TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEVELOPMENT OF AUTONOMOUS NAVIGATION ROBOT IN OUTDOOR ENVIRONMENTS USING SLAM TOOLBOX AND NAV2

Balkrishna Poudel (077BME005)
Biswash Khatiwada (077BME014)
Manoj Bhatta (077BME020)
Susil Chhetri (077BME044)

A PROJECT REPORT
SUBMITTED TO THE DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR IN MECHANICAL ENGINEERING

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
LALITPUR, NEPAL

March 2025

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A PROJECT REPORT ON

DEVELOPMENT OF AUTONOMOUS NAVIGATION ROBOT IN OUTDOOR ENVIRONMENTS USING SLAM TOOLBOX AND NAV2

By:

Balkrishna Poudel (077BME005)

Biswash Khatiwada (077BME014)

Manoj Bhatta (077BME020)

Susil Chhetri (077BME044)

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

LALITPUR, NEPAL

March 2025

LETTER OF APPROVAL

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING, PULCHOWK CAMPUS
DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled **"DEVELOPMENT OF AUTONOMOUS NAVIGATION ROBOT IN OUTDOOR ENVIRONMENTS USING SLAM TOOLBOX AND NAV2"** submitted by Balkrishna Poudel, Biswash Khatiwada, Manoj Bhatta, and Susil Chhetri in partial fulfillment of the requirement for the degree of Bachelor of Mechanical Engineering.

Supervisor, Dr.Laxman Poudel
Professor
Department of Mechanical and Aerospace Engineering
IOE, Pulchowk Campus

Supervisor, Biman Rimal
Assistant Professor
Department of Mechanical and Aerospace Engineering
IOE, Pulchowk Campus

External Examiner, Er. Ramraj Khanal
Managing Director
Calcgen Solutions

Committee Chairperson, Dr. Sudip Bhattrai
Assistant Professor
Department of Mechanical and Aerospace Engineering
IOE, Pulchowk Campus

Date: 2025/03/11

ABSTRACT

This study focuses on the development of an Autonomous Mobile Robot (AMR) system capable of outdoor navigation through real-time mapping, localization, and navigation. The system implements ROS2 as the main communication framework to connect and manage the different hardware and software components, combining LiDAR-based mapping and motion estimation. For environmental mapping, Fast-LIO, a fast and efficient LiDAR-Inertial Odometry method, is used to generate 3D point cloud maps and odometry. Communication between hardware components was achieved through UART communication, allowing smooth and efficient data transfer between the sensors and processing unit. By combining LiDAR data for mapping and IMU data for motion tracking, the system improves its understanding of the outdoor environment and ensures precise mapping. Mapping is done using the SLAM toolbox, and navigation is done using Nav2 algorithms. The robot's obstacle avoidance, path optimization, and performance were improved and tested in complex outdoor environments.This robot can be used for delivery, security, and cleaning purposes within the campus.

Keywords: *AMR, SLAM, ROS2, UART, LIO, 3D Mapping*

ACKNOWLEDGEMENT

We are sincerely grateful to the Department of Mechanical and Aerospace Engineering, IOE, Pulchowk Campus, Lalitpur, for providing us with a valuable opportunity to undertake a project that allows us to explore, enhance and apply the knowledge and skills that we have acquired throughout our Bachelor of Mechanical Engineering journey. We are thankful to our supervisors, Prof. Dr. Laxman Poudel, Asst. Prof. Biman Rimal, and Asst. Prof. Ashish Karki for guiding us throughout this project. We are especially thankful to Asst. Prof. Biman Rimal for providing us with MID360 LiDAR.

We want to acknowledge and express our sincere gratitude to the senior group members Ankit Kharel, Anusha Acharya, Nitesh Subedi, and Prajwal Koirala, who laid the foundation for the trajectory optimization of autonomous navigation of two-wheeled robots. Additionally, we extend our thanks to Nirmal Prasad Panta, Pawan Shrestha, Prince Panta, and Saki Basnet for their valuable insights and foundational contributions to the indoor navigation of two-wheeled robots, which have greatly inspired and supported our efforts.

We would also like to thank the Robotics Club, Pulchowk Campus, for providing us with the required workspace for the project. We are really indebted to all our friends, seniors, and juniors at the Robotics Club, who assisted us with the problems that we encountered during the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robot
CAD	Computer Aided Design
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IDE	Integrated Development Environment
IKD-Tree	Incremental K-Dimension Tree
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
LIO	Lidar Inertial Odometry
LOAM	Lidar Odometry and Mapping
RGB-D	Red Green Blue-Depth
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
UKF	Unscented Kalman Filter
URDF	Unified Robot Description Format

Chapter One: INTRODUCTION

1.1 Background

Robotics is an interdisciplinary domain involving science and engineering that focuses on developing robots. The journey of robotics has come a long way and has undergone much rapid development in recent years. In the rapidly evolving field of robotics, autonomous robotics has emerged as a game changer to make intelligent robots. There are many types of autonomous systems. The term "Autonomous" means that the system is capable of moving through its environment without direct human intervention or custom scripts to define the actions that the bot will perform to control the steering, acceleration, braking, etc. The term "automated" in Automated Guided Vehicle (AGV) defines the action of "doing the same thing over and over again based on a simple cue." An Autonomous Mobile Robot (AMR), however, is "autonomous," which indicates an added awareness of its surroundings to overcome unexpected challenges based on its perception.

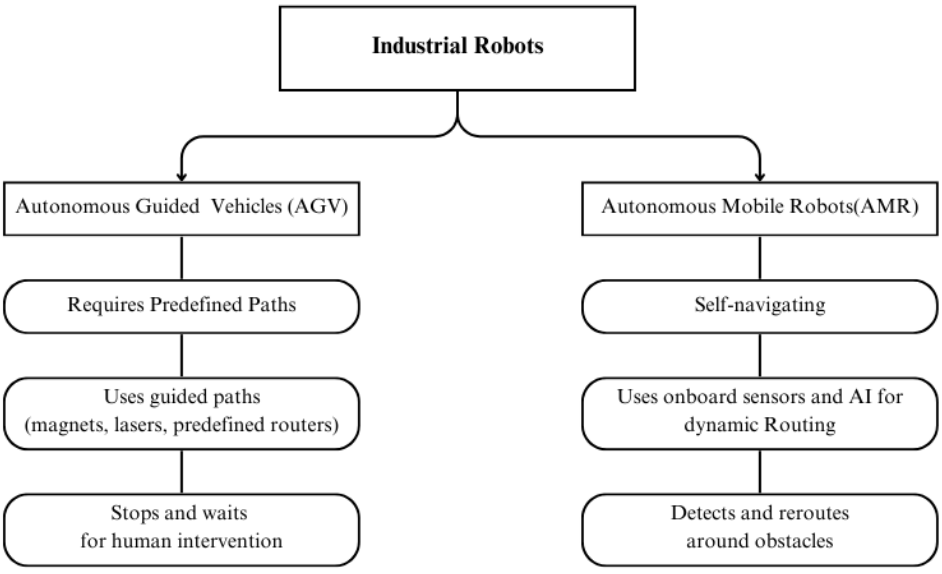


Figure 1.1: Comparison of AGV and AMR

The level of human intervention in the system defines the level of autonomy. This term is used to define the autonomous vehicle.

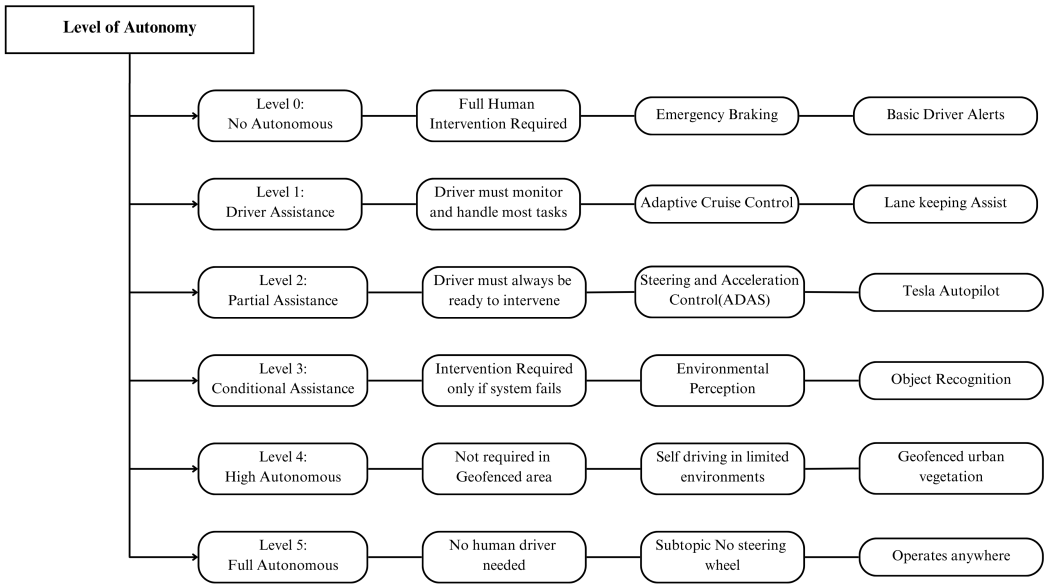


Figure 1.2: Levels of Autonomy

The development of robots capable of operating autonomously and making independent decisions depends heavily on two main elements: sensing capabilities and decision-making algorithms. Sensing capabilities include the ability to perceive, interpret and understand the surrounding environment through various sensors such as cameras, LiDAR, and radar so that autonomous robots can make accurately informed decisions using decision-making algorithms. The sensors collect data, and those algorithms use that information to understand the environment, helping the robot make intelligent decisions and respond appropriately. When combined, sensing capabilities and decision-making algorithms form the foundation for creating autonomous robots capable of operating independently. When a robot operates in an unknown area, it is essential to create a map of the environment. Simultaneous Localization and Mapping (SLAM) addresses this issue when GPS is unavailable. However, suppose a pre-existing map is available, and the GPS signal is reliable; then, in that case, we prioritize using GPS for mapping over the computationally intensive task of creating a new map from scratch.

The critical objective is to generate maps that are sufficiently similar to allow comparison between a known map and the local map created during a task. In the GPS-denied environment, localization is performed by using landmarks to generate maps and determine the robot's position relative to these points of reference for robots to navigate and perform functions like steering and braking autonomously, they must perceive and understand their surroundings like humans. AI, computer vision, and sensor technology advancements have made autonomous robots a reality. Thus, using a fusion of these technologies, robots can now determine their location and make real-time navigation

decisions based on their perception of the environment.

The rapid advancements in robotics and artificial intelligence have revolutionized the capabilities of Autonomous Mobile Robots (AMRs), making them a vital component in various industries today. Their ability to operate in ever-changing dynamic, unstructured and unproductive environments has made them indispensable in sectors such as logistics, manufacturing, healthcare, and agriculture. In particular, the demand for AMRs is growing significantly due to the need for efficient, cost-effective, and reliable solutions to automate tasks like material handling, transportation, and inspection. With labour shortages and rising costs, AMR offers a scalable way to improve productivity and simplify operations.

AMRs are gaining traction in outdoor environments for applications such as last-mile delivery, environmental monitoring, and infrastructure maintenance. Urban areas and large campuses, such as universities, are ideal testing grounds for these robots, as they simulate real-world conditions, including variable terrains, obstacles, and dynamic traffic. Their role in outdoor operations is particularly critical as industries and institutions look to reduce carbon footprints and improve logistics efficiency through sustainable automation technologies.Studies are being done to make the system fully reliable and deployable. A 2023 report by ABI Research predicts that over 4 million AMRs will be deployed in various industries worldwide by 2030, underscoring the increasing role of these technologies in solving real-world problems. Despite the rapid progress in robotics and artificial intelligence, developing reliable autonomous mobile robots (AMRs) for outdoor environments remains a significant challenge.

This project aims to address the challenges associated with autonomous navigation by implementing and evaluating a system designed for outdoor environments with a focus on campus roads during the testing phase. This project centers on the development of an autonomous differential mobile robot capable of navigating in semi-structured environments. By conducting a comparative analysis of existing methods, this study seeks to enhance the understanding of autonomous navigation performance in semi-structured outdoor settings.

The findings are expected to contribute to ongoing advancements in AMR technology, supporting future developments in urban mobility, smart logistics, and industrial automation. Thus, by leveraging state-of-the-art technologies, this project will not only demonstrate the feasibility of campus-wide autonomous navigation but also pave the way for future applications of AMRs in Nepal.

1.2 Problem Statement

One of the primary challenges in the development of autonomous robots is enabling them to identify and model their surrounding environment. Therefore, the first step is to perceive the environment, after which the fundamental building blocks: Mapping, Localization, Obstacle Detection, and Path Planning, can be implemented.

Localization is the problem of determining a robot's current position in the environment. This localization challenge extends across various environmental domains from robotics and human navigation in indoor spaces to oceanic depths, underground environments, aerial contexts, and even space exploration due to the frequent lack of GPS signals in these environments. In outdoor settings where Global Navigation Satellite Systems (GNSS) such as GPS are accessible, positioning becomes trivial if the accuracy meets the application's requirements. However, if the robot does not have access to GPS or the environment is not well known, the problem of performing autonomous navigation becomes much more complex and far from trivial. In such scenarios, the robot needs to continuously estimate its position accurately in real time without relying on satellite signals. Localization often relies on estimation techniques to update the robot's position based on sensors and algorithms. The estimation can be performed with the help of GPS or the fusion of sensors like (IMU, LiDAR, and odometry).

Therefore, different localization and position estimating methods are necessary, each suited to the specific environmental conditions. In localization, the robot's location can be determined either using a global coordinate system like GPS or by referencing a local map. The solutions (position estimation) must be reliable, and all other available information must be used to get the best possible positioning estimate. That is the problem of localization.

Navigation is the problem of finding traversable paths to different goal locations within the environment by avoiding obstacles. An example task for a mobile robot in an outdoor domain is to go from point A to point B using only visual landmarks or GPS waypoints.

The problem at hand, therefore, is to implement a SLAM algorithm in a mobile robotic system so that during the task's execution, the robot localizes itself relative to a pre-built map while simultaneously generating a drivable path to solve and navigate in an outdoor environment. This approach eliminates the reliance on GPS signals, using only local sensors (such as LiDAR, IMUs, and odometry) to continuously estimate the robot's position and map the environment in real-time.

1.3 Objectives

This project aims to explore and implement autonomous navigation technologies for mobile robots. The primary focus is on developing a system that can operate in outdoor settings.

1.3.1 Main Objective:

- To develop an autonomous mobile robot capable of navigating in an outdoor environment.

1.3.2 Specific Objective:

- To design and fabricate an AMR prototype.
- To implement a suitable SLAM algorithm and generate a map of campus premises.
- To implement the plan planning using Nav2 stack for autonomous navigation in campus environment
- To integrate obstacle avoidance for static as well as dynamic obstacles.

1.4 System Requirements (For Development/Implementation)

This section specifies the essential hardware and software elements that form the foundation of our AMR.

1.4.1 Hardware Requirements

The major hardware components required for our project are:

1. DC Motor
2. Motor Driver
3. STM32f4 Discovery
4. Encoders
5. Laptop
6. Mid-360 LiDAR
7. Battery

1.4.2 Software Requirements

The required softwares for the project include:

1. Solidworks
2. ROS2 (Humble)
3. VSCode
4. STMCubeMX
5. OpenCV

Chapter Two: THEORETICAL BACKGROUND

2.1 Autonomous Mobile Robot

An autonomous mobile robot is a type of robot that can understand and move through its environment independently. AMRs differ from their predecessors, autonomous guided vehicles (AGVs), which rely on tracks or predefined paths and often require operator oversight [1]. When the robot is left in the environment and given the certain goal to make it autonomous, it is our prime concern that how does the robot move. Autonomous here means that robot localizes in its environment, changes its speed, rotates in certain direction to reach the given path. And we are interested how all these action happens, which sensors perceives the environmental features and information to minimize the error to reach the goal.
Autonomous mobile robots have many use cases like inspection of crops in agriculture, material transportation in industrial sites, post-disaster exploration in rescue operations, etc. Depending on the use case scenarios and environments of use of the robots, different types of drive systems are available for use.

2.2 Drive Systems for AMRs

To achieve effective mobility, AMRs utilize different types of drive systems, each offering unique advantages depending on the application and operational environment. The choice of a drive system significantly impacts the robot's maneuverability, stability, and energy efficiency. The following section explores the various drive mechanisms used in AMRs. The most common types of drive systems are

2.2.1 Differential Drive:

Differential drive is one of the most commonly used drive systems for AMRs. It consists of two independently controlled wheels on either side of the robot, with optional passive caster wheels for stability. The robot moves forward or backward by rotating both wheels at the same speed and turns by varying the speed of the two wheels.

Pros:

- Simple design and easy to implement.
- Provides good maneuverability and can turn in place.
- Cost-effective and widely used in mobile robots.

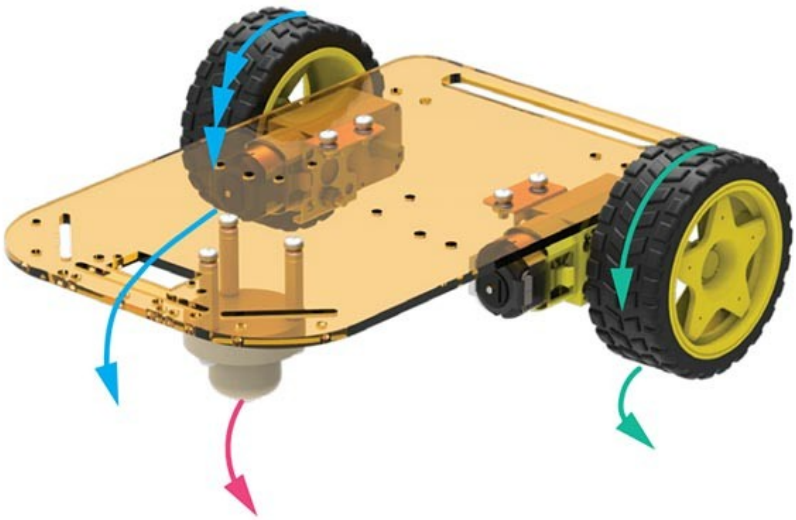


Figure 2.1: Differential Drive Robot[2]

Cons:

- Struggles with traction and control on uneven surfaces.
- Requires precise wheel synchronization for accurate movement.
- May suffer from slippage, leading to localization errors.

2.2.2 Omnidirectional Drive:

Omnidirectional drive systems allow robots to move in any direction without the need to rotate first. These systems often use Mecanum wheels or omni-wheels, which have rollers positioned at an angle to the main wheel.

Figure 2.2: Omnidirectional Drive[3]

Pros:

- Excellent maneuverability with the ability to move in any direction instantly.
- Useful in constrained environments requiring precise positioning.
- Smooth navigation without requiring complex turning maneuvers.

Cons:

- More complex in terms of mechanical design and control.
- Reduced traction and stability compared to standard wheels.
- Higher energy consumption due to increased friction and slippage.

2.2.3 Skid-Steer Drive:

Skid-steer drive is commonly used in off-road and rugged environments. It involves multiple wheels or tracks on each side, which are driven in pairs. The robot turns by applying different speeds or directions to each side, causing the wheels or tracks to skid.

Figure 2.3: Skid-Steer Drive Robot[4]

Pros:

- Suitable for rough and uneven terrain.
- Simple mechanical design with robust structure.
- Provides high traction and stability.

Cons:

- High energy consumption due to friction during turns.
- Causes wear on tires or tracks due to skidding.
- Less precise turning compared to other drive systems.

2.2.4 Ackermann Drive:

Ackermann drive is modeled after the steering mechanism used in cars, where the front wheels steer while the rear wheels provide propulsion. This system ensures that all wheels follow a smooth turning path without skidding.

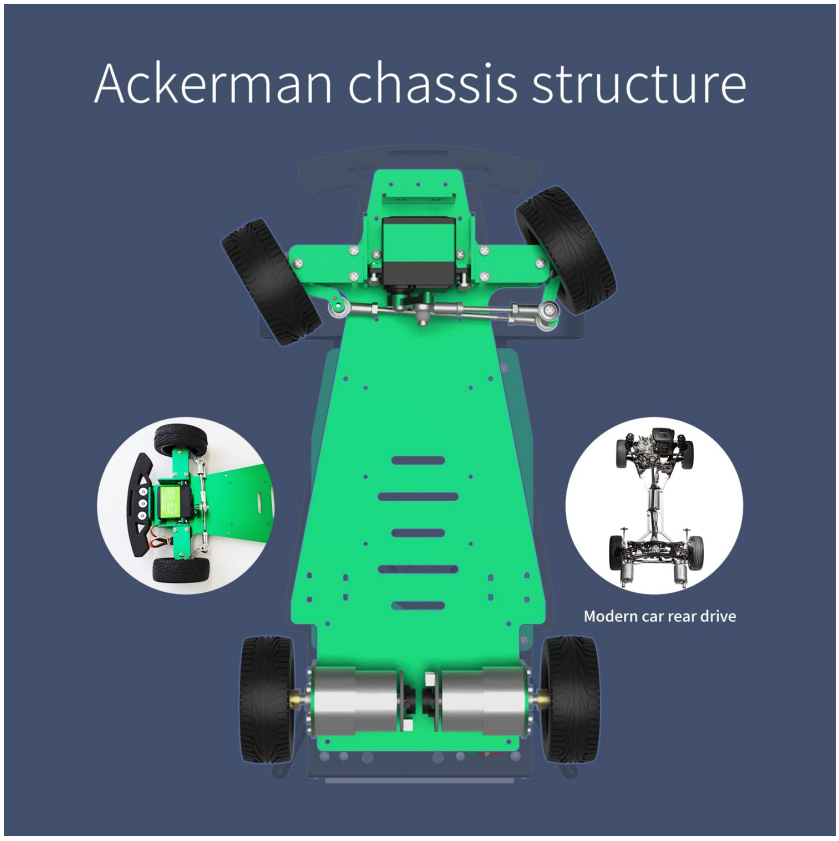


Figure 2.4: Ackermann Drive[5]

Pros:

- Efficient on paved surfaces with minimal energy loss.
- Provides smooth and stable motion, making it ideal for road-based applications.
- Reduced tire wear compared to skid-steer drive.

Cons:

- Limited maneuverability, as it cannot turn in place.
- Complex steering mechanism requiring additional components.
- Less effective in highly cluttered or dynamic environments.

Here is a table comparing all the above drive systems, highlighting their advantages and disadvantages: From the above options, skid-steer drive is chosen due to its simple mechanical structure, higher maneuverability, and less complexity compared to other drive

Table 2.1: Comparison of Different Drive Systems

Drive System	**Pros**	**Cons**
Differential Drive	Simple design, good maneuverability, cost-effective	Struggles on uneven surfaces, requires precise wheel synchronization, localization errors due to slippage
Omnidirectional Drive	Excellent maneuverability, precise positioning, smooth navigation	Complex design, reduced traction, higher energy consumption
Skid-Steer Drive	Suitable for rough terrain, robust structure, high traction	High energy consumption, tire wear due to skidding, less precise turning
Ackermann Drive	Efficient on paved surfaces, smooth and stable motion, reduced tire wear	Limited maneuverability, complex steering mechanism, less effective in cluttered environments

systems. Unlike Ackermann drive, which requires a complex steering mechanism, or omnidirectional drive, which demands intricate control and additional hardware, skid-steer drive offers a robust and straightforward solution for applications requiring high traction and rugged terrain adaptability.

2.3 Robot Kinematics

Skid-steer is a type of drive system, in which the wheels or tracks on each side of the vehicle are driven independently and turning is realized by means of driving the left and right wheels at different velocities[6] Wheeled skid-steering drive mechanisms are commonly used in various all-terrain vehicles, including loaders, agricultural machinery, mining equipment, and military applications. This traction method is also valuable for off-road mobile robots, with applications in planetary exploration, land-mine detection, and search-and-rescue operations. Additionally, some commercial robotic research platforms incorporate this locomotion system.

The steering mechanism relies on controlling the relative velocities of the left and right wheels, similar to differential drive vehicles. However, since all wheels are aligned with the vehicle's longitudinal axis, turning requires wheel slippage. This mechanism operates similarly to tracked vehicles, which generally provide better traction but involve greater mechanical complexity.

Wheeled skid-steering offers two key advantages over alternative wheel configurations like Ackerman steering or axle-articulated systems. It is mechanically simple and robust

while also enabling superior maneuverability, including zero-radius turning, without requiring additional components beyond those needed for straight-line motion.

Despite its advantages, this locomotion system presents challenges in motion control and odometry. The kinematics of skid-steering are not straightforward, as the vehicle's motion cannot be precisely predicted based solely on control inputs. Traditional kinematic models for non-holonomic wheeled vehicles, which assume pure rolling and no-slip conditions, do not apply in this case.

To address this challenge, various kinematic and dynamic models have been developed for this category of robots. Historically, kinematic models have been more widely used due to their simplicity and robustness, even when parameter estimates are imprecise. Existing kinematic models for skid-steer wheeled mobile robots (SSWMRs) and, more broadly, skid-steer mobile robots (SSMRs) often either neglect slip altogether or account for it through empirical relationships. However, these empirical approaches lack a clear physical interpretation, making them difficult to generalize across different robotic platforms.

Extended differential drive [7] is the most common and the simplest kinematic model for SSMRs. It based on the model used for differential drive systems. In this model the relation between the wheel velocities and the robot velocity is expressed as:

$$\begin{bmatrix} v_x \\ v_y \\ v_\phi \end{bmatrix} = \frac{r}{B\chi} \begin{bmatrix} \frac{B\chi}{2} & \frac{B\chi}{2} \\ 0 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} \tag{2.1}$$

where ω_l and ω_r represent the angular velocity of the left and right wheels, respectively. v_x, v_y, and v_ϕ denote the longitudinal, lateral, and angular velocities of the robot, respectively. r is the effective wheel radius, and B is the track width of the robot, as shown in Figure2.5.

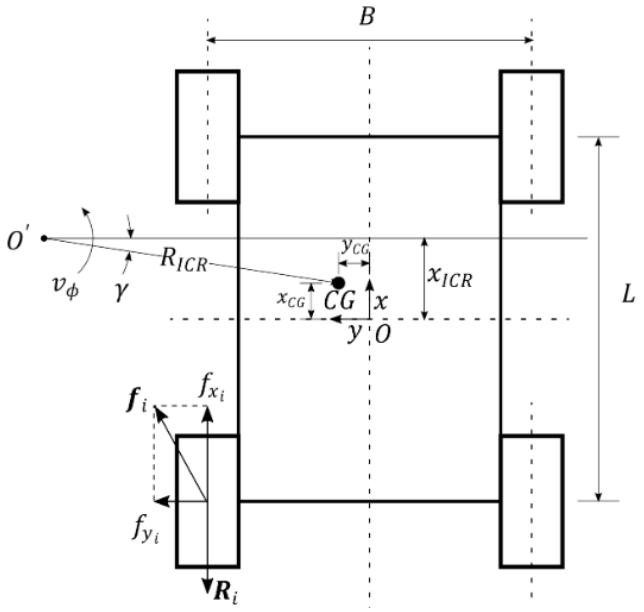


Figure 2.5: A Skid-Steer Mobile Robot Performing a Turning Maneuver

χ is a terrain-dependent parameter, called the ICR coefficient, where ICR stands for the instantaneous center of rotation. χ has values in the range $[1, \infty)$, where $\chi = 1$ corresponds to the case when there is no slippage, and the kinematic model would be equivalent to that of an ideal differential drive system.

The extended differential drive model assumes zero lateral velocity for the robot, i.e. $v_y = 0$, and that robot is also also symmetric. From the assumption that $v_y = 0$, the equation 2.1 can be simplified to,

$$\begin{bmatrix} v_x \\ v_\phi \end{bmatrix} = \frac{r}{B\chi} \begin{bmatrix} \frac{B\chi}{2} & \frac{B\chi}{2} \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix}$$

To solve for ω_l and ω_r, we need to calculate the inverse of the matrix on the right-hand side. The inverse of a 2x2 matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

is given by the formula:

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

For our matrix

$$\begin{bmatrix} \frac{B\chi}{2} & \frac{B\chi}{2} \\ -1 & 1 \end{bmatrix}$$

we compute the determinant Δ as follows:

$$\Delta = \left(\frac{B\chi}{2}\right)(1) - \left(\frac{B\chi}{2}\right)(-1) = \frac{B\chi}{2} + \frac{B\chi}{2} = B\chi$$

Thus, the inverse of the matrix is:

$$\frac{1}{\Delta}\begin{bmatrix} 1 & \frac{B\chi}{2} \\ 1 & -\frac{B\chi}{2} \end{bmatrix} = \frac{2}{B\chi}\begin{bmatrix} 1 & \frac{B\chi}{2} \\ 1 & -\frac{B\chi}{2} \end{bmatrix}$$

Now, applying this inverse matrix to the original equation, we get:

$$\begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} = \frac{2r}{B\chi}\begin{bmatrix} 1 & \frac{B\chi}{2} \\ 1 & -\frac{B\chi}{2} \end{bmatrix}\begin{bmatrix} v_x \\ v_\phi \end{bmatrix} \tag{2.2}$$

These equations allow us to compute the wheel angular velocities needed to achieve the desired linear velocity v_x and angular velocity v_ϕ for a skid-steer mobile robot.

2.4 LiDAR

LiDAR(Light Detection and Ranging) is a remote sensing technology which utilizes a short laser pulse to measure the range of its surroundings. It is the key component for perception of environment for out autonomous mobile robot. The fundamental operating principle involves emitting laser pulses and measuring the time taken for the reflected light to return, enabling highly accurate spatial mapping and object detection.

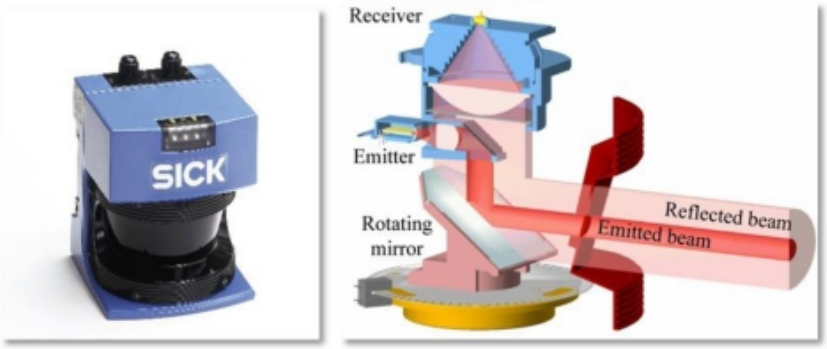


Figure 2.6: LiDAR and its Internals[8]

2.4.1 Operational Mechanism

The core principle of LiDAR can be mathematically represented by the distance calculation:

$$D = \frac{c \times t}{2} \tag{2.3}$$

- D = Distance

- c = Speed of light
- t = Time between laser emission and return

A typical LiDAR instrument is made up of:

- a rapid pulse(near-infrared light) emitting laser scanner
- a LiDAR sensor for detecting and collecting the returning light pulses, and
- a processor for calculating the time and distance and for builds the resultant data set, called point cloud.

For accurate remote sensing, measurements of time and space must be exact, so a LiDAR system will also use time-keeping electronics, an inertial measurement unit (IMU) and GPS.

2.5 ROS

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source[9]. ROS prevents "reinventing the wheel" meaning developers don't need to create basic robotics software components from scratch every time they start a new project.

ROS was originally developed by Willow Garage in 2007 to address the growing complexity of robotic software development. It provided essential features such as message-passing, package management, and visualization tools. However, as robotics applications expanded into industrial, real-time, and multi-robot domains, several limitations in ROS1 became evident, including:

1. Lack of real-time capabilities
2. Poor support for multi-robot systems
3. Limited cross-platform compatibility

To overcome these challenges, ROS2 was introduced, built on the Data Distribution Service (DDS) for better performance and scalability. Key improvements in ROS2 include:

1. Real-time capabilities: Support for deterministic execution
2. Improved security and reliability
3. Multi-platform support: Runs on Linux, Windows, and macOS
4. Better multi-robot communication

2.5.1 ROS Architecture

ROS2 is built to provide a more scalable, reliable, and real-time capable framework for robotic applications. Its architecture is designed to support multi-robot systems, industrial automation, and safety-critical applications.

1. **Data Distribution Service (DDS) Middleware**

   ROS2 uses the Data Distribution Service (DDS) for efficient, real-time, and scalable communication. DDS enables automatic discovery of nodes without a central broker, simplifying setup. It ensures reliable data exchange using configurable policies and supports security features like encryption and authentication. With built-in Quality of Service (QoS) controls, DDS allows developers to optimize latency, reliability, and message durability for different applications.

2. **Node-Based System**

   ROS2 follows a modular, node-based structure where independent nodes handle specific functionalities. Nodes communicate using well-defined interfaces, promoting flexibility and scalability. Managed nodes in ROS2 provide structured lifecycle management, improving resource efficiency and system predictability. This architecture enhances fault tolerance, making it suitable for complex robotic applications.

3. **Communication Mechanisms**

   ROS2 provides multiple ways for nodes to exchange information:

   - **Publish-Subscribe (Topics)**: Enables asynchronous data exchange, where publishers send messages on a topic, and multiple subscribers receive them. Commonly used for sensor data and control commands.
   - **Request-Response (Services)**: Allows synchronous communication where a client requests data or an action from a server, which then responds. Useful for on-demand operations.
   - **Action Interface**: Designed for long-duration tasks requiring periodic feedback. The client can send a goal, receive progress updates, and cancel the task if needed. Used in motion planning and navigation.

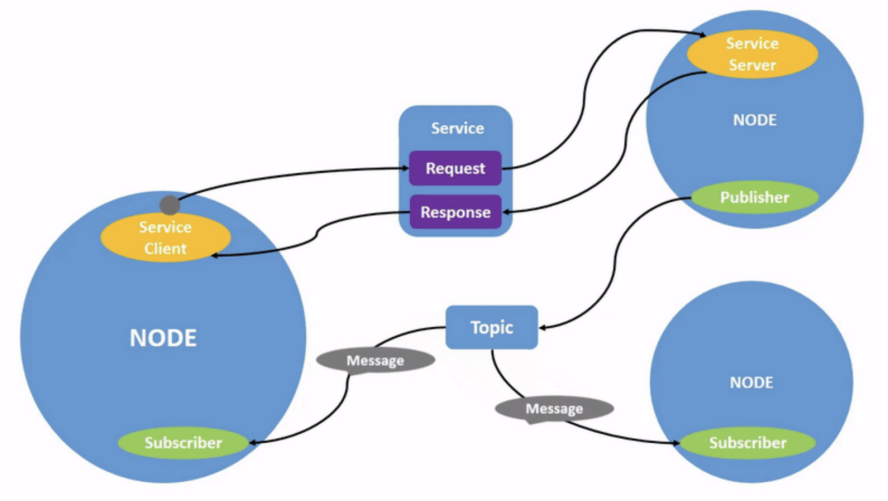


Figure 2.7: Core Concepts of ROS - Node, Topic and Service[10]

4. **Quality of Service (QoS) for Reliable Communication**

   ROS2 introduces QoS policies to enhance communication reliability. Developers can choose between best-effort delivery for lower latency or reliable delivery to ensure message integrity. Durability settings allow messages to persist for late-joining subscribers, while history depth determines how many past messages are stored. The deadline policy ensures timely data updates, and the latency budget helps maintain real-time constraints. These QoS settings are crucial for mission-critical applications like autonomous vehicles and robotic surgery, where delayed or lost data can lead to system failures.

2.6 Transformations

Coordinate frames are a set of axes (x,y,z) used to describe the position and orientation of objects in space.
To understand this, let us consider a simple two-wheeled differential robot with wheels, bases, and lidar as its main parts of dimensions as follows:

- Robot Base Dimensions: 200 mm x 200 mm
- Wheel Diameter: 75 mm
- LIDAR Height: 100 mm
- Obstacle Distance: 0.8 meters (800 mm) away from the robot's starting position
- Robot Speed: 80 mm/s

And let us define a three coordinate-frame for the robot: Base frame, wheel frame and LiDAR Frame, which has been described below:
Base Frame: This is typically the main reference frame for the robot. It is attached to the robot's base and is the origin for describing its position and orientation in the

environment.
Wheel Frame: Each wheel of the robot has its frame. These frames describe the rotation and movement of the wheels relative to the base frame.
LiDAR Frame: The LiDAR sensor also has its frame. This frame describes the LiDAR's position and orientation relative to the robot's base.
Each of these frames has a different origin, which means they provide unique perspectives. Static transformations describe the positions of components relative to each other at a fixed point in time.

1. **Base Position in World Frame**: Origin of the robot base.

$$\text{Base Position} = (0, 0, 0)$$

2. **Wheel Position Relative to Base**: Assuming the wheel is centered at the edge of the base:

$$\text{Wheel Position}=(100, 0, -37.5)$$

   Here, 100 mm is half the length of the base, and -37.5 mm is half the negative diameter of the wheel.
3. **LiDAR Position Relative to Base**: Attached at the height of 100mm

$$\text{LiDAR Position}=(0, 0, 100)$$

4. **Obstacle Position in World Frame**:

$$\text{Obstacle Position}=(800, 0, 0)$$

Therefore, the static and dynamic transformations calculations are calculated at each discrete time frame of 1 sec.

The above distance and position are calculated analytically in each sample, but what if we want to calculate the distance at some discrete time series or the distance between any two dynamic objects? We need to write a program. Hence, the same scenario can be achieved in ROS2 through tf2, and reinventing the wheel is not necessary. Transformations in ROS2 are useful for managing multiple frames over time. Transformations use tf2 packages.

Transformations are necessary because robots often have multiple parts that move relative to each other and to the environment. By keeping track of these movements, robots can accurately compute the positions and orientations of objects in their surroundings.

Time (s)	Robot Position (X,Y,Z)	Obstacle Position (X,Y,Z)	Distance (mm)
0	(0, 0, 0)	(800, 0, 0)	800
1	(80, 0, 0)	(800, 0, 0)	720
2	(160, 0, 0)	(800, 0, 0)	640
3	(240, 0, 0)	(800, 0, 0)	560
4	(320, 0, 0)	(800, 0, 0)	480
5	(400, 0, 0)	(800, 0, 0)	400
6	(480, 0, 0)	(800, 0, 0)	320
7	(560, 0, 0)	(800, 0, 0)	240
8	(640, 0, 0)	(800, 0, 0)	160
9	(720, 0, 0)	(800, 0, 0)	80
10	(800, 0, 0)	(800, 0, 0)	0

Table 2.2: Dynamic Transformations

2.7 Gazebo and RVIZ

Autonomous systems require extensive testing to ensure robustness before deploying in the real world environment and conducting physical experiments for each scenario is costly and time-consuming. Since the real world is affected by physical forces such as gravity, friction, inertia, collision and more phenomena. Therefore, there is the need for a physics based simulation platform to make the interaction realistic. Various simulation platforms like Webots, CoppeliaSim, Gazebo are available but among them gazebo is widely used because of its integration with ROS2.

Gazebo is a robot simulation environment that allows you to test and develop robots in a realistic 3D environment without needing physical hardware. The realistic real world simulations is accompanied by physics engines like ODE (Open Dynamics Engine), Simbody (Multibody Physics API), Bullet (a physics engine which simulates collision detection as well as soft and rigid body dynamics), and DART (Dynamic Animation and Robotics Toolkit).

2.7.1 Gazebo Architecture

Gazebo has plugin-based architecture and it can be customized by integrating various sensors, physic engines, robot models, and controller.The architecture of gazebo simulation can be explained as:

2.7.1.1 Gazebo Server (gzserver)

When executed, generates sensor data and real-world physics but nothing is displayed (hidden).
Libraries: Physics, Sensors, Rendering, Transport etc.

2.7.1.2 Gazebo Client (gzclient)

When executed, displays the GUI, robot, and world model, and visualizes the simulation.
Libraries: Transport, Rendering, GUI etc.

S.N.	Ubuntu Version	Recommended Gazebo Version	LTS Status
1	24.04 (Noble)	Harmonic	LTS
2	22.04 (Jammy)	Harmonic	LTS
3	20.04 (Focal)	Citadel	LTS

Table 2.3: Gazebo Version and Ubuntu Compatibility

2.7.2 Gazebo Plugin

The plugin are the chunks of code attached to a world, model or sensor. The messages are published in the /topic mentioned in plugin definition.
There are currently 6 types of plugins

- World
- Model
- Sensor
- System
- Visual
- GUI

The types of plugins and their contributions to simulating a two-wheel differential drive robot with a LiDAR sensor are as follows:

- **Robot Model (URDF/Xacro file)**: This file describes the robot's physical properties, including its dimensions, links, joints, and other kinematic and dynamic properties.
- **Gazebo Plugins for Differential Drive**: These plugins simulate the motion and behavior of the robot based on the differential drive system.
  - **Differential Drive Plugin**: Simulates the movement of the differential drive robot, using the inputs from the motor commands to control the wheel velocities.
  - **Controller Plugin for ROS**: Interfaces with the robot's control system through ROS, enabling the robot to be controlled via ROS messages for tasks such as velocity control and path following.
  - **Additional Plugin for Simulating Sensors**: Plugins that simulate various sensors on the robot, such as cameras, IMUs, and LIDAR sensors.

- **LIDAR Sensor Plugin**: Simulates the behavior of a LiDAR sensor, generating point clouds that represent the surrounding environment and enabling obstacle detection.
- **Wheel Encoder Plugin**: Simulates the wheel encoders, providing feedback on the robot's wheel rotations and enabling odometry calculations.

The types of plugin and their contribution in simulating a two-wheel differential drive robot with a LiDAR is shown as:

Plugin Type	Components for Differential Drive Robot with LiDAR
World	- Physics engine settings - Ambient lighting - Gravity settings
Model	- Differential drive control - Wheel joint control - Robot state management
Sensor	- LiDAR sensor configuration - LiDAR data acquisition - Sensor properties control
System	- Startup process control - Initial robot positioning
Visual	- Robot appearance - LiDAR visualization
GUI	- Custom user interface elements - Robot control panels

Table 2.4: Components for Differential Drive Robot with LiDAR

2.7.3 RVIZ

RViz commonly known as (ROS Visualization) is a visualization software that is used to visualize the message and topic. It allows users to see the perception of the environment.

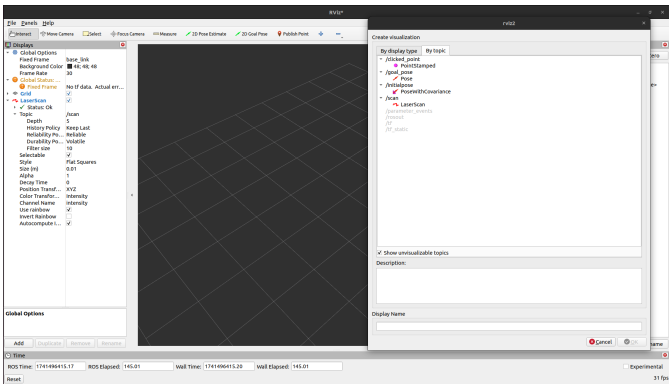


Figure 2.8: RViz Display

RViz2 is essentially a port of the original RViz visualization tool from ROS1 to the ROS2 framework, maintaining similar functionality while adapting to the new architecture.

2.8 SLAM

Mapping means to gather and tally measurements from the sensor into the world representation. These are classified on the basis of including features or not as feature-based maps, and other featureless metric maps which just represents the shape of the world. So here mapping relates to planning with the exploration of the surrounding.

Planning is the taking the actions what robot should do after each step. Suppose the robot comes near the obstacle or robot is taken to the unknown environment now it must plan each time to reach the goal. So There comes the term path planning, which is how robot approaches the goal. The robot should go through the most promising and low cost path to a given goal. In case of our robot, planing is handled through the NAV2 parameters.

Localization means where the robot is currently locating in the given environment, referencing to the map. There may occur error in the localization due to various reason like sensor noise, uncertainty with in the map.

It seems like for the localization, map is required and mapping cannot occur without localizing the robot. So the concept of SLAM is introduced.SLAM (simultaneous localization and mapping) is a method used for autonomous vehicles that lets you build a map and localize your vehicle in that map at the same time [11].

SLAM are classified into different types on the basis of how they represent and solve the localization and mapping problem as :

1. **Filter-Based SLAM:**
   Filter approaches treat SLAM as a state-estimation challenge, using probability models for continuous updates. The basic Kalman Filter works well for linear systems with Gaussian noise but has trouble with non-linear situations. The Extended Kalman Filter attempts to handle non-linearity through approximation techniques, though this can create inaccuracies. The Unscented Kalman Filter improves non-linear performance by using representative points to better approximate distributions, despite requiring more processing power. Particle Filters manage highly non-linear and non-Gaussian systems by using multiple samples to represent possible states, though they struggle with complex spaces. These filter-

ing methods perform best in organized environments with predictable movements and are good choices for robot positioning in controlled settings.

2. **Graph-Based SLAM:**
   Graph methods view SLAM as an optimization problem where robot positions and environmental features become nodes in a network, with relationships between them forming connections. Square Root methods efficiently optimize sparse networks through mathematical factorization. Incremental approaches allow real-time updates without needing to reprocess all data, making them suitable for ongoing mapping. General optimization frameworks provide flexible platforms supporting various error models and calculation methods. This approach handles environments with significant uncertainty effectively, though it typically requires more computational resources.

3. **Deep Learning-Based SLAM:**
   Machine learning approaches apply neural networks to identify features, create maps, and optimize results. Early bio-inspired systems were designed to copy animal navigation patterns. Some approaches combine traditional techniques with neural network feature identification to improve matching reliability. Streamlined neural designs attempt to balance accuracy with processing requirements to achieve real-time performance. While this area continues to develop, it shows promise for complex, unstructured environments, despite current limitations in processing demands and adaptability to new situations.

The other types of slam based on the trajectory estimation, optimization scope and computational complexity are full SLAM and Online SLAM.

1. **Full SLAM:**
   It is also referred to as the offline SLAM. The main aim of this SLAM is that it keeps record of the entire path it has been to make the map of the environment. After tracking all the detailed path of the map, it will be easier to estimate the best of past poses and maps using all available sensor data collected over time. In simple words, Full SLAM is remembering everywhere the robot has been and optimizing the full journey. This algorithm is computationally heavy. Examples of this SLAM are Graph-based SLAM, Pose Graph Optimization, and SLAM using Batch Optimization techniques.

2. **Online SLAM:**
   As opposed to full SLAM, Online SLAM focuses on estimating the current position of the robot and updating the map. It keeps a record of only the latest state and continuously updates the map. In simple words, Online SLAM only cares

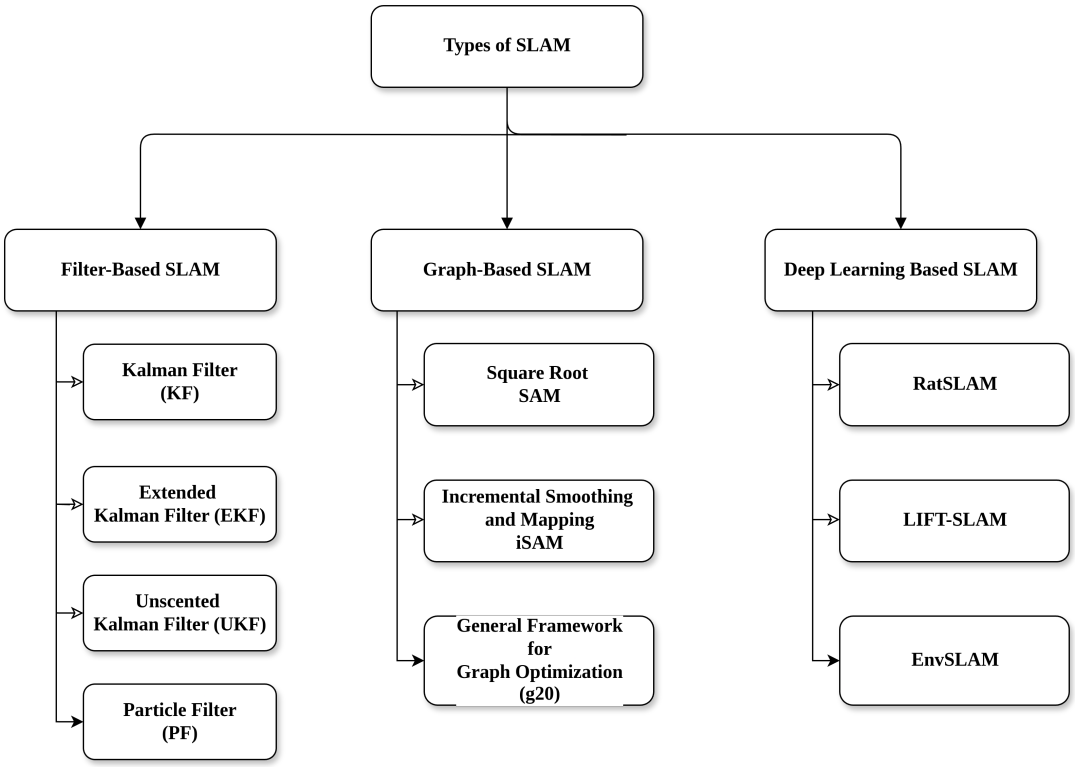


Figure 2.9: Types of SLAM Based on the Algorithm

about where the robot is currently located and what is ahead of the robot. This is more efficient for real-time navigation and robotic applications. Example algorithms of this SLAM are: Extended Kalman filter-SLAM, Particle Filter SLAM, and Google cartographer.

Similarly, on the basis of how the different slam processes (sensor input, localization and mapping) are executed and synchronized , SLAM can be classified as :

1. **Synchronous SLAM:**
   Synchronous SLAM is also known as traditional SLAM. It follows the step-by-step processes where all activities are occurs in sequential manner. The robot collects sensor data, processes it to determine its position (localization), updates the map, and then repeats this cycle. This SLAM ensures higher accuracy and consistency, as the next step will be carried out once the previous one is completed. This results in expensive computation in real-time applications. It is one of the limitations for the robot requiring quick decision-making, such as in self-driving cars or drone navigating in changing environment.

2. **Asynchronous SLAM:**
   This SLAM overcomes the limitations of the traditional SLAM. The step-by-step processes are now run independently and in parallel. Async SLAM continuously

updates different components whenever new sensor data is available. This approach improves efficiency and real-time performance, For example, in a self-driving car, LiDAR, cameras, and IMU sensors provide data at different rates, and Async SLAM ensures that each data source is processed as soon as it arrives, without causing delays in navigation.

The above-mentioned SLAM types form the foundation of various algorithms, yet the most prominent approaches include Lidar SLAM, VSLAM, and Hybrid SLAM. Lidar SLAM uses laser sensors to create accurate maps and is ideal for structured environments. VSLAM relies on camera-based visual data for mapping and localization, while Hybrid SLAM combines multiple sensor modalities, enhancing robustness in complex and dynamic environments.

These can be visually explained through a hierarchy as:

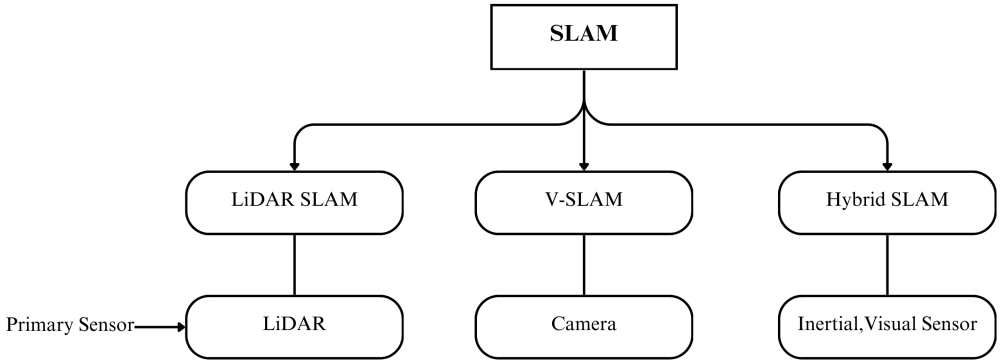


Figure 2.10: Types of SLAM Based on Sensor

2.9 Nav2

Movement is everything in the case of autonomous mobile robot. The robot has to move on the unpredictable environment where path gets blocked, obstacle appears suddenly. Now the robot has to make an instant decision, which is a challenging task. Hence, Nav2 is used for the navigation propose. ROS Navigation Stack is upgraded successfully to Nav2 to optimize autonomous vehicles. The robot can move through initial pose (location) to the goal using this package. Furthermore, Intermediary poses can be assigned between these two poses and the robot can easily navigate along with following the object or complete coverage navigation. This package now supports multiple robot types (Differential-drive, Ackermann, leggged). Nav2 expects standard inputs like TF transformations, map sources, and sensor (Camera, LIDAR) data, transforming these into precise velocity commands for robot motors. This leads the robot to perceive

their environment, plan sophisticated paths, control motor movements, avoid obstacles and create complex behavior. Its plugin-based architecture allows for easy customization, making it an extensible solution for robotics researchers and engineers seeking advanced navigation capabilities.

There are several servers in the Nav2 to handle the specific functions.

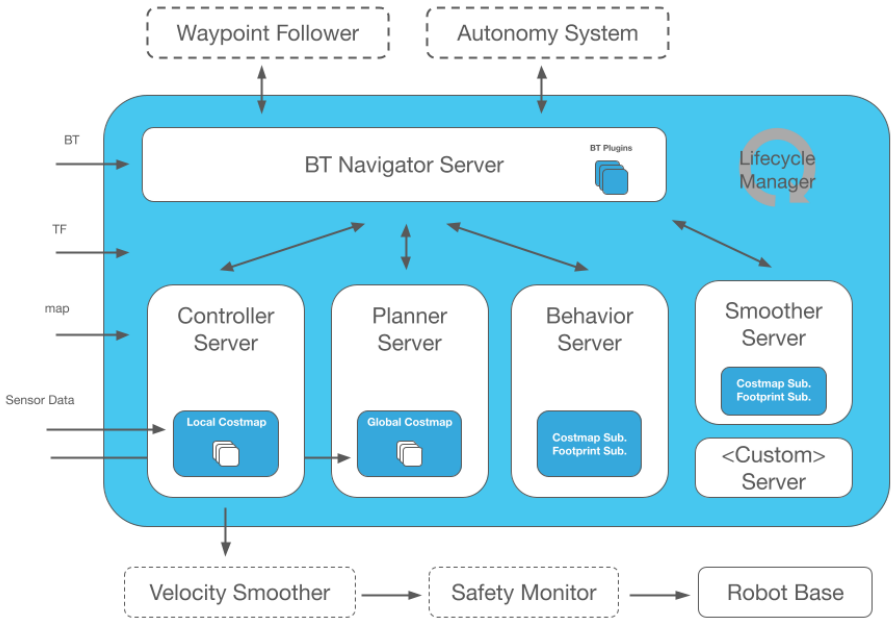


Figure 2.11: Various Servers in Nav2[12]

1. **Planner Server:**
   Planner server helps to plan the route for the robot to move from initial pose to desired goal. The task of this server is to determine the optimal path for the robot to navigate and cover all the other possible path. These planners can be divided into two types:

   - **Global Planner:**
     There would be the map of the environment consisting of obstacles, and terrains. Whenever the desired goal pose is fed into the robot, the robot will now generate the path from the initial (current) position to the goal position. This path is obtained from the global planner. Global planner uses the algorithms like A*, Dijkstra's algorithm etc. for path planning.
   - **Local Planner:**
     The robot, when moving in the map, meets the obstacle (static or dynamic) and need to change the path generated by the global planner. It uses the real time information from the sensor like LIDAR or camera. Hence, the robot now can move through the close spaces and avoid collisions.

2. **Controller Server:**
   Planner server is responsible for generation of the path and controller now uses that path and says the robot how to go in the planned path. It provides velocity commands by continuously adjusting speed and direction that control the robot's motors. If an obstacle appears, it makes changes in speed and direction to avoid collision. In other words, this server is the real time decision-making server.

3. **Behavior Server :**
   Controller server just changes the speed whenever the obstacle appears, but that mayn't be the smartest decision. So, behavior server ensures robot make smart decisions and can handle the situations like blocked path, obstacle temporarily blocking the way (example, moving vehicle, pedestrian), backing up or rotating in place when stuck. If any unexpected challenges occur, this server ensures no failure. Thus, this server also can be referred as problem-solving unit.

4. **Smoother Server:**
   Whenever the planner server plans the path for the robot to move, the path may be jerky or inefficient with unnecessary sharp turns. This leads to awkward movement. The smoother server changes these paths to more optimize path to ensure quality motion of the robot. It applies a smoothing algorithm that makes the robot to move in optimal path and without sudden stop.

5. **Waypoint Follower:**
   In a structured environment or let's say that a robot needing to do the same task repetitively, robots need to follow a predefined set of waypoints rather than going to a single goal. The robot must go to the specific checkpoints, pause for some time, perform any task if any like scanning a barcode or picking up an object, again resume functionalities and follow the same path in the structured environment. In such scenario, waypoint follower helps for a fixed sequence of movement.

2.9.1 Costmap

Robot requires costmap to understand the environment. Costmap is a grid based map that simply represents the cost of travelling various regions of the map. Cost here means difficulty. Nav2 uses a layered costmap system that contains:

- **Static Layer:**
  This layer includes all the known permanent obstacles in the environment. The static map layer is generally created with a SLAM algorithm.

- **Obstacle Layer:**
  The provided real time sensor data continuously updates the local map and creates the layer for any obstacle seen. This layer is for any obstacle that are not represented in the map during SLAM mapping.

- **Inflation Layer:**
  This is the layer that add a buffer zone around the obstacles. This increase the cost value in the surrounding area to prevent a robot from getting too close to an obstacle. Simply. It represents the robot's collision space within the map.

Chapter Three: LITERATURE REVIEW

The journey of Autonomous Mobile Robots (AMRS) began in the mid-20th century with the development of the early guide wire-floating robot. Shaki, developed at the Stanford Research Institute from 1966 to 1972, is considered the first mobile robot with widely autonomous logic capabilities[13]. Those developed systems depend on the predetermined paths and simple sensing mechanisms, limiting their application scope.

The first commercial application of mobile robots emerged in the 1980s with Automated Guided Vehicles (AGVs) that followed magnetic strips or embedded wires in factory floors[14]. While effective in controlled environments, these systems lacked true autonomy as they could not deviate from predefined paths or adapt to environmental changes. A significant breakthrough came in the 1990s with the development of more sophisticated localization and mapping techniques. The probabilistic robotics approach introduced by [15]enabled robots to build and maintain maps of their environment while simultaneously tracking their position within it. This development marked the transition from guided vehicles to truly autonomous systems capable of navigating without predetermined infrastructure.

The early 2000s saw rapid advancement in indoor robotics applications, particularly in warehousing and manufacturing environments. Kiva Systems (later acquired by Amazon) revolutionized warehouse logistics with their mobile robotic fulfillment system in 2003, demonstrating the commercial viability of large-scale AMR deployments [16]. The development of affordable laser range finders and RGB-D cameras accelerated indoor AMR adoption. The introduction of the Robot Operating System (ROS) in 2007 provided a standardized software framework that significantly reduced development time and facilitated knowledge sharing across the robotics community [17]. Furthermore, Indoor environments have flat surfaces, regular features, and controlled lighting conditions that simplify the navigation challenges. Thus,AMRs have currently seen its use in various fields like hospital environments to deliver medicine and transport samples, office buildings for security service, retail environments for material handling,etc.

While indoor applications flourished, outdoor autonomous navigation remained significantly more challenging. The DARPA Grand Challenge competitions (2004, 2005) and Urban Challenge (2007) served as critical catalysts for outdoor autonomous vehicle research, focusing primarily on road-based navigation [18]. These competitions demonstrated both the potential and limitations of existing technologies when applied

to less structured environments.The advancement in perception, localization, and path-planning algorithms result in various outdoor navigating robots like: last-mile delivery robot that tracks pedestrian areas, agriculture robots,campus delivery robots,etc.Outdoor navigation also comes with the challenges of variable terrain conditions,dynamic lighting and weather effects,multi-level features such as slope, ditches in roads,GPS signal variability and multipath errors,highly dynamic environments.

LiDAR is used to map and navigate the robots in certain environments. 2D LiDARs are mostly used in the indoor environment as they are simple and cost-effective; but in the case of outdoor environments these LiDARs are irrelevant. As noted by [19], 2D LiDARs operate on a single scanning plane, which makes them highly susceptible to misclassification of terrain features such as slopes, curbs, and uneven surfaces. Additionally, [20] demonstrated that 2D LiDARs often fail to detect obstacles with profiles that fall outside their scanning plane, creating potential safety hazards in dynamic outdoor settings. As the outdoor environment comes with multiple challenges as discussed earlier, use of 2D LiDAR brings inaccuracy in map and hazards in navigation as well. [21] found that maps generated using 2D LiDAR in outdoor environments contained up to 73% more localization errors compared to those produced using 3D LiDAR systems, particularly in environments with slopes greater than 10 degrees.3D LiDAR is thus preferable in outdoor mobile robotics.As [22] observed, 3D LiDAR sensors provide comprehensive spatial information that enables robots to perceive and navigate complex three-dimensional environments.

Our selection of the Mid-360 3D LiDAR was driven by several key advantages identified in recent literature. Mid-360 can capture the entire geometry of the environment,i.e it can identify the overhanging objects or obstacles,sloped terrain [23]. The Mid-360 LiDAR's multi-echo technology provides superior performance in adverse weather conditions by distinguishing between solid objects and particulate matter such as rain, dust, or light fog[24].This LiDAR have the maximum range of 200 meters that means it can plan long-range and navigate at particular speed in outdoor environments.

The point cloud data from Mid-360 LiDAR is accessed using the Livox SDK2 provided by Livox company. This sdk does the low level task of communicating with the LiDAR and getting the data to computer. From this point 'livox ros driver2' takes the raw data from sdk and converts it to ros2 compatible standard messages like 'sensor msgs/PointCloud2.msg'. Another key innovation is the use of an incremental k-d tree data structure, the iKD-Tree, to maintain the map. This structure supports incremental updates, including point insertion and deletion, and ensures dynamic re-balancing for efficient mapping[25].

The data from the MID360 LiDAR and integrated IMU can be extracted using the FastLIO package. FastLIO represents a significant advancement over previous LiDAR odometry approaches such as LOAM and LIO-SAM. Unlike the traditional two-stage (feature extraction and matching) approach used in LOAM-based methods, FastLIO employs an iterated Kalman filter that directly registers raw points to a map within a single framework. [26] demonstrated that this integrated approach reduces accumulated drift by approximately 25% compared to separated frameworks. The incorporation of IMU data provides critical motion estimates between LiDAR scans, addressing what [27] identified as a fundamental limitation of pure LiDAR methods—the inability to capture high-frequency motion. This is particularly relevant for outdoor navigation where uneven terrain induces complex robot motion patterns. A comparative study by [28] demonstrated that FastLIO achieves localization accuracy comparable to optimization-based methods like LIO-SAM while maintaining significantly lower computational requirements. FastLIO processed point clouds at 30Hz on an Intel i7 processor, while LIO-SAM required GPU acceleration to achieve similar frame rates. [29] found that FastLIO's tightly-coupled formulation provides superior performance in feature-poor environments and during rapid motion, two conditions frequently encountered in outdoor navigation scenarios. The selection of FastLIO for our implementation was further supported by its performance in comparative evaluations.[30]tested five state-of-the-art LiDAR-inertial odometry algorithms across diverse outdoor environments and found that FastLIO demonstrated the lowest average translation error (1.2% of distance traveled) in environments with significant elevation changes—a critical consideration for our outdoor application. We have though used FASTLIO2 which is upgraded from FASTLIO.FAST-LIO2 represents a significant advancement in real-time mapping and position tracking technology. By implementing the innovative ikd-Tree data structure, it drastically reduces computation time compared to traditional systems like LOAM that rely on resource-intensive processes. Its tightly integrated approach to combining LiDAR and inertial measurement unit (IMU) data offers superior performance in challenging environments with limited distinguishable features. Unlike competing methods such as LIO-SAM and LILI-OM that accumulate positional errors over time, FAST-LIO2 eliminates drift by continuously registering raw sensor data directly to the global map. The system's efficient design enables operation in large-scale environments while maintaining accuracy, and its optimized mathematical formulations further enhance real-time performance. These innovations collectively make FAST-LIO2 particularly valuable for applications requiring precise simultaneous localization and mapping in diverse settings.

SLAM algorithm is used to create map of the outdoor environment. SLAM algorithms

can be classified into two groups: the earlier algorithms that use Bayes-based filter approaches, and newer graph-based methods. Significant filter-based implementations available as ROS packages are GMapping and HectorSLAM. Cartographer and KartoSLAM are the major graph-based implementations available. GMapping is one of the most commonly used SLAM libraries, presented in 2007. It uses a particle filter approach to SLAM for the purpose of building grid maps from 2D lidar data. However, GMapping is not well suited for large spaces and fails to accurately close loops at an industrial scale. Additionally, filter-based approaches cannot be easily reinitialized across multiple sessions. HectorSLAM relies on lidar scan matching and 3D navigation filter based on EKF state estimation. This method focuses on real-time robot pose estimation and generates 2D maps with high update rate. Unlike other mentioned methods, Hector does not use odometry data, which can cause inaccurate pose and map updates when lidar scans arrive at a lower rate, or when mapping large or featureless spaces. HectorSLAM however does not provide loop closure capabilities, making it unsuitable for reliable mapping of large spaces or when using laser scanners with low update rates. KartoSLAM and Cartographer are both graph-based algorithms that store a graph of robot poses and features. Graph-based algorithms have to maintain only the pose-graph, which usually makes it efficient in handling resources, especially while building maps of a large scale. KartoSLAM uses Sparse Bundle Adjustment for loop-closure graph optimization. Cartographer consists of a front-end which is in charge of scan matching and building trajectory and submaps, and a back-end that does the loop closure procedure. The graph solver used in Cartographer is Google Ceres. Cartographer provides pure localization mode, when the user has a satisfactory map for usage. It also provides data serialization for storing processed submaps only. However, Cartographer has stopped maintenance and support from Google and has been largely abandoned. In addition, it does not not build suitable maps for annotation and localization with other localization software packages on robotic platforms without exceptional odometry. The unusual complexity of the software makes it challenging to modify or resolve seemingly simple issues, making it not suitable for many applications.[31]

Later, at ROSCon 2019, Macenski introduced a 2D SLAM package. It is built on Open Karto (Konolige et al., 2010), a pose-graph optimization SLAM method, allowing for efficient use of the robot's resources while mapping large environments and employing similar local and global approaches.The research on autonomous navigation robots in outdoor environments spans several studies, each contributing unique insights and methodologies. Brenneke et al. proposed a method using 3D Laser Range Data for Simultaneous Localization and Mapping (SLAM) in outdoor environments. This approach combines 3D and 2D algorithms to improve accuracy in complex terrains [33]. In recent developments, Wang et al. introduced a robust and real-time outdoor local-

Table 3.1: Comparison of Different Popular SLAM Algorithms [32]

Algorithm	**Type**	**Advantages**	**Limitations**	**Recommended Environment Size**
GMapping (2007)	Particle Filter-Based	Simple implementation, low computational requirements	Struggles with large environments, requires odometry data	Small Up to 20,000 sqft
Karto (2010)	Graph-Based	Efficient loop closure, good balance of accuracy	Performance can degrade in larger environments	Medium Up to 50,000 sqft
Hector SLAM (2011)	Grid-Based	No odometry required, Can work with low cost lidars, works well in gps-denied environment	Sensitive to high-speed motion, struggles in dynamic environments	Large Medium (Up to 50,000 sqft)
Cartographer (2016)	Graph-Based	High accuracy, handles complex environments	Complex configuration, less community support	Large Up to 100,000 sqft
SLAM Toolbox (2019)	Hybrid	Versatile with multiple mapping modes	Requires more computational resources	Very Large Over 100,000 sqft

ization technique based solely on LiDAR. Their grid matching algorithm enhances data association and reduces computational overhead [34]. Similarly, Liu et al. developed a fusion method integrating binocular vision, 2D LiDAR, and IMU for outdoor localization, presenting a theoretical framework grounded in graph theory to enhance localization accuracy [35].

Further advancements were made by Hong and Huang, who conducted a comprehensive comparison of 3D reconstruction techniques between SLAM and Structure-from-Motion (SfM) in outdoor settings. Their findings highlight the strengths and weaknesses of each approach, offering insights for selecting appropriate techniques based on specific application requirements [36].Wijayathunga et al. addressed the challenges and solutions for autonomous ground vehicles in outdoor environments, emphasizing the significant role of sensor fusion and adaptive algorithms in navigating dynamic and unpredictable terrains [37]. Concurrently, Jia et al. compared 2D and 3D LiDAR-based person detection on mobile robots, finding that 2D LiDAR sensors are more susceptible to errors in complex environments, while 3D LiDAR offers superior detection capabilities[38]. Finally, Kim et al. focused on the development of an autonomous mobile robot designed for urban environments. Their work highlighted comparative experiments that reveal the weaknesses and strengths of different sensor configurations and algorithms[39]. These studies collectively advance the field of autonomous outdoor navigation, addressing various challenges through innovative methodologies and comprehensive evaluations.

Though there are various SLAM algorithm, we use SLAM toolbox which is capa-

ble of maintaining consistent maps and support loop closures. [40] demonstrated that SLAM Toolbox achieves real-time mapping performance on large-scale environments ($>8000m^2$) without requiring downsampling or map reduction techniques.SLAM toolbox also has the ability to real time update and refine of maps during runtime.SLAM Toolbox provides seamless integration with the ROS 2 Navigation2 framework, enabling efficient planning and execution in the generated maps.The conversion from 3D point clouds to 2D laser scans represents a pragmatic approach to outdoor navigation that balances the comprehensive sensing capabilities of 3D LiDAR with the computational efficiency of 2D navigation algorithms. [41] demonstrated that while this approach necessarily discards some 3D information, properly configured conversions that account for terrain slope and obstacle height can maintain navigation safety while significantly reducing computational requirements. Our implementation utilizes a height-sliced approach similar to that proposed by [42], where multiple virtual scan planes are extracted from the 3D point cloud and combined into a consolidated 2D representation. This approach preserves critical obstacle information while enabling efficient processing by SLAM Toolbox. The combination of FastLIO for robust odometry and SLAM Toolbox for mapping creates a comprehensive SLAM solution that addresses the unique challenges of outdoor navigation. As [43] observed in their comparative study of integrated SLAM systems, this hybrid approach effectively leverages the strengths of both 3D sensing and 2D mapping to achieve reliable localization and navigation in complex outdoor environments.

Nav2(Navigation2) is used to plan the path and navigate the robot in the given path. Navigation2's most significant strength lies in its robust, community-driven development model. As documented by [40], Nav2 has over 150 direct contributors and represents one of the most actively maintained navigation frameworks in robotics. This collaborative approach offers several advantages.There are continuous pu;l requests every month for the navigation2 and the contributors are from academia,industry and research institutions. Nav2 is supported for multiple robotic platforms and is hardware configured.Despite being widely used, the navigation system Nav2 has significant shortcomings when it comes to handling difficult outdoor situations, particularly when it comes to non-flat terrain. According to studies by [29], Nav2 struggles with 3D topography and is primarily designed for flat, 2D maps. It struggles to navigate complicated natural landscapes (such as forests or rocky trails), high slopes (over 15 degrees), rough or uneven terrain, and places where surfaces vary (such as from grass to gravel). Because of these drawbacks, it is less dependable for outdoor navigation in the real world where the terrain is not entirely level or consistent.The decision to use a 2D-based navigation approach is primarily based on its computational efficiency. It requires fewer resources, operates faster, and is compatible with a wide range of robotic systems, mak-

ing it ideal for robots with limited computing power. However, 2D navigation has its limitations. Research shows that it performs up to 62% worse in unstructured environments, struggles with obstacle detection in complex terrains, and lacks the ability to optimize paths effectively in challenging conditions. To address these limitations, various strategies have been proposed, such as incorporating layered costmap techniques, using height-based filtering from 3D point clouds, and applying machine learning for terrain classification and adaptive planning.Despite its limitations, Nav2 remains a preferred system due to its maturity, extensive community support, and compatibility with the ROS 2 framework. It offers a modular architecture, allowing for easy customization and plugin integration. Additionally, it performs well in controlled, structured environments, where obstacles are predictable, and its behavior is reliable. In our research, we acknowledge the limitations of Nav2 but utilize its strengths in relatively flat areas, enhancing its performance with custom costmap layers and point cloud preprocessing to better represent terrain.When comparing different navigation approaches, traditional 2D systems like Nav2 are efficient in structured environments but struggle with complex terrain. In contrast, 3D navigation systems offer better terrain handling but require higher computational resources and are suited for unstructured environments. Hybrid approaches combine the advantages of both 2D and 3D systems, providing a balance between computational efficiency and adaptability for mixed environments.

Chapter Four: METHODOLOGY

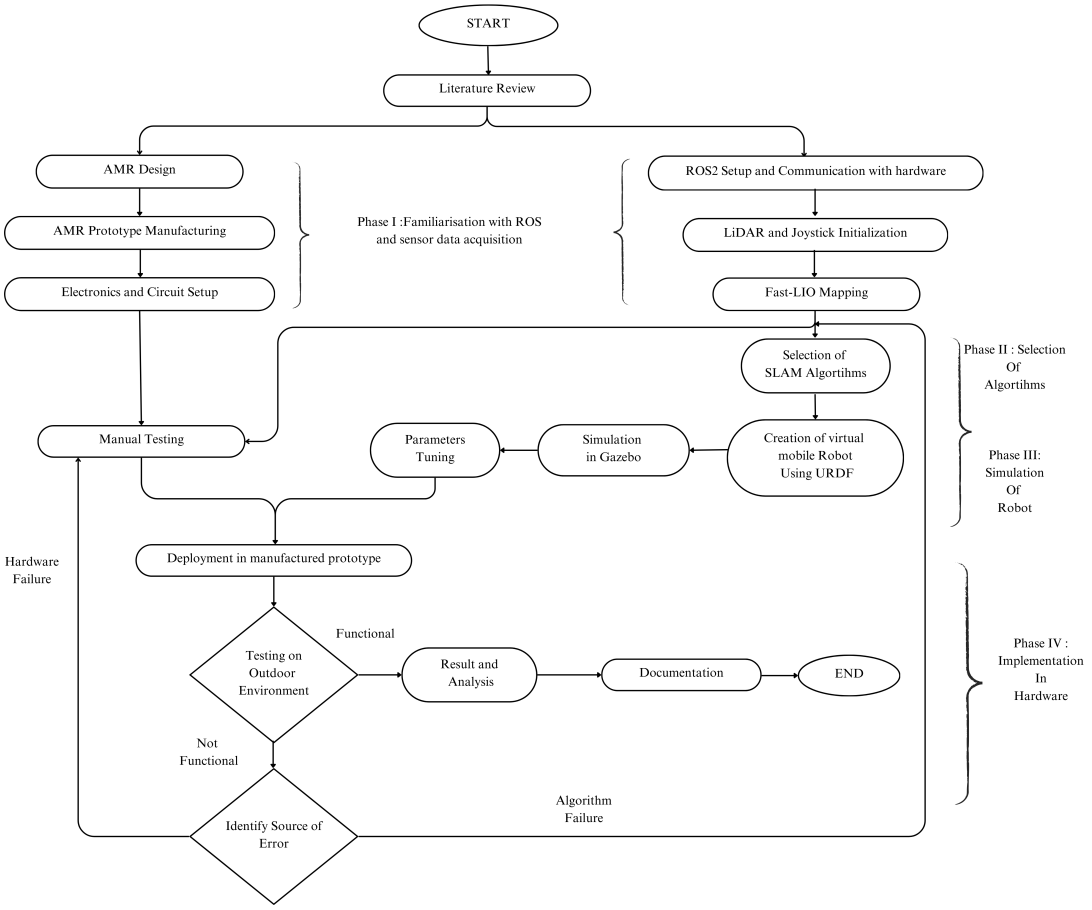


Figure 4.1: Flowchart showing Methodology of the Project.

This project was completed in four major steps: Familiarization with ROS and sensor data acquisition, selection of algorithm for SLAM, Path Planning, simulation of robot in virtual environment, and implementation in hardware.

4.1 AMR Design and Fabrication

The fabrication of the AMR involved using the existing materials and 3D printing custom components for cost effectiveness, durability and functionality. This included following phases:

4.1.1 Conceptual Planning and Design Considerations:

The design of the AMR chassis began with careful conceptual planning where following importance factors were considered:

1. **Stability:**The frame had to be able to hold the weight of the robot and all its components without sacrificing structural integrity.

2. **Clearance:**Clearance was needed in order to ensure free movement by ground and road.

3. **Efficient Mobility:**The chassis needed to be made for quick movement, particularly at low turning radius and maneuvering in small places.

During the initial planning phase, we thought of designing a custom chassis. We discovered a scrap hoverboard chassis that met the clearance we required. The hoverboard chassis provided the clearance, but only two wheels, which was unstable and lacked maneuverability. This led to the redesign of the chassis.

4.1.2 CAD:

To solve the stability problem of the hoverboard chassis, we chose to weld two individual chassis together. We did so because the materials of the hoverboard chassis were weldable, and welding provided us with the flexibility to make changes to the structure if needed. Therefore, we created the detailed CAD of the prototype to manufacture, with custom components needed for mounting.

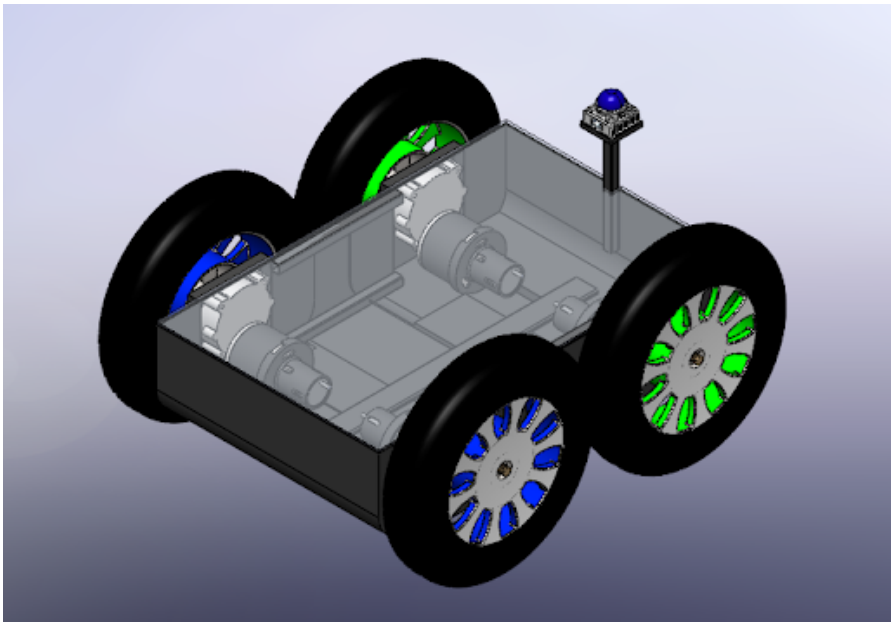


Figure 4.2: Assembled CAD Model of Robot

4.1.3 Material Selection:

The pre-existing structure of the hoverboard was made up of mild steel so we arc welding was used for joining the two hoverboard chassis. Remaining structural components

like tubes and supporting plates were also made up of mild steel because of its weldability, strength, and low cost. For components like sensor and encoder mounts and brackets, PLA (Polylactic Acid) was utilized for 3D printing.

4.1.4 Chassis Fabrication:

The structures of the frame consisting of mild steel components were welded based on the dimensions from the CAD. This ensured a precise alignment when welding to achieve a rigid and stable structure. After welding, the chassis underwent an evaluation process in terms of strength and balance, to confirm its ability to withstand the expected operating conditions without deformation.

Initially, the center-to-center distance of the side wheels was set at 700mm. However, during testing, it was observed that this configuration resulted in significant resistance while turning due to skidding. To improve maneuverability, the wheel distance was reduced to 420mm, which was the least possible for our configuration while maintaining stability.

We used plywood to cover the top of the robot chassis, securing it by welding bolts to the chassis and fastening them with wing nuts. This design allows for easy tightening and loosening, facilitating modifications or debugging of any circuitry faults inside the chassis. Additionally, we integrated foam into the plywood, cutting slots to securely hold the laptop. This ensures that the laptop remains stable and protected while the robot is in operation.

4.1.4.1 LiDAR Mounting System:

The LIVOX Mid-360 comes with four mounting holes at its base for secure attachment. We thus designed the 3D printed mounting component that can attach LiDAR and also be fitted to a quarter-inch square pipe connected to the robot's chassis. The LiDAR is positioned in such a way that its field of view (FOV) is just above the highest point of the wheels; so that there would not be any interference by the wheels to the point cloud data.

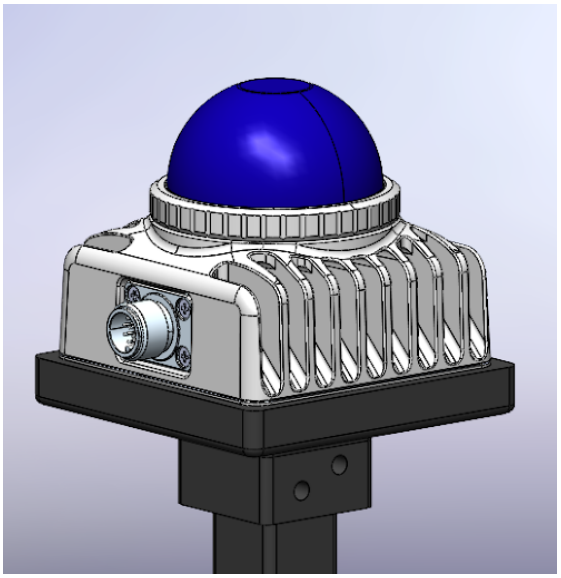


Figure 4.3: Mid-360 LiDAR with Mount.

4.1.4.2 Encoder Mounts and Coupler Fabrication:

The encoders were attached to the wheel system to enable the movement tracking with high precision. Custom designed 3D-printed encoder mounts were made to hold the encoders in fixed position, keeping center alignment with the backend shaft of the motors. A coupler was designed and fabricated to connect the encoder shaft to the wheel's rotational system.

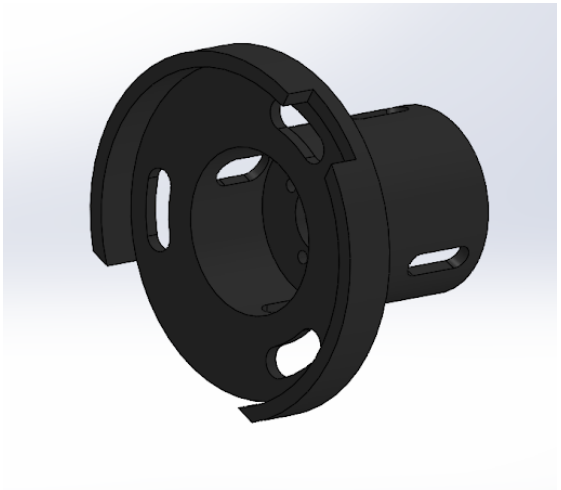


Figure 4.4: CAD model of Encoder Mount.

4.1.4.3 Assembly:

Once all individual components were fabricated, they were assembled together to build the full AMR prototype. Proper integration process included fixing all mechanical and electronic components firmly but proportionately to allow modifications in the model

later. After assembly, the AMR prototype have the following key dimension:

1. Overall length: 830mm
2. Overall width: 702mm
3. Overall height (including LiDAR): 457mm
4. Wheel center-to-center distance (after modification): 420mm

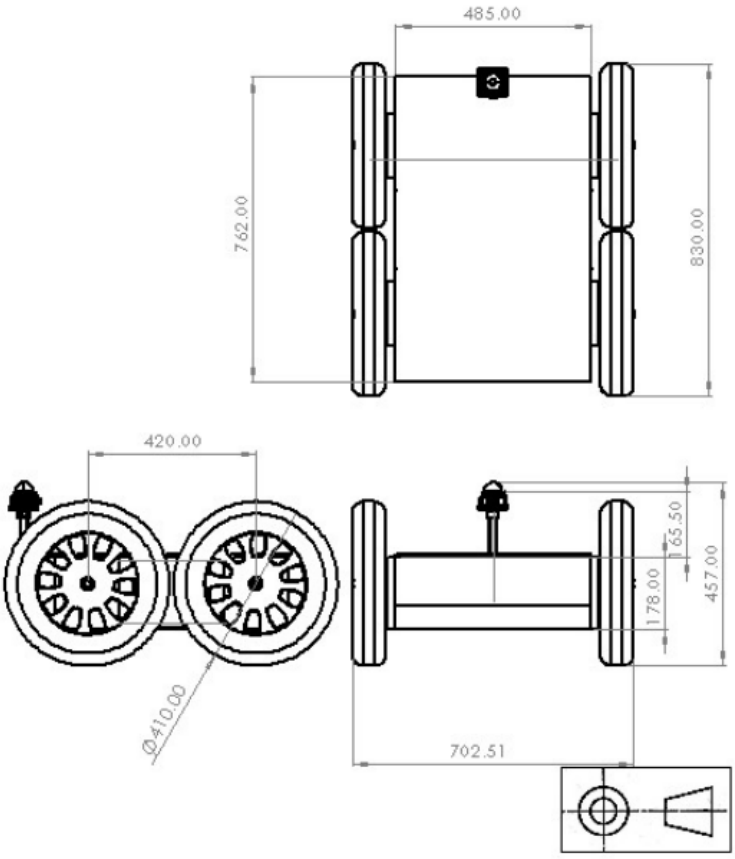


Figure 4.5: CAD Model of Robot

4.2 Electronics and Circuit setup

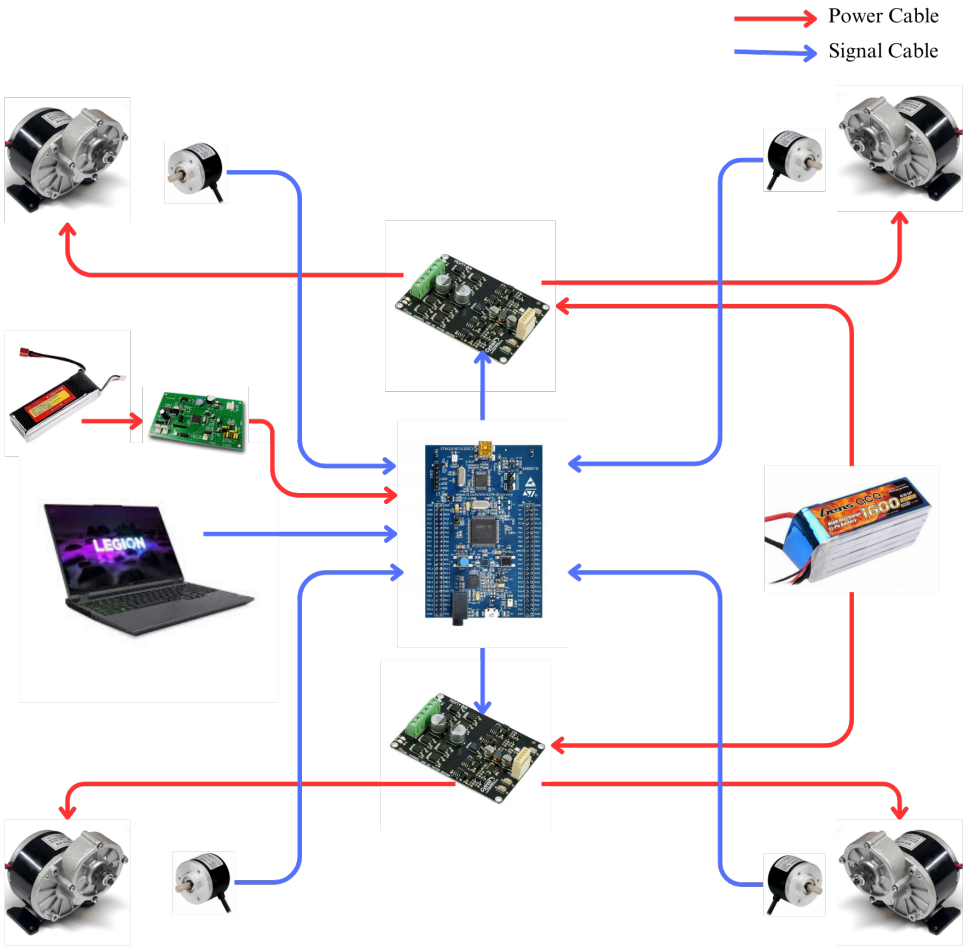


Figure 4.6: Complete Schematic of Electronic Circuit

After fabrication of the chassis and assembly of the manufactured parts along with motors and wheels, we moved on to setting the circuitry required to make the robot move. The following are the three circuit boards use in the robot:

1. **STM32 Breakout Board:**
   The breakout circuit board used for STM32 was prepared by robotics club pulchowk campus. It consists of all the necessary pinouts and ports for encoders and motor signal pins.

2. **Power Distribution Board:**
   The power distribution board takes 12V from a LiPo battery and it uses buck convertors to produce standard voltages of 5V and 3.3V required for parts like encoder and stm32. It is responsible for distribution of power to the parts.

3. **Motor Driver:**

The motor driver used is Cytron MD20A motor driver. It is a dual channel H-bridge motor driver with maximum current rating of 20A; which was more than enough for our motors.

4. **DC motor:**
   Four 24 V, 250 W DC motor salvaged from a scrap hoverboard was used to drive the robot. The motor is powerful enough to carry loads of up to two persons.

5. **Encoder:**
   And for each of the wheels, a quadrature rotary encoder was used to measure the position and velocity. It provides a resolution of 400 pulses per revolution which is enough for our application

For the setup of microcontroller, all the necessary timers and peripherals like UART are setup using stm32cubeMx software

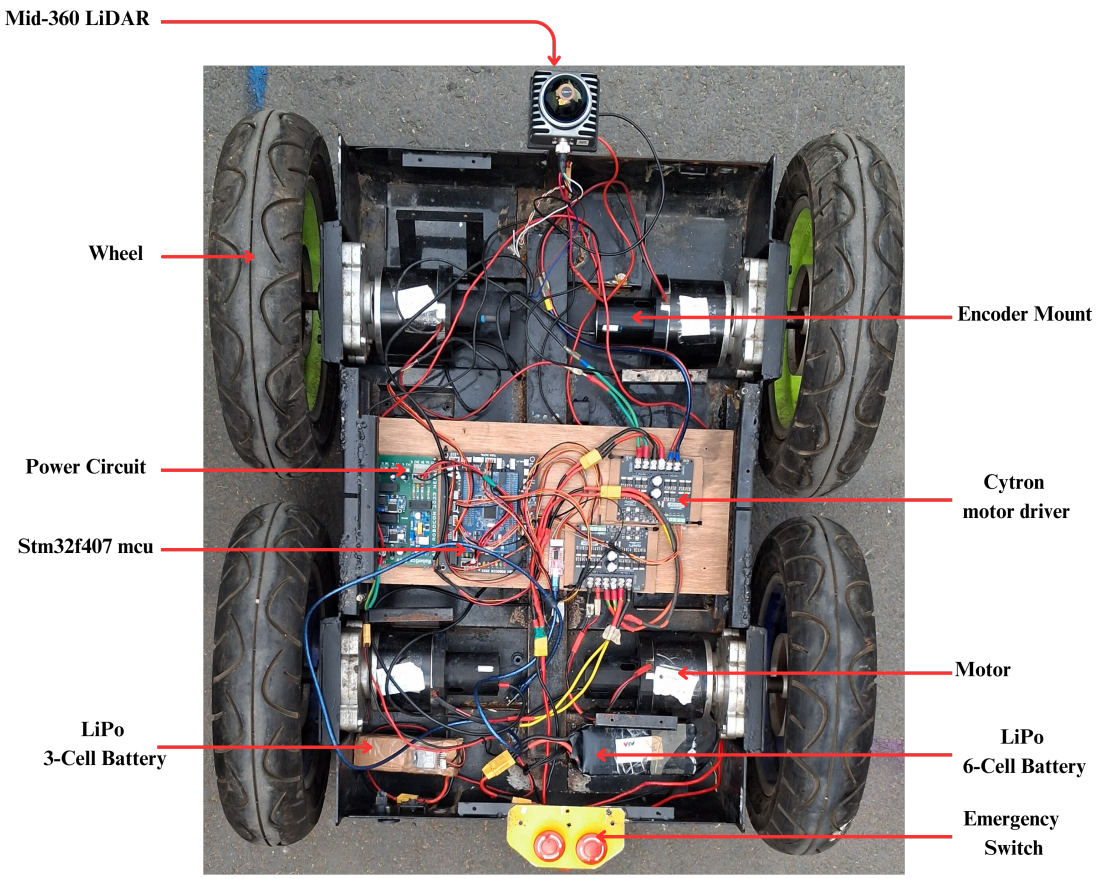


Figure 4.7: Complete Assembly of the System

4.3 ROS2 Setup and Communication with Hardware

To power the robot's computing and control, a laptop running Ubuntu 22.04 was used instead of a Raspberry Pi. This decision was made due to the unavailability of the Rasp-

berry Pi board and its potential performance limitations. ROS2 Humble was installed following the official documentation, and all necessary dependencies were set up to create a stable development environment.

After setting up ROS2, we explored its architecture and communication system by following the official tutorials. The main goal was to establish reliable communication between a joystick controller, the STM32 microcontroller, and the robot's motors to enable both manual and autonomous operation.

For manual control, the `joy` node was used to interface with the joystick and publish its data in a ROS2-compatible format. A custom ROS2 node was developed to extract the relevant joystick axis values and publish them as `/cmd_vel` (Twist) messages. Another custom node was responsible for subscribing to the `/cmd_vel` topic and transmitting these commands to the STM32 microcontroller over UART.

UART communication is a key component of the robot's operation, as it allows the laptop to send motion commands to the microcontroller. To ensure reliable data transmission, a structured serial communication protocol was implemented. Each data packet contains the following components:

- **Start Byte**: Marks the beginning of a packet to synchronize communication between the laptop and the STM32.
- **Twist Message**: Contains velocity commands (linear and angular) to control the robot's movement.
- **Cyclic Redundancy Check (CRC)**: Ensures data integrity by detecting any errors during transmission.

The STM32 microcontroller processes these packets, extracts the velocity commands, and adjusts motor speeds accordingly. The system was tested under various conditions to verify the robustness of UART communication, including handling packet loss and error detection.

The received Twist message consists of linear velocities (v_x, v_y) and angular velocity (ω). To translate these into motor commands, an inverse kinematics algorithm based on the skid-steer kinematic equation (Equation 2.1) is implemented. This algorithm calculates the angular velocities required for each wheel based on the desired motion of the robot.

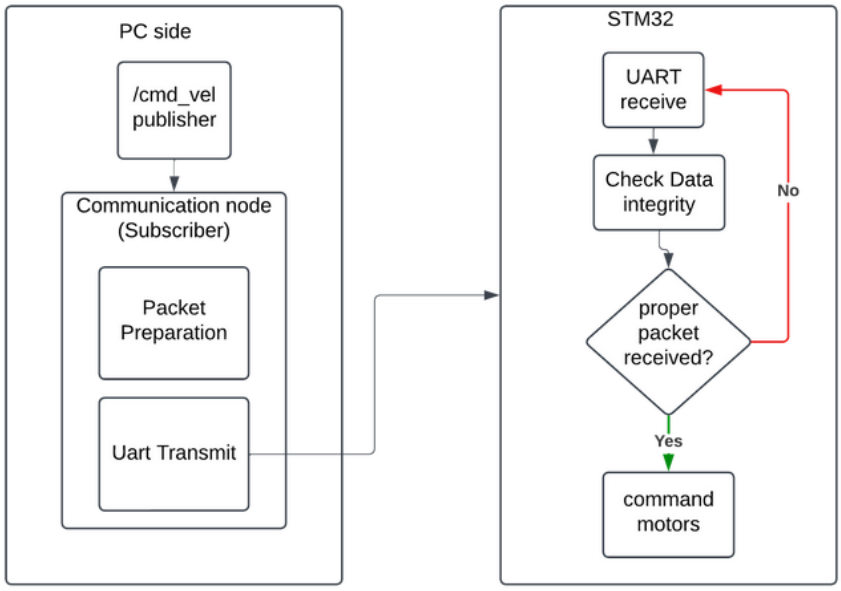


Figure 4.8: Schematic for Communication Between ROS and STM

To maintain accurate motor control, encoders measure the actual angular velocities of the wheels. A closed-loop PID controller is then used to adjust motor speeds and ensure they match the commanded values. This feedback system helps maintain smooth and precise movement, even in varying terrain conditions, by correcting any discrepancies between the desired and actual velocities.

4.4 Lidar Initialization

The LiDAR sensor (Mid-360) is interfaced using the Livox Software Development Kit (Livox SDK), which is provided by the manufacturer. This SDK handles low-level communication with the sensor and extracts point cloud data.

To integrate the LiDAR with the system, the Livox SDK was installed and set up on the PC following the official documentation from Livox Tech. Once configured, the raw point cloud data could be extracted from the sensor. However, this data is not directly compatible with ROS2, as ROS2 uses the `sensor_msgs/PointCloud2` format for point cloud representation.

To handle this, the `livox_ros_driver2` package was used. This package converts the raw data from LiDAR to ROS2 compatible `sensor_msgs/PointCloud2` format . This ensures seamless integration of the LiDAR data with other ROS2-based modules, allowing for further processing and use in navigation, mapping, and perception tasks.

However, the published point cloud data is in a custom message format that includes lidar id and some custom bytes. `livox_interfaces::msg::CustomMsg` within the lidar frame. To be usable by other ROS2 components, it must be converted into the standard `PointCloud2` message type `sensor_msgs::msg::PointCloud2`. This conversion allows the point cloud data to be easily integrated with other parts of the ROS2 ecosystem.

4.5 Fast-LIO Mapping

After reviewing multiple algorithms for odometry and 3D map generation, Fast-LIO was selected due to its superior accuracy and ability to create highly detailed 3D maps. Fast-LIO fuses LiDAR feature points with IMU data using a tightly coupled iterated extended Kalman filter (iEKF), enabling robust navigation even in fast-motion, noisy, or cluttered environments where traditional approaches may struggle.

Fast-LIO implements incremental mapping using an iKD-Tree structure, allowing it to achieve real-time performance at LiDAR rates exceeding 100Hz. Given the accuracy of the odometry provided by Fast-LIO, only its output was used for localization. The encoder data from the wheels was not used for odometry but was instead utilized solely for closed-loop PID motor control.

4.6 Selection of SLAM algorithms

This section involved exploring and reviewing different SLAM algorithms and libraries available for LiDAR Sensor based on their strengths and weaknesses.

Table 4.1: Comparison of SLAM Algorithms

SLAM Algorithm	**Description**
Hector SLAM	Works without odometry, appropriate for lightweight applications.
GMapping	Simple, lacks loop closure, lower accuracy compared to modern methods.
SLAM Karto	Graph-based SLAM, accurate for 2D mapping.
SLAM Toolbox	Advanced, flexible as per need, supports lifelong mapping capabilities.
Cartographer	Supports both 2D and 3D SLAM, but requires high computational power.

After assessment,we concluded that Slam_toolbox is the best choice.It offers balance between performance and flexibility, supports large-scale mapping, and has a toolset that can be tuned to our particular needs.

It has been modified to take a laserscan converted from a 3d point cloud given by MID-360 using FastLIO and create a 2D map.

4.7 Setting up Gazebo Simulation

Gazebo simulation is an excellent way to test algorithms and validate their performance before implementing them on actual hardware. This approach minimizes the risk of damaging expensive components, especially when testing navigation and control algorithms. Since our robot carries essential onboard electronics, including a laptop, avoiding real-world crashes during early-stage testing is crucial. Therefore, we first set up a simulation environment and selected appropriate algorithms for SLAM and navigation within Gazebo.

The simulation system consists of the following two parts:

4.7.1 Setting Up the Robot URDF

To perform the simulation, the first step was to create the URDF (Unified Robot Description Format) model of the robot. The URDF file provides a complete description of the robot, including its physical structure, visual appearance, and collision properties. Essentially, it acts as a digital blueprint, defining the robot's components, their connections, and how they interact with the environment.

The robot model is constructed using two fundamental elements, links and joints.Links represent different parts of the robot, such as the base, wheels, and LiDAR sensor. Define how links are connected and move relative to each other. For example, a wheel is attached to the base using a revolute joint (allowing rotation), while a LiDAR sensor is typically mounted with a fixed joint (no movement).

Each joint in the URDF defines a transformation from one link to another, ultimately forming a transformation tree. The transformation structure for our robot is illustrated in Figure 4.9.

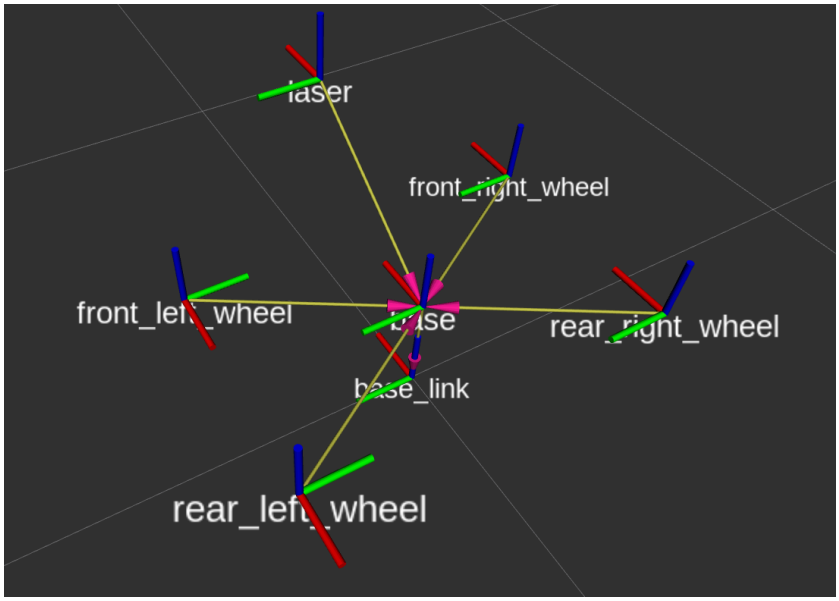


Figure 4.9: Transformation Tree of the Robot in URDF

After defining the robot's structure, mesh files exported from the SolidWorks CAD model (in STL format) were integrated into the URDF to achieve a realistic visual representation of the robot. However, to simulate actual behavior, Gazebo plugins were incorporated to model sensor data and movement. Some of the key plugins used include:

- **Gazebo Ray Sensor Plugin (libgazebo_ros_ray_sensor.so):** Simulates LiDAR by generating range data. It can be customized with parameters like scan range, update rate, and noise model to match real-world performance.
- **Gazebo IMU Plugin (libgazebo_ros_imu_sensor.so):** Simulates an Inertial Measurement Unit (IMU), providing acceleration and orientation data.
- **Gazebo Differential Drive Plugin (gazebo_plugins::GazeboRosDiffDrive):** Simulates the motion of a differential drive robot by varying wheel speeds.

With the URDF setup and plugins configured, the robot model in Gazebo closely resembles its physical counterpart in both appearance and functionality.

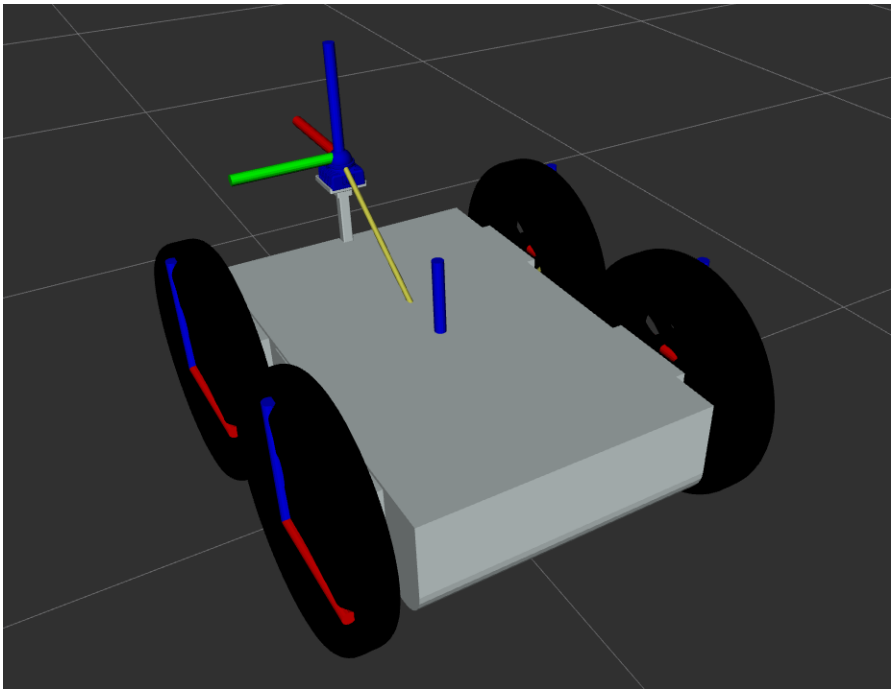


Figure 4.10: Robot URDF

4.7.2 Setting Up the Simulated World

To test SLAM and navigation algorithms, the robot requires an environment that mimics real-world conditions. A simple world was initially designed using Gazebo's built-in building editor. Default models such as walls, rooms, and pathways were placed to resemble an indoor testing environment. To simulate realistic obstacles, various default models like construction cones, tables, and poles were added. These objects help evaluate the robot's ability to navigate around obstacles and perform SLAM effectively.

Once the environment was set up, the robot was spawned into the world, positioned at a suitable starting location. Now, With the robot placed in the simulated world, SLAM and navigation algorithms were implemented and tested. The performance was analyzed by observing the generated maps and testing the robot's ability to navigate autonomously. By designing an environment similar to real world, we ensured that the robot could be tested under controlled conditions before deploying it in a real environment.

4.8 Implementing SLAM in Simulation

Testing algorithms in simulation is crucial before actually testing them on real hardware. SLAM requires the transformation of the robot, which is defined by the URDF of the robot. For 2D laser scan data, we could either directly use a 2D LiDAR plugin in ROS2, but to better mimic the actual hardware, a 3D LiDAR plugin was used, which

provided point cloud data. However, the point cloud data published by the plugin was in the standard ROS2 format `sensor_msgs/PointCloud2`, while Fast-LIO expected it in the `livox_ros_driver2/CustomMsg` format. To address this, an additional package was used to convert the standard ROS2 message to a Fast-LIO compatible message format.

After ensuring the proper message types and topic naming conventions, Fast-LIO was implemented. Since SLAM requires 2D laser scan data on the `/scan` topic, the `pointcloud_to_laserscan` node was used to convert the 3D point cloud data to 2D laser scan data.

Once the above packages were set up and confirmed to be working, the robot was loaded into a simulated environment in Gazebo, and the nodes were launched through a launch file. The robot was then controlled using `teleop_twist_keyboard` to send `cmd_vel` commands through the keyboard. By issuing commands via the keyboard, the robot moved through the environment, and a 2D occupancy grid of the environment was obtained.

4.9 Implementing Nav2 Stack in Simulation

After successfully generating a 2D occupancy grid using SLAM, the next step was to implement autonomous navigation using the Nav2 stack. The Nav2 stack enables path planning, localization, and obstacle avoidance, allowing the robot to navigate autonomously in the simulated environment.

First, the generated occupancy grid map was saved and used as an input for the Nav2 stack. The robot's URDF model was configured to include necessary transformations and sensor plugins, ensuring compatibility with navigation requirements. The `amcl` (Adaptive Monte Carlo Localization) node was used to localize the robot within the pre-generated map.

For motion planning, the Nav2 stack requires an appropriate global and local planner. The `NavfnPlanner` was chosen as the global planner to compute the optimal path from the start position to the goal, while the `DWBPlanner` was used as the local planner to handle obstacle avoidance and smooth trajectory generation.

To facilitate sensor data integration, the robot's LiDAR was configured to publish 2D laser scan data on the `/scan` topic. Additionally, the `costmap_2d` package was set up to generate global and local costmaps for safe navigation.

Once all configurations were completed, the Nav2 stack was launched using a launch file, initializing the required nodes for localization, path planning, and control. The `rviz2` visualization tool was used to monitor the robot's position and planned path.Using the `rviz2` GUI, an initial pose estimate was provided for the robot within the map, activating the `amcl` node and allowing it to generate pose estimates. Once the robot was roughly localized, a navigation goal was set using the `rviz2` GUI. The Nav2 stack then successfully generated a feasible path and executed smooth autonomous navigation while dynamically avoiding obstacles.

4.10 Deployment of SLAM in hardware

After successfully testing algorithms like SLAM Toolbox for mapping and the Nav2 stack for autonomous navigation in a simulated environment, these methods were implemented and tested on real hardware. The implementation of SLAM in hardware is quite similar to that in simulation. One key difference between simulation and real-world testing is how transformations between different robot links are handled. In simulation, these transformations are defined within the URDF (Unified Robot Description Format) of the robot. However, on the actual hardware, these transformations were set up using the static transform publisher. These transformations are crucial as they define the relationship between the LiDAR sensor and the robot's base, which is essential for accurate map generation and for the controller server to effectively command the robot.

To build a map using SLAM Toolbox, a manual control interface was first set up between the joystick and the robot using the `joy_node` along with a custom node for serial communication. Once the communication was established, it was ensured that the robot moved according to the commands given by the controller.

Next, the system required a 2D laser scan of the environment. To achieve this, the LiDAR was initialized, and the Fast-LIO mapping node was launched, which published both the point cloud and odometry data. However, since we had 3D point cloud data instead of a 2D laser scan, we used the `pointcloud_to_laserscan` node to convert the 3D point cloud into a 2D representation. This process involved "squishing" all 3D points onto a 2D plane. A minimum and maximum height range were specified for the point cloud conversion, ensuring that only points representing potential obstacles for the robot were included.

After this stage, everything was ready to launch SLAM in mapping mode. We had our transformations from `odom` to `base_link`, along with scan data available in the

`/scan` topic and odometry data in the `/Odometry` topic.

Once the proper transformations were set up and all the nodes were publishing data to their respective topics, SLAM Toolbox was launched in online asynchronous mode. The robot was manually moved through the environment, and the map was incrementally constructed in the 2D occupancy grid. The robot was driven through the entire campus Pulchowk Campus for mapping, resulting in the successful development of a 2D occupancy grid representation of the campus environment.

4.11 Tuning of Nav2 Parameters

The default parameters of the Nav2 stack are designed to work well for most differential drive robots. However, to improve the performance, smoothness, speed, and accuracy of the robot while following the path provided by the Nav2 server, various parameters were tuned according to our environment for optimal results.The major parameters tuned are as follows:

1. **Size of the Local Cost Map:** The size was increased from 3 meters to 10 meters by 10 meters. This is the size of the window over which the local planner, which uses the Dynamic Window Approach, plans locally. The size was increased to accommodate the dynamic nature of the campus environment.

2. **Robot Radius:** The robot radius was adjusted to match the size of our robot. It was changed from 0.1 meters to 0.4 meters, which closely approximates the size of the circle that covers our robot. This reduced the chances of collision by robot as it could then judge its size relative to environment properly and maintain the required distance from the obstacles.

3. **Inflation Radius:** The inflation radius defines the buffer distance to be kept from obstacles in the map. By increasing this value, we saw a decrease in the number of collisions and near-miss incidents.

4. **Other Parameters for the Local Planner:** These include the minimum and maximum velocities achievable by the robot. After tuning the velocity limits, the robot's average speed increased by *100%*, allowing for faster completion of tasks without sacrificing accuracy.

5. **Velocity Smoother:** The velocity smoother is responsible for making the motion of the robot smooth while operating. The `scale_velocity` parameter was set to true to further smoothen the motion of the robot. This resulted in a significant reduction in sudden velocity changes, improving the robot's overall stability and making its path-following smoother.

Table 4.2: Tuned Parameters and Their Default vs. Changed Values

Parameter	**Default Value**	**Changed To**
Size of the Local Cost Map	$3m \times 3m$	$10m \times 10m$
`robot_radius`	$0.1m$	$0.42m$
Inflation Radius	0.55	0.7
`amcl (max_particles)`	2000	4000
controller server `max_vel_x`	0.26	2.0
controller server `max_vel_x`	0.26	1.0
Velocity Smoother (`scale_velocity`)	`false`	`true`

The final schematic for the data flow throughout the system for navigation is illustrated by Figure 4.11.

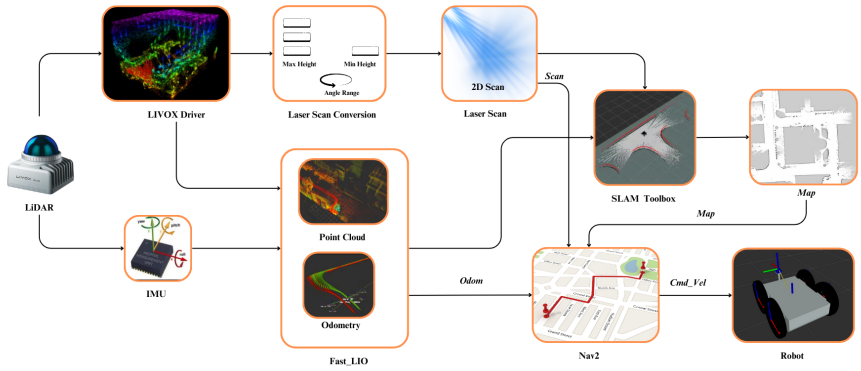


Figure 4.11: Final System Architecture for Navigation

4.12 System Testing

We conducted system testing to check whether the system performs as expected. Several tests are carried out to ensure accuracy, reliability, and smooth operation.

4.12.1 Goal-Reaching Performance Test

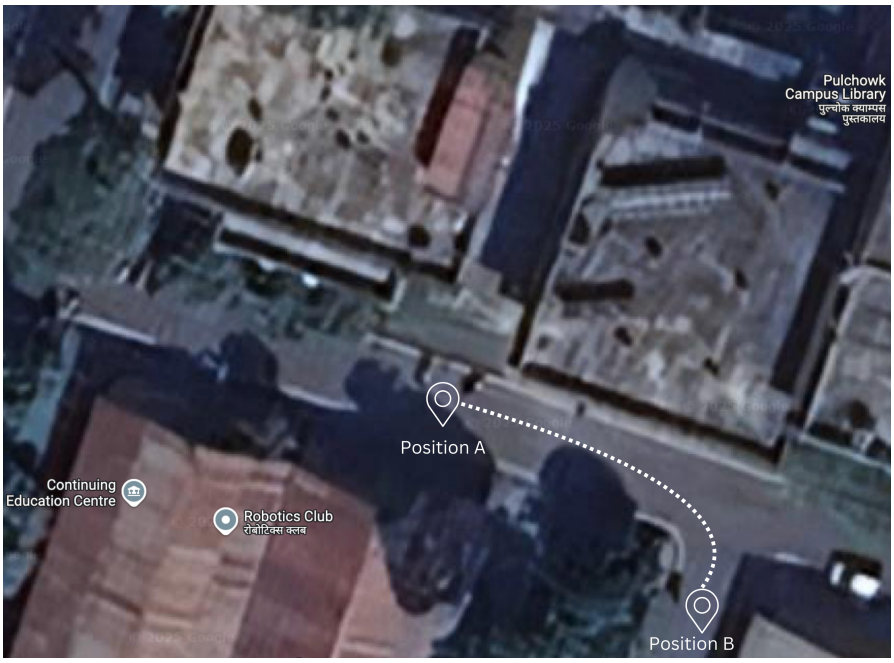


Figure 4.12: Assigned Position A and B in Google Map

We chose two positions for the testing of the robot.The initial position was named as A and the goal was named as B.This positions are around Robotics Club which is shown in google maps represented by Figure 4.12 In this test, the robot traversed from position A to B. We marked both the positions with paint and a jig fixture was used.

A jig fixture was designed and manufactured inorder to ensure the robot's precise positioning at both its initial pose and final goal. The jig fixture was constructed using plywood, as it was easily available and simply cut. A 3D CAD model was created using SolidWorks as shown in Figure 4.13. The design includes the precision slots so that the jig can be tight once upon installation.

Then, that jig was fabricated through a laser cutter for precision cutting. All jigs were tested for fit and alignment before final assembly. After fabrication, the individual components were assembled, and fine adjustments were made to ensure that the robot could be accurately placed in its starting position and final goal coordinate.

This fixture helped to provide a repeatable positioning system that guides the robot accurately during the setup and operation phases of testing. This is required for reproducibility in testing, as it eliminates variations caused by inaccurate positioning. The initial and goal position were marked with spray paints shown in Figure 4.15

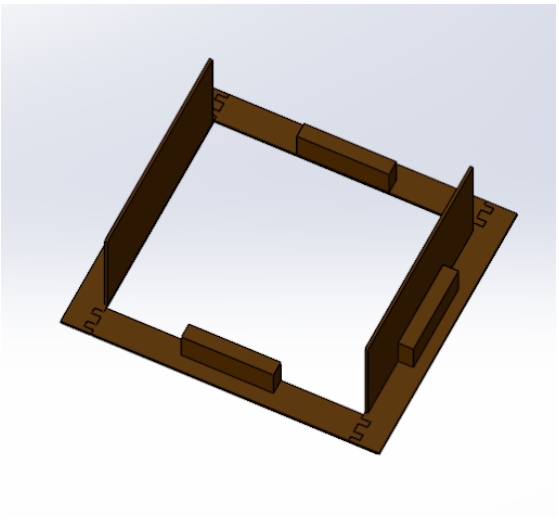


Figure 4.13: Jig Design in SolidWorks

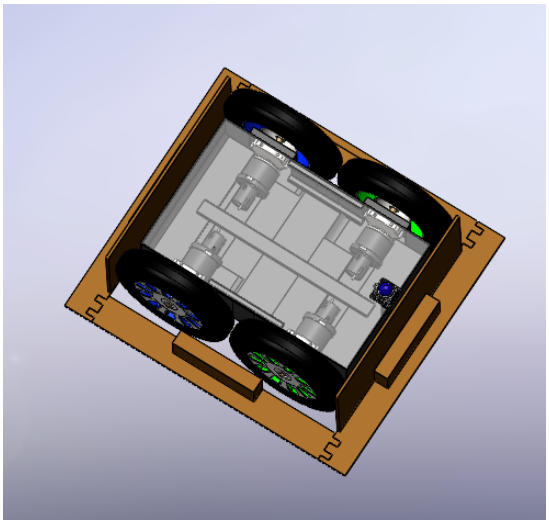


Figure 4.14: Jig Guiding the Robot

Figure 4.15: Marking Position of Robot with Jig

The test was carried out for **four** test.The robot was set to position pre-defined point A with the help of the jig. After the robot was set in the initial position, odometry was echoed and recorded to take the information of the position of the initial location. We then provide the goal to the robot by the Nav2 goal command. Once the robot reached the goal, the odometry was echoed again and recorded.

The difference in the pose goal given and the echoed odometry gives the variation of position of the robot. The difference in goal pose is shown in figure below. We took LiDAR odometry as the ground truth for our test.

4.12.1.1 Clearance Test

The primary goal of the clearance test is to determine whether the robot can successfully pass through obstacles in a given gap or not without colliding. This test helps to evaluate the minimum required clearance for safe navigation and assesses how the robot's dimensions affect its ability to traverse narrow spaces.

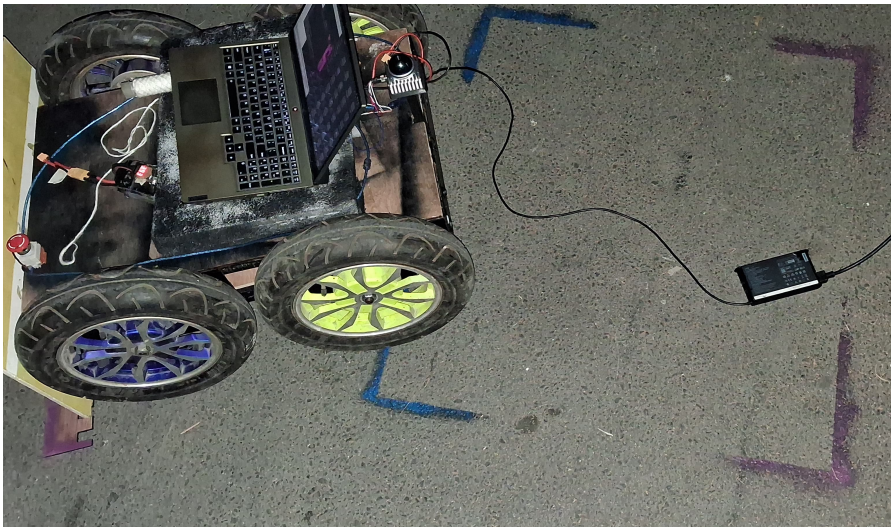


Figure 4.16: Difference in Goal Position and Orientation

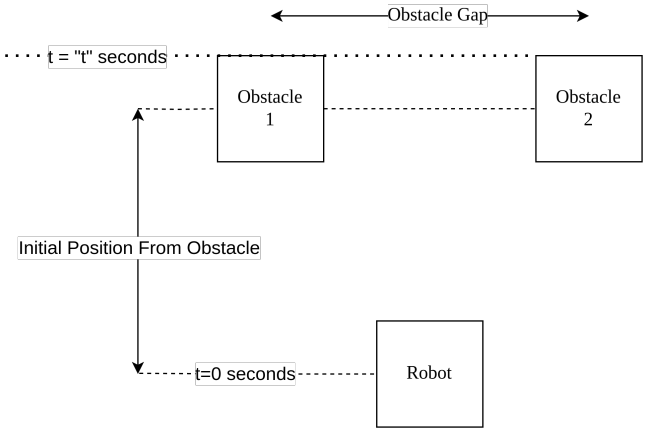


Figure 4.17: Setup of Clearance Test

The test setup consists of two obstacles placed at a fixed distance apart, creating an obstacle gap. Figure 4.17 provides an overview of the system testing arrangement. The robot starts from an initial position at a set distance from the obstacles, and its movement is tracked from t = 0 seconds until it either successfully passes through the gap or collides with the obstacles. The key parameters measured include the initial position from the obstacle (cm), the obstacle gap (cm), and the journey time (seconds) taken by the robot to cross the gap. The outcome of each test is categorized as either ”Pass” if the robot clears the gap without collision, ”Chance of Collision” if there is slight contact or hesitation, or ”Fail” if the robot collides and stops. By conducting multiple trials with varying obstacle gaps, the test determines the minimum clearance required for successful navigation.

4.13 Camera Integration for Lane Detection

A monocular camera was used to detect lanes for our robot's navigation. We first designed the camera mount that fits on our robot.The CAD model of camera mount is shown in Figure 4.18 and it was 3D printed. The camera used is a logitech webcam. The process began with camera calibration, where we determined the focal length, principal point, and distortion coefficients using ArUco markers. Calibration parameters were extracted and stored in YAML files from the captured images.

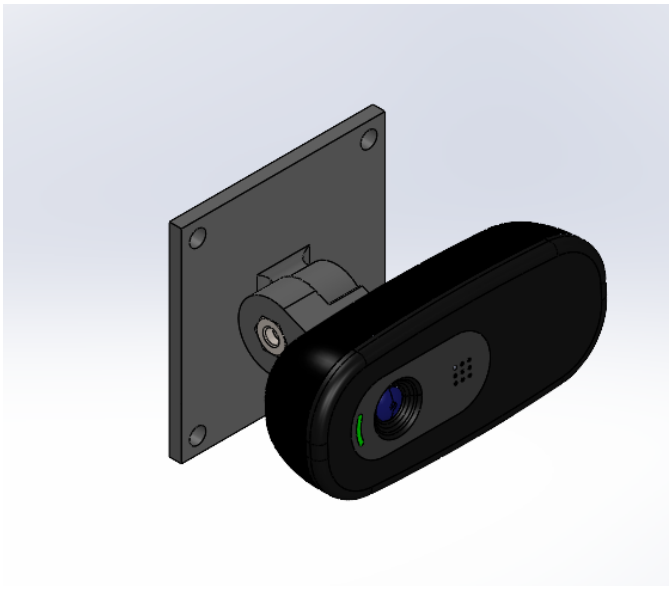


Figure 4.18: CAD Model of Camera Mount

Next, we developed a lane detection algorithm by training a model using Roboflow, which generated the model weights optimized for our pre-designed model. Once the model successfully detected lanes, we further segmented the ground from the camera frames to isolate the lane markings. This step was essential to ensure that the robot recognized lanes as obstacles and prevent it from navigating into ditches along the sides of the lane.The robot could determine the actual drivable path, mimicking human driving behavior by treating the lane as an obstacle.

After segmentation, we developed a custom plugin to integrate the detected lane into the costmap.This enhances the robot's path-planning capabilities. While the plugin functioned as intended, aligning the resolution of the detected lane with the actual map remained a challenge. we were unable to fully resolve this issue within the project timeline due to time constraints, but we have included to address it in future works.

Chapter Five: RESULTS AND DISCUSSION

This chapter presents the outcomes of the autonomous robot's deployment in both real and simulated environments of varying complexities. The results demonstrate the robot's ability to autonomously navigate these settings, with metrics indicating significant reliability and accuracy.

5.1 Simulated Environment

First, the robot's SLAM algorithms and navigation capabilities were tested in a simulated environment using Gazebo. The robot was simulated in both a simple indoor environment and a complex environment that mimicked the complexity of an outdoor setting.

5.1.1 Simple Indoor Environment

A simple room with obstacles, such as construction cones, was created in Gazebo as shown in Figure 5.1. The robot was spawned in this room, and both SLAM and Nav2 were implemented.

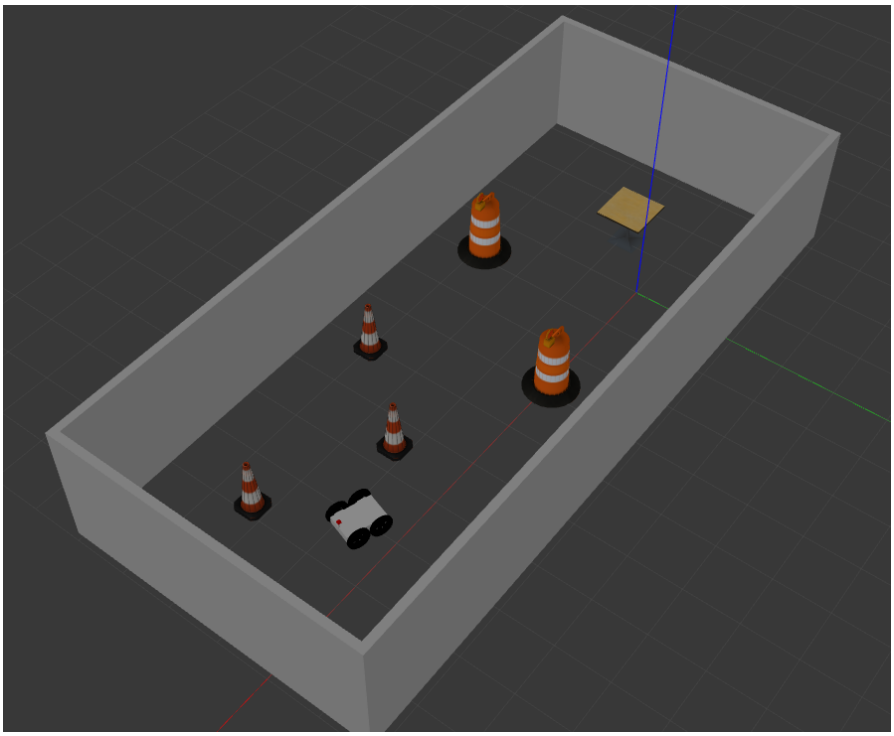


Figure 5.1: Simple Indoor Environment in Gazebo

Fast LIO provided robust odometry as well as a 3D point cloud map of the environment was obtained as shown in Figure 5.2. To reduce computational load the number of

points in simulation are reduced than that of real lidar without compromising in the quality of odometry. Hence, the point cloud obtained in simulation is a bit sparse.

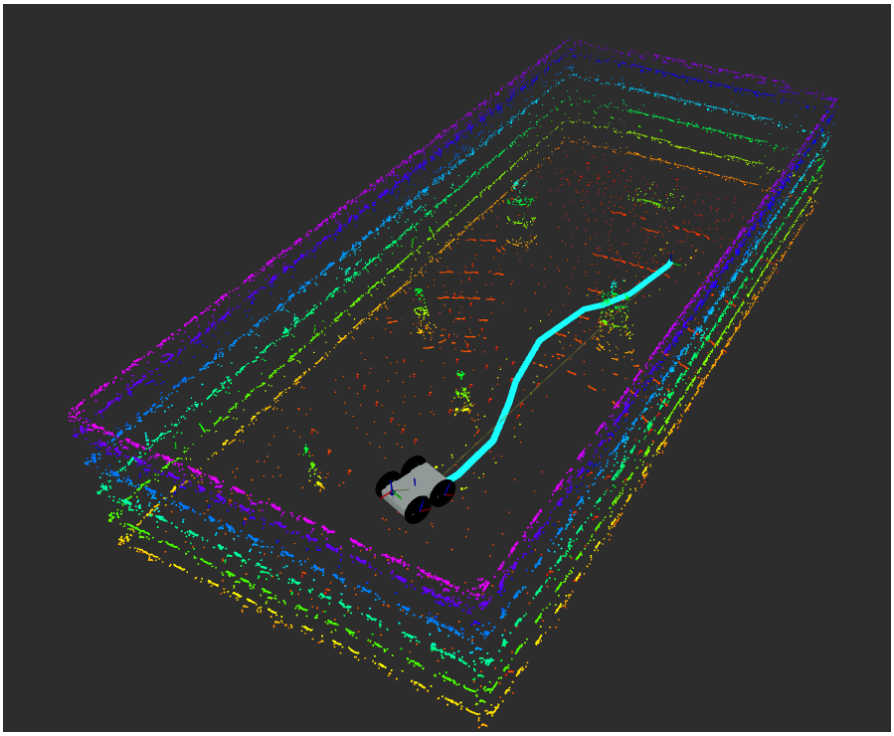


Figure 5.2: Fast LIO Point Cloud Data

Since SLAM was performed using 2D laser scan data, the 3D point cloud data was converted into a 2D occupancy grid representation of the environment. Due to simpler environment and fluctuating odometry , the occupancy grid in Figure 5.3 has slight inconsistencies and map shifting in some areas.

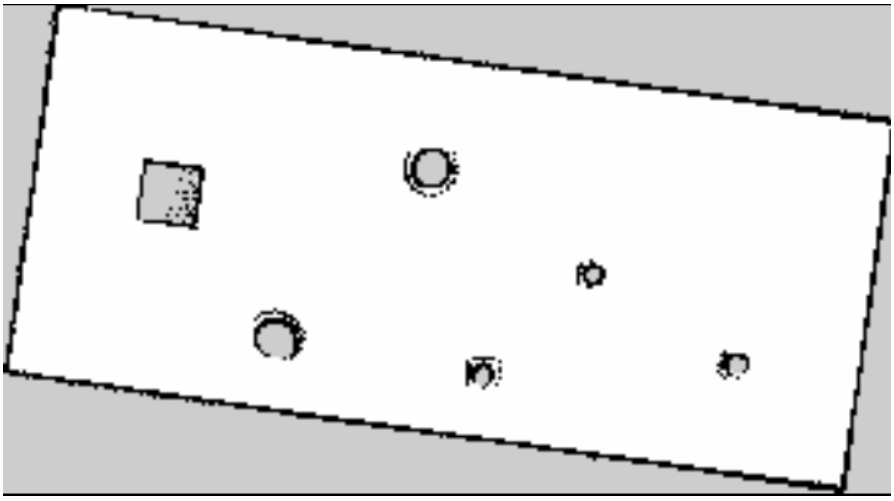


Figure 5.3: 2D Occupancy Grid of the Simple Environment

5.1.2 Complex Environment

To test navigation algorithms in a realistic environment, an STL file representing an outdoor world was created from an occupancy grid obtained from a real environment. This STL file was then loaded into Gazebo for simulation. The 2D occupancy grid used for this purpose was generated from the outdoor environment surrounding the Robotics Club at Pulchowk Campus.
The STL file, when loaded into Gazebo, resulted in the following simulated world:

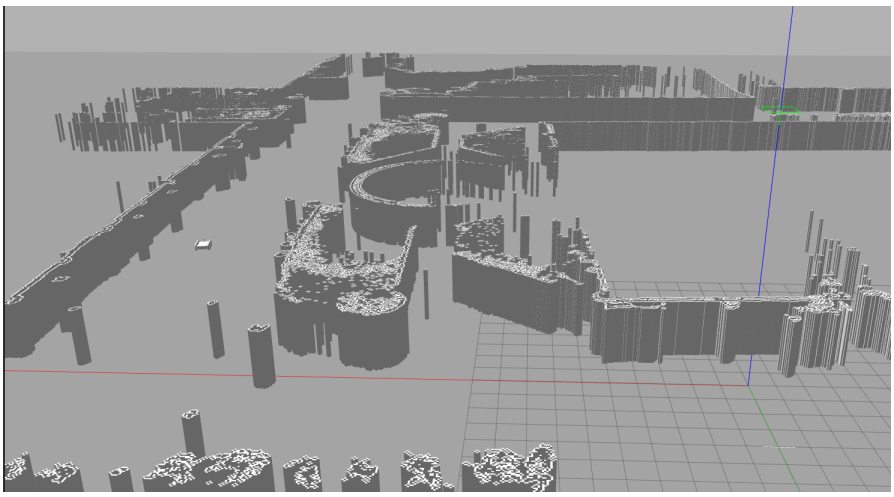


Figure 5.4: Simulated Environment of the Robotics Club

To evaluate the accuracy of navigation, the robot was commanded to move from position A to position B within this environment. The white line in Figure 5.5 represents the path taken by the robot.

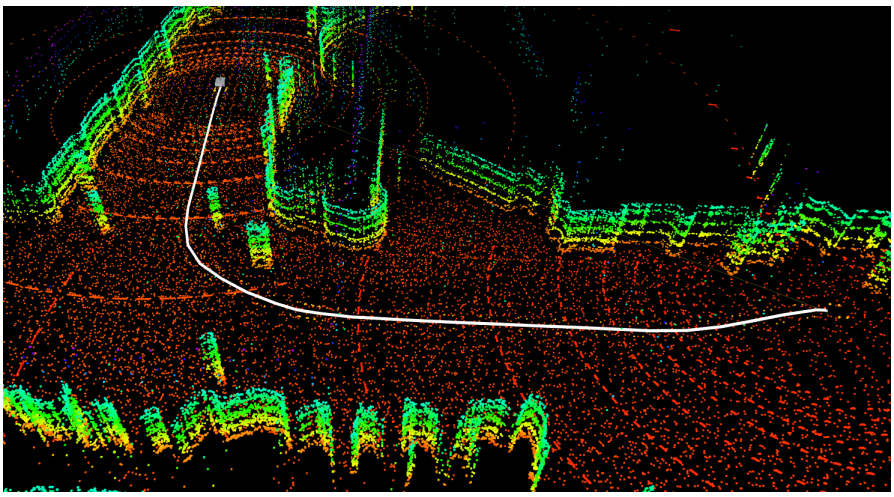


Figure 5.5: Point Cloud Representation of the Environment Outside Robotics Club

5.1.3 Analysis

The navigation performance of the robot in the simple environment was found to be worse than in the complex environment. This is primarily due to the lack of distinct features in the simple environment, which negatively impacts the robot's localization accuracy. In contrast, the complex environment contains a greater number of structural features, allowing the robot to localize more effectively and, consequently, navigate with higher accuracy. Additionally, the presence of more faces and edges enhances odometry, thereby improving the overall autonomy of the robot.

5.2 Real environment

For testing of robot in outdoor environment , the environment outside of Robotics Club pulchowk campus was choosen due to the presesnce of good surface for robot to travel and sufficient features for it to localize. Mapping was done while manually moving the robot throughout the environment and the point cloud map generated can be seen in Figure 5.8 and the 2D occupancy grid is shown in Figure 5.6.

Figure 5.6: 2D Occupancy Grid Map of the Robotics Club Environment

The obtained occupancy grid initially marked potholes and drainages as drivable spaces, requiring refinement. To improve accuracy, we manually added lane markings to the map. Figure 5.7 presents the final version after incorporating these adjustments.

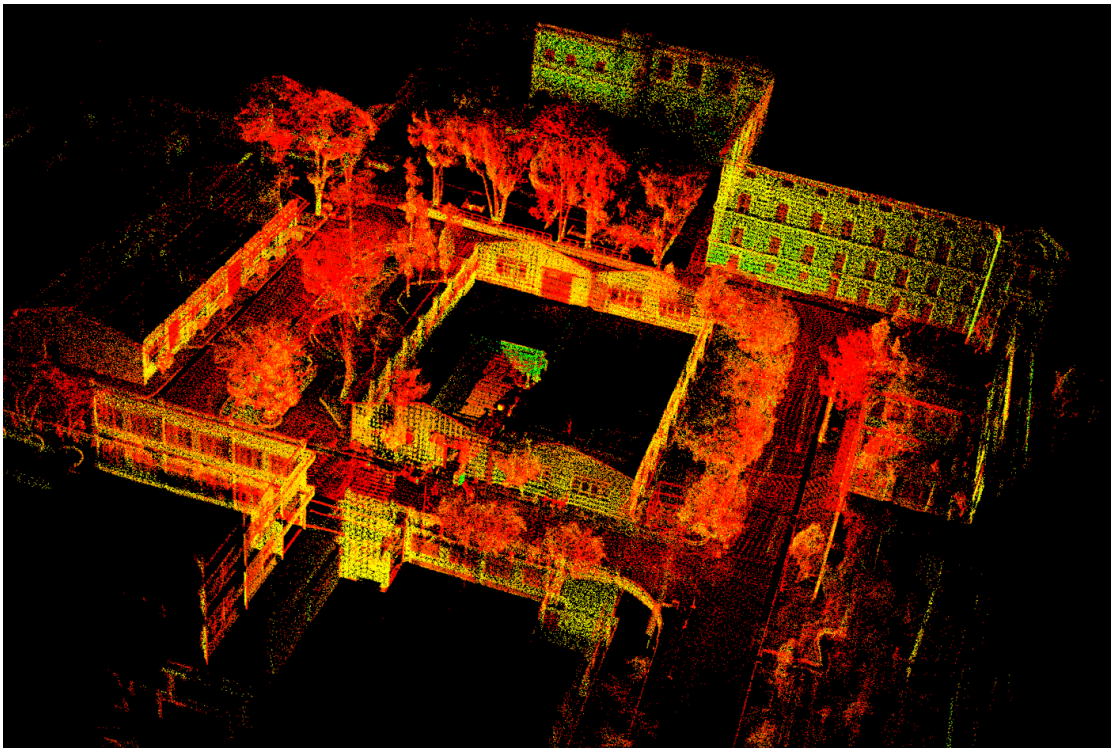


Figure 5.8: Point Cloud Data of the Robotics Club Environment

Figure 5.7: 2D map after adding lane marking

After running the Nav2 stack on the robot, the generated map enabled successful autonomous navigation. The robot effectively traversed the given waypoints while avoiding obstacles along its path. Figure 5.9 illustrates the path traced by the robot as it

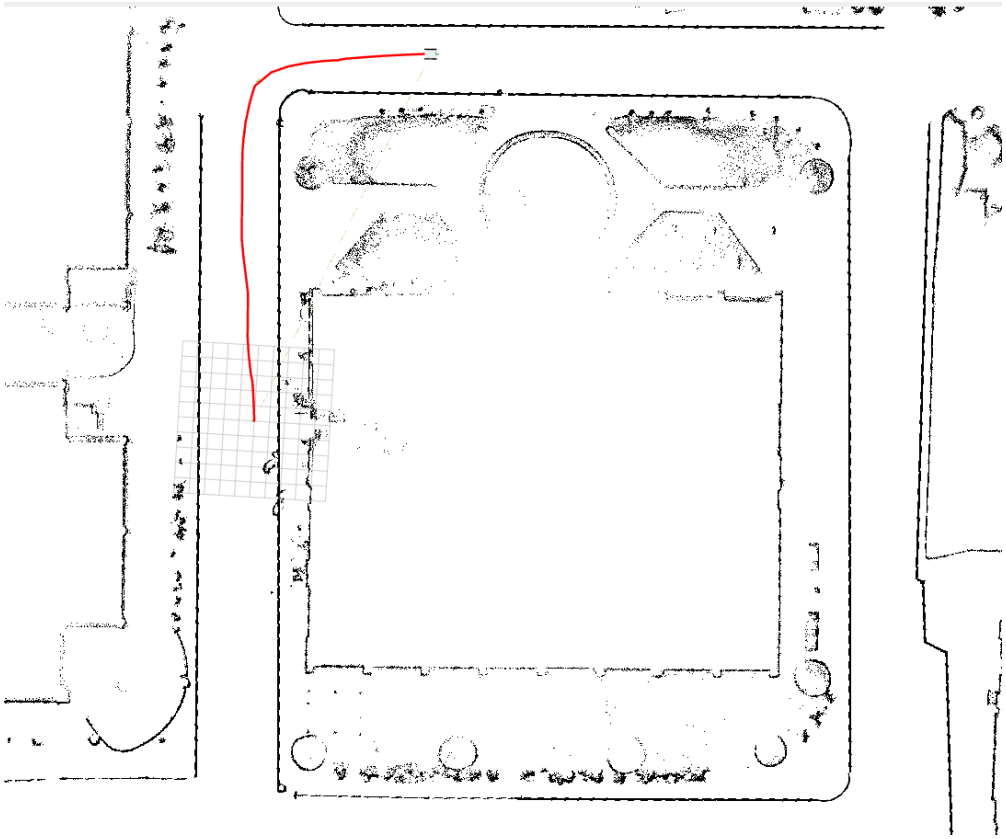


Figure 5.9: Path traced

navigated through the waypoints.

To evaluate the robot's mapping capability over a large area, we mapped the entire campus premises. The generated map encompasses all roads, from Love Garden in front of the Dean's office to the campus teacher's quarters. Overall, the map quality was satisfactory, with minor shifts in certain areas, likely due to odometry uncertainties. The complete map is shown in Figure 5.10.

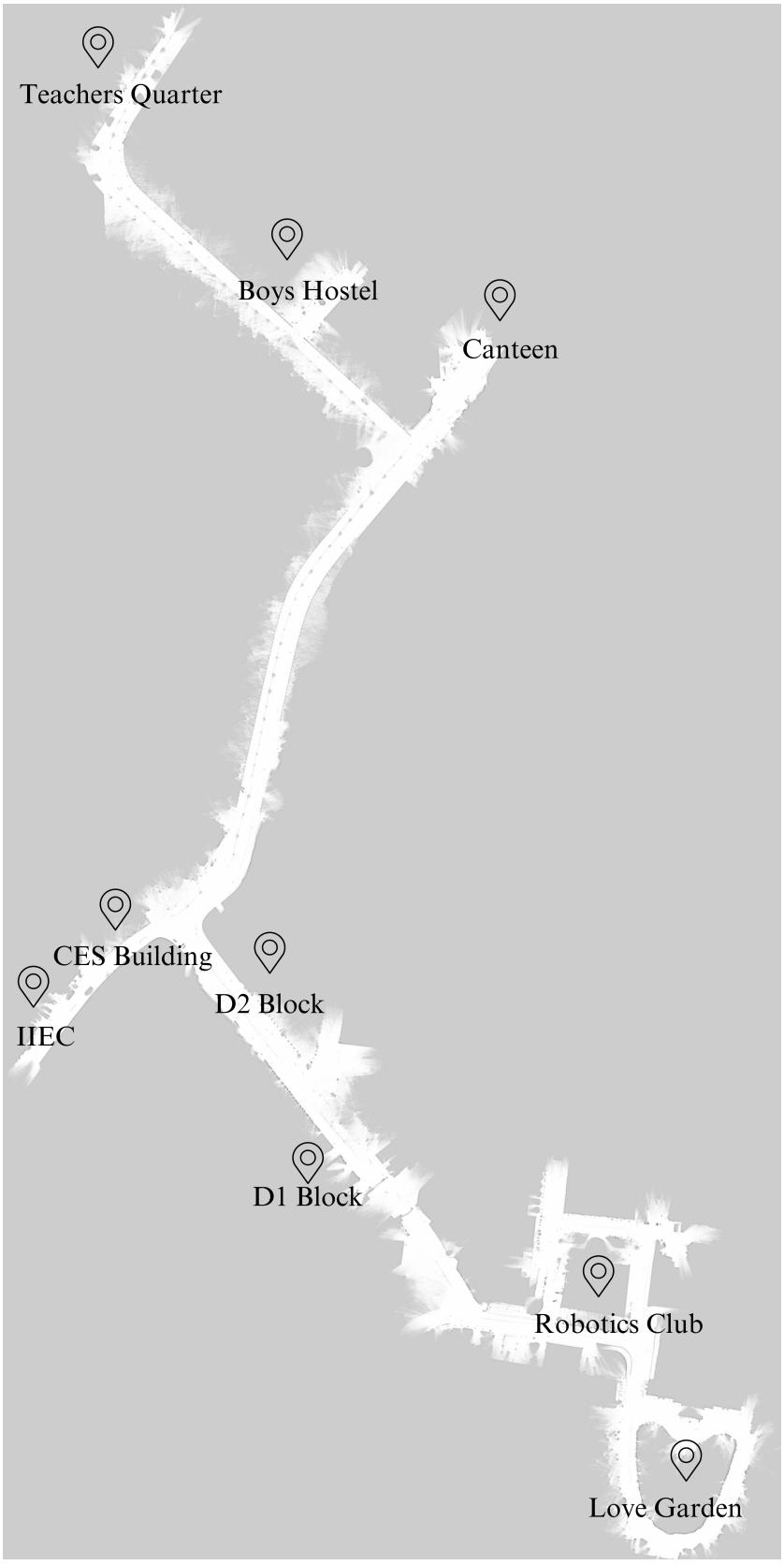


Figure 5.10: Entire Map of Pulchowk Campus

5.3 Results

We conducted four test runs to evaluate the positioning accuracy of the robot when navigating from position A to position B. For these tests, both positions were physically marked with paint, and a jig fixture was used to ensure consistent measurement reference points.

Table 5.1: Table of Positions and Errors

S.N.	**A (position):** (x_1, y_1, θ_1)	**B (goal):** (x_2, y_2, θ_2)	**B (position):** (x_2', y_2', θ_2')	**B (error)**
1	(-0.27, 0.01, 0.00)	(25.06, -2.65, -82.19)	(25.30, -3.18, -58.31)	(0.23, -0.53, 23.88)
2	(0.05, -0.01, 0.13)	(25.06, -2.65, -82.19)	(25.31, -3.22, -58.40)	(0.25, -0.57, 23.80)
3	(0.03, -0.01, 0.12)	(25.06, -2.65, -82.19)	(25.32, -3.32, -58.31)	(0.26, -0.67, 23.88)
4	(0.02, 0.00, 0.12)	(25.06, -2.65, -82.19)	(25.21, -2.86, -61.81)	(0.14, -0.21, 20.38)

The table 5.1 represents the error data collected from four tests where the robot's positions (both initial and final) were measured at different stages. The error is calculated as the difference between the robot's recorded position at the goal (from the odometry) and the intended position.

The results obtained from the four test runs indicate an average error of approximately 0.221 m in the X-axis, -0.495 m in the Y-axis, and 22.986° in orientation. The similarity in error values across all test runs suggests the presence of a systematic bias rather than random fluctuations. This consistency indicates that a fixed error source-such as a sensor misalignment or persistent calibration error is affecting each test run in a similar way.

5.4 Trajectory

The data from ROS was logged using rosbag, and the odometry data was visualized using a Python script. Figure 5.11 illustrates the robot's trajectory.

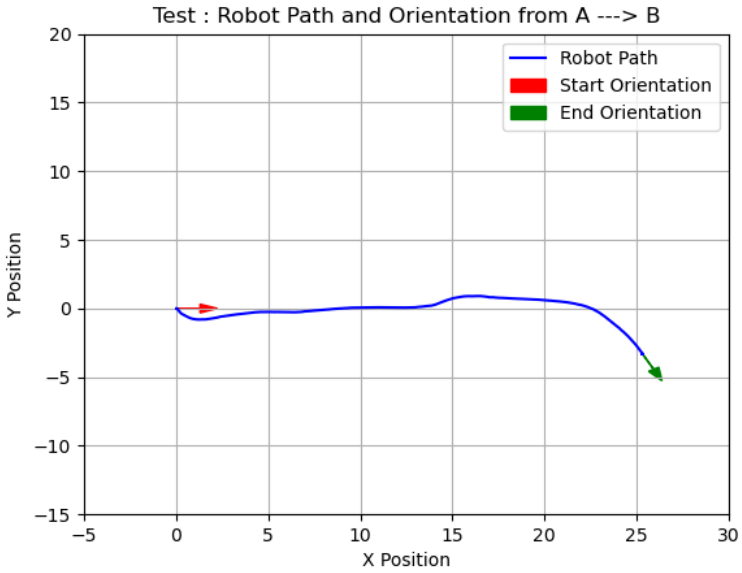


Figure 5.11: Trajectory of Robot Test 4

5.5 Clearance Test

The robot was assigned a goal of 3 meters in the x-direction and tested under the setup shown in Figure 4.17. Initially, the robot's radius was set to 0.3 meters and was made to pass through gaps of 93 cm and 92 cm, resulting in a slight collision. The results are described in table 5.2.

Table 5.2: Navigability of Robot when Robot Radius = 0.3m

Robot Radius (m)	0.3			
S.N.	**Initial Position from obstacle (cm)**	**Obstacle Gap (cm)**	**Remarks**	**Journey Time (sec)**
1	90	93	Passed but slightly grazed	10.91
2	90	92	Passed but slightly grazed	12.12

These results indicate that even a small change in the gap width can significantly increase the likelihood of a collision. This demonstrates that the gap size is crucial for the robot's navigability; a smaller gap forces more careful maneuvers, which can increase travel time, even though it might reduce the chance of collision.

To assess the effect of the obstacle gap with a larger robot radius, we performed Test 2 (Part 1) with the robot's radius set to 0.42 meters (as per its dimensions). Similarly, a goal of 3 meters in the x-direction was assigned, and the setup was tested by varying the obstacle gap. The results obtained are described in table 5.3. To identify the minimum

required gap for successful navigation, we manually tested different gap sizes, starting from 90 cm and increasing to 105 cm. This approach allowed us to explore the range of possible passing distances and progressively narrow it down. By testing both smaller and larger gaps, we could determine the critical threshold at which the robot could successfully pass.

Table 5.3: Navigability of Robot when Robot Radius = 0.42 (Part 1)

Robot Radius (m)	0.42			
S.N.	**Initial Position from obstacle (cm)**	**Obstacle Gap (cm)**	**Remarks**	**Journey Time (sec)**
1	90	90	Not Passed	-
2	90	105	Passed	6.47
3	90	96	Passed	7.18
4	90	92	Not Passed	-
5	90	94	Not Passed	-

These results indicate that the robot's radius plays a crucial role in determining whether it can navigate through a given gap. When the radius was 0.3 meters, the robot successfully passed through the same obstacle gap. However, when the radius was increased to 0.42 meters, the robot was unable to pass and instead avoided the obstacle. The results demonstrate that the robot requires a gap of at least 96 cm to navigate through without collisions.

To assess the repeatability and journey time of the robot while maneuvering through a 96 cm gap, we fixed both the obstacle gap and the robot's initial position and conducted test (Part 2) under the same setup. The results obtained are described in table 5.4.

Table 5.4: Navigability of Robot when Robot Radius = 0.42m (Part 2)

Robot Radius (m)	0.42			
S.N.	**Initial Position from obstacle (cm)**	**Obstacle Gap (cm)**	**Remarks**	**Journey Time (sec)**
1	90	96	Passed	8.5
2	90	96	Passed	6.87
3	90	96	Passed	7.71
4	90	96	Passed	9.82
Average Time				8.225

The results demonstrate that the robot successfully navigates the 96 cm obstacle gap with an average travel time of 8.255 seconds.

5.6 Lane Detection and Segmentation

The model for lane detection and segmentation was trained using YOLO v.. The model could successfully detect the lane and generate a binary mask across multiple scenarios in campus environment as shown in the Figure 5.12.

Figure 5.12: Images with correct lane predictions.

But the model encountered specific challenges in certain scenarios as shown in Figure 5.13. The system exhibited difficulties in two primary conditions:

1. Areas with excessive brightness where lane markings were overexposed
2. Locations where dust or debris obscured the lane features.

Figure 5.13: Images with incorrect lane predictions

5.7 Limitations

1. This project focuses on the implementation and integration of existing pre-built packages and algorithms rather than the development of new ones.

2. Robot is not equipped to handle complex or rough terrain due to the lack of advanced suspension systems required for such tasks.

3. The robot is equipped with a single LiDAR sensor, positioned at a height of 437mm to maximize its field of view. However, with a vertical field of view of only 7° below the horizon, it has blind spots and cannot detect objects below a certain height at a given distance, limiting obstacle detection for low-lying objects.

4. The robot lacks the capability to identify potholes, road markings, and drains, as its current sensing system is not designed for surface-level perception or detecting negative spaces.

5. Due to testing constraints and the lack of high-performance embedded computing resources, a laptop has been used onboard for processing. This setup is not an end-product solution, as a more compact and efficient processing unit would be required for practical deployment.

6. The use of a laptop for computation increases power consumption and reduces portability, limiting the robot's operational time and making it less feasible for long-duration tasks in outdoor environments.

5.8 Problems Faced

1. **Turning resistance due to long wheel-base:**
   The initial design of the robot's chassis was overly long, which caused significant problems with turning and maneuverability during tests. The extended length made it difficult for the robot to navigate sharp turns efficiently. To resolve this, the chassis was shortened by removing the middle section and bringing the wheels on each side closer together. This adjustment improved the robot's turning ability and overall movement during testing.

2. **Debugging and Insufficient Documentation:**
   We encountered problems during debugging due to incomplete documentation and compatibility issues with certain libraries. The lack of detailed and clear resources complicated troubleshooting and led to time-consuming delays, as additional effort was required to identify and resolve the software-related issues.

3. **Integration of Camera for Lane Detection:**
   A camera was integrated into the system for lane detection, and the detected lane was successfully overlaid onto the local costmap through plugins. However, achieving precise lane fitting with accurate resolution remained a challenge. This challenge paved the way to further improvement.

4. **Testing Limitations Due to Hardware Constraints:**
   Since all processing was conducted on a single laptop, tuning algorithms parameters and computing pose errors placed a heavy load on the system. This high power consumption required frequent recharging of the laptop that slowed down the overall testing process

5.9 Budget Analysis

The tables below present the estimated cost of the project and the estimated cost of all the items that we plan to borrow from different parties.

Table 5.5: Cost Estimation Table

S.N	Description	Estimation (NPR)
1	Fabrication Cost	10,000
2	Scrap Metals	5,000
3	Documentation	4,000
4	Miscellaneous Costs	5,000
	Total	**24,000**

Table 5.6: Items Pledged Table

S.N	Description	Qty.	Estimation (NPR)
1	24V DC Motors	4	20,000
2	STM32 Discovery Board	1	8,000
3	Cytron Motor Driver	2	10,000
4	LIVOX MID360	1	102,000
5	Battery 6S	1	8,000
		Total	**128,000**

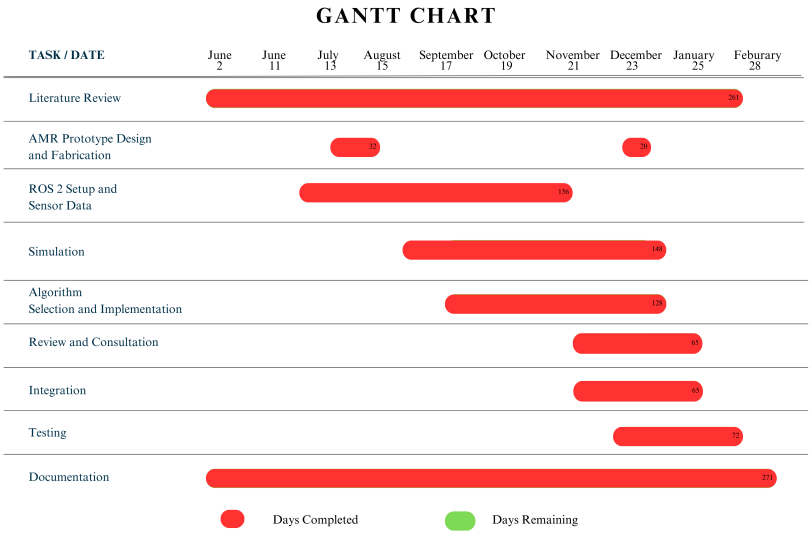
5.10 Work Schedule(Gantt Chart)

The following table shows the tasks, their starting date (expected starting date) and the end date (expected end date) along with the expected days for each task.

Table 5.7: Project Schedule

S.N	Task	Start Date	End Date	Duration(days)
1	Literature Review	2-Jun	18-Feb	261
2	Prototype Design	15-Jul	25-Jul	10
3	Prototype Fabrication	26-Jul	17-Aug	22
4	ROS Setup & Sensor Data	22-Jun	21-Nov	156
5	Algorithm Selection and Implementation	16-Sep	10-Jan	115
6	Simulation	17-Sep	1-Jan	105
7	Review and Consultation	21-Nov	25-Jan	65
8	Integration	21-Nov	25-Jan	65
9	Testing	09-Dec	18-Feb	72
10	Documentation	2-Jun	28-Feb	271

Figure 5.14: Gantt Chart



Chapter Six: CONCLUSION AND FUTURE ENHANCEMENTS

The development of an autonomous mobile robot for outdoor navigation using SLAM and Nav2 successfully achieved all the specific objectives. The system was tested in both simulated and real-world environments, demonstrating its effectiveness within structured environments such as our campus.

The robot chassis was successfully built, ensuring stability and mobility within the campus premises. Initial testing was conducted using a joystick to manually operate the robot to confirm that the mechanical and electronic components functioned as expected. A 3D point cloud map of the surroundings was generated that shows the robot's ability to accurately perceive and map its environment. Additionally, the FastLIO odometry and SLAM algorithm were implemented and tested to effectively enable localization and mapping in both simulation and real-world scenarios. Nav2 was successfully utilized for path planning, autonomous navigation, and obstacle avoidance.

Despite the successful completion of the project, several challenges were encountered. Understanding and integrating various software packages proved to be more complex than expected. So,we needed to adjust the project timeline. Additionally, minor pose and orientation errors were observed at the goal position, though they are within tolerable limits. Indoor testing was conducted in the Robotics Club of Pulchowk Campus, while outdoor testing took place in various locations around the campus. While camera integration reached a certain milestone, it could not be fully implemented in the final hardware.

Overall, the project successfully demonstrated the feasibility of an autonomous mobile robot for structured outdoor environments. Each phase contributed to successful completion of this project.

The possibility of further improvements hence now can be enlisted as:

1. **Integration of Additional Sensors:**
   Currently, the system relies only on LiDAR, limiting its applicability to certain outdoor environments. Although lane detection using a camera was explored, it could not be fully integrated into the final hardware. Incorporating additional sensors such as GPS, depth cameras, and ultrasonic sensors, combined with sensor fusion techniques, would enhance environmental perception. A detailed understanding of the surroundings would improve the overall system performance that

makes robust navigation in diverse outdoor conditions.

2. **Refinement for a Product-Ready System:**
   While the use of a laptop provides sufficient computational power, it adds bulk to the system, making commercialization challenging. Replacing the laptop with a more compact yet powerful embedded computing device would make the system more modular and practical for real-world deployment. Additionally, upgrading the current 6-cell LiPo battery would improve the robot's operational efficiency and longevity.

3. **SLAM and Path Planning Optimization:**
   The parameters of the SLAM algorithm and path planning process should be fine-tuned to optimize efficiency. Adjusting algorithm parameters can refine the robot's behavior in outdoor navigation. Furthermore, implementing advanced control strategies such as Model Predictive Control (MPC) could enhance the planner's decision-making capabilities which leads to more efficient and adaptive navigation.

4. **Motion Capture for Ground Truth Validation:**
   Integrating a motion capture system would provide precise ground truth positioning of the robot, enabling a quantitative comparison between the tracked and estimated trajectories. This would facilitate a more accurate evaluation of the SLAM and path planning algorithms that helps to refine their performance.

REFERENCES

[1] Intel, *Autonomous mobile robots overview*, Accessed: 2025-03-06, 2025. [Online]. Available: https://www.intel.com/content/www/us/en/robotics/autonomous-mobile-robots/overview.html.

[2] StackExchange, *Free body diagram of a differential drive mobile robot*, Accessed: Mar. 10, 2025, 2025. [Online]. Available: https://engineering.stackexchange.com/questions/43474/free-body-diagram-of-a-differential-drive-mobile-robot.

[3] OSOYOO, *Osoyoo robotic mecanum wheel platform for raspberry pi*, Accessed: Mar. 10, 2025, 2025. [Online]. Available: https://www.amazon.co.uk/OSOYOO-Robotic-Mecanum-Platform-Raspberry/dp/B07WZJYVB5.

[4] ResearchGate, *Robot skid steering rover 4wd*, Accessed: Mar. 10, 2025, 2025. [Online]. Available: https://www.researchgate.net/figure/Robot-skid-steering-Rover-4WD1_fig2_321249467.

[5] Yahboom, *Ros chassis - ackermann steering robot*, Accessed: Mar. 10, 2025, 2025. [Online]. Available: https://category.yahboom.net/products/ros-chassis?_pos=1&_psq=ackerman&_ss=e&_v=1.0&variant=44178907070780.

[6] S. Rabiee and J. Biswas, "A friction-based kinematic model for skid-steer wheeled mobile robots," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019. DOI: 10.1109/ICRA.2019.8794216.

[7] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo, and J. Gonzalez, "Experimental kinematics for wheeled skid-steer mobile robots," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, IEEE, 2007, pp. 1222–1227.

[8] ResearchGate, *Lms200 laser sensor - sick ag and its principal internal components*, Accessed: Mar. 10, 2025, 2025. [Online]. Available: https://www.researchgate.net/figure/LMS200-laser-sensor-SICK-AG-and-its-principal-internal-components_fig1_51873403.

[9] Open Source Robotics Foundation, *Robot Operating System (ROS)*, Accessed: March 6, 2025, 2025. [Online]. Available: https://www.ros.org.

[10] ROS 2 Documentation, *Understanding nodes — ros 2 documentation: Humble*, Accessed on January 15, 2025, n.d. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html.

[11] *What is slam (simultaneous localization and mapping) – matlab & simulink - matlab & simulink*, [Online; accessed 2024-12-16]. [Online]. Available: https://www.mathworks.com/discovery/slam.html.

[12] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[13] N. J. Nilsson, "Shakey the robot," *SRI International Technical Note*, p. 323, 1984.

[14] G. Ullrich, "Automated guided vehicle systems," *Springer*, 2015.

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

[16] R. D'Andrea, "Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 4, 638–639, 2012.

[17] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," *ICRA workshop on open source software*, vol. 3, no. 3.2, p. 5, 2009.

[18] M. Buehler, K. Iagnemma, and S. Singh, Eds., *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, 2009, vol. 56.

[19] J. P. Fentanes, B. Lacerda, T. Krajník, N. Hawes, and M. Hanheide, "Now or later? predicting and maximising success of navigation actions from long-term experience," in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7533–7539. DOI: 10.1109/ICRA40940.2020.9197037.

[20] M. A. Arain, E. Schaffernicht, and V. H. Bennetts, "A comparative evaluation of robot navigation in complex environments using 2d and 3d perception systems," *Robotics and Autonomous Systems*, vol. 121, p. 103 258, 2019. DOI: 10.1016/j.robot.2019.103258.

[21] J. Zhang and S. Singh, "Laser–visual–inertial odometry and mapping with high robustness and low drift," *Journal of Field Robotics*, vol. 35, no. 8, pp. 1242–1264, 2018. DOI: 10.1002/rob.21807.

[22] E. Jelavic, D. Jud, F. Ramos, and M. Hutter, "Terrain mapping for navigation in complex outdoor environments using 3d lidar," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1380–1387, 2021. DOI: 10.1109/LRA.2021.3053045.

[23] W. Gao and R. Tedrake, "Filterreg: Robust and efficient probabilistic point-set registration using gaussian filter and twist parameterization," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11 095–11 104. DOI: 10.1109/CVPR.2019.01135.

[24] Y. Wang, Y. Sun, Z. Liu, and S. E. Sarkar, "Multi-echo lidar for robust perception in adverse weather conditions," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 3835–3841. DOI: 10.1109/ICRA40940.2021.9576596.

[25] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022. DOI: 10.1109/TRO.2022.3141876.

[26] G. Huang, K. Eckenhoff, and P. Geneva, "Bias: Back-end inertial-aided system for 3d lidar slam," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 7864–7870. DOI: 10.1109/ICRA46639.2022.9811689.

[27] X. Chen, A. Millane, A. Jacobson, and D. Scaramuzza, "Resource-constrained 3d mapping using lidar and visual-inertial odometry," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1670–1677, 2020. DOI: 10.1109/LRA.2020.2970983.

[28] X. Wu, S. Liu, and Y. Yang, "Comparative evaluation of lidar-inertial odometry algorithms for aerial and ground robots," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 976–991, 2023. DOI: 10.1109/TRO.2023.3245678.

[29] T. Zhang, K. Wu, and D. Su, "Analyzing the robustness of lidar-based slam systems to environmental challenges," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 9112–9118. DOI: 10.1109/ICRA46639.2022.9812066.

[30] J. Lin and F. Zhang, "A comprehensive comparison of lidar-inertial odometry for off-road navigation," *Autonomous Robots*, vol. 46, no. 3, pp. 359–374, 2022. DOI: 10.1007/s10514-021-10019-3.

[31] S. Macenski and I. Jambrecic, "Slam toolbox: Slam for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021. DOI: 10.21105/joss.02783. [Online]. Available: https://doi.org/10.21105/joss.02783.

[32] *On use of the slam toolbox*, [Online; accessed 2024-12-16]. [Online]. Available: https://roscon.ros.org/2019/talks/roscon2019_slamtoolbox.pdf.

[33] C. Brenneke, O. Wulf, and B. Wagner, "Using 3d laser range data for slam in outdoor environments," *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 1, 188–193 vol.1, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:1149990.

[34] L. Wang, Z. Liu, and H. Li, "Robust and real-time outdoor localization only with a single 2-d lidar," *IEEE Sensors Journal*, vol. 22, no. 24, pp. 24 516–24 525, 2022. DOI: 10.1109/JSEN.2022.3218676.

[35] Z. Liu, Z. Li, A. Liu, Y. Sun, and S. Jing, "Fusion of binocular vision, 2d lidar and imu for outdoor localization and indoor planar mapping," *Measurement Science and Technology*, vol. 34, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253291070.

[36] Y.-T. Hong and H.-P. Huang, "A comparison of outdoor 3d reconstruction between visual slam and lidar slam," *2023 International Automatic Control Conference (CACS)*, pp. 1–6, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:265470430.

[37] L. Wijayathunga, A. Rassau, and D. Chai, "Challenges and solutions for autonomous ground robot scene understanding and navigation in unstructured outdoor environments: A review," *Applied Sciences*, vol. 13, no. 17, 2023, ISSN: 2076-3417. DOI: 10.3390/app13179877. [Online]. Available: https://www.mdpi.com/2076-3417/13/17/9877.

[38] D. Jia, A. Hermans, and B. Leibe, "2d vs. 3d lidar-based person detection on mobile robots," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3604–3611, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:251041150.

[39] M. Kim, M. Zhou, S. Lee, and H. Lee, "Development of an autonomous mobile robot in the outdoor environments with a comparative survey of lidar slam," Nov. 2022, pp. 1990–1995. DOI: 10.23919/ICCAS55662.2022.10003762.

[40] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 2013–2020. DOI: 10.1109/IROS51168.2021.9636712.

[41] A. Thomas, F. Mastrogiovanni, and M. Baglietto, "An adaptive approach for projecting 3d point clouds into 2d laser scans for mobile robot navigation," *IEEE Robotics and Automation Letters*, vol. 8, no. 1, pp. 423–430, 2023. DOI: 10.1109/LRA.2022.3202123.

[42] C. Zhao, H. Zhang, and S. Song, "Point-to-scan conversion for efficient integration of 3d lidar into 2d slam frameworks," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7829–7836, 2021. DOI: 10.1109/LRA.2021.3085354.

[43] C. Ren, Y. Ji, and J. Liu, "Hybrid 2d-3d mapping approaches for mobile robot navigation: A comparative study," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1782–1795, 2022. DOI: 10.1109/TASE.2021.3096644.

APPENDICES:

Figure 1: Grinding the Hoverboard's Chassis for Proper Alignment During Assembly

Figure 2: Measuring and Verifying the Dimensions of the Setup Before Welding

Figure 3: Preparing the Jig Fixture for the Assembly Setup Prior to Welding

Figure 4: Chassis Fabrication in Progress through Welding

Figure 5: Final Robot

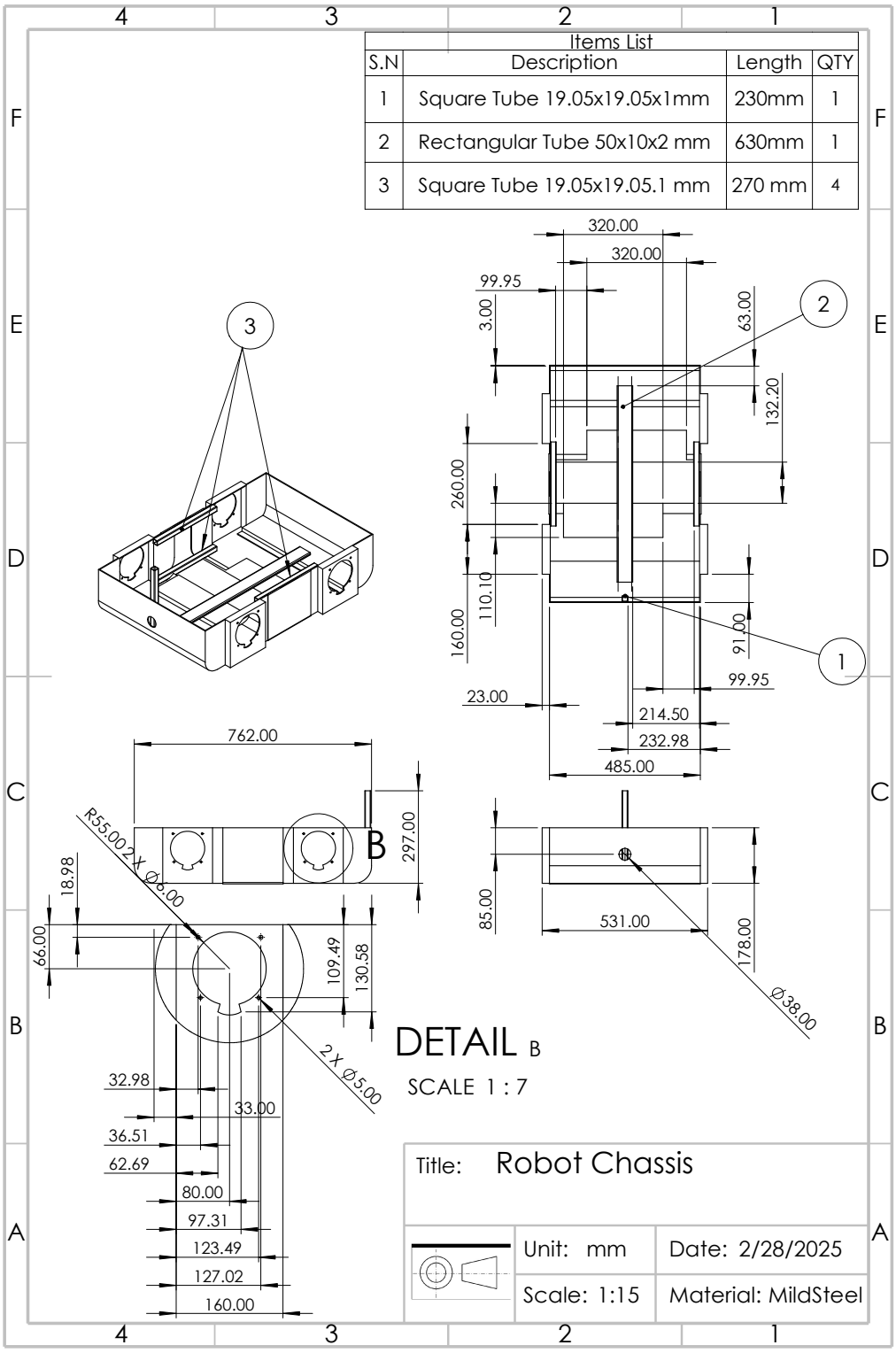


Figure 6: Detailed Drawing of Chassis of Robot

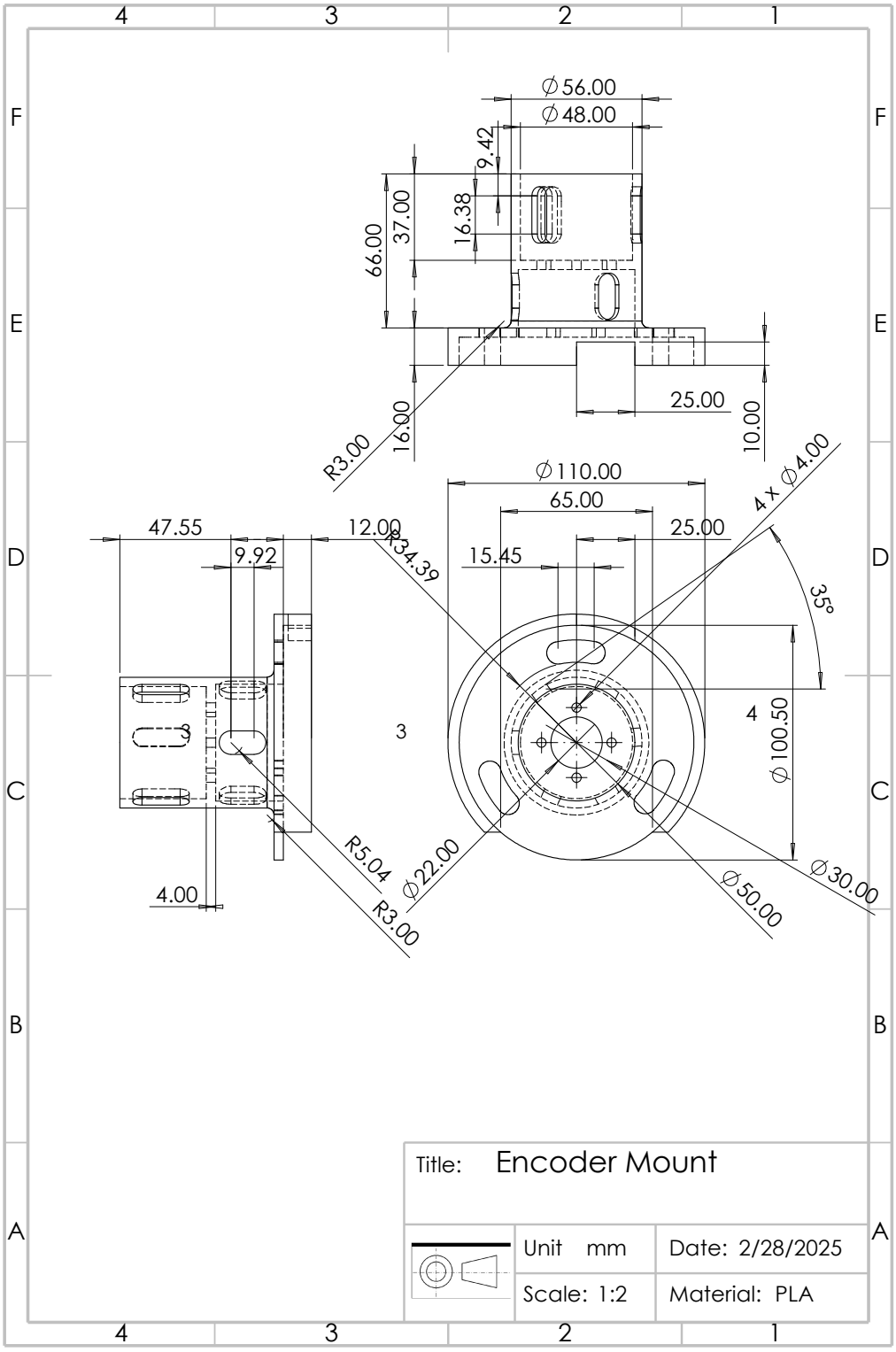


Figure 7: Detailed Drawing of Encoder Mount

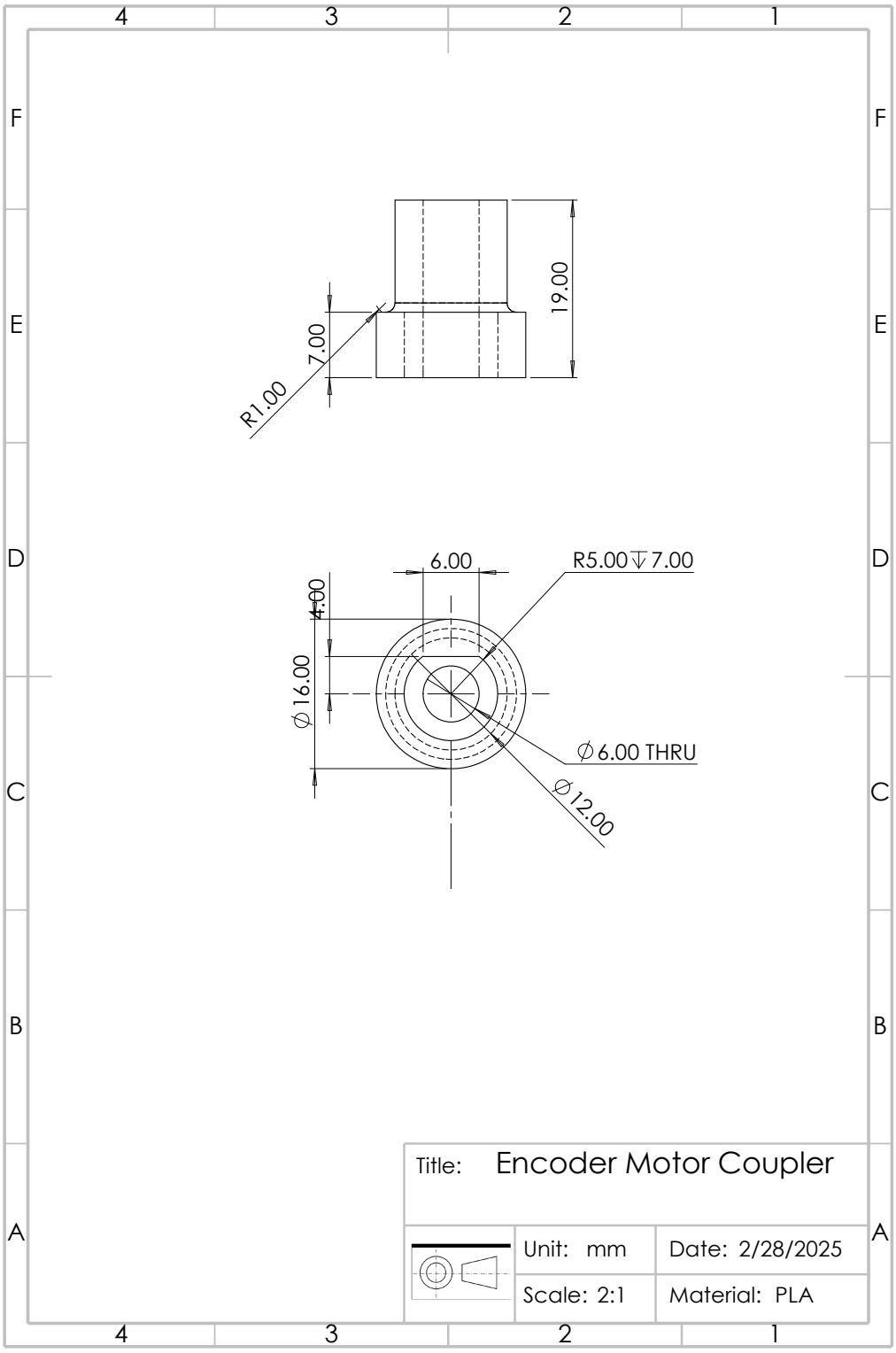


Figure 8: Detailed Drawing of Motor and Encoder Couple

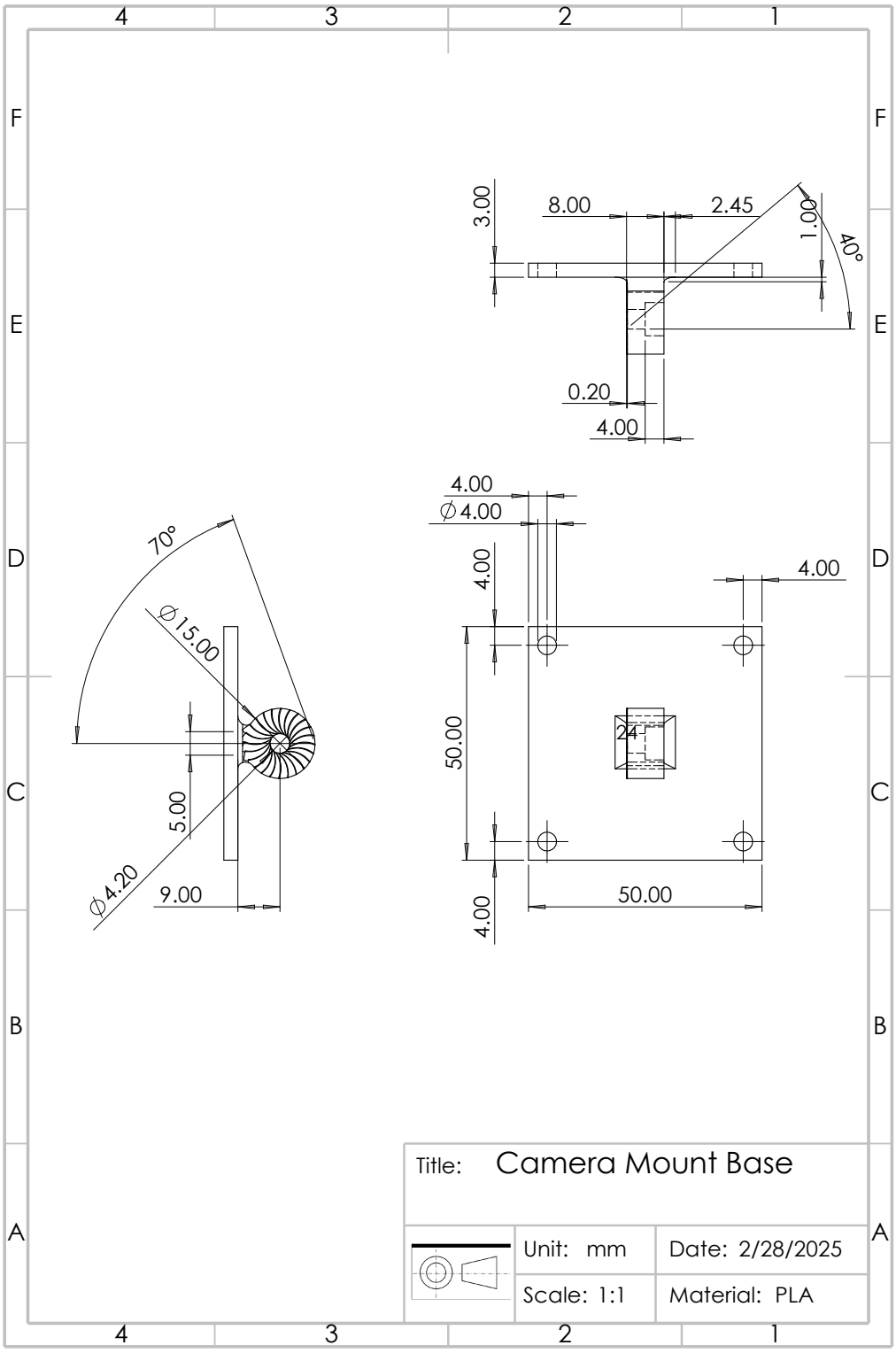


Figure 9: Detailed Drawing of Base of Camera Mount

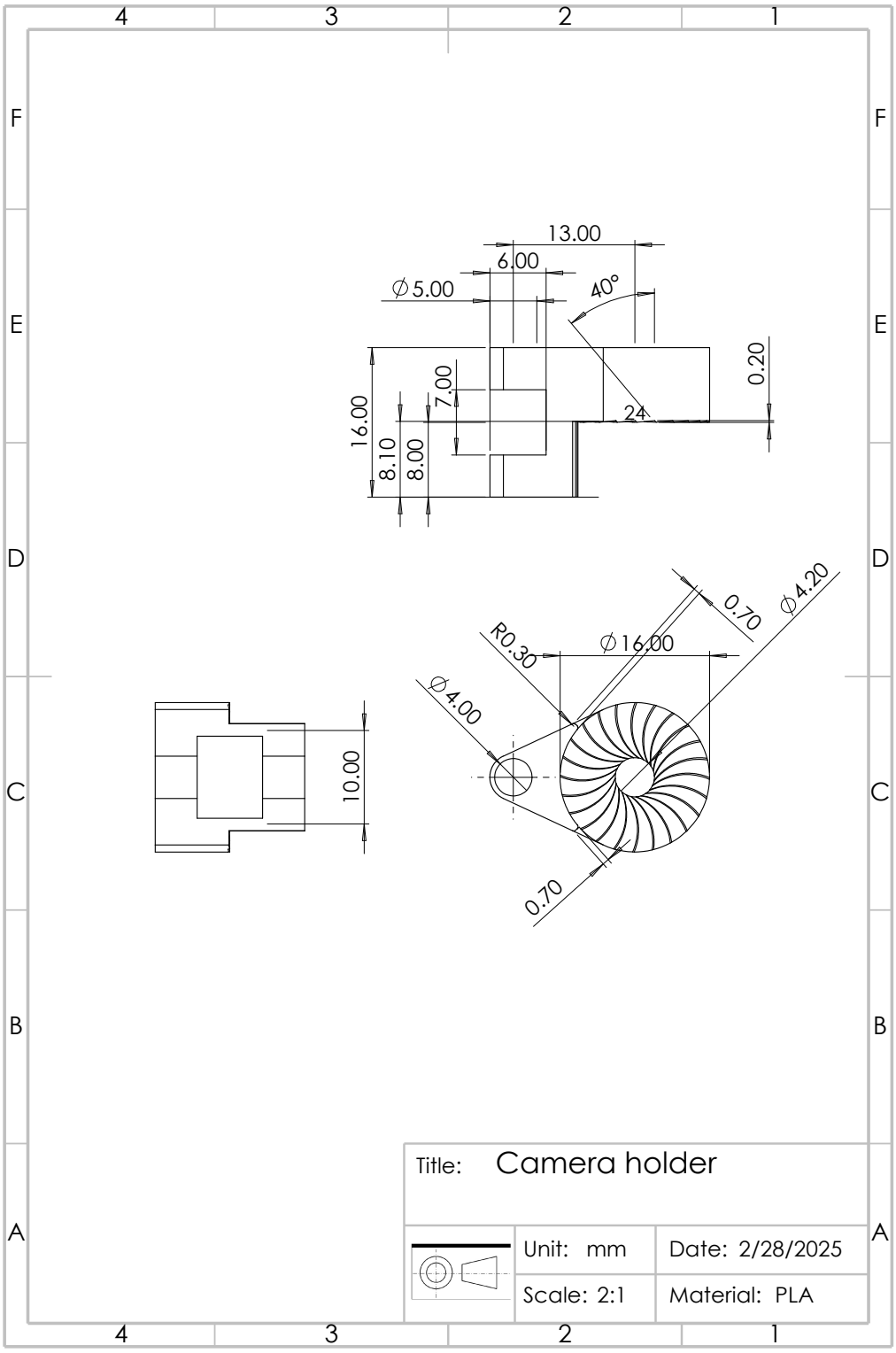


Figure 10: Detailed Drawing of Camera Mount

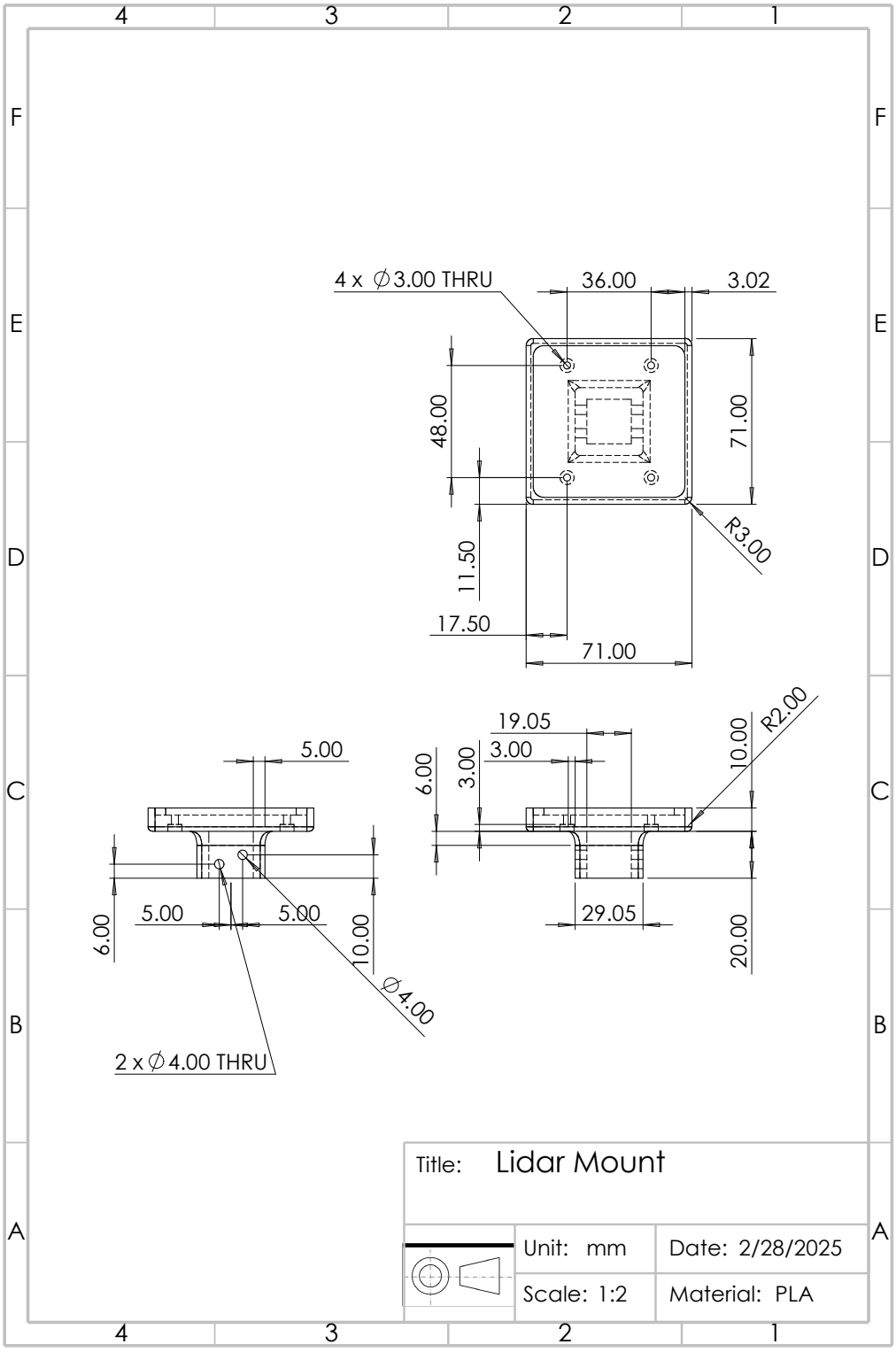


Figure 11: Detailed Drawing of LiDAR Mount