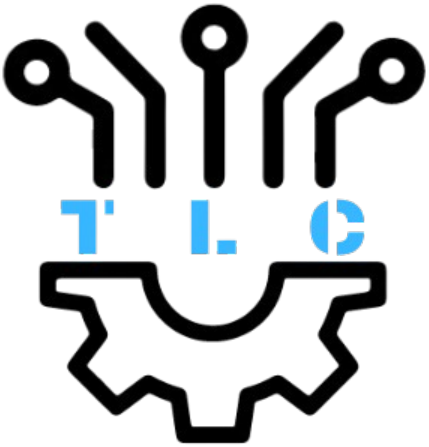
Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2025



Team TLC
Roam_Bot

Abubakar Kassim
Andrew Howard
Christopher Davis
Madison Gage
Raya Sultan

Revision History

Revision	Date	Author(s)	Description
0.1	2.4.2025	CC	Document Creation
0.2	2.25.2025	AH	Added Introduction Information
0.3	2.27.2025	CD, RS, AK, AH	Added Layer Information
1.0	2.28.2025	CD, RS, MG, AK, AH	Official Release
1.1	4.30.2025	AK	Updated Layer Information
2.0	4.30.2025	CD, RS, MG, AK, AH	Official Release

CONTENTS

List of Figures

1 INTRODUCTION

The RoamBot should assist with learning to implement autonomous system software through real test simulations. This will be achieved using actual robotic systems. The overall design is controlled by a Raspberry Pi, which acts as the host connecting the different components of the rover. The first section describes the rover's motion and functionality. The second section entails the interface which allows users to upload their algorithm or manually control the rover. The third section is about the path-finding function, which dictates where the rover can move with the use of LIDAR and other sensors.

2 SYSTEM OVERVIEW

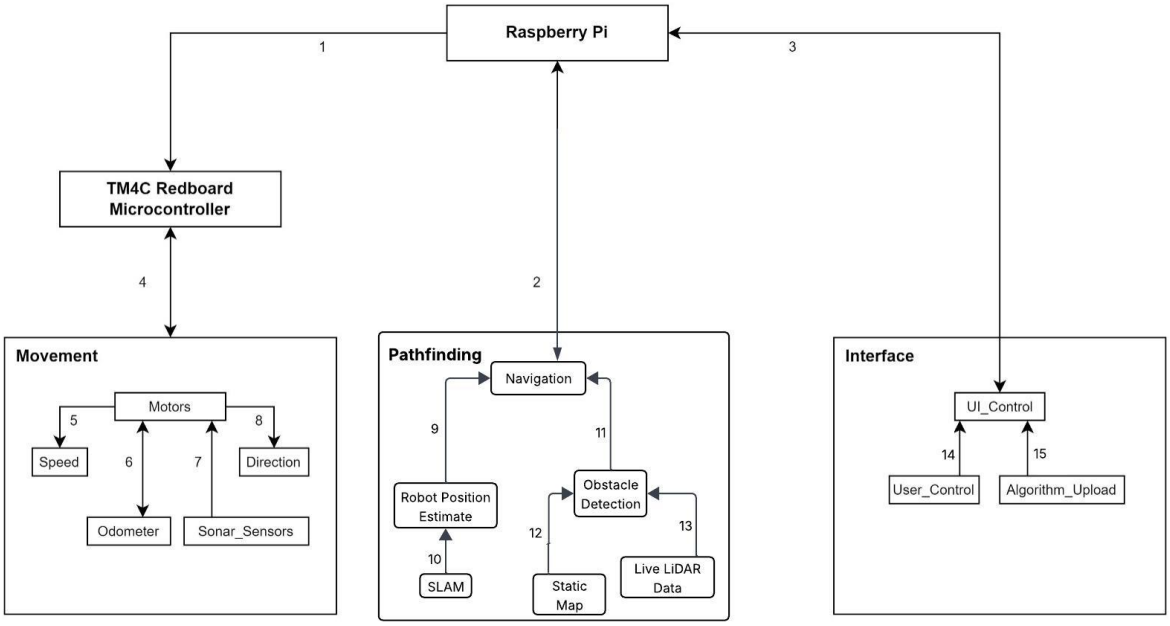


Figure 1: Data Flow Diagram

The architecture of the RoamBot system is structured into three distinct layers: Movement, Path Finding, and Interface, each serving a critical role in the rover's functionality. The Movement Layer is responsible for controlling the rover's motion, including speed, direction, and crash prevention. It utilizes UART to control speed and direction and GPIO for safety features like stopping when obstacles are detected. The Path Finding Layer enables autonomous navigation through LIDAR-based mapping and obstacle detection, using GPIO communication with a TM4C microcontroller to control and guide the movement. Lastly, the Interface Layer acts as the bridge between the user and the system, handling user command interactions, including direct control of the rover and algorithm uploads.

These layers work together to ensure efficient and autonomous operation of the RoamBot. The Movement Layer executes physical actions based on inputs from the Path Finding Layer, which interprets environmental data to determine safe navigation paths. The Interface Layer facilitates user interaction, allowing for both manual and automated control. This structured approach ensures modularity, making each layer functionally independent while maintaining well-defined communication interfaces for seamless operation.

3 Movement Layer Subsystems

The Movement Subsystem controls the rover's motion by managing speed, direction, and collision avoidance. It receives navigation commands from the Raspberry Pi and sends signals for motor control, ensuring responsive navigation.

3.1 Microcontroller

The microcontroller [2] is a hardware component responsible for processing commands and controlling the rover's movement. It manages communication between the control systems and the motor drivers, sending directional and speed commands. Additionally, it handles sensor inputs for obstacle detection and emergency stops, ensuring safe and efficient operation.

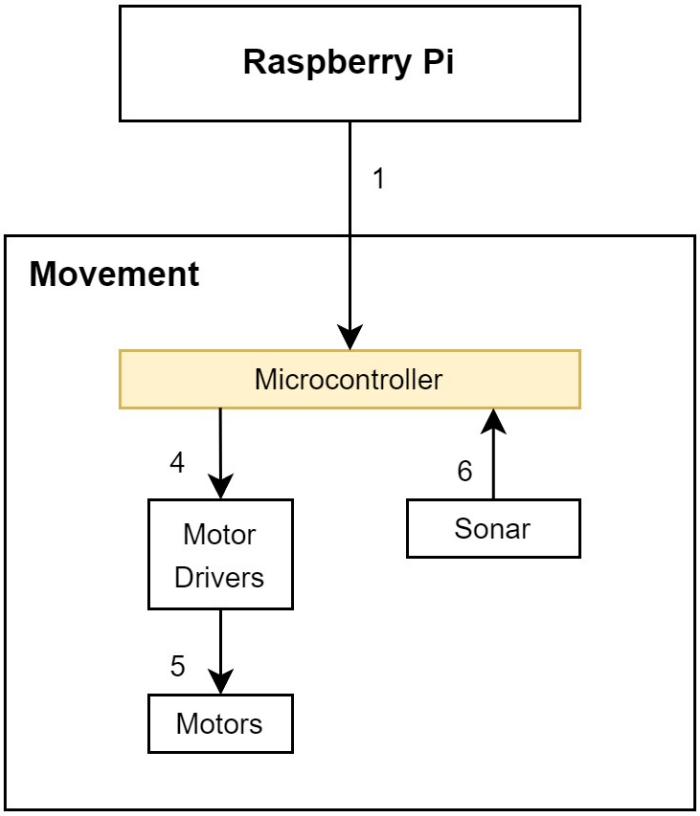


Figure 2: Movement Layer - Microcontroller Subsystem

3.1.1 Subsystem Hardware

- TM4C RedBoard - A microcontroller with built-in UART and GPIO modules, responsible for processing motor control commands and handling sensor inputs (Figure 15).

3.1.2 Subsystem Software Dependencies

TI Driver Library (Tivaware) - Provides low-level control over the TM4C RedBoard's peripherals, including UART and GPIO.

3.1.3 Subsystem Programming Languages

The TM4C RedBoard firmware uses C to implement the communication protocols and to control movement functions.

3.1.4 Subsystem Data Structures

- (4x) 1-bit input GPIO control signals, one for each movement direction.
- 8-bit control packets are sent via UART to regulate each motor. Each packet consists of a single byte representing motor-specific commands such as speed and direction. Transmission occurs at 9600 baud with 5V, 8N1 settings, controlling one motor per packet.

3.1.5 Subsystem Data Processing

The microcontroller polls movement direction inputs every 100ms, capturing the required state and transmitting four motor control commands (one per motor).

3.2 Motor Drivers

The motor drivers [1] are hardware components that regulate power to the DC motors based on control packets received from the TM4C RedBoard. These are Sabertooth controllers that process incoming commands to adjust motor speed and direction. They ensure efficient and precise movement of the rover.

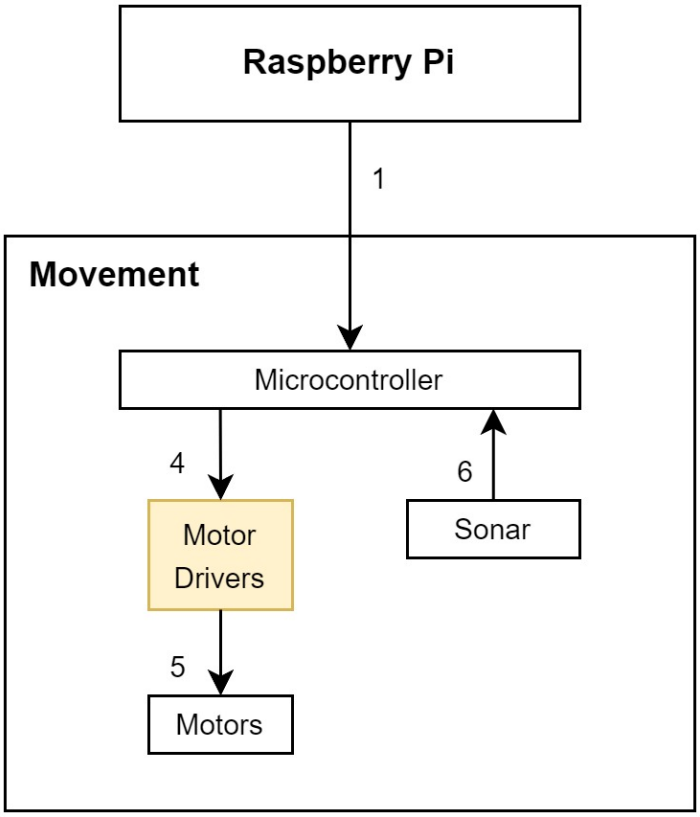


Figure 3: Movement Layer - Motor Drivers Subsystem

3.2.1 Subsystem Hardware

- Two Sabertooth 2x25 Motor Controllers - High-performance dual motor drivers that receive UART commands from the TM4C RedBoard. Each controller independently drives two motors, enabling precise movement control (Figure 16).

3.2.2 Subsystem Data Structures

- 8-bit control packets regulate motor operations. Each packet consists of a single byte representing speed and direction commands. The packets are received over UART at 9600 baud with 3.3V, 8N1 (1 start bit, 8 data bits, 1 stop bit) settings, controlling one motor per packet.

3.2.3 Subsystem Data Processing

Each Sabertooth controller processes incoming UART packets every 100ms. Two packets (one per motor) are received per controller, spaced 50us apart to ensure proper sequencing and avoid command overlap.

3.3 MOTORS

The motors are the primary actuators responsible for the physical movement of the rover. These are high-performance DC motors controlled by the motor drivers to achieve the desired speed and direction. The motors receive regulated power from the motor drivers, ensuring smooth movement.

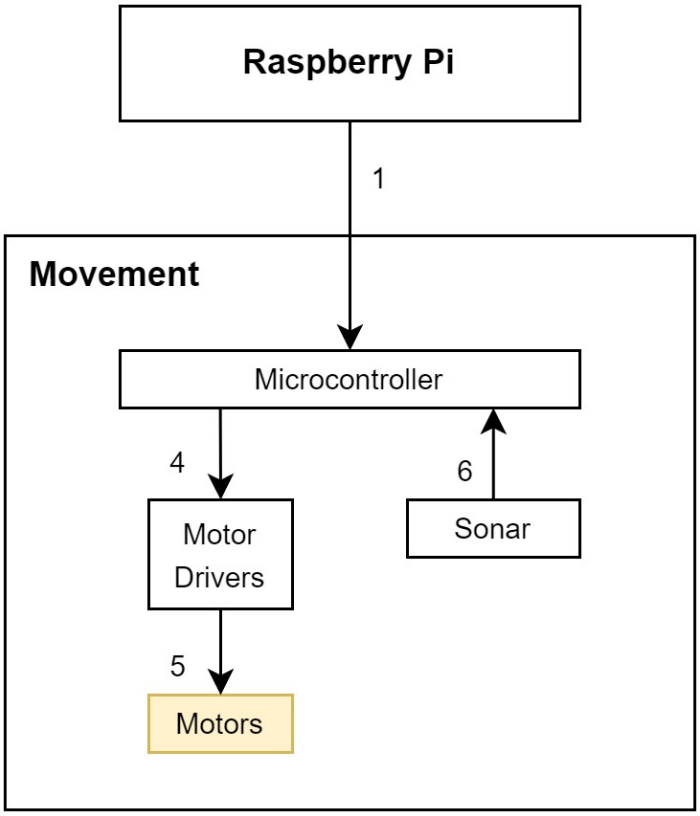


Figure 4: Movement Layer - Motors Subsystem

3.3.1 SUBSYSTEM HARDWARE

- Four DC Motors - Generate the mechanical power required for rover movement. The motors are grouped into front and rear pairs, each controlled independently by the motor drivers to regulate speed and direction.

3.4 SONAR

The Sonar subsystem utilizes ultrasonic sensors to detect obstacles in the rover's path. The system works by emitting sound waves and measuring the time it takes for the waves to bounce back after hitting an obstacle. The data is processed to determine the distance to the nearest object, enabling the rover to take last-minute actions to avoid collisions.

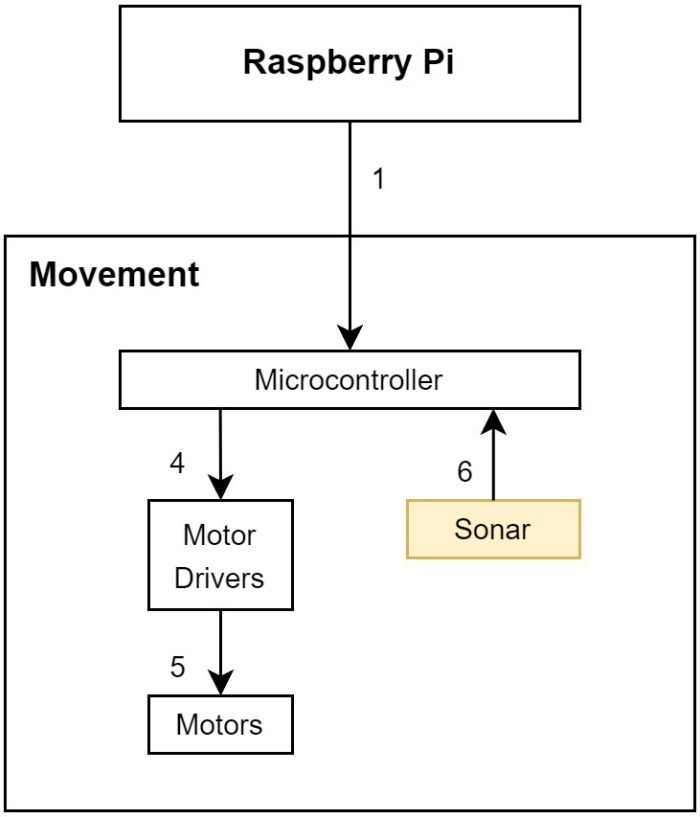


Figure 5: Movement Layer - Sonar Subsystem

3.4.1 SUBSYSTEM HARDWARE

- Ultrasonic Distance Sensor (HC-SR04) - Commonly used sensors that emit sound waves and listen for the echo, allowing for precise distance measurements to obstacles in the rover's path.

3.4.2 SUBSYSTEM DATA STRUCTURES

- 8-bit distance value (in centimeters) transmitted from the sensor to the microcontroller. This value represents the distance to the nearest obstacle detected by the sonar.

3.4.3 SUBSYSTEM DATA PROCESSING

The microcontroller sends a 10us pulse to the sensor's trigger pin, which causes the sensor to emit an ultrasonic wave. The time taken for the echo to return is measured to compute the distance. The system continuously polls the sensor at regular intervals (every 100ms) to update the obstacle detection distance. If an obstacle is detected within a defined threshold (50 cm), the rover will stop.

4 Pathfinding Layer Subsystems

The Pathfinding Subsystem controls the rover's movement, by utilizing the rover's LIDAR to create a map of the terrain. The subsystem will than create a costmap grid, and than applying an A* pathfinding algorithm to find the most efficient and safest path.

4.1 Navigation

The Navigation system is a core component responsible for guiding the rover through its environment safely and efficiently. It combines sensor input, mapping, localization, and path planning to enable autonomous movement. Using real-time data from perception subsystems, the navigation system constructs a map of the surroundings, determines the rover's position within that map, and plans feasible routes to target destinations. It continuously updates the robot's trajectory to avoid obstacles and adapt to changes in the environment, ensuring smooth and collision-free traversal. This system is tightly integrated with motion control and localization components to maintain accurate positioning and responsive movement.

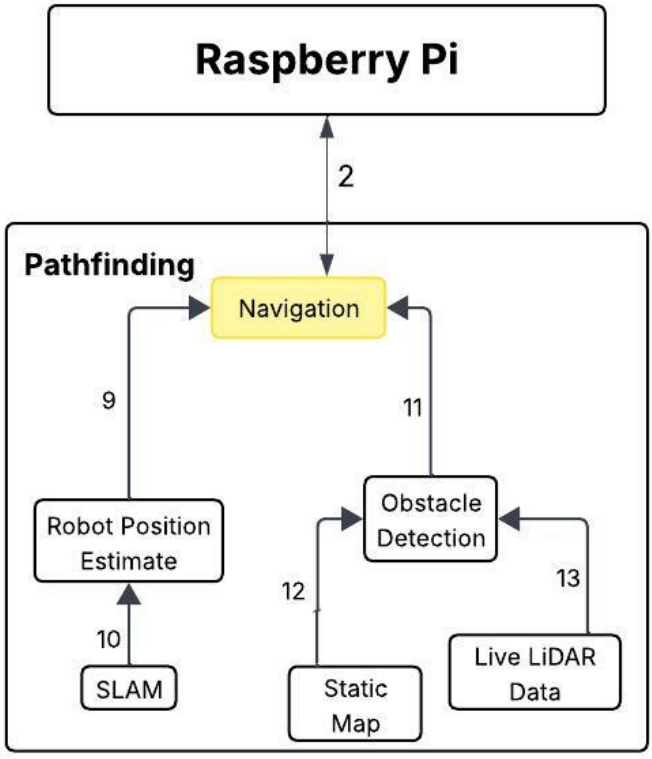


Figure 6: Pathfinding Layer - LIDAR Subsystem

4.1.1 Subsystem Hardware

Slamtec RPLIDAR A1 - 360-degree laser range scanner. With a maximum range of 100 meters and a sampling frequency of a little over 2500 samples a second, measures distances and angles to create a 2D cloud point representation of the surroundings.

4.1.2 Subsystem Operating System

The navigation subsystem will be run on ROS2 Jazzy which requires Ubuntu 24.04 (Noble Numbat) or higher for compatibility.

4.1.3 Subsystem Dependencies

The Navigation Subsystem uses Nav2 and ROS2 to develop the pathfinding algorithm and design the path for the rover to follow, and ROS2 to manage the robot's movements.

4.1.4 Subsystem Programming Languages

Nav2 is primarily implemented in C++ to ensure optimal compatibility with ROS2, which is also C++-based. However, it also provides a Python API for additional flexibility.

4.1.5 Subsystem Data Structures

The pathfinding subsystem employs the A* algorithm due to its ability to determine the most optimal path while maintaining lower computational overhead compared to alternative pathfinding methods.

4.1.6 Subsystem Data Processing

The pathfinding subsystem employs the A* algorithm due to its ability to determine the most optimal path while maintaining lower computational overhead compared to alternative pathfinding methods. (such as Djikstra, BFS)

4.2 Obstacle Detection

The Obstacle Detection subsystem leverages the costmap generated by the Grid Management Subsystem to assess the robot's environment. It assigns a numerical value to each grid cell based on the proximity of surrounding objects, with higher values indicating closer objects and lower values representing areas free from obstacles.

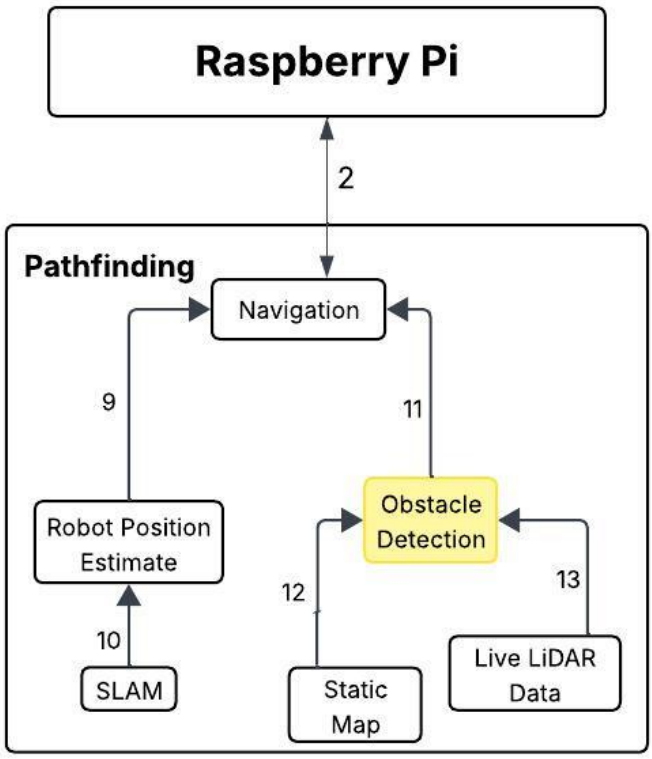


Figure 7: Pathfinding Layer - Navigation Subsystem

4.2.1 Subsystem Operating System

ROS2 Jazzy requires Ubuntu 24.04 (Noble Numbat) or higher for compatibility.

4.2.2 Subsystem Software Dependencies

The Obstacle Detection Subsystem uses Nav2 to develop the pathfinding algorithm and design the path for the rover to follow, and ROS2 to manage the robot's movements.

4.2.3 Subsystem Programming Languages

Nav2 is primarily in C++, so it can better configure with ROS2 which is also in C++, but Nav2 does allow for a Python API.

4.3 Static Map

The Static Map subsystem provides foundational spatial information for the rover's autonomous navigation. It consists of a pre-built occupancy grid defines known obstacles and traversable areas within a fixed environment, creating a costmap. In a ROS 2 and Nav2-based system, this map is loaded at runtime and used by the global planner to calculate optimal paths from the rover's current position to a designated goal. The static map enables efficient pathfinding by offering a reliable reference frame for localization and route generation.

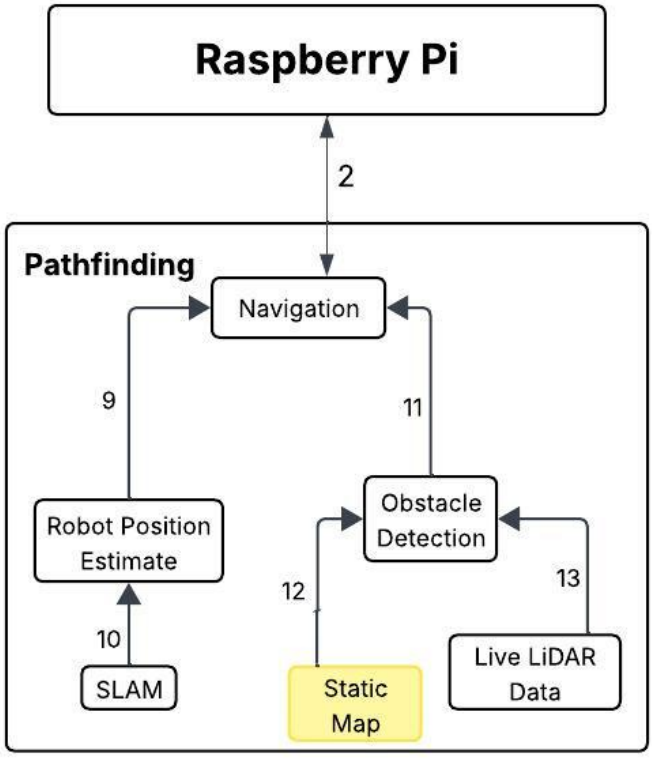


Figure 8: Pathfinding Layer - Static Map Subsystem

4.3.1 Subsystem Operating System

Our Roam_Bot utilizes a Slamtec RPLIDAR A1 - 360 Laser Range Scanner to create the static map

4.3.2 Subsystem Software Dependencies

The Static Map Subsystem uses Nav2 to to create the static map.

4.4 Live LiDAR Data

The Live LiDAR Data subsystem provides real-time environmental awareness for the rover's navigation stack. As the LIDAR sensor continuously scans the surroundings, it generates a dynamic stream of range measurements that are processed into a real-time occupancy grid.

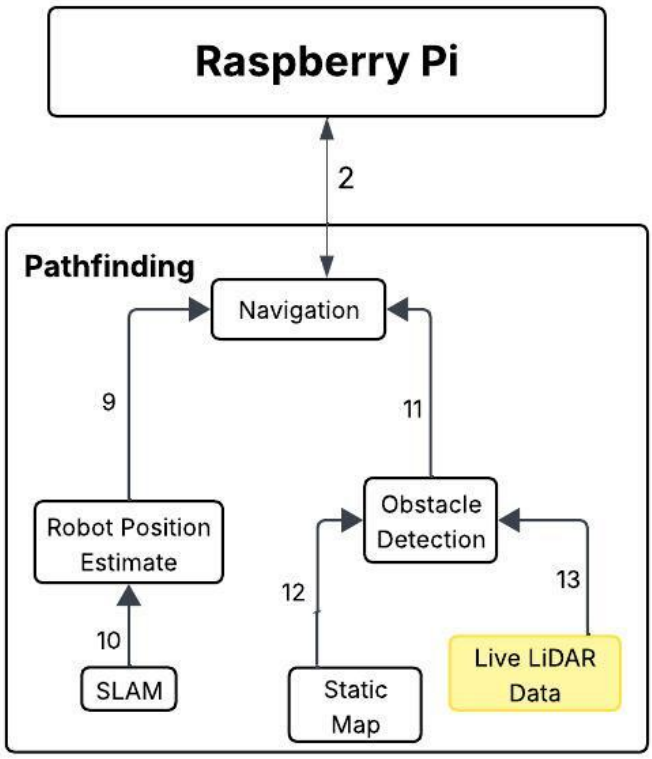


Figure 9: Pathfinding Layer - Live LiDAR Data Subsystem

4.4.1 Subsystem Hardware

Our Roam_Bot utilizes a Slamtec RPLIDAR A1 - 360 Laser Range Scanner.

4.4.2 Subsystem Operating System

ROS2 Jazzy requires Ubuntu 24.04 (Noble Numbat) or higher for compatibility.

4.5 Robot Position Estimate

The robot position estimate subsystem is a software component responsible for determining the rover's location and orientation within its environment. It operates as part of the localization layer and integrates data from odometry, and LIDAR. The subsystem continuously fuses sensor inputs using probabilistic filtering techniques to maintain a stable and consistent position estimate, which is critical for navigation, path planning, and obstacle avoidance.

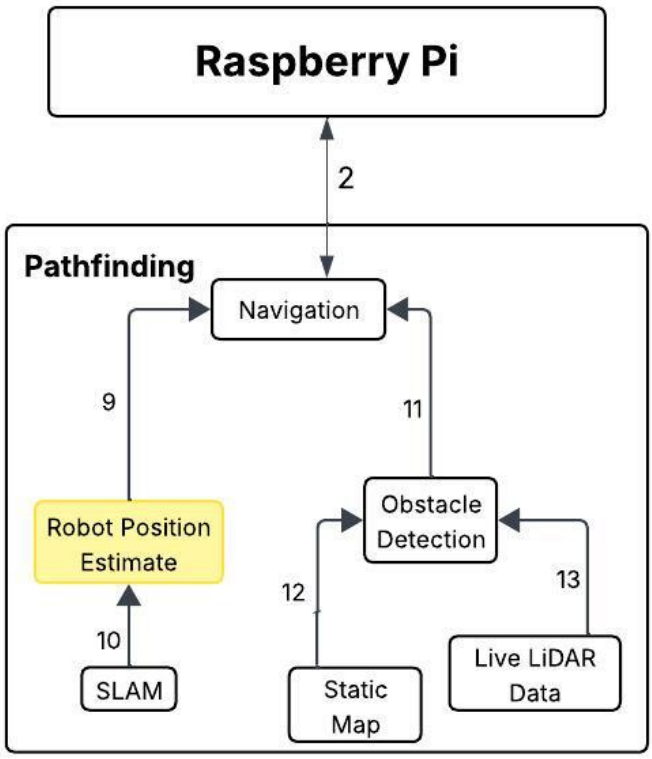


Figure 10: Robot Position Estimate diagram

4.5.1 Subsystem Hardware

To create a position estimate for our Roam_Bot we utilize our Slamtec RPLIDAR A1.

4.5.2 Subsystem Operating System

ROS2 Jazzy requires Ubuntu 24.04 (Noble Numbat) or higher for compatibility.

4.5.3 Subsystem Software Dependencies

The Robot Position Estimate is calculated using ROS2 Jazzy and Nav2

4.5.4 Subsystem Programming Languages

the Robot Position Estimate is programmed in C++.

4.6 SLAM

The SLAM (Simultaneous Localization and Mapping) subsystem is a software module that enables the rover to navigate unknown environments by building a map while simultaneously estimating its position within it. As the robot moves, SLAM compares incoming sensor data to previous scans to update both its map and its own position.

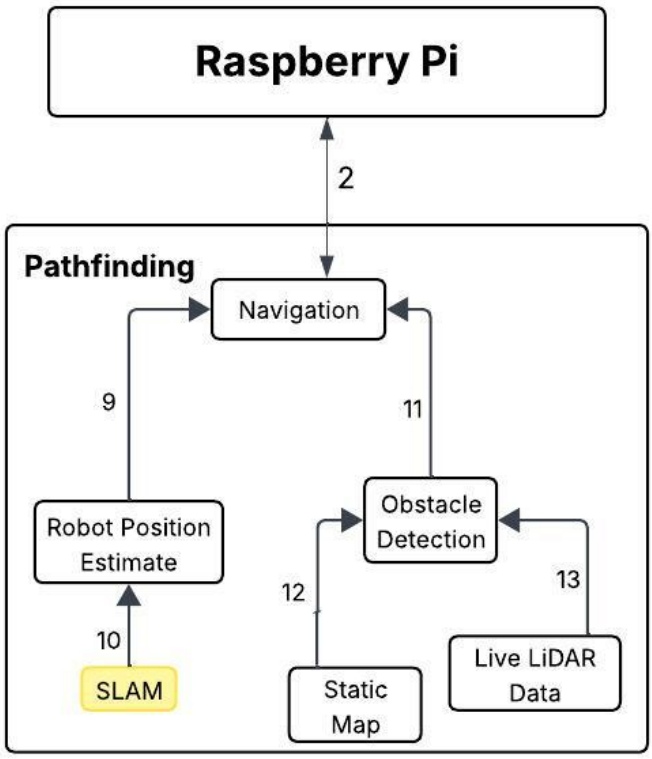


Figure 11: SLAM diagram

4.6.1 Subsystem Hardware

SLAM relies on the SlamTec Lidar to derive the robot position estimate and map the floor,

4.6.2 Subsystem Operating System

ROS2 Jazzy requires Ubuntu 24.04 (Noble Numbat) or higher for compatibility.

4.6.3 Subsystem Software Dependencies

SLAM is calculated using ROS2 Jazzy and Nav2

4.6.4 Subsystem Programming Languages

Nav2 is primarily in C++, so it can better configure with ROS2 which is also in C++, but Nav2 does allow for a Python API.

5 Interface Layer Subsystems

The interface subsystems of the RoamBot are responsible for processing user input and translating it into commands for the rover. These subsystems are implemented using C, C++, and Python and operate on the Raspberry Pi 5.

5.1 Layer Hardware

The Raspberry Pi 5 serves as the core interface hardware for the RoamBot, providing robust computational power with its quad-core ARM Cortex-A76 processor and 16GB of LPDDR4x RAM. It efficiently handles real-time pathfinding and decision-making processes. Additionally, its USB 3.0 ports and GPIO pins allow for direct connection to sensors and motor controllers.

5.2 Layer Operating System

The operating system used for the interface layer is Ubuntu 24.04, providing a stable environment for development and execution.

5.3 Layer Software Dependencies

The interface subsystems rely on the ROS2 framework for robotic control. These libraries are utilized for implementing communication protocols, control algorithms, and user input processing.

5.3.1 Layer Programming Languages

This layer is implemented using C, C++, and Python to efficiently process user commands and algorithm decisions and communicate with the motor control system.

5.4 UI CONTROL

The UI control subsystem serves as the top-level software interface for managing user interactions. It enables switching between direct user control and automated algorithmic control.

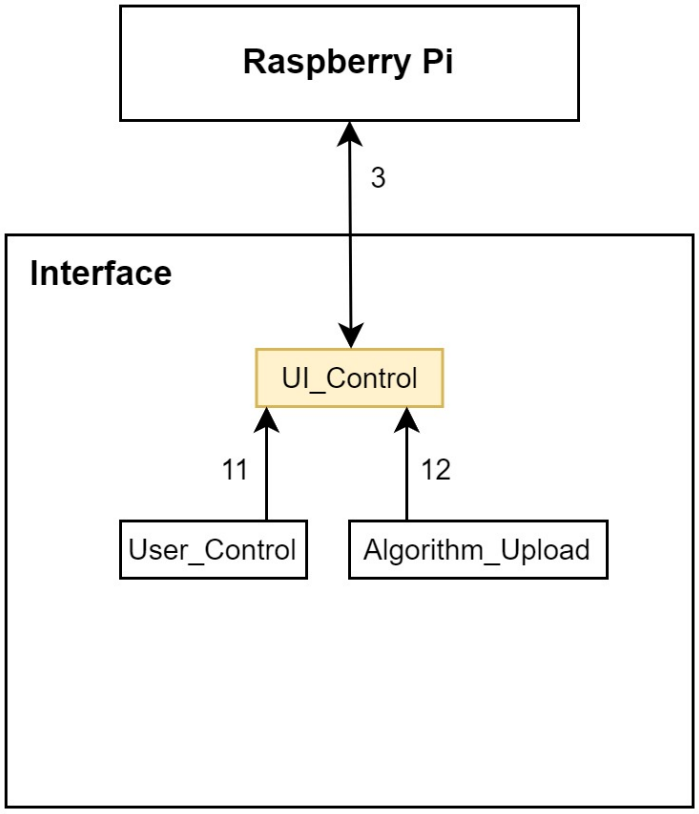


Figure 12: Interface Layer - UI Control Subsystem

5.4.1 SUBSYSTEM DATA STRUCTURES

The subsystem uses flag-based data structures to manage control mode status, with flags indicating whether the rover is in manual or autonomous mode. These flags are checked in real-time to determine which set of functions or commands should be executed.

5.4.2 SUBSYSTEM DATA PROCESSING

The UI control subsystem processes incoming user commands to switch between manual and automated modes by interpreting the mode. When a switch occurs, the system either routes input to the manual control system, allowing the user to directly guide the rover, or to the autonomous system, where the rover executes pre-programmed algorithms.

5.5 User Control

The User Control subsystem allows direct manipulation of the RoamBot's movement in real-time. It processes immediate user inputs, and translates the input into motion instructions. This subsystem ensures that users have precise control over speed and direction, allowing manual operation when needed.

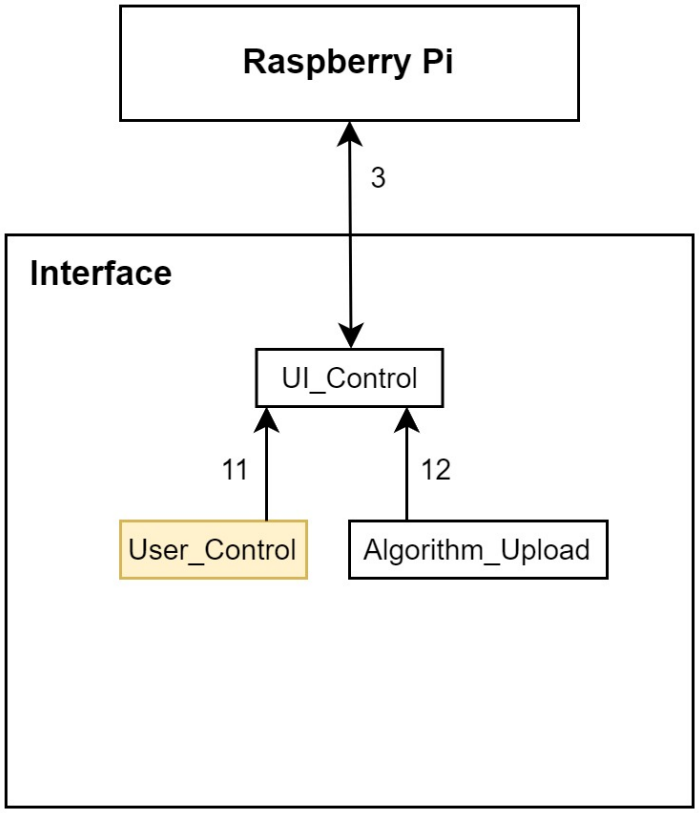


Figure 13: Interface Layer - User Control Subsystem

5.5.1 Subsystem Programming Languages

This subsystem is implemented using C, C++, and Python to efficiently process user commands and communicate with the motor control system.

5.5.2 Subsystem Data Processing

User inputs are converted into structured motion commands that control the rover's speed and direction. These commands are processed in real time to ensure responsiveness.

5.6 Algorithm Upload

The Algorithm Upload subsystem allows users to upload pre-programmed pathfinding algorithms to enable autonomous navigation. It processes these algorithms and integrates them into the rover's control system, ensuring that the RoamBot can follow predefined paths without manual intervention.

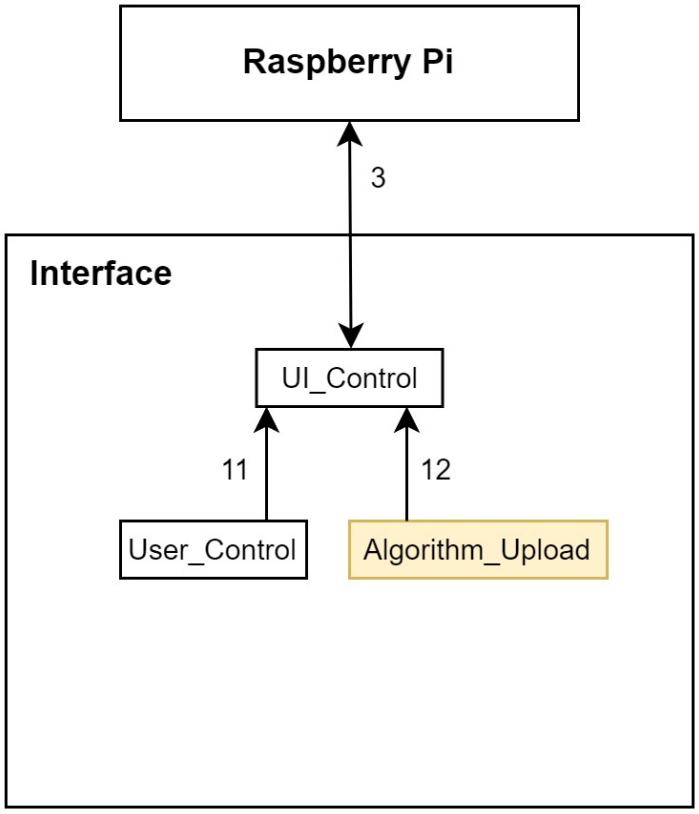


Figure 14: Interface Layer - Algorithm Upload Subsystem

5.6.1 Subsystem Data Structures

The Algorithm Upload subsystem uses data structures like pathfinding grids or graphs to represent the environment, with nodes storing position and cost values values. The resulting path is stored as a sequence of waypoints.

5.6.2 Subsystem Data Processing

Data processing involves parsing the algorithm, running pathfinding algorithms like A* to generate a path, and converting it into motion commands. These commands are then sent as ROS2 messages to the motor control system for autonomous navigation.

6 APPENDIX

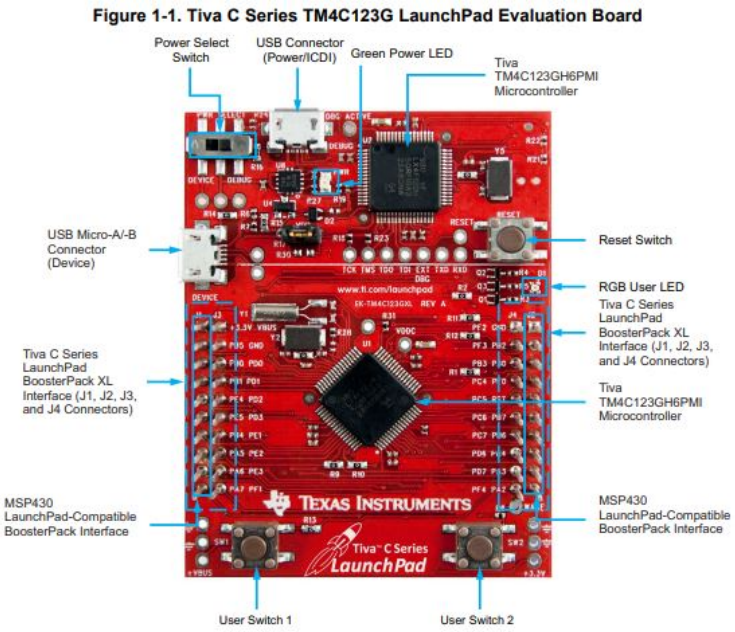


Figure 15: Movement Layer - Microcontroller Circuit

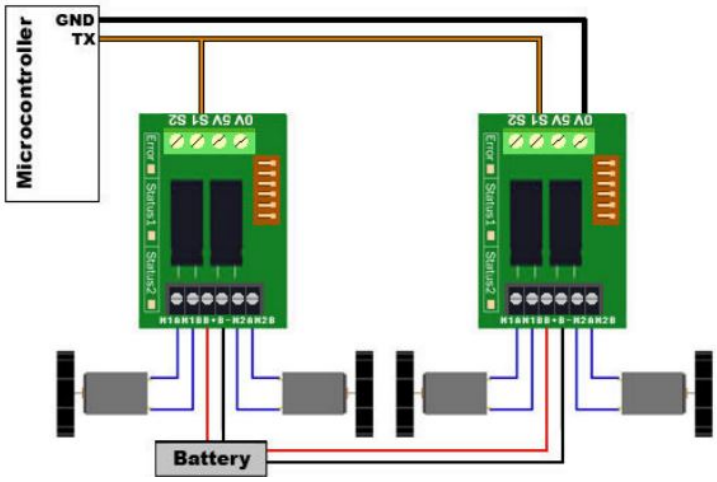


Figure 16: Movement Layer - Motor Drivers Circuit

REFERENCES

[1] Dimension Engineering. Sabertooth 2x25 motor driver datasheet. Accessed: 2025-02-28. URL: https://www.dimensionengineering.com/datasheets/Sabertooth2x25.pdf.

[2] Texas Instruments. Tiva c series tm4c123g launchpad evaluation board user guide. Accessed: 2025-02-28. URL: https://www.ti.com/lit/ug/spmu296/spmu296.pdf?ts=1740733005566.