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

Architectural 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	11.05.2024	CD	document creation
0.2	11.19.2024	CD	added the introduction
0.3	11.19.2024	MG	created architecture diagram
0.4	11.20.2024	AH	added movement subsystems
0.5	11.24.2024	AK	added pathfinding subsystems
0.6	11.24.2024	RS	added interface subsystems
1.0	11.25.2024	CD, RS, MG, AK, AH	official release
1.1	04.30.2025	MG, AK	updated pathfinding subsystems
2.0	04.30.2025	CD, RS, MG, AK, AH	official release

Contents

List of Figures

List of Tables

1 Introduction

The Roam_Bot 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. For now, the Raspberry Pi has three sections that connect to it. 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

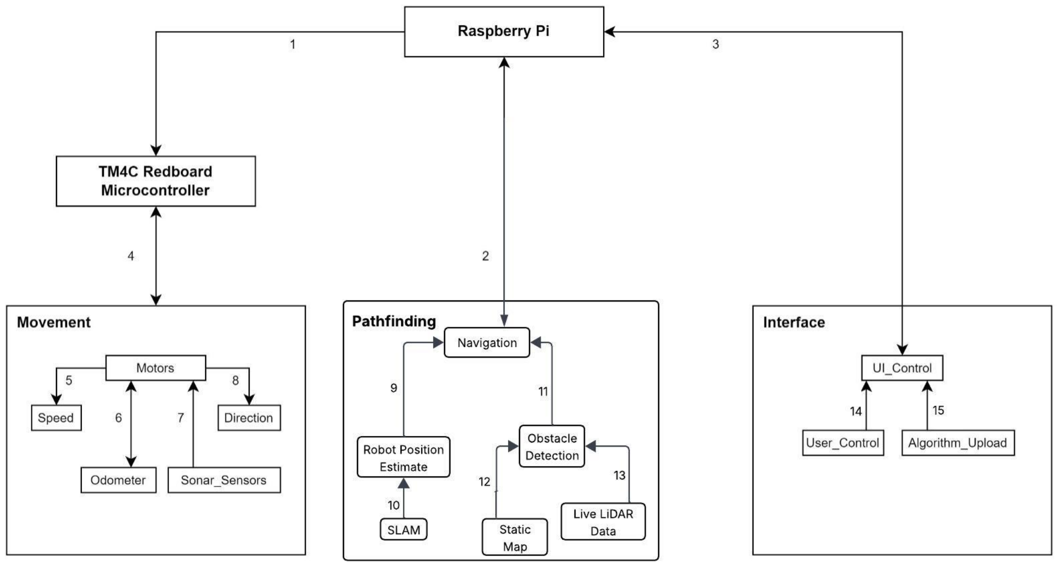

Figure 1: Architectural Layer Diagram

2.1 MOVEMENT - DESCRIPTION

The movement layer of the Roam_Bot is responsible for facilitating the rover's movement including the crash prevention feature, odometer, speed, and direction control. All instructions in this layer pass through Motor_Control using PWM and GPIO. The speed and direction subsystems both use PWM to achieve a user-provided speed and the algorithm determines the direction. The crash_prevention and odometer subsystems both use GPIO to let the rover know when to stop.

2.2 PATH FINDING - DESCRIPTION

The path-finding layer of the Roam_Bot is responsible for the navigation of the rover. This layer communicated with a TM4C to facilitate communication with the LIDAR. LIDAR will be used in the navigation to create a mapping of the room and communicate obstacles. The LIDAR and navigation will use UART.

2.3 INTERFACE - DESCRIPTION

The user interface layer is responsible for receiving user input. This layer is made up of 3 subsystems. The starting UI will display and redirect the user to algorithm upload or user control. User Control will use UART to accept rover commands. The algorithm upload uses UART to read a file into the system.

3 Subsystem Definitions & Data Flow

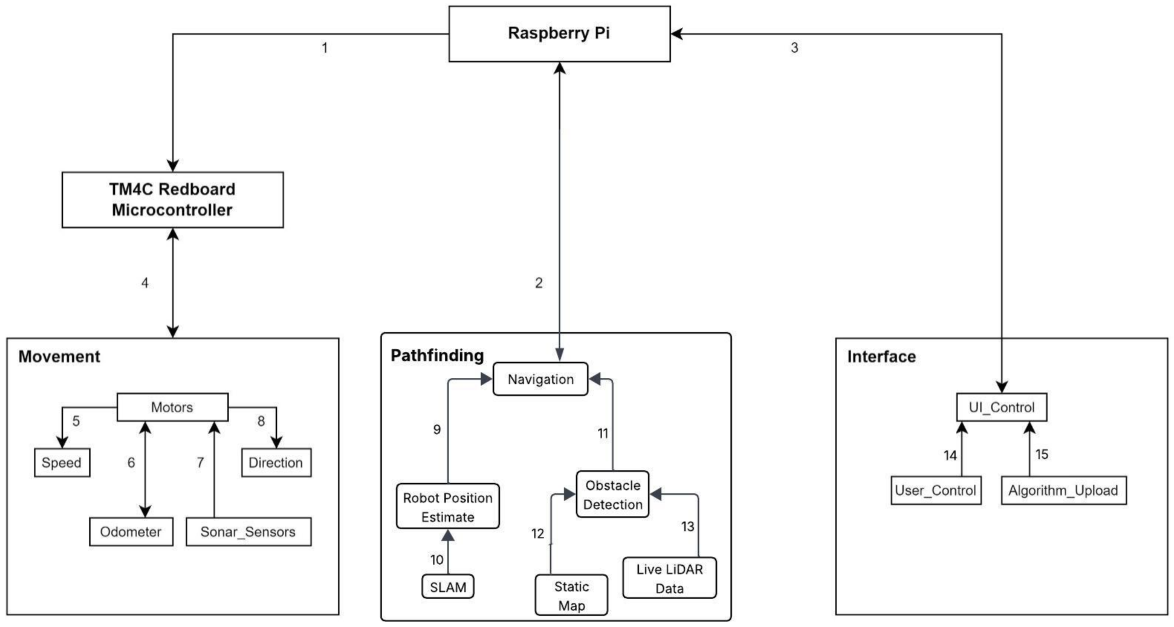

Figure 2: Data Flow Diagram

4 Movement Subsystems

4.1 Speed Control

The speed control subsystem manages the power of each motor to maintain a specific set speed. It receives feedback from the odometer and crash prevention to ensure safe operation.

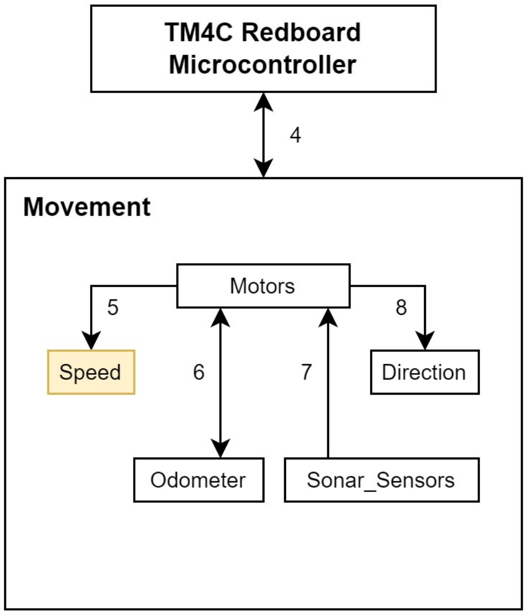

Figure 3: Movement Subsystem - Speed Control

4.1.1 Assumptions

The motor driver supports PWM input to control motor speed. The odometer feedback is accurate within 10% from wheel encoder measurements [1]. Communication latency between the TM4C and Raspberry Pi isn't noticeable for real-time feedback.

4.1.2 Responsibilities

Adjust motor PWM signals to match the target speed set. Monitor odometer feedback to ensure the actual speed matches the desired speed. Stop the speed if a collision is detected.

4.1.3 Subsystem Interfaces

Table 2: Movement Subsystem - Speed Control

ID	Description	Inputs	Outputs
#5	Pulse Width Modulation (PWM)	Power In	N/A

4.2 Direction Control

The direction control subsystem manages the robot's steering by adjusting the motor signals for the left and right wheels. It receives feedback from the steering sensors and ensures the rover follows the correct path.

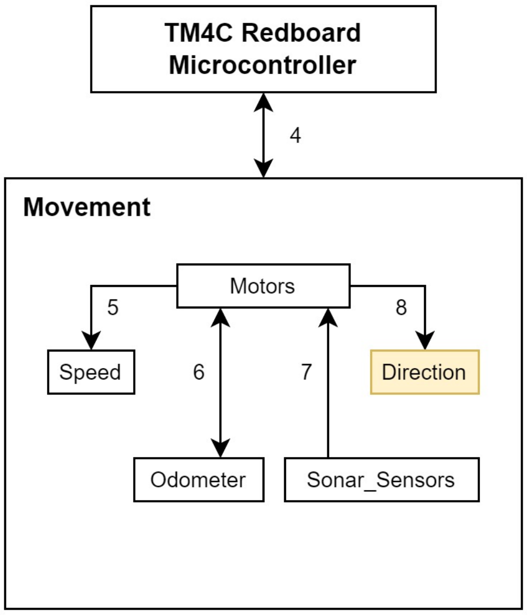

Figure 4: Movement Subsystem - Direction Control

4.2.1 Assumptions

The motor driver supports PWM input to control motor direction. The steering feedback system is accurate within 5% from LIDAR measurements [2]. Communication latency between the TM4C and Raspberry Pi is minimal for real-time control.

4.2.2 Responsibilities

Adjust motor PWM signals to steer the robot in the desired direction. Monitor steering feedback to ensure the robot is following the intended path. Correct any misalignment detected by the steering sensors.

4.2.3 Subsystem Interfaces

Table 3: Movement Subsystem - Direction Control

ID	Description	Inputs	Outputs
#8	Pulse Width Modulation (PWM)	Power In	N/A

4.3 Motor Control

The motor control subsystem manages the power supplied to the motors to achieve the desired movement. It receives feedback from the motor encoders and adjusts the motor signals to control both speed and direction.

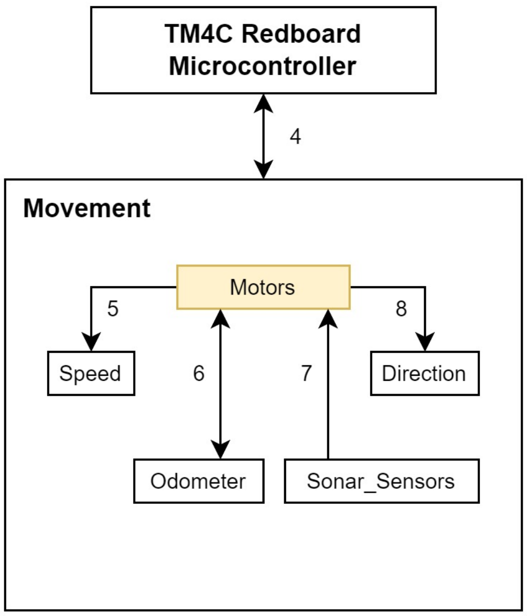

Figure 5: Movement Subsystem - Motor Control

4.3.1 Assumptions

The motor driver supports both PWM and direction control signals to drive motor speed and direction [3]. The motor encoders provide accurate feedback to control speed and direction adjustments. Communication latency between the TM4C and Raspberry Pi does not impact real-time motor control.

4.3.2 Responsibilities

Control motor power to achieve the desired speed and direction. Monitor motor feedback to ensure correct operation and make adjustments when necessary. Handle error states, such as motor stall or failure, and stop the motors if a critical fault is detected.

4.3.3 Subsystem Interfaces

Table 4: Movement Subsystem - Motor Control

ID	Description	Inputs	Outputs
#5	Pulse Width Modulation (PWM)	N/A	Power Output
#6	General Purpose Input Output (GPIO)	Odometer Feedback	Odometer Control
#7	General Purpose Input Output (GPIO)	Crash Detect	N/A
#8	Pulse Width Modulation (PWM)	N/A	Power Output

4.4 Odometer

The odometer subsystem is responsible for measuring the distance traveled by the vehicle. It provides feedback to the motor control subsystem to help adjust the speed and ensure accurate movement over time.

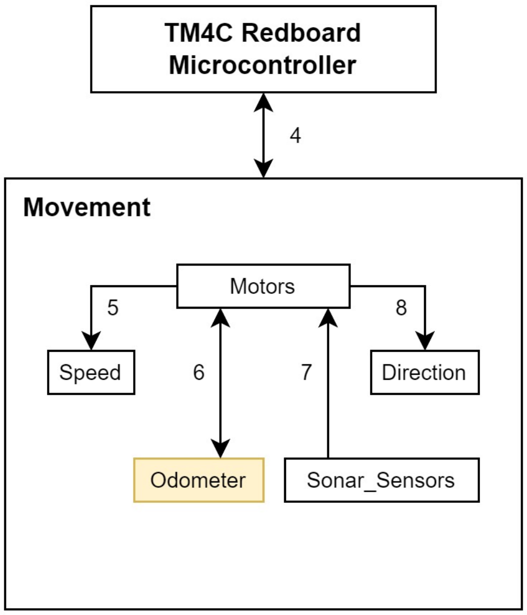

Figure 6: Movement Subsystem - Odometer

4.4.1 Assumptions

The odometer system uses wheel encoders and LIDAR to measure the distance traveled. The feedback is accurate within 10% for distance measurement [1] [2]. The odometer data is updated in real time, with no significant communication delays between the TM4C and Raspberry Pi affecting its accuracy.

4.4.2 Responsibilities

Measure the distance traveled by the vehicle based on wheel rotations. Provide real-time feedback to the motor control subsystem to adjust the speed accordingly. Alert the system if abnormal readings are detected, such as wheel slippage or encoder malfunction.

4.4.3 Subsystem Interfaces

Table 5: Movement Subsystem - Odometer

ID	Description	Inputs	Outputs
#6	General Purpose Input Output (GPIO)	Odometer Control	Odometer Feedback

4.5 Crash Detection

The crash detection subsystem is responsible for identifying any potential collisions or obstacles in the vehicle's path. It monitors sensor data to detect obstacles or sudden changes in velocity that may indicate a crash or near-crash event.

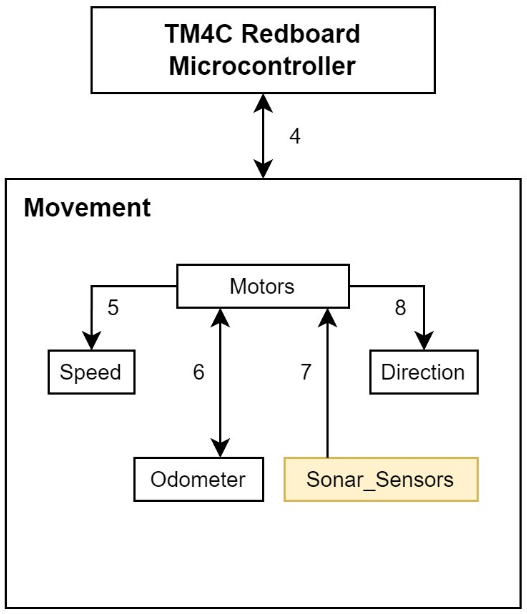

Figure 7: Movement Subsystem - Crash Detection

4.5.1 Assumptions

The crash detection system uses a combination of proximity sensors and accelerometers to detect obstacles and sudden impacts [4].The sensor system is accurate within a range of 5 cm for obstacle detection, and the accelerometers can detect changes in velocity. Communication latency between the TM4C and Raspberry Pi is minimal and does not affect real-time collision detection.

4.5.2 Responsibilities

Monitor sensor data to detect any obstacles or sudden changes in velocity. Alert the motor control subsystem to stop or adjust the vehicle's movement if a crash is detected. Ensure that the vehicle responds promptly to prevent further damage or injury in case of a collision.

4.5.3 Subsystem Interfaces

Table 6: Movement Subsystem - Crash Detection

ID	Description	Inputs	Outputs
#7	General Purpose Input Output (GPIO)	Crash Detect	N/A

5 Pathfinding Subsystems

The pathfinding subsystem manages the components or modules within a larger pathfinding system that handles specific aspects of the

5.1 Static Map Subsystem

The Static Map subsystem allows the display of the LiDAR range as a grid of cells and allows them to be marked as either traversable or non-traversable, as needed.

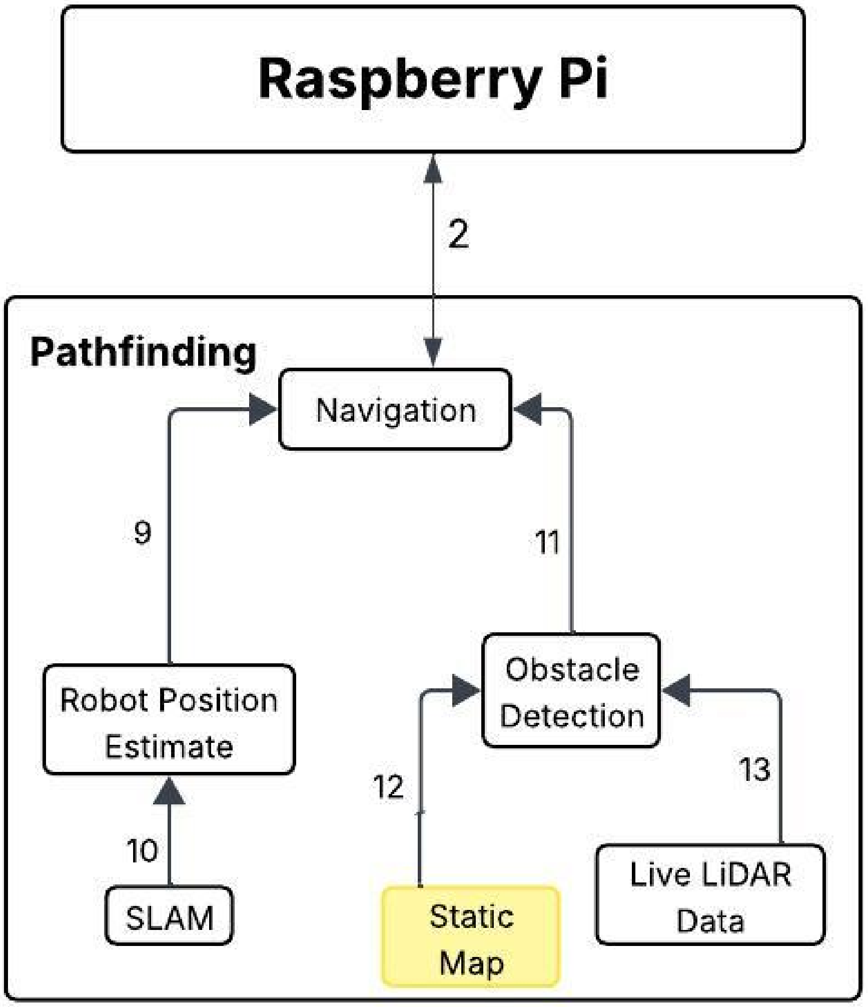

Figure 8: Pathfinding Subsystem - Static Map

5.1.1 Assumptions

Static Map assumes that the Obstacle Detection will be functional, and there will not be dynamic obstacles or other movement during pathfinding.

5.1.2 Responsibilities

The static map creates a map using the SlamTec LiDAR to than feed into the Obstacle Detection Subystem, so that it can create a costmap.

5.1.3 Subsystem Interfaces

Table 7: Pathfinding Subsystem - Static Map

ID	Description	Inputs	Outputs
#12	Data converts Laser Scan into Map	N/A	Map (.yaml file)

5.2 Obstacle Detection Subsystems

The obstacle detection subsystem is designed to process feedback from the LiDAR, identifying objects within its range and marking them as impassable when necessary.

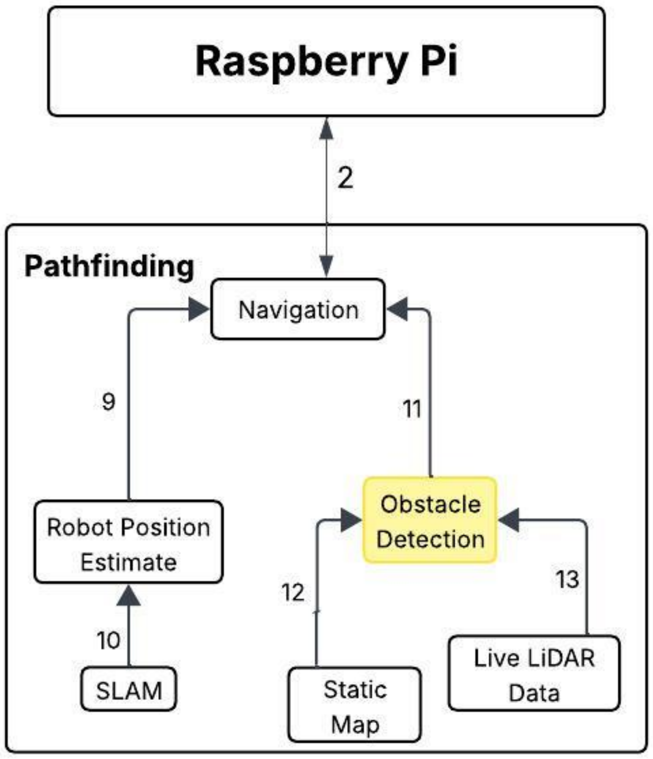

Figure 9: Pathfinding Subsystem - Obstacle Detection

5.2.1 Assumptions

The rover will be able to register static and dynamic obstacles and be detected with LiDAR.

5.2.2 Responsibilities

The rover will be able to detect obstacles and update the graph representation Subsystem by marking a cell as untraversable. The obstacle detection subsystem should also be able to be integrated with LiDAR and other sensors. If the close-range sensors get triggered the rover should halt movement and retreat a set distance before reactivating its pathfinding.

The pathfinding algorithm creates a grid using the map the static map subsystem provided, representing the environment with the LiDAR Range. The grid will contain nodes that contain properties such as:

- Traverability (true/false).
- Coordinates in world space.

5.2.3 Subsystem Interfaces

Table 8: Pathfinding Subsystem - Obstacle Detection

ID	Description	Inputs	Outputs
#11	Algorithm will send a signal to the navigation.	N/A	Marked Obstacles
#12	Algorithm combines the data from map.	Map Data	N/A
#13	Algorithm combines the data from LiDAR.	Scan Data	N/A

5.3 Navigation Subsystems

The Navigation Subsystem enables the Roam_Bot to autonomously navigate the environment by combining real-time position data from the Robot Position Estimate Subsystem with obstacle information from the SLAM-generated map. It generates an optimal path through the terrain, considering both the robot's current pose and the locations of obstacles, and continuously adjusts the path as the environment changes.

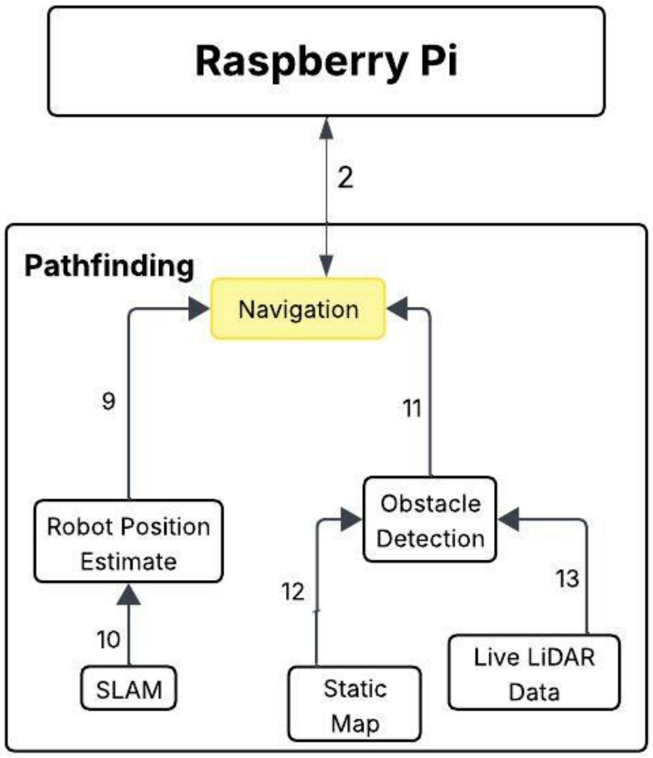

Figure 10: Pathfinding Subsystem - Navigation

5.3.1 Assumptions

The pathfinding algorithm requires that the graph is already constructed, and LiDAR is active.

5.3.2 Responsibilities

The pathfinding algorithm calculates either the optimal or heuristic-based path from a start to end node. The pathfinding algorithms should also allow for multiple algorithms, such as A*, Dijkstra's, Breadth-First-Search, etc. While also handling static and dynamic obstacles.

5.3.3 Subsystem Interfaces

Table 9: Pathfinding Subsystem - Navigation

ID	Description	Inputs	Outputs
#2	Sending movement decision to Raspberry Pi.	N/A	Data to TM4C (Microcontroller)
#9	Receives approximate position odometry.	Position from SLAM	N/A
#11	Receives obstacles detected from scan.	Obstacle Detection	N/A

5.4 Live LiDAR Data Subsystems

The LiDAR subsystem is implemented as a ROS package that interfaces with and drives the LiDAR hardware module, providing the Roam_Bot with real time scan data used to perceive and interpret the surrounding terrain.

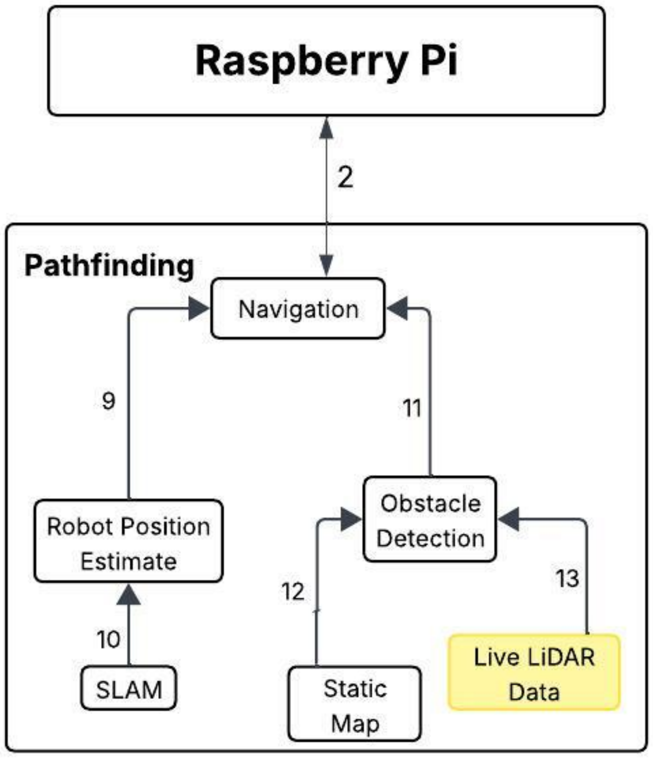

Figure 11: Pathfinding Subsystem - Live LiDAR Data

5.4.1 Assumptions

- Static Obstacles are Detected Within Range
- Robot Operates on a Relatively Flat Plane
- Costmap is Regularly Updated with LiDAR Data
- No Excessive Sensor Noise or Clutter

The LiDAR subsystem requires that the Roam_Bot be turned on.

5.4.2 Responsibilities

The LiDAR subsystem takes the input given by its sensors and sends all relevant information to the Obstacle Detection subsystem, which creates a costmap.

5.4.3 Subsystem Interfaces

Table 10: Pathfinding Subsystem - Live LiDAR Data

ID	Description	Inputs	Outputs
#13	The LiDAR subsystem will transmit laser scan data to the Obstacle Detection subsystem, which will process the input, convert it into a structured grid, and divide it into individual cells for further analysis.	N/A	Obstacle Detection

5.5 Robot Position Estimate Subsystem

The Robot Position Estimate Subsystem is responsible for determining the robot's current pose (position and orientation) within a map generated by SLAM. It continuously updates the pose using LiDAR data and provides this information to the Navigation Decision Subsystem for path planning and movement control.

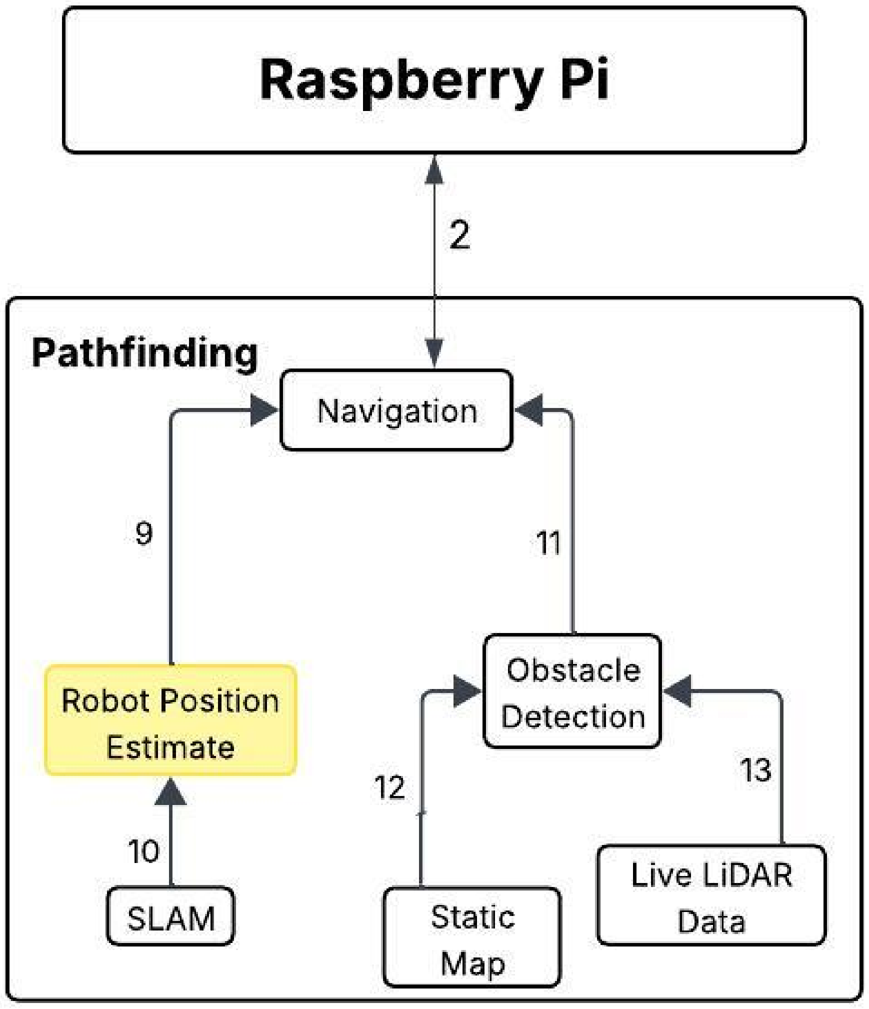

Figure 12: Pathfinding Subsystem - Position Estimate

5.5.1 Assumptions

- SLAM is functioning correctly and consistently receives LiDAR input.
- The robot operates in a primarily static environment suitable for SLAM-based localization.
- The initial pose is known or initialized at the origin.
- The Raspberry Pi is fully operational and can run SLAM in real time.

5.5.2 Responsibilities

The Robot Position Estimate Subsystem performs the following functions:

- Accepts pose data from the SLAM subsystem.
- Maintains the robotâs estimated pose in the global coordinate frame.
- Provides updated pose data to the Navigation Decision Subsystem for route planning and motion.
- Supports coordinate conversion as necessary for integration with grid-based navigation.

5.5.3 Subsystem Interfaces

Table 11: Pathfinding Subsystem - Position Estimate

ID	Description	Inputs	Outputs
#9	Current estimated robot pose for navigation decisions	N/A	Robot Pose
#10	Pose data provided by SLAM	SLAM Pose	N/A

5.6 SLAM

The SLAM subsystem processes data to simultaneously localize the robot within an environment and build a global occupancy grid map. This enables the Robot Position Estimate Subsystem to determine the robot's real time pose (x, y, Î¸) in relation to the mapped surroundings, which is critical for accurate navigation and planning.

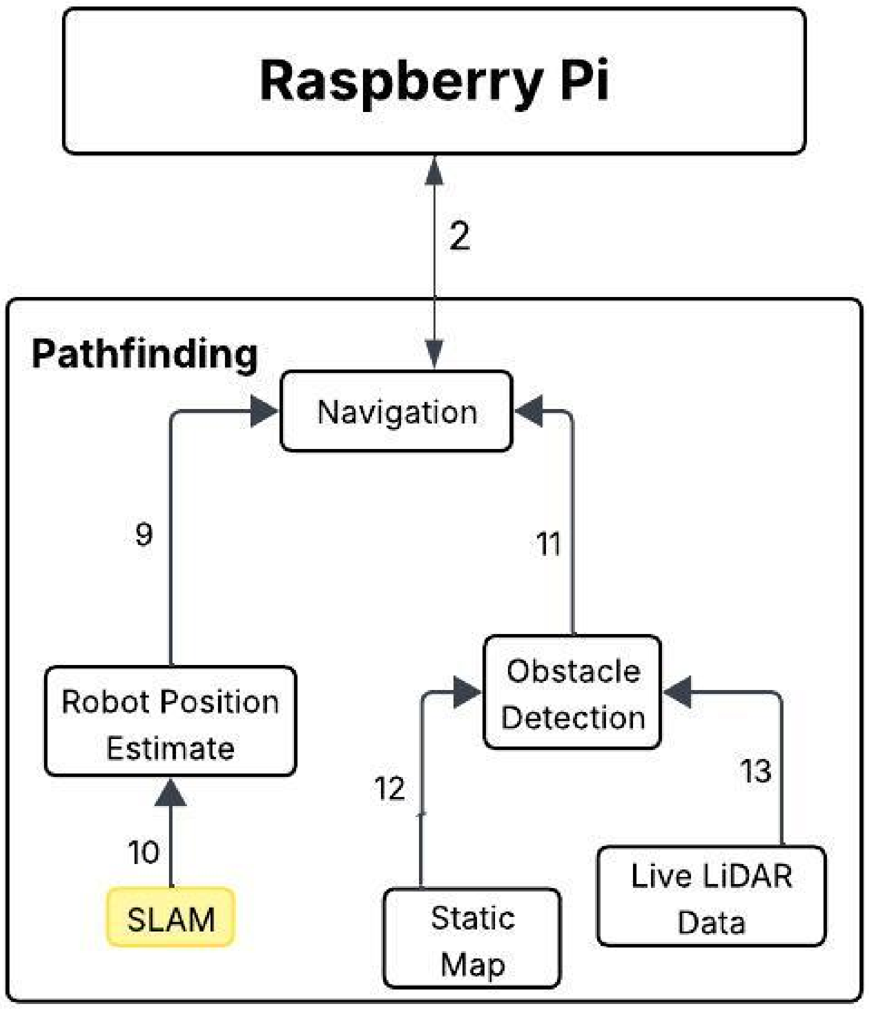

Figure 13: Pathfinding Subsystem - SLAM

5.6.1 Assumptions

- ROS2 Jazzy is installed on the Raspberry Pi 5.
- Nav2 Jazzy is installed on the Raspberry Pi 5.
- The Static Map is locatable on the Raspberry Pi 5.
- Robot operates on a relatively flat plane.

5.6.2 Responsibilities

The responsibilites that SLAM uses is:

- Real-Time Map Building: (Mapping) Continuously generates a 2D or 3D map of the environment based on sensor data (typically LiDAR or depth camera).
- Robot Localization: Estimates the robot's pose (x, y, Î¸) in the global map frame in real time. Uses scan matching or visual feature matching to align sensor data with the evolving map.

5.6.3 Subsystem Interfaces

Table 12: Pathfinding Subsystem - SLAM

ID	Description	Inputs	Outputs
#10	Slam's localization function outputs the robot's position estimate	N/A	SLAM Pose

6 Interface Subsystems

The interface subsystems are responsible for getting the user input and translating it to the motion of the rover.

6.1 User Interface

This is the user interface, where users will be able to input which path-finding algorithm they want the rover to utilize, and control the motion of the rover.

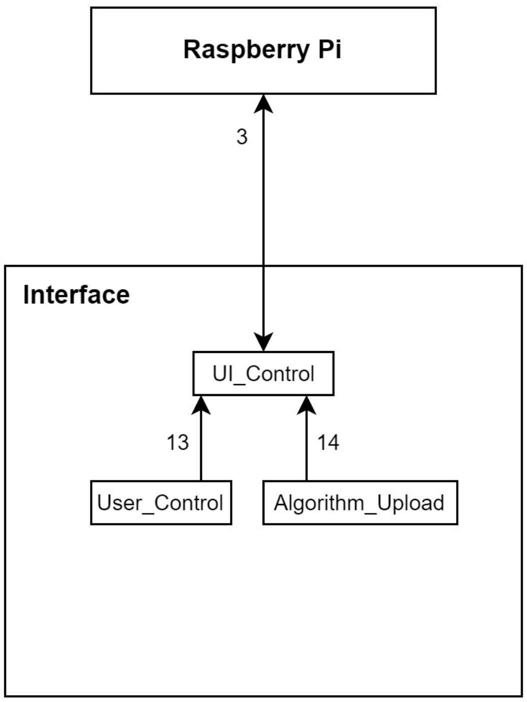

Figure 14: Interface Subsystem

6.1.1 Assumptions

We are assuming that the user will be able to utilize the UI. We will be printing out specific instructions, and cover for any possible cases.

6.1.2 Responsibilities

It should be user-friendly and easy to use, so that users can simply run the program and send commands to the rover.

6.1.3 Subsystem Interfaces

Table 13: Interface Subsystem

ID	Description	Inputs	Outputs
#3	UI	N/A	Screen
#13	User Control	Speed Direction	Motion
#14	Algorithm Upload	Algorithm Name	Path-finding

References

[1] X. Zhang, Y. Li, and L. Zhang, “A new method of angle measurement error analysis of rotary encoders,” *IEEE Access*, vol. 8, pp. 74 539–74 549, 2020.

[2] T. Epton, “Odometry correction of a mobile robot using a range-finding laser,” Master’s Thesis, Clemson University, 2012.

[3] D. Engineering, “Sabertooth 2x25 motor driver,” 2024, accessed: 2024-11-25.

[4] M. Hassan, M. Raza, F. Gul, M. Anwar, and M. Durrani, “A sensor-based robotic model for vehicle collision reduction,” *ResearchGate*, 2012, accessed: 2024-11-25.