16 tf

To reuse software components and integrate them better, developers require a shared convention for coordinate frames. The `tf` package enables users to track multiple coordinate frames over time, maintaining relationships in a tree structure. It allows transformation of vectors and points between any two coordinate frames at any time. Further information can be found on the ROS website and the ROS Wiki site.

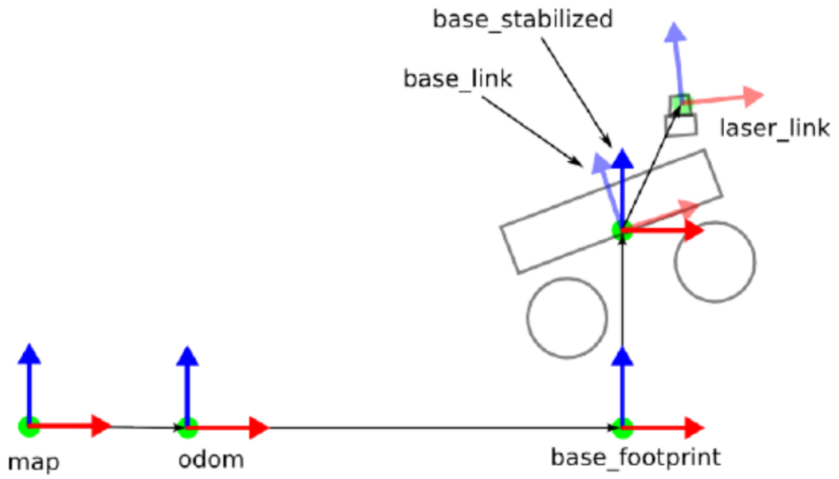

Figure 10: The `tf` Transformed Coordinate Frame for a Differential Drive Robot

`tf` manages various transforms necessary for a robot, defining standard terms like base frame and laser frame. The robot's base frame is also known as the base link. The laser frame represents the LIDAR's coordinate frame. Continuous publishing of transformations from base link to laser link is handled by `tf`. The package also maintains dynamic transforms, such as the `odom` to `base_link` transform, representing the robot base frame's position with respect to the starting position (odometry). A visual representation of the system transform tree is shown in Figure 10.

7.17 Transforms in ROS

With ROS, both static and dynamic transforms can be published. An example of a static transform is the `base_link` → `laser_link` transform. The laser is rigidly fixed to the robot, making its transform static with respect to the robot. On the other hand, the `odom` → `base_link` transform is dynamic since it changes as the robot moves, and these changes are published by the `amcl` node.

7.18 Differential Drive Package

The differential drive package provides tools to interface a differential drive wheeled mobile robot with the ROS Navigation Stack. Its purpose is to implement a differential drive controller independent of specific hardware, such as a particular microcontroller unit used for robot implementation.

This package takes messages from the Navigation Stack and converts them into left wheel and right wheel messages, serving as voltage levels transferred to the motor driver in the form of PWM signals. Additionally, it receives encoder signals from the hardware, generating messages for odometry and location referencing in the form of `tf` transform messages. To maintain hardware abstraction, speed messages are communicated in the form of x-axis meters/second and z-axis radians/second.

The package includes a PID controller that utilizes feedback from rotary encoders attached to the motors, forming a closed-loop control system to control wheel velocities. Two PID controllers are instantiated, corresponding to each wheel. The package also provides a node for odometry calculations through the encoders of the motors. Ticks detected by each encoder are counted and published by the low-level controller (Arduino) to the differential drive package, using the serial communication package, Rosserial. The odometry node subscribes to these messages and calculates the odometry values of the robot.